

PROGRAMMĒŠANAS UN DATORIZĒTU SISTĒMU PAMATI

Mācību kursa programma • pasniedzēja rokasgrāmata • skolēna/studentu rokasgrāmata •
v2.0 FINAL

Profesionālās izglītības un universitātes ievadlīmeņa kurss

Algoritmi • Python • C++ • SQL • IoT • mākoņdati • mehatronika • robotika • autonomās sistēmas

Dokumenta statuss	v2.0 FINAL • gala redakcija 18.08.2026
Pamatkursa apjoms	36 × 90 min = 54 astronomiskās stundas = 72 akadēmiskās stundas
● PAMATI	Obligātais koledžas/profesionālās izglītības saturs un praktiskie uzdevumi
● PROFESIONĀLI	Paplašinātā integrācija: SQL, IoT, mehatronika, robotika, industriālās sistēmas
● UNIVERSITĀTE / PASNIEDZĒJS	Padziļinātā matemātika, vadība, ROS 2, SLAM, AI, Safety, Digital Twin un pašmācība

Lietošanas piezīme. Pamatkursa mācību plāns ir paredzēts 36 nodarbībām. Paplašinātās universitātes, robotikas, drošības un Digital Twin nodaļas ir modulārs turpinājums un nav obligāti jāiekļauj vienā mācību gadā. “Teacher Note” stila bloki ir paredzēti pasniedzējam; laboratoriju darba lapas un ātrās atsauces — skolēnam/studentam.

Saturs

1. Kursa koncepcija un pielietojums.....	5
2. Mērķauditorija, apjoms un kursa arhitektūra.....	5
3. Mērķi un sasniedzamie rezultāti.....	6
4. Vērtēšana un mācību metodika.....	6
5. Gada mācību plāns.....	7
6. Pirmais semestris: algoritmi un Python.....	9
7. Otrais semestris: C++ un sistēmu integrācija.....	22
8. SQL, IoT un mākoņdati.....	30
9. Mehatronikas pamati.....	34
10. Robotikas pamati.....	36
11. Python robotikā un kinemātikā.....	37
12. LiDAR, Machine Vision, IMU un sensoru sapludināšana.....	39
13. Kā domā autonomi robots?.....	41
14. ROS 2, SLAM un navigācijas sistēmas.....	44
15. Noslēguma patstāvīgais projekts.....	46
16. Universitātes līmeņa paplašinājums.....	50
17. Skolēna/studentu rokasgrāmatas kodols.....	52
18. Pārbaudes darbu un projektu vērtēšana.....	53
19. Terminoloģijas minimums.....	55
20. [ARHĪVS] Izstrādes statuss un nākamie posmi.....	56
21. Skolēnu darba lapas un diferencētie A/B/C uzdevumi.....	57
22. Pilni pārbaudes darbi un vērtēšanas atslēgas.....	62
23. Ātrās atsauces lapas: Python, C++, SQL, IoT un robotika.....	65
24. Laboratorijas darbu rokasgrāmata.....	70
25. Pasniedzēja/universitātes padziļinājuma pielikums.....	84
26. Kursa apstiprināšanas pielikums un ieviešanas kontrolsaraksts.....	86
27. Laboratoriju skolēna darba lapas un pasniedzēja risinājumi.....	88
28. Projekta inženiertehniskās dokumentācijas veidnes.....	133
29. Mācību procesa pierādījumi, pašvērtējums un portfolio.....	138
30. Paplašinātā mācību grāmatas teorija.....	141
31. Universitātes robotikas lekciju modulis.....	147
32. Aprēķinu piemēri un formulas.....	153
33. Vizuālie sistēmu modeļi un arhitektūras shēmas.....	155
34. Pasniedzēja profesionālās pašmācības modulis.....	157
35. 36 nodarbību pasniedzēja runas scenāriju karte.....	158
36. [ARHĪVS] Versijas 0.6 izstrādes statuss un nākamie soļi.....	162
37. Pilnie 36 nodarbību pasniedzēja runas scenāriji.....	163
38. Fiziskās laboratorijas: Arduino, ESP32, sensori, motori un robots.....	236
39. Tehnisko avotu un aparatūras drošības piezīmes.....	263
40. [ARHĪVS] Versijas 0.7 izstrādes statuss.....	264
41. Izdrukājamās H1–H12 fizisko laboratoriju darba lapas.....	265
42. Etalona programmas: Arduino, ESP32, Python un IoT.....	290
43. 2WD mobilā robota mehāniskā un odometrijas kalibrēšana.....	299
44. [ARHĪVS] Pāreja uz ROS 2: /odom, /imu, /scan un TF.....	303
45. Stāvokļa novērtēšana: EKF un robot_localization.....	310
46. SLAM: kartēšana, lokalizācija un slam_toolbox.....	315
47. Nav2: costmaps, plānotāji, kontrolieri un Behavior Trees.....	319
48. rosbag2: datu reģistrēšana, atkārtošana un eksperimentu analīze.....	324
49. Git un GitHub profesionālai programmatūras izstrādei.....	327
50. Eksāmenu, praktisko uzdevumu un aizstāvēšanas jautājumu banka.....	331
51. Integrētais ROS 2 autonomā robota ceļš: R4–R9.....	341
52. [ARHĪVS] Versijas 0.9 izstrādes statuss un nākamais slānis.....	343
53. R4–R9 izdrukājamās ROS 2 laboratoriju darba lapas.....	344
54. Etalona konfigurācijas: EKF, SLAM un Nav2.....	350
55. URDF/Xacro un TF — mobilā robota digitālais modelis.....	354
56. Robotu simulācija: Gazebo un Webots mācību plūsma.....	357

57. rosbag2 datu analīze ar Python un eksperimentu metriku.....	359
58. Continuous Integration: GitHub Actions, testi un kvalitātes vērti.....	361
59. Visual SLAM un kameras ģeometrija.....	364
60. 3D LiDAR, PointCloud2 un punktu mākoņu apstrāde.....	367
61. Integrētais universitātes projekts — multisensoru autonomi robots.....	370
62. [ARHĪVS] Versijas 1.0 statuss un turpmākā attīstības karte.....	373
63. Factor graphs, nelineārā optimizācija un Bundle Adjustment.....	375
64. Visual-Inertial Odometry: kamera + IMU.....	379
65. 3D LiDAR reģistrācija: ICP un NDT.....	382
66. 3D kartēšana: voxel, occupancy un OctoMap.....	385
67. ROS 2 Lifecycle Nodes, diagnostics un sistēmas veselība.....	388
68. Docker un reproducējama robotikas programmatūras vide.....	391
69. Industriālie protokoli: OPC UA, Modbus un CAN/CANopen.....	394
70. RTOS un FreeRTOS robotikā un mehatronikā.....	397
71. Industriālās sistēmas integrācija: no sensora līdz uzņēmuma datiem.....	400
72. Uzticamība, observability, watchdog un fault injection.....	403
73. Paplašinātie universitātes/industrijas projektu virzieni.....	405
74. v1.1 eksāmenu un aizstāvēšanas jautājumu banka.....	406
75. Pasniedzēja padziļinātais pašmācības ceļš.....	408
76. [ARHĪVS] Versijas 1.1 statuss un nākamā attīstības karte.....	410
77. State-space modeļi un modernās vadības teorijas pamati.....	412
78. System Identification — modeļa iegūšana no mērījumiem.....	415
79. LQR — optimālā stāvokļa atgriezeniskā saite.....	417
80. MPC — Model Predictive Control.....	420
81. Manipulatoru dinamika un torque control.....	422
82. Trajektoriju ģenerēšana, impedance un force control.....	425
83. MoveIt 2 — manipulatoru kustības plānošana un izpilde.....	427
84. v1.2 praktiskās laboratorijas un integrētais projekts.....	432
85. v1.2 eksāmenu un aizstāvēšanas jautājumu banka.....	434
86. Pasniedzēja pašmācības ceļš — advanced control un manipulatori.....	436
87. [ARHĪVS] Versijas 1.2 statuss un nākamais slānis.....	438
88. Deep Learning robotu redzei — pamati un sistēmas domāšana.....	439
89. CNN — konvolūciju neironu tīklu pamati.....	441
90. Dataset inženierija, anotēšana un datu kvalitāte.....	443
91. Transfer learning un fine-tuning robotu redzes uzdevumiem.....	445
92. Object detection — objektu atrašana un klasificēšana.....	447
93. Semantic un instance segmentation.....	449
94. Pose, keypoints, tracking un laika dimensija.....	450
95. Modeļu novērtēšana, kļūdu analīze un robustums.....	451
96. Edge inference, ONNX un reāllaika izvietošana robotā.....	453
97. ROS 2 integrācija robotu redzei.....	455
98. 3D objektu uztvere — RGB-D, LiDAR un sensoru sapludināšana.....	457
99. v1.3 praktiskās AI/robotu redzes laboratorijas.....	460
100. Integrētais projekts — AI uztvere autonomam robotam.....	467
101. v1.3 eksāmenu un aizstāvēšanas jautājumu banka.....	470
102. Pasniedzēja pašmācības ceļš — Deep Learning un robotu redze.....	472
103. [ARHĪVS] Versijas 1.3 statuss un nākamais slānis.....	473
104. Robotu un mehatronisko sistēmu drošības inženierijas pamati.....	474
105. Risku novērtēšana un risku samazināšana.....	476
106. Safety functions, PL/SIL un funkcionālā drošība.....	478
107. Industriālo robotu šūnu drošība — ISO 10218 konteksts.....	480
108. Collaborative robotika un cilvēka–robotu mijiedarbības drošība.....	482
109. Drošības funkciju arhitektūras — E-stop, interlock, STO, safe motion.....	483
110. Verification, validation, traceability un safety case.....	485
111. Kiberdrošības un funkcionālās drošības mijiedarbība.....	487
112. ES mašīnu regulējuma un standartu versiju pārvaldība.....	488
113. Digital Twin pamati un ISO 23247 ietvars.....	489
114. Asset Administration Shell un standartizēta aktīva digitālā reprezentācija.....	491
115. Digital Thread, konfigurācijas pārvaldība un datu izsekojamība.....	492

116. Digital Twin modeļa ticamība, kalibrācija un validācija.....	494
117. Digital Twin mehatronikā, robotikā un autonomās sistēmās.....	496
118. v1.4 praktiskās Safety/Digital Twin laboratorijas.....	498
119. Integrētais projekts — drošs robots ar Digital Twin un pierādījumiem.....	505
120. v1.4 eksāmenu un aizstāvēšanas jautājumu banka.....	507
121. Pasniedzēja pašmācības ceļš — Safety Engineering + Digital Twin.....	509
122. [ARHĪVS] Versijas 1.4 statuss un nākamais slānis.....	510
123. Skolēna/studenta rokasgrāmata — ātrais starts un navigācija.....	511
124. Zināšanu progresijas karte — PAMATI / PROFESIONĀLI / UNIVERSITĀTE.....	513
125. Kā mācīties ar šo rokasgrāmatu — 90 minūšu un pašmācības cikls.....	515
126. Kodola koncepti — algoritmi, Python, C++, SQL, IoT, mehatronika un robotika.....	516
127. Formulu un aprēķinu rokasgrāmata.....	519
128. Ātrās atsaucē kartes — kods, komandas un protokoli.....	521
129. Debugging un problēmu risināšanas ceļvedis.....	523
130. Projektu ceļveži — no idejas līdz aizstāvēšanai.....	524
131. Portfolio un pierādījumu komplekts profesionālai izaugsmei.....	526
132. Terminoloģijas vārdnīca A–Z.....	527
133. Pašvērtējuma un kompetenču kontrolesaraksti.....	532
134. Profesionālās izaugsmes ceļi pēc kursa.....	536
135. Laboratoriju, pārbaudes darbu un projektu navigācijas indekss.....	537
136. [ARHĪVS] Versijas 1.5 statuss un pāreja uz pasniedzēja gala komplektu.....	538
137. Pasniedzēja gala komplekts — lietošanas princips un dokumentu kontrole.....	539
138. 36 nodarbību teacher checklist un minimālais pierādījumu komplekts.....	541
139. Vienota 90 minūšu lesson-delivery karte un nodarbību veidnes.....	544
140. Vērtēšanas sistēmas normalizācija un semestra/gada modelis.....	546
141. Pārbaudes darbu variantu matrica un godīgas vērtēšanas principi.....	548
142. Laboratoriju atbilžu, etalonu un verifikācijas indekss.....	550
143. Noslēguma projekta gala rubrika un aizstāvēšanas protokols.....	552
144. Kursa apstiprināšanas kopsavilkums un ieviešanas pakete.....	554
145. Pasniedzēja sagatavošanās, kvalitātes kontrole un kursa uzturēšana.....	556
146. [ARHĪVS] Versijas 1.6 statuss un pāreja uz v2.0 FINAL.....	558
147. v2.0 FINAL — avoti, standartu statuss un versiju vēsture.....	559

1. Kursa koncepcija un pielietojums

Kurss “Programmēšanas un datorizētu sistēmu pamati” veidots kā starpdisciplinārs ievadkurss, kas sasaista algoritmisko domāšanu, programmēšanu un datu apstrādi ar reālām tehniskām sistēmām. Kursa centrā nav konkrētas programmēšanas valodas sintakses iegaumēšana, bet spēja analizēt problēmu un sistemātiski izveidot risinājumu.

Kursa pamata ķēde

PROBLĒMA → PRASĪBAS → ALGORITMS → KODS → TESTĒŠANA → DATI → SISTĒMAS
INTEGRĀCIJA → DOKUMENTĒŠANA → REZULTĀTA AIZSTĀVĒŠANA

Kursa piemēri primāri balstīti tehniskā vidē: sensori, motori, konveijeri, temperatūras regulēšana, ražošanas iekārtas, datu reģistrēšana, IoT mezgli, mobilie roboti, manipulatori un autonomās sistēmas. Tas ļauj audzēknim saprast programmēšanas profesionālo nozīmi un sagatavo pārejai uz automatizāciju, mehatroniku, robotiku, iegultajām sistēmām un datu inženieriju.

1.1. Trīs dziļuma līmeņi vienā dokumentā

Līmenis	Mērķis	Raksturīgais saturs
A — obligātais	Profesionālās izglītības pamats	Algoritmi, Python, C++ pamati, SQL pamati, IoT principi, projekti.
B — paplašinātais	Profesionālā izaugsme	Git, API, MQTT, Linux, testēšana, mikrokontroleri, mehatronika, sistēmu integrācija.
C — universitātes/pasniedzēja	Padziļināta izpratne	Datu struktūras, sarežģītība, OOP, atmiņa, kinemātika, vadības teorija, SLAM, sensor fusion, ROS 2.

2. Mērķauditorija, apjoms un kursa arhitektūra

Pamata mērķauditorija: audzēkņi pēc 9. klases, kuri mācās tehniskajā koledžā/profesionālās izglītības programmā. Nodarbības ilgums: 90 minūtes, 1 reizi nedēļā. Plānotais mācību gads: aptuveni 34–36 nodarbības; precīzs skaits pielāgojams iestādes kalendāram.

Periods	Apt. nodarbības	Galvenais fokuss
1.–2. mēnesis	8	Algoritmiskā domāšana, blokshēmas, mainīgie, loģika, cikli, dekompozīcija, pārbaudes darbs.
3. mēnesis	4	Python ievads, nosacījumi, cikli, debugging, mini projekts.

Periods	Apt. nodarbības	Galvenais fokuss
2. semestra sākums	8–9	C++ pamati, datu tipi, I/O, nosacījumi, cikli, funkcijas, masīvi, OOP ievads, pārbaudes darbs.
Sistēmu integrācija	3–4	SQL, datu formāti, API, IoT, MQTT/mākonis, tehnisko datu plūsma.
Pēdējie 3 mēneši	10–11	Patstāvīgs integrēts projekts Python + C++, ar izvēles SQL/IoT/robotikas komponentēm.

3. Mērķi un sasniedzamie rezultāti

Kursa mērķis: attīstīt audzēkņu algoritmisko un sistēmisko domāšanu, spēju izstrādāt, analizēt, testēt un dokumentēt programmatūras risinājumus, kā arī izpratni par programmatūras saikni ar sensoriem, aktuatoriem, datubāzēm, tīklu un autonomām tehniskām sistēmām.

- analizēt tehnisku problēmu un sadalīt to apakšuzdevumos;
- izveidot algoritmu tekstā, pseidokodā un blokshēmā;
- izmantot mainīgos, nosacījumus, ciklus, funkcijas un datu struktūras;
- izstrādāt un testēt vienkāršas Python un C++ programmas;
- izmantot SQL pamata vaicājumus un izprast datubāzes struktūru;
- izprast IoT datu ķēdi no sensora līdz mākoņpakalpojumam un datubāzei;
- izprast sensora, kontroliera, aktuatora un atgriezeniskās saites lomu mehatronikā;
- izskaidrot autonomā robota uztveres, lokalizācijas, plānošanas un vadības ciklu;
- izmantot Python vienkāršiem robotu kinemātikas un sensoru datu apstrādes uzdevumiem;
- dokumentēt programmu, testēt robežgadījumus un pamatot izstrādāto risinājumu.

4. Vērtēšana un mācību metodika

Mācību metodika balstās uz problēmorientētu pieeju: jauna konstrukcija tiek ieviesta, sākot no problēmas, kuras atrisināšanai tā ir nepieciešama. Piemēram, cikls tiek ieviests, kad nepieciešams atkārtoti nolasīt sensoru, nevis kā izolēts sintakses elements.

Komponente	Ieteicamais īpatsvars	Ko vērtē
Īsie praktiskie darbi	20 %	Algoritma loģika, koda korektums, spēja izskaidrot.
Pārbaudes darbi	25 %	Teorijas izpratne, algoritma analīze, patstāvīgs risinājums.
Python/C++ mini projekti	20 %	Plānošana, realizācija, testēšana, aizstāvēšana.
Noslēguma projekts	30 %	Integrēta sistēma, dokumentācija, tests, demonstrācija.

Komponente	Ieteicamais īpatsvars	Ko vērtē
Profesionālā iesaiste	5 %	Darba organizācija, Git/žurnāls, refleksija, sadarbība.

AI izmantošanas princips

AI rīkus drīkst izmantot skaidrojumiem, kļūdu meklēšanai, testu idejām un dokumentācijas uzlabošanai, taču audzēknim jāspēj izskaidrot sava risinājuma algoritmu un katru būtisko koda daļu. AI ģenerēts kods bez izpratnes netiek uzskatīts par apgūtu rezultātu.

5. Gada mācību plāns

Nr.	Tēma	Galvenais saturs
1	Algoritms un algoritmiskā domāšana	Algoritma īpašības, ievade–apstrāde–izvade, tehniskie piemēri.
2	Algoritma struktūras un pseidokods	Secība, zarošanās, atkārtotās; pseidokoda veidošana.
3	Blokshēmas	Standarta simboli, tehniska procesa blokshēma.
4	Mainīgie un datu tipi	Skaitļi, teksts, Boolean; datu loma sistēmā.
5	Loģika un nosacījumi	Salīdzināšana, AND/OR/NOT, interlock piemēri.
6	Cikli	FOR/WHILE ideja, bezgalīgs cikls, watchdog ideja.
7	Dekompozīcija un sistēmas algoritms	Lielas problēmas sadalīšana blokos.
8	Pārbaudes darbs: algoritmi	Teorija, analīze, blokshēma, patstāvīgs tehnisks algoritms.
9	Python ievads	Vide, print/input, mainīgie, tipi, aprēķini.
10	Python nosacījumi	if/elif/else, loģiskie operatori, tehnisks lēmumu algoritms.
11	Python cikli un debugging	for/while, kļūdu tipi, testēšana.
12	Python mini projekts	Programmas plāns, realizācija, tests, demonstrācija.
13	C++ un kompilācijas princips	Programmas struktūra, kompilators, main(), cout/cin.

Nr.	Tēma	Galvenais saturs
14	C++ datu tipi un operatori	int, double, bool, char/string, matemātika.
15	C++ zarošanās	if/else, switch, salikti nosacījumi.
16	C++ cikli	for, while, do-while.
17	C++ funkcijas	Parametri, atgrieztā vērtība, dekompozīcija.
18	Masīvi un datu apstrāde	Masīvi, min/max/vidējais, mērījumu dati.
19	OOP ievads	Class/object, Sensor un Motor piemēri.
20	C++ pārbaudes darbs	Sintakse + algoritma realizācija + debugging.
21	SQL pamati	Tabula, ieraksts, lauks, PK; SELECT/INSERT/WHERE.
22	Datu formāti un Python + SQL	CSV/JSON, SQLite ideja, mērījumu uzglabāšana.
23	IoT un MQTT	Sensors–edge–tikls–brokeris–mākonis–datubāze.
24	Sistēmu integrācijas mini darbs	Simulēta datu plūsma: sensors → kods → DB/dashboard.
25	Noslēguma projekta prasības	Problēma, prasības, arhitektūra, izvēlētās tehnoloģijas.
26	Algoritms un sistēmas projekts	Blokshēma, moduļi, datu plūsma, testu plāns.
27	Python projekta realizācija I	Galvenā loģika, datu apstrāde.
28	Python projekta realizācija II	Testi, kļūdu apstrāde, vizualizācija/SQL pēc izvēles.
29	C++ projekta realizācija I	Algoritma pārnese vai aparatūrai tuvā komponente.
30	C++ projekta realizācija II	Funkcijas/OOP, ievade/izvade, testi.
31	IoT/robotikas integrācija	Pēc projekta: sensori, MQTT, simulācija vai robota vadība.
32	Testēšana un validācija	Robežvērtības, atteices, logs, testu protokols.

Nr.	Tēma	Galvenais saturs
33	Dokumentācija	README, sistēmas apraksts, blokhēma, lietotāja instrukcija.
34	Projekta pilnveidošana	Kļūdu labošana, optimizācija, atkārtota testēšana.
35	Projekta aizstāvēšana I	Demonstrācija, tehniskie jautājumi.
36	Projekta aizstāvēšana II un refleksija	Rezultāti, salīdzinājums Python/C++, nākamais attīstības solis.

6. Pirmais semestris: algoritmi un Python

Pirmās astoņas nodarbības paredzētas algoritmiskās domāšanas izveidei pirms intensīvas programmēšanas. Katras 90 minūšu nodarbības ieteicamā struktūra: 10–15 min iepriekšējo zināšanu aktivizēšana, 20–25 min teorija un demonstrācija, 35–40 min vadīts/patstāvīgs praktiskais darbs, 10 min rezultātu analīze un refleksija.

6.1. Nodarbība 1 — Kas ir algoritms un kā domā dators?

Ko stāstīt / teorija:

Algoritms kā galīga, precīzi definētu darbību secība.

Algoritms ir noteiktā secībā sakārtotu darbību kopums, kas paredzēts konkrēta uzdevuma atrisināšanai vai noteikta rezultāta sasniegšanai. Algoritms nav tikai programmēšanas jēdziens: algoritmiski procesi ir sastopami bankomātā, liftā, automātiskajās durvīs, termostātā, veļas mašīnā, ražošanas iekārtā un daudzās citās tehniskās sistēmās.

Lai darbību secību varētu izmantot datorizētā sistēmā, katrai darbībai jābūt formulētai tā, lai izpildītājam nebūtu jāmin, ko autors ir domājis. Komanda “atver durvis” cilvēkam šķiet pietiekama, bet automatizētai sistēmai tā jāsadala darbībās: nolasīt sensoru, pārbaudīt drošības nosacījumus, ieslēgt piedziņu, kontrolēt gala stāvokli, apstādināt motoru un pāriet nākamajā stāvoklī.

Pasniedzējam uzsvērt: dators neizpilda mūsu nodomu - tas izpilda precīzi aprakstītas instrukcijas. Neprecīzs algoritms var radīt ne tikai nepareizu rezultātu, bet tehniskā sistēmā arī mehānisku, procesa vai drošības problēmu.

Algoritma īpašības: diskretums, noteiktība, galīgums, rezultativitāte, pielietojamība līdzīgu problēmu klasei.

Diskretums nozīmē, ka algoritms sastāv no atsevišķiem, secīgi izpildāmiem soļiem. Pat tad, ja process fizikāli šķiet nepārtraukts, programma to apstrādā noteiktos ciklos, stāvokļos vai mērījumu momentos.

Noteiktība nozīmē, ka vienādos sākuma apstākļos algoritma darbībām jābūt nepārprotami definētām. Formulējumi “pagaidi mazliet” vai “ja ir pārāk karsts” nav pietiekami, kamēr nav noteikts laiks vai temperatūras robežvērtība.

Galīgums nozīmē, ka uzdevumu risinošam algoritmam jāspēj nonākt līdz noslēgumam vai skaidri definētam stabilam darba režīmam. Vadības sistēmās pastāv arī apzināti nepārtraukti cikli, tomēr tiem jāparedz STOP, kļūdas, timeout vai cita droša izešanas loģika.

Rezultatīvitate nozīmē, ka algoritma izpildei jārada pārbaudāms rezultāts: aprēķināta vērtība, izvadīts paziņojums, mainīts sistēmas stāvoklis, izpildīta kustība vai pieņemts lēmums.

Pielietojamība līdzīgu problēmu klasei nozīmē, ka labs algoritms nav uzrakstīts tikai vienam skaitlim vai vienam konkrētam gadījumam. Piemēram, temperatūras novērtēšanas algoritms jāspēj izmantot dažādiem sensora mērījumiem, nevis tikai 25 °C vērtībai.

Ievade → apstrāde → izvade.

Šis ir viens no vienkāršākajiem un svarīgākajiem datorizētas sistēmas modeļiem. Ievade ir dati vai signāli, ko sistēma saņem. Apstrāde ir darbības, ko programma ar šiem datiem veic. Izvade ir rezultāts vai komanda, ko sistēma rada.

Automātisko durvju piemērā ievades var būt kustības sensors, drošības sensors un gala slēdži; apstrāde ir vadības loģika, kas pārbauda sensoru stāvokļus un nosaka nākamo darbību; izvades ir motora vadības signāli, indikācija vai trauksme.

Šo modeli vēlāk var paplašināt līdz slēgtai vadības cilpai: sensors → kontrolieris → aktuators → fiziskais process → sensors. Tas ir tilts no algoritmiem uz mehatroniku, automatizāciju un robotiku.

Atšķirība starp cilvēka intuitīvu rīcību un datoram nepieciešamu precīzu instrukciju.

Cilvēks izmanto iepriekšējo pieredzi, kontekstu un veselo saprātu, tāpēc spēj aizpildīt trūkstošu informāciju. Dators pats par sevi šādu kontekstu neizsecina, ja vien tas nav ieprogrammēts vai iegūts no datiem ar konkrētu modeli.

Piemēram, norāde “ja telpā ir tumšs, ieslēdz gaismu” cilvēkam ir saprotama, bet programmai vajag definēt, kā tiek mērīts apgaismojums, kāda skaitliska robeža nozīmē “tumšs”, cik bieži sensors tiek nolasīts un kas notiek, ja sensors ir bojāts.

Šī atšķirība veido algoritmiskās domāšanas pamatu: problēmu jāpārveido no ikdienas valodas uz precīzi pārbaudāmiem nosacījumiem, datiem un darbībām.

Praktiskais darbs un diskusija:

Sadzīves un tehniskie piemēri: bankomāts, lifts, termostats, konveijers, automātiskās durvis.

Praktiskais darbs: aprakstīt automātisko durvju darbību 8–12 precīzos soļos un identificēt ievades/izvades.

6.2. Nodarbība 2 — Secība, zarošanās un atkārtošāns

Ko stāstīt / teorija:

Trīs vadības struktūras: sequence, selection, iteration.

Secība (sequence) nozīmē, ka darbības izpilda vienu pēc otras noteiktā kārtībā. Piemēram: nolasīt temperatūru → aprēķināt starpību → izvadīt rezultātu. Secības maiņa var mainīt rezultātu vai padarīt algoritmu neizpildāmu.

Zarošanās (selection) ļauj algoritmam izvēlēties atšķirīgu rīcību atkarībā no nosacījuma. Tipiska forma ir IF/ELSE: ja nosacījums ir patiess, izpilda vienu darbību, citādi - citu.

Atkārtošanās (iteration) ļauj vienu darbību vai darbību grupu izpildīt vairākas reizes. Atkārtojumu skaits var būt iepriekš zināms vai atkarīgs no nosacījuma.

Pasniedzējam uzsvērt: ļoti lielu daļu programmu var aprakstīt, kombinējot tieši šīs trīs struktūras. Tāpēc to izpratne ir svarīgāka par konkrētas programmēšanas valodas sintaksi.

Pseudokoda veidošanas principi.

Pseudokods ir algoritma apraksts strukturētā, cilvēkam saprotamā formā, kas līdzinās programmēšanas valodai, bet nav piesaistīts konkrētai sintaksei. Tā mērķis ir vispirms sakārtot domāšanu un tikai pēc tam rakstīt kodu.

Labam pseudokodam jābūt secīgam, ar skaidriem darbības vārdiem, redzamām atkāpēm zarošanās un ciklu blokos, saprotamiem mainīgo nosaukumiem un viennozīmīgiem nosacījumiem.

Pseudokods ir īpaši noderīgs pirms Python vai C++ programmēšanas, jo ļauj pārbaudīt algoritmu neatkarīgi no semikoliem, iekavām, datu tipu deklarācijām un citām valodas detaļām.

Nosacījumu nepieciešamība reālā sistēmā.

Reālas sistēmas gandrīz nekad nedarbojas tikai kā nemainīga darbību secība. Tām jāreaģē uz sensoriem, lietotāja komandām, robežvērtībām, kļūmēm un drošības stāvokļiem. Tāpēc nosacījums ir mehānisms, kas savieno mērījumu vai stāvokli ar lēmumu.

Piemēram, sūknis var darboties tikai tad, ja tvertnē nav sasniegts maksimālais līmenis; motors var tikt palaists tikai tad, ja drošības durvis ir aizvērtas; ventilators ieslēdzas tikai tad, ja temperatūra pārsniedz robežvērtību.

Jāuzsver arī robežgadījumi: kas notiek tieši pie robežvērtības, ko darīt ar nederīgu sensora signālu un kā rīkoties, ja vairāki nosacījumi ir pretrunā.

Praktiskais darbs un diskusija:

Uzdevums: izveidot pseudokodu tvertnes uzpildes sistēmai, kur sūknis darbojas līdz līmeņa sensora signālam; papildus variants — kļūdas stāvoklis, ja sensors nedod signālu noteiktā laikā.

6.3. Nodarbība 3 — Blokslēmas

Ko stāstīt / teorija:

Sākums/beigas, process, ievade/izvade, lēmums, bultas.

Blokslēma izmanto standartizētus simbolus, lai vizuāli parādītu algoritma vadības plūsmu.

Sākuma/beigu simbols norāda procesa robežas; procesa taisnstūris apzīmē darbību vai aprēķinu; ievades/izvades simbols apzīmē datu saņemšanu vai rezultāta izvadišanu; rombs attēlo nosacījumu un lēmuma zarus.

Bultas nosaka izpildes virzienu. Tās nav dekorācija - bez skaidras plūsmas nav iespējams saprast, kādā secībā algoritms darbojas. Nosacījuma zaru bultēm ieteicams pierakstīt JĀ/NĒ vai TRUE/FALSE.

Blokshēmai jābūt lasāmai no augšas uz leju vai pa skaidri definētu virzienu, izvairoties no nevajadzīgiem krustojumiem un “lēcieniem”.

Blokshēmas un programmas vadības plūsmas saistība.

Katram blokshēmas elementam vēlāk ir tieša analogija programmā. Procesa bloks bieži kļūst par aprēķinu vai funkcijas izsaukumu, lēmuma rombs - par if/else konstrukciju, bet atpakaļejoša bulta - par ciklu.

Blokshēma tādēļ nav atsevišķs “zīmēšanas uzdevums”. Tā ir programmas struktūras modelis pirms koda rakstīšanas un īpaši noderīga, ja sistēmā ir vairāki stāvokļi vai sarežģīta drošības loģika.

Vēlāk skolēniem var parādīt vienu un to pašu algoritmu trīs formās: blokshēma → pseidokods → Python/C++ kods.

Kā pārbaudīt blokshēmu ar testa datiem.

Blokshēmu iespējams “izpildīt ar roku”, ievadot konkrētas sākuma vērtības un soli pa solim sekojot bultām. Šo procesu var saukt par dry run vai trace. Tas ļauj atrast loģikas kļūdas vēl pirms programmas rakstīšanas.

Jāpārbauda ne tikai tipisks gadījums, bet arī robežgadījumi: vērtība tieši uz robežas, minimālā un maksimālā vērtība, nederīga ievade, sensora kļūda vai stāvoklis, kas var izraisīt bezgalīgu ciklu.

Pasniedzējam uzsvērt: testēšana sākas jau algoritma projektēšanas posmā, nevis tikai pēc tam, kad programma ir uzrakstīta.

Praktiskais darbs un diskusija:

Uzdevums: izveidot motora temperatūras aizsardzības blokshēmu: normāls darbs, brīdinājums, avārijas apstādināšana.

6.4. Nodarbība 4 — Mainīgie un datu tipi

Ko stāstīt / teorija:

Mainīgais kā nosaukta datu vērtība un konceptuāli — vieta atmiņā.

Mainīgais ir nosaukums, ar kuru programmā atsaucas uz noteiktu vērtību. Konceptuāli to var iztēloties kā marķētu vietu atmiņā, kurā programma glabā pašreizējo informāciju.

Mainīgā vērtība programmas darbības laikā var mainīties. Piemēram, temperature var būt 21.4, pēc nākamā sensora nolasījuma 21.7, bet people_count var palielināties, cilvēkam ienākot telpā.

Labs mainīgā nosaukums apraksta nozīmi, nevis tikai datu tipu. Nosaukums sensor_temperature ir informatīvāks par x. Šis ieradums kļūst īpaši svarīgs lielākās programmās un komanddarba projektos.

Integer, real/float, Boolean, string.

Integer jeb vesels skaitlis tiek izmantots skaitītājiem, indeksiem un citām vērtībām bez daļas, piemēram, detaļu skaits vai kļūdu skaits.

Float/real tiek izmantots mērījumiem un aprēķiniem ar decimāldaļu, piemēram, temperatūrai, spiedienam, spriegumam vai attālumam. Jāpiemin, ka datorā peldošā komata skaitļi ir aptuveni un ne visas decimālvērtības iespējams attēlot pilnīgi precīzi.

Boolean satur divus loģiskus stāvokļus - TRUE/FALSE. Tehniskās sistēmās tas ir ļoti dabisks datu tips: door_closed, motor_running, alarm_active.

String ir teksta simbolu virkne, ko izmanto nosaukumiem, statusa ziņojumiem, lietotāja ievadei vai datiem, kas nav paredzēti aritmētiskām darbībām.

Mērījums, mērvienība, datu tips, validācija.

Sensora vērtība nav tikai skaitlis. Lai to pareizi interpretētu, jāzina fiziskais lielums un mērvienība: 25 var nozīmēt 25 °C, 25 %, 25 mm vai 25 bar. Programmas projektēšanā mērvienībai jābūt dokumentētai.

Datu tips nosaka, kā vērtību glabā un kādas darbības ar to drīkst veikt. Taču arī pareiza tipa vērtība var būt fiziski nederīga, tāpēc nepieciešama validācija.

Validācija pārbauda, vai dati atbilst sagaidāmajam diapazonam un formātam. Piemēram, telpas temperatūra -500 °C visticamāk norāda sensora, komunikācijas vai datu apstrādes kļūdu, nevis reālu mērījumu.

Praktiskais darbs un diskusija:

Uzdevums: definēt nepieciešamos mainīgos “viedās telpas” sistēmai: temperature, humidity, door_open, people_count, alarm_message; noteikt datu tipu un pieļaujamās vērtības.

6.5. Nodarbība 5 — Loģika un nosacījumi

Ko stāstīt / teorija:

Salīdzināšanas operatori >, <, >=, <=, ==, !=.

Salīdzināšanas operatori ļauj no skaitliskas vai citas vērtības iegūt loģisku rezultātu TRUE vai FALSE. Tie ir nosacījumu pamats.

Īpaši jāizskaidro atšķirība starp “=” kā piešķiršanas ideju daudzās programmēšanas valodās un “==” kā vienādības pārbaudi. Šī ir viena no tipiskākajām iesācēju kļūdām.

Robežoperatoru izvēle ir svarīga. Nosacījumi temperature > 30 un temperature >= 30 atšķiras tieši pie 30 °C, tāpēc prasībām jābūt formulētām precīzi.

Boolean loģika AND, OR, NOT.

AND ir patiess tikai tad, ja patiesi ir visi apvienotie nosacījumi. To bieži izmanto atļaujas loģikā: motoru drīkst palaist, ja durvis aizvērtas UN drošības ķēde gatava UN nav avārijas.

OR ir patiess, ja patiess ir vismaz viens nosacījums. To var izmantot trauksmei: alarm = high_temperature OR low_pressure OR emergency_stop.

NOT apgriež loģisko vērtību. Tas ļauj ērti izteikt nosacījumus, piemēram, NOT emergency_stop vai NOT door_open.

Jāuzsver operatoru prioritāte un iekavu izmantošana sarežģītās izteiksmēs, lai loģika būtu nepārprotama.

Patiesības tabulas un interlock loģikas princips.

Patiesības tabula sistemātiski parāda visas iespējamo Boolean ieeju kombinācijas un paredzēto rezultātu. Tā ir vienkārša, bet ļoti efektīva metode loģikas pārbaudei.

Interlock ir nosacījums, kas neļauj izpildīt darbību, kamēr nav izpildīti drošības vai procesa priekšnoteikumi. Piemēram, motors nedrīkst palaisties, ja aizsargdurvis ir atvērtas.

Pasniedzējam svarīgi nošķirt mācību loģiku no reālas funkcionālās drošības sistēmas: rūpnieciskā iekārtā drošības funkciju realizācijai var būt nepieciešama sertificēta drošības aparatūra un atbilstošs riska novērtējums.

Praktiskais darbs un diskusija:

Tehniskais piemērs: motoru drīkst palaist tikai tad, ja DoorClosed = TRUE, EmergencyStop = FALSE un SafetyRelayOK = TRUE. Uzdevums: izveidot loģisko izteiksmi un analizēt 8 ieejas kombinācijas.

6.6. Nodarbība 6 — Cikli un atkārtota sistēmas darbība

Ko stāstīt / teorija:

FOR: zināms atkārtojumu skaits.

FOR cikls ir piemērots situācijām, kad atkārtojumu skaits ir iepriekš zināms vai to var noteikt pirms cikla sākuma. Piemēram, nolasīt 10 mērījumus, apstrādāt 100 detaļu ierakstus vai izpildīt darbību katram masīva elementam.

Ciklā parasti ir skaitītājs, sākuma vērtība, beigu nosacījums un skaitītāja maiņa. Vēlāk Python un C++ šo pašu ideju realizē ar atšķirīgu sintaksi.

WHILE: atkārtošana līdz nosacījumam.

WHILE cikls ir piemērots situācijām, kad nav zināms precīzs atkārtojumu skaits, bet ir zināms nosacījums, kamēr darbība jāturpina. Piemēram, “kamēr tvertne nav pilna, turpināt uzpildīt”.

Pirms katras iterācijas tiek pārbaudīts nosacījums. Ja tas jau sākumā ir aplams, cikla ķermenis var netikt izpildīts nevienu reizi.

Jāuzsver, ka cikla nosacījumam programmas darbības laikā jāspēj mainīties, citādi var rasties bezgalīgs cikls.

Bezgalīga cikla risks, timeout un watchdog ideja.

Bezgalīgs cikls rodas, ja izešanas nosacījums nekad neklūst aplams vai sistēma nekad nesaņem gaidīto signālu. Datorprogrammā tas var “iesaldēt” procesu, bet fiziskā sistēmā var izraisīt daudz nopietnākas sekas.

Timeout nosaka maksimāli pieļaujamo gaidīšanas laiku. Piemēram, ja gala sensors nenostādā 5 sekunžu laikā, sistēma nevis turpina darbināt motoru bezgalīgi, bet pāriet kļūdas stāvoklī.

Watchdog ir uzraudzības mehānisms, kas pārbauda, vai programma vai kontrolleris joprojām darbojas paredzētajā režīmā. Ja vadības programmas cikls apstājas vai neatjauno watchdog signālu, sistēma var veikt restartu vai pāriet drošā stāvoklī.

Praktiskais darbs un diskusija:

Uzdevums: algoritms, kas nolasa 10 temperatūras mērījumus, atrod vidējo, min un max. Papildu uzdevums: nepārtraukta nolasīšana līdz STOP komandai.

6.7. Nodarbība 7 — Dekompozīcija, moduļi un sistēmas domāšana

Ko stāstīt / teorija:

Sarežģītu problēmu sadalīšana funkcionālos blokos.

Dekompozīcija nozīmē lielu problēmu sadalīt mazākās, saprotamākās daļās. Tas samazina kognitīvo slodzi un ļauj katru daļu projektēt, programmēt un testēt atsevišķi.

Piemēram, automātisku autostāvvietu var sadalīt transportlīdzekļa noteikšanas, barjeras vadības, vietu uzskaites, displeja, maksājumu un kļūdu apstrādes moduļos.

Šāds domāšanas veids vēlāk pāriet funkcijās, klasēs, programmatūras moduļos, PLC funkcionālajos blokos un robotikas nodes.

Ievades, izejas un atbildības katram modulim.

Katram modulim jābūt skaidri definētam: ko tas saņem, ko tas dara un ko atdod. Šādu aprakstu var uzskatīt par vienkāršotu interfeisu jeb līgumu starp sistēmas daļām.

Piemēram, “vietu skaitītāja” modulis var saņemt event_in un event_out signālus, uzturēt current_count un izvadīt free_spaces. Tam nav pašam jāvada barjeras motors.

Ja moduļa atbildība ir pārāk plaša, programmu kļūst grūtāk testēt un mainīt. Tāpēc viens no profesionālas izstrādes principiem ir skaidrs atbildību sadalījums.

Saikne ar funkcijām, klasēm un vēlāk sistēmu arhitektūru.

Funkcija programmā ir pirmais praktiskais veids, kā dekompozīciju realizēt kodā: viena funkcija veic vienu saprotamu uzdevumu un to var izsaukt no citām programmas daļām.

Klase apvieno datus un ar tiem saistīto uzvedību vienā struktūrā. Piemēram, Sensor klase var saturēt sensora vērtību, statusu, kalibrācijas parametrus un nolasīšanas metodes.

Lielākā sistēmā funkcijas un klases veido moduļus, servisus vai komponentes. Robotikā līdzīgu principu var redzēt ROS 2 mezglos, bet industriālajā automatizācijā - PLC funkciju blokos.

Praktiskais darbs un diskusija:

Grupas uzdevums: sadalīt “automātisko autostāvvietu” moduļos: transportlīdzekļa noteikšana, barjera, vietu skaitīšana, displejs, kļūdu stāvokļi.

6.8. Nodarbība 8 — Pārbaudes darbs: algoritmi

Ko stāstīt / teorija:

20 % teorija.

Teorijas daļa pārbauda pamatjēdzienu izpratni, nevis definīciju mehānisku iegaumēšanu.

Jautājumiem jāaptver algoritma jēdziens, pamatīpašības, vadības struktūras, mainīgie, nosacījumi un cikli.

Ieteicams kombinēt īsas definīcijas ar jautājumiem “kāpēc?” un “kur to izmantotu?”, lai skolēns demonstrētu izpratni un spēju sasaistīt terminus ar tehnisku sistēmu.

20 % dota algoritma analīze.

Analīzes uzdevumā skolēns saņem gatavu pseidokodu, blokshēmu vai īsu algoritma aprakstu un prognozē tā darbību ar konkrētiem ievades datiem.

Šāda daļa pārbauda spēju izsekot vadības plūsmai, pamanīt nosacījumus, ciklus un kļūdas. Tas ir būtisks debugging priekšnosacījums.

30 % blokshēma.

Blokshēmas uzdevumā skolēns no teksta prasībām izveido vizuālu algoritma modeli. Vērtē ne tikai simbolu pareizību, bet arī procesa loģiku, lēmumu zarus, plūsmas nepārtrauktību un robežgadījumu apstrādi.

Augstāku vērtējumu var piešķirt par drošu kļūdu stāvokli, timeout vai skaidri modelētu atkārtoto, ja tas atbilst uzdevumam.

30 % patstāvīgs tehnisks algoritms.

Patstāvīgajā daļā skolēns saņem tehnisku problēmu un pats izvēlas algoritma struktūru. Šeit galvenais ir problēmas sadalīšana, nepieciešamo ievades/izvades datu identificēšana, lēmumu formulēšana un risinājuma pamatotība.

Vērtēšanas kritērijos jāatstāj vieta vairākiem korektiem risinājumiem. Programmēšanā un inženierijā viena problēma bieži var tikt atrisināta vairākos veidos, ja risinājums ir loģiski pareizs, pārbaudāms un atbilst prasībām.

Praktiskais darbs un diskusija:

Ieteicamais praktiskais uzdevums: ventilācijas vadība ar histerēzi — ieslēgt virs 28 °C, izslēgt zem 24 °C; starpstāvoklī saglabāt iepriekšējo stāvokli. Augstāka līmeņa variants: sensora kļūdas stāvoklis.

6.9. Python ievada bloks (nodarbības 9–12)

Nodarbība	Teorija	Prakse
9. Python vide un datu plūsma	Interpreteris, skripts, print(), input(), mainīgie, tipi, operatori.	Temperatūras konvertētājs; sensora mērījumu kalkulators.
10. Nosacījumi	if/elif/else, Boolean loģika, validācija.	Iekārtas statusa noteikšana pēc temperatūras/spiediena.

Nodarbība	Teorija	Prakse
11. Cikli un debugging	for/while, syntax/runtime/logic errors, test cases.	10 mērījumu apstrāde; speciāli bojāta koda labošana.
12. Python mini projekts	Prasības, algoritms, kods, tests, demonstrācija.	Iekārtas vadības simulators / viedās telpas monitors / datu analizators.

```

temperature = float(input('Temperature: '))
if temperature > 30:
    print('Cooling ON')
elif temperature < 24:
    print('Cooling OFF')
else:
    print('Keep previous state')

```

6.9.1. Python mācīšanas princips: algoritms nemainās, mainās realizācijas valoda

Python blokā skolēnam nevajag sākt domāt no nulles. Pirmo divu mēnešu algoritmi tiek pārrakstīti izpildāmā kodā. Tādēļ katra jauna sintakses konstrukcija tiek piesaistīta jau zināmai algoritma struktūrai: secība kļūst par komandu virkni, zarošanās — par if/elif/else, atkārtošanos — par for/while, bet dekompozīcija — par funkcijām un moduļiem.

Python ir īpaši piemērots pirmajai programmēšanas pieredzei, jo sintakse ļauj koncentrēties uz algoritmu un datu plūsmu. Vienlaikus jau no pirmās nodarbības jāveido profesionāli ieradumi: saprotami mainīgo nosaukumi, mērvienības, ievades validācija, mazi testējami soļi un spēja izskaidrot katru koda rindu.

Attēls. Functional tree — tehnisko datu apstrāde ar Python.

Nodarbība 9 — Python vide, programmas izpilde un datu plūsma

Sasniedzamie rezultāti:

- Atšķir skriptu, interpreteru un izpildes vidi.
- Izmanto print(), input(), mainīgos, datu tipu pārveidošanu un aritmētiskos operatorus.
- Spēj sasaistīt ievade → apstrāde → izvade modeli ar konkrētu Python programmu.

Interpreteris, skripts un programmas izpildes secība.

Python avota kods ir cilvēkam lasāms teksts. Interpreteris nolasa programmas instrukcijas un organizē to izpildi. Iesācējam svarīgākā doma: programma tiek izpildīta noteiktā plūsmā, un katra rinda var mainīt programmas stāvokli.

Atšķirībā no C++ kursa sākumā nav nepieciešams detalizēti analizēt kompilāciju. Tomēr jāizskaidro, ka arī Python izpildei nepieciešama programmatūras vide un ka kļūda vienā instrukcijā var pārtraukt izpildi.

Tehniskais piemērs: Temperatūras pārveidotājs saņem °C, aprēķina K vai °F un parāda rezultātu.

Ko pasniedzējam uzsvērt: Vispirms izsekojam programmas stāvoklim uz papīra: kāda vērtība ir katrā mainīgajā pēc katras rindas.

print(), input() un datu tipu pārveidošana.

input() rezultāts sākotnēji ir teksts. Ja vēlamies veikt matemātiku, nepieciešama skaidra tipa konvertēšana, piemēram, int() vai float(). Tas ir labs brīdis, lai atkārtotu datu tipu jēdzienu un parādītu, kāpēc nepareizs tips rada kļūdu vai nepareizu rezultātu.

print() nav tikai teksta parādīšana; tā ir vienkāršākā diagnostikas un novērošanas metode. Profesionālā darbā to aizstāj strukturēti logi, taču pamatprincips ir tas pats — padarīt programmas iekšējo stāvokli redzamu.

Tehniskais piemērs: Operators ievada sensora mērījumu; programma pārbauda tipu un aprēķina novirzi no setpoint.

Ko pasniedzējam uzsvērt: Vienmēr prasīt: kāda ir ievades mērvienība un kāds datu tips to glabās?

Mainīgie, nosaukumi un mērvienības.

Mainīgā nosaukumam jāpasaka, ko vērtība nozīmē. x var būt pieņemams īsā matemātiskā formulā, bet tehniskā programmā labāk izmantot temperature_c, motor_speed_rpm vai pressure_bar. Tas samazina kļūdu risku un uzlabo koda uzturējamību.

Mērvienība pati par sevi Python tipā netiek glabāta. Tādēļ mērvienības jāpadara redzamas nosaukumos, komentāros, dokumentācijā vai specializētās bibliotēkās.

Tehniskais piemērs: temperature_c = 24.5 ir daudz drošāks nosaukums nekā t = 24.5, ja sistēmā ir vairāki temperatūras avoti.

Ko pasniedzējam uzsvērt: Programmēšanas kļūda var būt arī inženiertehniska kļūda — piemēram, sajaukt °C ar K vai mm ar m.

90 minūšu nodarbības scenārijs:

- 10 min — Iepriekšējo algoritmu tēmu aktivizēšana un mērķa formulēšana.
- 20 min — Demonstrācija: pirmais Python skripts un programmas izpildes izsekošana.
- 20 min — Vadīts darbs ar input(), float(), aritmētiku un formatētu print().
- 30 min — Patstāvīgais tehniskais uzdevums un 2–3 testa gadījumi.
- 10 min — Kļūdu analīze, refleksija un īss exit-ticket.

Praktiskais darbs: Izveidot “sensora mērījumu kalkulatoru”: ievadīt 3 temperatūras mērījumus, aprēķināt vidējo, starpību starp max/min un formatēti izvadīt rezultātu ar mērvienībām.

Patstāvīgais darbs / mājasdarbs: Pārveidot vienu no iepriekš zīmētajiem algoritmiem 8–15 rindās Python kodā.

Universitātes/pasniedzēja paplašinājums: Paplašināti: CPython izpildes modelis, bytecode ideja, virtual environment un reproducējama izstrādes vide.

Nodarbība 10 — Nosacījumi, Boolean loģika un datu validācija Python

Sasniedzamie rezultāti:

- Raksta if/elif/else konstrukcijas.
- Izmanto salīdzināšanas un loģiskos operatorus.
- Veido validāciju un atšķir normālu, brīdinājuma un kļūdas stāvokli.

if/elif/else kā algoritma zarošanās realizācija.

Nosacījums ir Boolean izteiksme, kuras rezultāts ir True vai False. Programma izvēlas vienu izpildes zaru un ignorē pārējos. Tas ļauj algoritmam reaģēt uz datiem un sistēmas stāvokli.

elif palīdz veidot savstarpēji izslēdzošus diapazonus. Svarīga ir nosacījumu secība — pārāk plašs nosacījums augstāk var padarīt zemāku zaru nenasniedzamu.

Tehniskais piemērs: Temperatūras statuss: LOW / NORMAL / HIGH / SENSOR_ERROR.

Ko pasniedzējam uzsvērt: Likst skolēniem ar konkrētu skaitli izsekot, kurš zars tiks izpildīts un kāpēc.

AND, OR, NOT tehniskā loģikā.

AND prasa, lai visi drošībai nepieciešamie nosacījumi būtu izpildīti. OR ļauj reaģēt uz jebkuru no vairākiem trigeriem. NOT apgriež loģisko vērtību.

Tehniskā sistēmā loģiku ieteicams veidot lasāmi: atsevišķi aprēķināt door_closed, e_stop_ok, pressure_ok un tikai tad kombinēt machine_can_start.

Tehniskais piemērs: machine_can_start = door_closed and e_stop_ok and pressure_ok.

Ko pasniedzējam uzsvērt: Kompleksu nosacījumu nevajag slēpt vienā ļoti garā if rindā, ja to var sadalīt saprotamos Boolean mainīgajos.

Ievades validācija un kļūdas stāvoklis.

Validācija atbild uz jautājumu: vai saņemtie dati vispār ir ticami un pieļaujami? Temperatūra -999 °C var būt sintaktiski derīgs skaitlis, bet fiziski norāda uz sensora kļūdu vai nekorektu ievadi.

Validāciju jāatdala no procesa lēmuma. Vispirms pārbaudām datu kvalitāti, tad izmantojam datus vadības algoritmā.

Tehniskais piemērs: Ja sensors paredzēts -40...125 °C, vērtība 250 °C tiek klasificēta kā SENSOR_ERROR, nevis tikai HIGH.

Ko pasniedzējam uzsvērt: Kļūdas stāvoklis nedrīkst automātiski tikt interpretēts kā normāls process.

90 minūšu nodarbības scenārijs:

- 10 min — Atkārtojums: zarošanās blokhēmā un pseidokodā.
- 20 min — if/elif/else demonstrācija ar stāvokļu tabulu.
- 20 min — AND/OR/NOT un drošības interlock piemērs.
- 30 min — Patstāvīgs iekārtas starta atļaujas uzdevums.
- 10 min — Testu gadījumi un kļūdu scenāriju apspriešana.

Praktiskais darbs: Izveidot iekārtas starta atļaujas programmu ar 3–4 drošības nosacījumiem un sensora diapazona validāciju.

Patstāvīgais darbs / mājasdarbs: Sagatavot vismaz 6 testa gadījumus, kas aptver katru programmas zaru.

Universitātes/pasniedzēja paplašinājums: Paplašināti: truth tables, short-circuit evaluation, guard clauses un state-machine pāreju validācija.

Nodarbība 11 — Cikli, datu kopas, debugging un testēšanas domāšana

Sasniedzamie rezultāti:

- Izmanto for un while atbilstoši uzdevumam.

- Apstrādā mērījumu virkni un aprēķina statistiku.
- Atšķir sintakses, izpildes un loģikas kļūdas; izmanto sistemātisku debugging pieeju.

for — atkātojums pār zināmu elementu kopu.

for ir ērts, ja atkātojumu skaits vai datu kopums ir zināms. Python for bieži iterē nevis “skaitītāju” abstrakti, bet tieši pār kolekcijas elementiem.

Mērījumu datu apstrādē for ļauj vienā algoritmā apstrādāt 10 vai 10 000 ierakstu, nemainot pašu loģikas principu.

Tehniskais piemērs: Aprēķināt 10 temperatūras mērījumu vidējo un saskaitīt, cik pārsniedz brīdinājuma robežu.

Ko pasniedzējam uzsvērt: Likst skolēnam paskaidrot, kas ir viena cikla iterācija un kas mainās pēc katras iterācijas.

while — atkātojums līdz stāvokļa maiņai.

while izmanto, ja atkātojumu skaits iepriekš nav zināms. Cikls turpinās, kamēr nosacījums ir patiess. Šī konstrukcija labi modelē vadības sistēmas, izvēlnes un gaidīšanu uz notikumu.

Galvenais risks ir nosacījums, kas nekad neklūst False. Tādēļ tehniskā sistēmā jādomā par timeout, maksimālo mēģinājumu skaitu vai ārēju stop nosacījumu.

Tehniskais piemērs: Kamēr pressure_bar < target_pressure, turpināt simulēt uzpildi; pēc timeout pāriet ERROR.

Ko pasniedzējam uzsvērt: Bezgalīgs cikls nav tikai “programma iestrēga”; iekārtas vadībā tas var nozīmēt nekontrolētu darbību.

Debugging kā hipotēžu pārbaude.

Debugging nav nejauša rindu mainīšana. Vispirms definē sagaidāmo rezultātu, reproducē kļūdu, atrod vietu, kur faktiskā izpilde sāk atšķirties, un pārbauda vienu hipotēzi vienlaikus.

Vienkāršākie rīki: print diagnostika, mainīgo vērtību pieraksts, mazi testa dati. Vēlāk — debugger, breakpoints, watch un logi.

Tehniskais piemērs: Dota programma nepareizi skaita brāķa procentu. Skolēns atrod pirmo rindu, kur rezultāts kļūst nepareizs.

Ko pasniedzējam uzsvērt: Atšķirt “kods darbojas” no “kods dod pareizo rezultātu visos svarīgajos gadījumos”.

90 minūšu nodarbības scenārijs:

- 10 min — for un while izvēles kritēriji.
- 20 min — Vadīta mērījumu cikla izveide.
- 20 min — Bezgalīga cikla/timeout demonstrācija.
- 30 min — Debugging laboratorija ar bojātu kodu.
- 10 min — Testu tabulas aizpildīšana un secinājumi.

Praktiskais darbs: Apstrādāt 10–20 mērījumus: vidējais, min, max, alarm_count. Pēc tam saņemt speciāli bojātu programmu un dokumentēt atrastās kļūdas.

Patstāvīgais darbs / mājasdarbs: Izveidot testa tabulu: input, expected, actual, pass/fail vismaz 5 gadījumiem.

Universitātes/pasniedzēja paplašinājums: Paplašināti: exception handling, assert, pytest ievads, logging līmeņi un deterministiska testa datu ģenerēšana.

Nodarbība 12 — Python mini projekts — no prasības līdz demonstrācijai

Sasniedzamie rezultāti:

- Noformulē problēmu un 3–5 funkcionālās prasības.
- Izveido functional tree vai blokshēmu pirms programmēšanas.
- Realizē, testē un īsi aizstāv Python programmu.

Prasība nav tas pats, kas risinājums.

Prasība apraksta, ko sistēmai jāspēj izdarīt, nevis obligāti to, kā tieši tā tiks realizēta. Piemēram, “sistēmai jābrīdina par temperatūru virs 30 °C” ir prasība; “izmantot `if temperature > 30`” jau ir realizācijas detaļa.

Šī atšķirība sagatavo skolēnu profesionālai programmatūras un sistēmu izstrādei, kur prasības, dizains un implementācija tiek pārvaldīti atsevišķi.

Tehniskais piemērs: Viedās telpas monitors: mērīt/simulēt T un RH , noteikt statusu, saglabāt vai parādīt rezultātu.

Ko pasniedzējam uzsvērt: Pirms koda prasīt skolēnam parādīt, kā viņš pierādīs, ka prasība ir izpildīta.

Mini projekta dokumentācijas minimums.

Pat nelielam darbam jā saglabā problēmas apraksts, ievades/izvades, algoritms, izmantotie mainīgie, testa gadījumi un rezultāts. Šī dokumentācija vēlāk kļūs par noslēguma projekta README un testu protokola pamatu.

Tehniskais piemērs: Vienas lapas projekta kartīte + kods + 5 testu tabula.

Ko pasniedzējam uzsvērt: Vērtēt ne tikai gala ekrānu, bet arī domu ķēdi no prasības līdz testam.

90 minūšu nodarbības scenārijs:

- 10 min — Projekta uzdevuma izvēle un prasību precizēšana.
- 15 min — Functional tree/blokshēma un testa plāns.
- 45 min — Patstāvīga realizācija ar konsultācijām.
- 10 min — Testēšana un kļūdu labošana.
- 10 min — Īsās demonstrācijas/refleksija; ja grupa liela, demonstrācijas pabeidz nākamajā konsultācijā.

Praktiskais darbs: Izvēlēties vienu: iekārtas statusa simulators, viedās telpas monitors, mērījumu datu analizators vai neliels diagnostikas rīks. Sagatavot 3–5 min demonstrāciju.

Patstāvīgais darbs / mājasdarbs: Pēc demonstrācijas pievienot 3 teikumus: kas izdevās, kas bija grūtākais, ko uzlabotu.

Universitātes/pasniedzēja paplašinājums: Paplašināti: failu struktūra, funkcijas, dataclasses, type hints, Git commit vēsture un automatizēti testi.

6.9.6. Python mini projekta vērtēšanas kritēriji

Kritērijs	Punkti	Pierādījums
Prasību un algoritma skaidrība	20	Functional tree/blokshēma, ievades un izejas.

Kritērijs	Punkti	Pierādījums
Python sintakse un datu tipi	20	Kods izpildās, tipi un konvertēšana lietoti korekti.
Loģika un cikli	20	Nosacījumi/cikli atbilst algoritmam, nav nevajadzīgas dublēšanās.
Testēšana un kļūdu apstrāde	20	Vismaz 5 testi, robežgadījums un viena nederīga ievade.
Demonstrācija un skaidrojums	20	Skolēns spēj izskaidrot būtiskās koda rindas un rezultātu.

7. Otrais semestris: C++ un sistēmu integrācija

C++ tiek mācīts kā otrā valoda, salīdzinot jau zināmo algoritmu ar stingrāku sintaksi, statisku tipu sistēmu, kompilāciju un aparatūrai tuvāku programmas izpildi. Galvenā pedagogiskā doma: algoritms nemainās — mainās realizācijas valoda un programmētāja kontroles līmenis.

Tēma	Paplašinātais skaidrojums	Tehniskais piemērs
Programmas struktūra	Preprocesors, main(), kompilācija, izpildfails.	Vienkāršs iekārtas statusa terminālis.
Datu tipi	int, double, bool, char/string; diapazons un precizitāte.	Sensoru un skaitītāju dati.
I/O un operatori	cin/cout, aritmētika, salīdzināšana.	Ražošanas apjoma un brāķa % aprēķins.
Nosacījumi	if/else, switch, kombinētie nosacījumi.	Režīmu izvēle: AUTO/MANUAL/STOP.
Cikli	for, while, do-while.	Mērījumu cikls un operatora izvēlne.
Funkcijas	Parametri, return, moduļu veidošana.	readSensor(), checkAlarm(), calculateAverage().
Masīvi	Fiksēts datu kopums, iterācija.	10–100 mērījumu statistika.
OOP ievads	class/object, īpašības, metodes, konstruktors.	Sensor, Motor, Robot komponentes.

```
class Sensor {
public:
    double value;
    bool alarm;
    void update(double newValue) {
        value = newValue;
        alarm = value > 30.0;
    }
};
```

7.1. C++ mācīšanas princips: tas pats algoritms ar lielāku kontroli pār tiem un programmas struktūru

C++ blokā jāizmanto nepārtraukta salīdzināšana ar Python. Skolēnam jāsaprot, ka nosacījuma, cikla vai funkcijas algoritmiskā jēga nemainās. Mainās sintakse, datu tipu deklarēšana, kompilācijas process un programmētāja atbildība par programmas struktūru un resursiem.

Tehniskās izglītības kontekstā C++ dod tiltu uz mikrokontrolleriem, iegultajām sistēmām, robotiku un augstas veiktspējas programmatūru. Tādēļ piemēros priekšroka tiek dota sensoriem, motoriem, iekārtu stāvokļiem, mērījumu masīviem un vienkāršiem komponentu modeļiem.

Attēls. Functional tree — tehniska vadības programma C++.

Nodarbība 13 — C++ programmas struktūra, kompilācija un pirmā programma

Sasniedzamie rezultāti:

- Atšķir avota kodu, kompilatoru, linker un izpildfailu konceptuālā līmenī.
- Izveido main() programmu ar cout/cin.
- Salīdzina Python un C++ izpildes modeli iesācēja līmenī.

Avota kods → kompilācija → izpildfails.

C++ avota kods pirms palaišanas tiek tulkots mašīnai izpildāmā formā. Kompilators pārbauda sintaksi un tipu atbilstību, bet linker savieno nepieciešamās programmas daļas un bibliotēkas.

Šis process nozīmē, ka daudzas kļūdas tiek atklātas pirms programmas palaišanas. Kompilatora kļūdas paziņojums ir diagnostikas informācija, nevis “sods”; studentam jāiemācās to lasīt no pirmās būtiskās kļūdas.

Tehniskais piemērs: Vienkāršs termināļa rīks saņem mērījumu un izvada statusu.

Ko pasniedzējam uzsvērt: Iemācīt atšķirt compile-time error no runtime/logic error.

main(), bloki un semikoli.

main() ir tipiska programmas ieejas funkcija. Figūriekavas definē blokus, bet semikols noslēdz daudzus paziņojumus. Šī stingrā struktūra sākumā rada vairāk sintakses kļūdu, bet palīdz skaidri redzēt programmas robežas.

Indentācija C++ nav sintaktiski obligāta tāpat kā Python, taču tā ir obligāta profesionālai lasāmībai.

Tehniskais piemērs: if bloks ar skaidri redzamām figūriekavām pat vienas rindas gadījumā.

Ko pasniedzējam uzsvērt: Stilu māca no pirmās dienas, nevis pēc tam “sakārto”.

90 minūšu nodarbības scenārijs:

- 10 min — Python/C++ salīdzinājums.
- 25 min — Kompilācijas procesa skaidrojums un pirmā programma.
- 20 min — cin/cout un aritmētika.
- 25 min — Patstāvīgs pārneses uzdevums.
- 10 min — Kompilatora kļūdu lasīšana.

Praktiskais darbs: Pārrakstīt Python temperatūras kalkulatoru C++ formā un salīdzināt abus kodus rindu pa rindai.

Patstāvīgais darbs / mājasdarbs: Uzrakstīt 5 atšķirības starp Python un C++ pirmās programmas realizāciju.

Universitātes/pasniedzēja paplašinājums: Paplašināti: translation units, header/source faili, linker simboli, build systems (CMake) un toolchain jēdziens.

Nodarbība 14 — C++ datu tipi, operatori, const un precizitāte

Sasniedzamie rezultāti:

- Izvēlas int/double/bool/string atbilstoši datiem.
- Saprot veselo un peldošā punkta skaitļu atšķirību.
- Izmanto const nemainīgām konfigurācijas vērtībām.

Statiska tipu sistēma.

C++ mainīgā tips parasti tiek noteikts deklarācijas brīdī. Tas ļauj kompilatoram pārbaudīt daudzas neatbilstības un efektīvi organizēt atmiņu.

Skolēnam jāiemācās izvēlēties tipu pēc datu nozīmes, diapazona un precizitātes, nevis tikai “kas kompilējas”.

Tehniskais piemērs: int part_count; double temperature_c; bool alarm_active;

Ko pasniedzējam uzsvērt: Tips ir daļa no tehniskā dizaina.

Skaitļu diapazons un precizitāte.

Veselo skaitļu tipiem ir ierobežots diapazons. Peldošā punkta skaitļi nav precīzs visu decimāldaļu attēlojums; tie ir binārs tuvinājums. Tādēļ tiešs equality salīdzinājums dažos aprēķinos var būt problemātisks.

Iesācēja līmenī pietiek ar intuīciju, ka mērījumam 24.73 vajag double, bet detaļu skaitam 125 — veselo tipu.

Tehniskais piemērs: Ražošanas detaļu skaits nevar būt 12.7; temperatūra var.

Ko pasniedzējam uzsvērt: Saistīt datu tipu ar fiziskās vērtības jēgu un sensoru izšķirtspēju.

const un konfigurācijas vērtības.

const norāda, ka vērtību pēc inicializācijas nevajadzētu mainīt. Tas padara koda nolūku skaidrāku un pasargā no nejaušas konfigurācijas parametra pārrakstīšanas.

Tehniskais piemērs: const double MAX_TEMPERATURE_C = 80.0;

Ko pasniedzējam uzsvērt: Atšķirt procesu mainīgos no konfigurācijas robežām.

90 minūšu nodarbības scenārijs:

- 10 min — Datu tipu izvēles diskusija.
- 25 min — int/double/bool/string un operatori.
- 15 min — const un mērvienības.
- 30 min — Ražošanas datu uzdevums.

- 10 min — Robežgadījumi: zero total, negatīva ievade.

Praktiskais darbs: Izveidot ražošanas maiņas kalkulatoru: total_count, reject_count, reject_percent un procesa statuss.

Patstāvīgais darbs / mājasdarbs: Pievienot validāciju, lai reject_count nevarētu pārsniegt total_count.

Universitātes/pasniedzēja paplašinājums: Paplašināti: fixed-width integer types, overflow/underflow, IEEE 754, constexpr un units bibliotēku ideja.

Nodarbība 15 — C++ zarošanās, switch un sistēmas stāvokļi

Sasniedzamie rezultāti:

- Realizē if/else un switch.
- Veido saliktus drošības nosacījumus.
- Atšķir stāvokli, režīmu un procesa mērījumu.

if/else un salikti nosacījumi.

Algoritmiskā zarošanās tiek realizēta ar Boolean izteiksmēm. C++ operatori &&, || un ! atbilst AND, OR un NOT loģikai.

Drošības nosacījumus vēlams sadalīt nosauktos Boolean mainīgajos, lai programmas nolūks būtu pārskatāms.

Tehniskais piemērs: bool canStart = doorClosed && eStopOk && pressureOk;

Ko pasniedzējam uzsvērt: Lasāmība ir daļa no drošuma.

switch diskrētiem režīmiem.

switch ir piemērots, ja viena mainīgā vērtība izvēlas vienu no vairākiem diskrētiem režīmiem. To nevajag izmantot diapazonu pārbaudei, kur if/else parasti ir skaidrāks.

Režīmu piemēri: STOP, MANUAL, AUTO, SERVICE. Vēlāk to var sasaistīt ar enum un state machine.

Tehniskais piemērs: Operatora izvēlne pārslēdz AUTO/MANUAL/STOP.

Ko pasniedzējam uzsvērt: Atšķirt režīma izvēli no drošības interlock — AUTO režīms pats par sevi vēl nedod starta atļauju.

90 minūšu nodarbības scenārijs:

- 10 min — Boolean loģikas atkārtojums.
- 20 min — if/else C++ sintakse.
- 20 min — switch un režīmu modelis.
- 30 min — Iekārtas režīmu simulators.
- 10 min — Testu tabula visiem režīmiem.

Praktiskais darbs: Izveidot iekārtas režīmu simulatoru ar switch un atsevišķu starta drošības pārbaudi.

Patstāvīgais darbs / mājasdarbs: Pievienot SERVICE režīmu, kurā atļautas tikai noteiktas darbības.

Universitātes/pasniedzēja paplašinājums: Paplašināti: enum class, guard conditions, finite state machines un transition tables.

Nodarbība 16 — C++ cikli, izvēlnes un atkārtota procesa vadība

Sasniedzamie rezultāti:

- Izmanto for, while un do-while.
- Izvēlas cikla tipu pēc procesa loģikas.
- Pievieno drošu izešanu un timeout/maksimālā mēģinājuma principu.

for, while, do-while semantika.

for ir kompakts cikls skaitītājiem un datu kopām. while pārbauda nosacījumu pirms iterācijas, bet do-while garantē vismaz vienu izpildi.

Izvēlnēs do-while var būt intuitīvs, jo izvēlne vispirms tiek parādīta un tikai pēc tam pārbaudīts, vai lietotājs izvēlējies EXIT.

Tehniskais piemērs: Mērījumu cikls + operatora izvēlne.

Ko pasniedzējam uzsvērt: Jautāt nevis “kuru ciklu zini?”, bet “kāds ir procesa pārtraukšanas nosacījums?”.

Droša cikla pārtraukšana.

break var pārtraukt ciklu, bet to jālieto apzināti. Tehniskā vadībā vēl svarīgāk ir modelēt stop nosacījumus un timeout tā, lai nevar rasties nekontrolēta bezgalīga darbība.

Tehniskais piemērs: Mēģināt nolasīt sensora vērtību ne vairāk kā 3 reizes, tad ERROR.

Ko pasniedzējam uzsvērt: Vienmēr definēt, kas notiek, ja gaidītais notikums nenotiek.

90 minūšu nodarbības scenārijs:

- 10 min — Ciklu salīdzinājums.
- 20 min — for piemēri.
- 20 min — while/do-while izvēlne.
- 30 min — Patstāvīga diagnostikas izvēlne.
- 10 min — Bezgalīga cikla un timeout analīze.

Praktiskais darbs: Izveidot konsoles programmu, kas atkārtoti ļauj ievadīt mērījumus, skatīt statistiku vai iziet.

Patstāvīgais darbs / mājasdarbs: Pievienot maksimālo mērījumu skaitu vai drošu ERROR stāvokli.

Universitātes/pasniedzēja paplašinājums: Paplašināti: range-based for, iterators, event loop un real-time control loop perioda jēdziens.

Nodarbība 17 — Funkcijas, parametri un programmas dekompozīcija

Sasniedzamie rezultāti:

- Sadala programmu funkcijās.
- Izmanto parametrus un return.
- Spēj definēt funkcijas atbildību un testējamu ieeju/izeju.

Funkcija kā līgums.

Funkcija saņem noteiktus datus, veic vienu saprotamu atbildību un atgriež rezultātu vai veic skaidri definētu darbību. Jo mazāka un skaidrāka funkcijas atbildība, jo vieglāk to testēt un atkārtoti izmantot.

Funkciju nosaukumiem jābūt darbību aprakstošiem: `calculateAverage()`, `isTemperatureValid()`, `checkAlarm()`.

Tehniskais piemērs: `double calculateAverage(...); bool isValid(...);`

Ko pasniedzējam uzsvērt: Dekompozīcija no 7. algoritmu nodarbības tagad kļūst par programmas struktūru.

Parametri, return un datu plūsma.

Parametrs ir funkcijas ievade. `return` ir viens no skaidrākajiem veidiem, kā atdot aprēķināto rezultātu. Jāizvairās no nevajadzīgiem globāliem mainīgajiem, jo tie slēpj datu plūsmu un sarežģī testēšanu.

Tehniskais piemērs: `checkAlarm(temperature, limit)` atgriež `bool`.

Ko pasniedzējam uzsvērt: Likst studentam uzzīmēt katras funkcijas `INPUT` → `PROCESS` → `OUTPUT`.

90 minūšu nodarbības scenārijs:

- 10 min — Dekompozīcijas atkātojums.
- 25 min — Funkcijas sintakse un `return`.
- 20 min — Parametri un lokālais scope.
- 25 min — Refaktorēšanas uzdevums.
- 10 min — Funkciju testēšanas diskusija.

Praktiskais darbs: Refaktorēt iepriekšējo programmu 3–5 funkcijās un katrai uzrakstīt vienu testa piemēru.

Patstāvīgais darbs / mājasdarbs: Pievienot funkciju datu validācijai un noņemt dublētu kodu.

Universitātes/pasniedzēja paplašinājums: Paplašināti: `pass-by-value/reference`, `const reference`, `overloads`, `header/source` sadalījums un `pure functions`.

Nodarbība 18 — Masīvi, mērījumu datu apstrāde un algoritmi

Sasniedzamie rezultāti:

- Glabā vairākas viena tipa vērtības masīvā.
- Iterē pār datiem un aprēķina min/max/vidējo.
- Analizē robežas un indeksu kļūdas.

Masīvs un indekss.

Masīvs glabā secīgu viena tipa elementu kopumu. Katram elementam ir indekss, un C++ tradicionāli indeksē no 0. Tas ir tipisks avots off-by-one kļūdām.

Tehniskajos piemēros masīvs var reprezentēt fiksētu mērījumu logu, sensora kanālus vai kalibrācijas punktus.

Tehniskais piemērs: `double measurements[10];`

Ko pasniedzējam uzsvērt: Vienmēr salīdzināt elementu skaitu ar atļauto indeksu diapazonu.

Datu apstrādes algoritmi.

Min, max, summa, vidējais un sliekšņa pārsniegumu skaits ir vienkārši, bet ļoti nozīmīgi algoritmi. Tie ļauj sasaistīt ciklus ar reāliem kvalitātes vai procesa datiem.

Svarīgi izskatīt tukša datu kopuma vai dalīšanas ar nulli gadījumu konceptuāli.

Tehniskais piemērs: 10 temperatūru logā noteikt max un alarm_count.

Ko pasniedzējam uzsvērt: Rezultāts ir tik labs, cik labi definēti ievades dati un robežgadījumi.

90 minūšu nodarbības scenārijs:

- 10 min — Masīva modelis uz tāfeles.
- 20 min — Indeksi un cikls.
- 20 min — Min/max/avg algoritmi.
- 30 min — Mērījumu analīzes uzdevums.
- 10 min — Off-by-one kļūdu meklēšana.

Praktiskais darbs: Apstrādāt 10 mērījumu masīvu un izveidot procesa kopsavilkumu.

Patstāvīgais darbs / mājasdarbs: Izveidot vēl vienu funkciju, kas atgriež alarm_count.

Universitātes/pasniedzēja paplašinājums: Paplašināti: std::array, std::vector, iterators, algorithms library, complexity un cache locality intuīcija.

Nodarbība 19 — OOP ievads: class, object un tehniska komponente

Sasniedzamie rezultāti:

- Izskaidro class un object atšķirību.
- Definē īpašības un metodes.
- Modelē vienkāršu Sensor vai Motor komponenti.

Class kā modeļa veidne.

Klase apraksta datu un uzvedības kopumu, kas pieder vienam konceptam. Objekts ir konkrēta šīs klases instance ar savu stāvokli.

Tehniskā sistēmā klasei ir jēga, ja tā palīdz modelēt komponenti vai atbildību, nevis tikai tāpēc, ka “OOP jāizmanto”.

Tehniskais piemērs: Sensor satur value, valid, alarmLimit un update()/isAlarm().

Ko pasniedzējam uzsvērt: OOP nav mērķis pats par sevi — tas ir viens organizācijas instruments.

Encapsulation intuīcija.

Komponentes iekšējo stāvokli vēlams mainīt caur skaidriem interfeisiem. Tas palīdz saglabāt invariantu — piemēram, ka sensoram nevar nejauši piešķirt neatļautu stāvokli, apejot validāciju.

Iesācējiem pietiek ar public/private ideju un vienkāršu metožu piemēru.

Tehniskais piemērs: Motor::setSpeed() pārbauda atļauto diapazonu, nevis ļauj brīvi mainīt iekšējo speed.

Ko pasniedzējam uzsvērt: Saistīt encapsulation ar drošāku komponentes lietošanu.

90 minūšu nodarbības scenārijs:

- 10 min — No struktūras uz komponenti.
- 25 min — class/object un konstruktora ideja.
- 20 min — Metodes un public/private.
- 25 min — Sensor klases uzdevums.
- 10 min — Python datu modeļa salīdzinājums.

Praktiskais darbs: Izveidot Sensor klasi un programmu, kas izveido 2 sensoru objektus, atjauno vērtības un izvada alarm statusu.

Patstāvīgais darbs / mājasdarbs: Pievienot Motor klasi ar start()/stop() un drošības stāvokli.

Universitātes/pasniedzēja paplašinājums: Paplašināti: constructors, composition, inheritance/polymorphism piesardzīgi, interfaces/abstract classes, RAII un SOLID intuīcija.

Nodarbība 20 — C++ pārbaudes darbs un debugging

Sasniedzamie rezultāti:

- Demonstrē C++ sintakses un tipu izpratni.
- Realizē algoritmu ar nosacījumiem, cikliem un funkcijām.
- Atrod un pamato kļūdas dotā kodā.

Pārbaudes darba mērķis.

Pārbaudes darbs nav tikai sintakses atcerēšanās. Tam jāpārbauda, vai students spēj no prasības izveidot algoritmu un korekti realizēt to C++ ar atbilstošiem tipiem un struktūru.

Daļu punktu jāpiešķir par kļūdas cēloņa skaidrojumu, ne tikai labotu gala rindu.

Tehniskais piemērs: Mērījumu analīze + interlock loģika + bojāta koda fragments.

Ko pasniedzējam uzsvērt: Vērtēt domāšanu, testus un skaidrojumu kopā ar kodu.

90 minūšu nodarbības scenārijs:

- 10 min — Instrukcija un uzdevumu precizēšana.
- 70 min — Individuāls pārbaudes darbs.
- 10 min — Darba nodošana un īsa refleksija par grūtākajiem punktiem.

Praktiskais darbs: Pārbaudes darbs: 20 % teorija/tipi, 25 % koda izsekošana, 35 % patstāvīgs tehnisks uzdevums, 20 % debugging.

Universitātes/pasniedzēja paplašinājums: Universitātes variantā pievienot references/pointers, std::vector, complexity vai nelielu OOP dizaina uzdevumu.

7.10. C++ pārbaudes darba ieteicamā punktu struktūra

Daļa	Punkti	Saturs
A. Jēdzieni un tipi	20	Programmas struktūra, tipi, kompilācija, const.
B. Koda analīze	25	Izsekot if/cikla/funkciju izpildei un noteikt rezultātu.
C. Patstāvīga realizācija	35	Tehnisks algoritms ar validāciju, funkciju un mērījumu datiem.

Daļa	Punkti	Saturs
D. Debugging	20	Atrast 3–5 kļūdas, labot un īsi pamatot cēloni.

8. SQL, IoT un mākoņdati

8.1. SQL un tehnisko datu uzglabāšana

SQL tiek izmantots, lai parādītu, ka sensora mērījumi un programmas rezultāti pēc aprēķina var kļūt par strukturētiem vēsturiskiem datiem. Obligātais minimums: tabula, rinda/ieraksts, kolonna/lauks, primārā atslēga, CREATE TABLE, INSERT, SELECT, WHERE, ORDER BY; paplašināti — agregācijas, JOIN un laika rindas.

```
CREATE TABLE measurements (
    id INTEGER PRIMARY KEY,
    temperature REAL,
    humidity REAL,
    timestamp TEXT
);

SELECT * FROM measurements
WHERE temperature > 25
ORDER BY timestamp DESC;
```

8.2. IoT datu ķēde

IoT arhitektūra

SENSORS → MIKROKONTROLLERIS/EDGE DEVICE → TĪKLS → MQTT/HTTP → SERVERIS/MĀKONIS → DATUBĀZE → API/DASHBOARD → LIETOTĀJS

- Wi-Fi, Ethernet, Bluetooth/BLE un LoRa/LoRaWAN jēdzieniska atšķirība.
- MQTT: broker, topic, publisher, subscriber; QoS un retained message — paplašinātajā līmenī.
- HTTP/HTTPS un REST API: GET/POST, statusa kodi, JSON.
- Edge computing: kad datu apstrādi veic ierīcē, nevis mākonī.
- Datu drošība: autentifikācija, API atslēgas, šifrēšana, piekļuves tiesības, tīklu segmentācija.

8.3. Praktiska integrācijas ideja

“Viedās telpas monitorings”: ESP32 vai simulators ģenerē temperatūras/mitruma datus; C++ vai Python veic sākotnējo apstrādi; dati tiek nosūtīti ar MQTT/HTTP; Python serviss tos saglabā SQL datubāzē un attēlo grafikā. Ja temperatūra pārsniedz robežu, sistēma ģenerē brīdinājumu.

8.4. Nodarbības 21–24 — datu glabāšana un sistēmu integrācija

Attēls. Functional tree — IoT tehnisko datu plūsma.

Nodarbība 21 — SQL pamati tehnisko mērījumu datiem

Sasniedzamie rezultāti:

- Atšķir datubāzi, tabulu, rindu, kolonnu un primāro atslēgu.
- Izmanto CREATE TABLE, INSERT, SELECT, WHERE un ORDER BY.
- Spēj formulēt vienkāršu vaicājumu tehniskam jautājumam.

Relāciju tabula kā strukturēts vēstures ieraksts.

Programmas mainīgais dzīvo izpildes laikā, bet datubāze ļauj saglabāt datus ilgākā periodā.

Tehniskā sistēmā tas ļauj analizēt tendences, notikumus, alarmus un kvalitātes rādītājus.

Tabulas kolonnām jābūt ar skaidru nozīmi: measurement_id, device_id, timestamp, temperature_c, quality_flag.

Tehniskais piemērs: Saglabāt telpas temperatūru ik pēc 5 minūtēm.

Ko pasniedzējam uzsvērt: Datubāzes shēma ir datu modeļa daļa — slikti nosaukumi un neskaidras vienības rada ilgtermiņa problēmas.

SELECT kā jautājums datiem.

SELECT izvēlas vajadzīgās kolonnas un rindas. WHERE filtrē, ORDER BY kārtot. Skolēnam jāmaca formulēt vispirms cilvēka valodas jautājumu un tikai tad SQL vaicājumu.

Piemērs: “Parādi pēdējos mērījumus, kuros temperatūra pārsniedza 30 °C.”

Tehniskais piemērs: SELECT timestamp, temperature_c FROM measurements WHERE temperature_c > 30 ORDER BY timestamp DESC;

Ko pasniedzējam uzsvērt: SQL nav tikai komandu iegaumēšana; tas ir strukturēts jautājums datu kopai.

90 minūšu nodarbības scenārijs:

- 15 min — No mainīgā uz vēsturisku datu ierakstu.
- 20 min — Tabula, PK un datu tipi.
- 20 min — INSERT un SELECT.
- 25 min — WHERE/ORDER BY praktiskie vaicājumi.
- 10 min — Rezultātu interpretācija un datu kvalitāte.

Praktiskais darbs: Izveidot measurements tabulu, ievietot vismaz 10 ierakstus un atbildēt ar SQL uz 5 tehniskiem jautājumiem.

Patstāvīgais darbs / mājasdarbs: Izveidot vaicājumu, kas atrod visus ALARM stāvokļus noteiktā periodā.

Universitātes/pasniedzēja paplašinājums: Paplašināti: normalization, foreign keys, transactions, indexes, GROUP BY, time-series DB jēdziens.

Nodarbība 22 — CSV, JSON un Python + SQLite

Sasniedzamie rezultāti:

- Atšķir CSV, JSON un SQL datubāzes lomas.
- Ar Python nolasa vai sagatavo strukturētus datus.
- Savieno programmas rezultātu ar SQLite datubāzi.

Datu formāts pret datubāzi.

CSV ir vienkāršs tabulārs teksta formāts. JSON ir hierarhisks datu apmaiņas formāts ar atslēgvērtību struktūru. SQL datubāze nodrošina vaicāšanu, ierobežojumus un strukturētu ilgtermiņa glabāšanu.

Nav viena “labākā” formāta. Izvēle atkarīga no sistēmas uzdevuma, datu apjoma un apmaiņas prasībām.

Tehniskais piemērs: Sensors → JSON ziņojums tīklā; serveris → SQL ieraksts; eksports → CSV analīzei.

Ko pasniedzējam uzsvērt: Atšķirt transporta formātu no glabāšanas modeļa.

Python datubāzes savienojums.

Python sqlite3 bibliotēka ļauj vienā datorā praktiski demonstrēt datubāzes integrāciju bez atsevišķa servera. Galvenais ir saprast connect → execute → commit → query ciklu.

Jāievieš parametrizētu vaicājumu ideja, lai neformētu SQL ar nekontrolētu tekstu.

Tehniskais piemērs: Python simulē sensora mērījumu un INSERT to SQLite.

Ko pasniedzējam uzsvērt: Pat mācību piemērā veidot drošus ieradumus: parametrizēti vaicājumi un datu validācija.

90 minūšu nodarbības scenārijs:

- 10 min — CSV/JSON/SQL salīdzinājums.
- 20 min — SQLite shēmas izveide.
- 25 min — Python sqlite3 demonstrācija.
- 25 min — Patstāvīga integrācija.
- 10 min — Datu pārbaude ar SELECT.

Praktiskais darbs: Python programma ģenerē/nolasa 20 mērījumus, saglabā SQLite un izvada pēdējos 5 alarmus.

Patstāvīgais darbs / mājasdarbs: Eksportēt datubāzes rezultātu CSV vai izveidot JSON kopsavilkumu.

Universitātes/pasniedzēja paplašinājums: Paplašināti: ORM jēdziens, PostgreSQL, schema migrations, parameterized queries un SQL injection riska skaidrojums.

Nodarbība 23 — IoT, MQTT, HTTP un mākoņdatu arhitektūra

Sasniedzamie rezultāti:

- Izskaidro sensor → edge → network → server/cloud → database → application ķēdi.
- Atšķir MQTT publish/subscribe no HTTP request/response pamatprincipa.
- Nosauc datu drošības un savienojuma uzticamības riskus.

IoT nav tikai “ierīce ar Wi-Fi”.

IoT sistēma sastāv no fiziska mērījuma vai darbības, lokālas apstrādes, sakaru kanāla, servera/brokera, datu glabāšanas un lietojuma. Katram posmam ir savi kļūdu režīmi un drošības prasības.

Edge computing nozīmē, ka daļu lēmumu veic lokāli ierīcē. Drošībai kritisku funkciju nevajadzētu padarīt pilnīgi atkarīgu no interneta savienojuma.

Tehniskais piemērs: ESP32 lokāli izslēdz sildītāju pārtemperatūrā pat tad, ja cloud nav pieejams.

Ko pasniedzējam uzsvērt: Mākonis nav vadības drošības aizstājējs.

MQTT publish/subscribe.

Publisher publicē ziņojumu topic, brokeris to izplata abonentiem. Ierīcēm nav obligāti jāzina viena otras IP vai tieši savienojumi, kas atvieglo daudzu komponentu integrāciju.

Topic struktūra ir daļa no sistēmas dizaina, piemēram factory/line1/motor3/temperature.

Tehniskais piemērs: sensor node publicē lab/room1/temperature; dashboard un logger abonē.

Ko pasniedzējam uzsvērt: Topic nosaukumiem jābūt konsekventiem un semantiskiem.

HTTP/REST un JSON.

HTTP modeli klients veic pieprasījumu serverim. REST API bieži izmanto GET datu saņemšanai un POST jaunu datu nosūtīšanai. JSON ir biežs ziņojuma ķermeņa formāts.

Jāpiemin statusa kodi, autentifikācija un HTTPS, bet nepadarīt šo nodarbību par pilnu tīklu kursu.

Tehniskais piemērs: POST /api/measurements ar JSON ķermeni.

Ko pasniedzējam uzsvērt: API ir interfeiss starp sistēmu komponentēm, nevis tikai URL.

90 minūšu nodarbības scenārijs:

- 10 min — IoT arhitektūras ievads.
- 25 min — MQTT modelis ar publisher/broker/subscriber.
- 20 min — HTTP/REST/JSON salīdzinājums.
- 20 min — Sistēmas arhitektūras zīmēšana.
- 15 min — Drošības un savienojuma atteices scenāriji.

Praktiskais darbs: Uzzīmēt savas IoT sistēmas arhitektūru un simulēt 3 MQTT topic ziņojumus vai HTTP pieprasījumus ar sagatavotiem datiem.

Patstāvīgais darbs / mājasdarbs: Aprakstīt, kas sistēmā notiek, ja uz 10 minūtēm pazūd internets.

Universitātes/pasniedzēja paplašinājums: Paplašināti: MQTT QoS 0/1/2, retained message, TLS, certificates, device identity, buffering/store-and-forward un OT/IT segmentācija.

Nodarbība 24 — Sistēmu integrācijas mini darbs — sensors → kods → DB → dashboard

Sasniedzamie rezultāti:

- Integrē vismaz 3 iepriekš apgūtos komponentus.
- Dokumentē datu plūsmu un interfeisus.
- Testē normālu un atteices scenāriju.

Integrācija kā interfeisu pārbaude.

Atsevišķi strādājoši komponenti vēl nenozīmē, ka sistēma strādās kopā. Integrācijā jāpārbauda datu formāts, vienības, timestamp, kļūdu kodi, savienojuma pārtraukumi un atbildība starp komponentēm.

Functional tree parāda, ko sistēmai jādara; arhitektūras shēma parāda, kurš komponents to dara; tests parāda, vai tas tiešām notiek.

Tehniskais piemērs: Viedās telpas monitors: Python sensors → JSON → SQLite → kopsavilkuma dashboard/grafiks.

Ko pasniedzējam uzsvērt: Prasīt skolēnam parādīt vienu datu vienību visā ceļā no ievades līdz datubāzei.

90 minūšu nodarbības scenārijs:

- 10 min — Prasību un arhitektūras pārbaude.
- 15 min — Functional tree/datu plūsmas precizēšana.
- 45 min — Integrācijas realizācija.
- 10 min — Atteices tests.
- 10 min — Demonstrācija un secinājumi.

Praktiskais darbs: Mini integrācijas darbs ar simulētiem datiem. Minimāli: Python + SQL; paplašināti: MQTT/HTTP vai vienkāršs dashboard.

Patstāvīgais darbs / mājasdarbs: Pievienot README ar arhitektūras attēlu un 3 testu rezultātiem.

Universitātes/pasniedzēja paplašinājums: Paplašināti: containerized service, time-series visualization, message schema/versioning un observability/logging.

9. Mehatronikas pamati

Mehatronika kursā tiek izmantota kā tilts starp programmu un fizisko procesu: mehānika + elektronika + vadība + programmatūra. Skolēnam jāspēj atšķirt mērījumu (sensoru), lēmumu/vadību (kontrolleri) un fizisko darbību (aktuatoru).

Bloks	Pamati	Padziļinājums pasniedzējam/universitātei
Sensori	Temperatūra, spiediens, attālums, gaismas, enkoders.	Linearitāte, precizitāte, izšķirtspēja, kalibrācija, troksnis.
Signāli	Digitāls/analogi, 0/1, spriegums, ADC.	Sampling, Nyquist ideja, filtrēšana, 4–20 mA industrijā.
Aktuatori	DC, servo, stepper, relejs, pneimatika.	H-bridge, PWM, torque-speed, driveri.
Kontrolleri	Arduino/ESP32/STM32/PLC ideja.	Interrupt, timer, real-time, watchdog.
Vadības cilpa	Setpoint, process, sensor, feedback.	Open/closed loop, PID, stabilitātes intuīcija.
Drošība	E-stop, interlock, fail-safe.	Safety relay/PLC, deterministic behavior, riska analīzes pamats.

Vadības cilpa

SETPOINT → CONTROLLER → ACTUATOR → PROCESS → SENSOR → FEEDBACK → CONTROLLER

9.1. Mehatroniskās sistēmas funkcionālā arhitektūra

Mehatroniskā sistēma apvieno fizisku procesu, sensorus, elektroniku, vadības algoritmu un aktuātorus. Programmēšanas kursā svarīgākais nav dziļi izrēķināt visu mehāniku vai elektroniku,

bet iemācīties redzēt robežas starp fizisko procesu un programmatūru: ko mēra, kas pieņem lēmumu, kas izpilda komandu un kā rezultāts tiek atgriezts atpakaļ kā feedback.

Attēls. Functional tree — mehatroniskas sistēmas vadība ar atgriezenisko saiti.

Sensors un mērījumu ķēde

Sensors fizisku lielumu pārvērš elektriskā vai digitālā signālā. Tehniskā sistēmā jāatšķir fiziskais lielums, sensora diapazons, izšķirtspēja, precizitāte, atkārtojamība un datu atjaunošanas ātrums.

Programma redz nevis “temperatūru”, bet skaitlisku vērtību, kuras ticamība ir atkarīga no sensora, kalibrācijas, trokšņa un datu apstrādes.

Tehniskais piemērs: Analogais 0–10 V vai 4–20 mA signāls tiek pārveidots engineering units, piemēram, bar vai °C.

Ko pasniedzējam uzsvērt: Mērījuma skaitlim vienmēr ir izcelsme, mērvienība un nenoteiktība.

Aktuatori un jaudas interfeiss

Programmatūra parasti nevar tieši darbināt motoru vai vārstu. Starp kontrolleri un aktuatoru ir draiveris, relejs, frekvenču pārveidotājs, servo drive vai cita jaudas elektronika.

Vadības komanda var būt digitāls ON/OFF, PWM, analogais setpoint, ātruma/pozīcijas komanda vai sakaru telegramma.

Tehniskais piemērs: Mikrokontrolleris ar PWM vada motor driver, nevis motoru tieši.

Ko pasniedzējam uzsvērt: Nošķirt loģikas līmeni no enerģijas/jaudas līmeņa.

Open-loop un closed-loop

Open-loop vadībā komanda tiek dota bez rezultāta mērījuma. Closed-loop sistēmā sensors mēra procesa faktisko rezultātu, un kontrolieris salīdzina to ar setpoint.

Feedback ļauj kompensēt slodzes un traucējumu ietekmi, taču slikti izstrādāta atgriezeniskā saite var radīt svārstības vai nestabilitāti.

Tehniskais piemērs: Ventilatora PWM komanda bez tachometra ir open-loop ātruma vadība; ar RPM feedback — closed-loop.

Ko pasniedzējam uzsvērt: Feedback nav garantija, ka sistēma ir laba; jāanalizē mērījums, dinamika un kontroles algoritms.

PID regulatora intuīcija

P daļa reaģē uz pašreizējo kļūdu, I uz uzkrāto kļūdu, D uz kļūdas izmaiņas ātrumu. Iesācēju līmenī mērķis ir saprast katras daļas intuitīvo ietekmi, nevis uzreiz iemācīt pilnu regulēšanas teoriju.

Programmēšanas laboratorijā PID var simulēt ar vienkāršu procesa modeli un grafiku, lai redzētu overshoot, settling un steady-state error.

Tehniskais piemērs: Motora ātrums vai temperatūra seko setpoint.

Ko pasniedzējam uzsvērt: PID jāpasniedz kā feedback algoritma piemērs, nevis “maģiska formula”.

Drošība, interlock un fail-safe

Drošības funkcijām ir cita prioritāte nekā normālai procesa loģikai. E-stop, aizsargdurvju interlock, limit switches un drošības releji/PLC var būt ārpus parastās programmas kontroles.

Fail-safe princips nozīmē, ka paredzamā atteices gadījumā sistēma nonāk drošākā stāvoklī. Timeout un watchdog palīdz atklāt programmas vai komunikācijas iestrēgšanu.

Tehniskais piemērs: Ja pazūd kritiska sensora signāls, sildīšana tiek atslēgta un ģenerēts alarm.

Ko pasniedzējam uzsvērt: Nedot skolēnam iespaidu, ka parasts if aizstāj sertificētu safety sistēmu.

9.2. Ieteicamā laboratorija — virtuāla temperatūras vadības cilpa

Python simulācijā modelē telpas vai sildītāja temperatūru. Students definē setpoint, vienkāršu procesa inerci, sensora troksni un ON/OFF regulatoru ar histerēzi. Paplašinātajā variantā tiek pievienots PID un salīdzināti grafiki.

- A līmenis: ON/OFF vadība ar histerēzi un sensora kļūdas stāvokli.
- B līmenis: datu žurnāls CSV/SQL un setpoint maiņa laikā.
- C līmenis: diskrets PID, sampling perioda ietekme un vienkārša stabilitātes analīze.

10. Robotikas pamati

- Robotu tipi: manipulators, mobilais robots, AGV/AMR, humanoīds, drons, ROV/AUV.
- Degrees of Freedom (DOF), joint, link, end-effector.
- Koordinātu sistēmas: world, base, joint/link, tool/end-effector, sensor frame.
- Pozīcija un orientācija: x, y, z; roll, pitch, yaw.
- Forward kinematics un inverse kinematics pamatjēga.
- Trajektorija, ātrums, paātrinājums, workspace, singularitātes jēdziens.
- Uztvere: kamera, LiDAR, IMU, enkoderi, ToF, GPS/GNSS.
- Mobilā robotika: differential drive, odometrija, pose (x, y, θ).
- Lēmumu loģika: finite state machine un behavior tree ievads.

10.1. Robots kā mehatroniska un datorizēta sistēma

Robots ir mehatroniska sistēma, kurai papildus vadībai bieži nepieciešama uztvere, koordinātu sistēmu pārvaldība, kustības plānošana un lēmumu loģika. Tādēļ robotika ir labs integrējošs piemērs visam kursam: algoritmi nosaka secību, Python palīdz prototipēt matemātiku un uztveri, C++ nodrošina aparatūrai tuvāku vadību, IoT/tīkli ļauj integrēt sistēmas, bet SQL/mākonis var glabāt telemetriju.

Robotu tipi un pielietojuma robežas

Industriāls manipulators galvenokārt veic precīzas kustības noteiktā darba telpā. Mobilais robots pārvietojas vidē un risina lokalizācijas/navigācijas problēmu. AGV parasti seko iepriekš definētai infrastruktūrai vai maršrutam, bet AMR spēj elastīgāk uztvert vidi un pārplānot ceļu.

Droni un ROV/AUV paplašina problēmu uz 3D kustību un sarežģītāku dinamiku.

Tehniskais piemērs: Nolikta AMR, 6DOF manipulators, quadrotor, zemūdens ROV.

Ko pasniedzējam uzsvērt: “Autonoms” nav tas pats, kas “bezvadu” vai “automātisks”.

Links, joints, DOF un end-effector

Link ir mehānisks posms, joint nosaka relatīvo kustību starp posmiem. Degrees of Freedom raksturo neatkarīgu kustības parametru skaitu.

Manipulatoram DOF palīdz saprast, kādas gala efektora pozīcijas/orientācijas ir sasniedzamas. Mobilam robotam konfigurācijas apraksts var būt x, y, θ .

Tehniskais piemērs: 2DOF plaknes roka pret 6DOF industriālu robotu.

Ko pasniedzējam uzsvērt: DOF ir kinemātisks jēdziens, nevis motoru skaita sinonīms visos gadījumos.

Koordinātu sistēmas un frames

Robots vienlaikus var izmantot world/map, base, link, tool un sensoru koordinātu sistēmas. Tas pats punkts dažādās koordinātu sistēmās iegūst atšķirīgas koordinātas.

Robota programmēšanā daudz kļūdu rodas nepareizi sajaucot frames vai vienības. Tādēļ transformācijas ir centrāls robotikas jēdziens.

Tehniskais piemērs: Kameras noteikts objekts camera frame jāpārveido robot base frame, pirms manipulators var pie tā kustēties.

Ko pasniedzējam uzsvērt: Vienmēr jautāt: “attiecībā pret kuru koordinātu sistēmu?”

State machine un behavior tree

Autonoma sistēma ne tikai aprēķina kustību; tai jāorganizē misijas stāvokļi. Finite State Machine definē diskrētus stāvokļus un pārejas. Behavior Tree organizē uzvedību hierarhiskāk un ir populārs sarežģītākās robotu misijās.

Iesācējiem FSM ir lielisks tilts no if/else uz sistēmu arhitektūru.

Tehniskais piemērs: SEARCH → APPROACH → PICK → RETURN → CHARGE/ERROR.

Ko pasniedzējam uzsvērt: Atšķirt “stāvoklis” no viena sensora mērījuma.

11. Python robotikā un kinemātikā

Python tiek izmantots kā ātrs matemātisko modeļu, sensoru datu un robotu algoritmu prototipēšanas līdzeklis. Pamata bibliotēkas: math, NumPy, matplotlib; paplašināti — SciPy, OpenCV un ROS 2 Python interfeisi.

11.1. 2DOF manipulatora forward kinematics

Divu posmu plaknes manipulatoram ar garumiem $L1$ un $L2$ gala efektora koordinātas var aprēķināt no locītavu leņķiem $\theta1$ un $\theta2$.

```
import math

L1 = 0.5
L2 = 0.3
q1 = math.radians(30)
q2 = math.radians(45)

x = L1*math.cos(q1) + L2*math.cos(q1 + q2)
```

```
y = L1*math.sin(q1) + L2*math.sin(q1 + q2)
print(x, y)
```

11.2. Turpmākie kinemātikas līmeņi

- Inverse kinematics: no vēlamās gala efektora pozīcijas uz locītavu leņķiem.
- Rotācijas matricas un homogēnās transformācijas 4×4 .
- Denavit–Hartenberg parametri manipulatorsistēmu aprakstam.
- Jacobian: saikne starp locītavu ātrumiem un gala efektora ātrumu.
- Singularitātes un darba telpa.
- Trajectory generation: pozīcija, ātrums, paātrinājums laikā.

Pedagoģiskā secība

Vispirms skolēns zīmē robotu un aprēķina vienkāršas koordinātas → pēc tam realizē formulu Python → vizualizē ar matplotlib → tikai universitātes līmenī pāriet uz matricām, Jacobian un vispārīgu n-DOF formulējumu.

11.3. Python kā robotu kinemātikas laboratorijas vide

Python ļauj atdalīt robotikas matemātiku no konkrēta robota ražotāja programmēšanas valodas. Studentam vispirms ir jāspēj aprēķināt un vizualizēt kustību modeli. Tikai pēc tam to pārnēs uz simulatoru vai reālu robotu.

Forward kinematics — no locītavu stāvokļa uz gala efektora pozu

Forward kinematics aprēķina, kur atrodas robota gala efektors, ja zināmi locītavu leņķi vai pārvietojumi un robota ģeometrija. 2DOF plaknes manipulatorā to var izskaidrot ar trigonometriju; n-DOF robotā izmanto transformāciju matricas.

Pedagoģiski svarīgi vispirms uzzīmēt posmus un leņķus, tad uzrakstīt formulas, pēc tam kodu un tikai beigās vizualizāciju. Tādējādi Python nav “formulu ģenerators”, bet izpildāms matemātiskais modelis.

Tehniskais piemērs: $L1/L2$ un $q1/q2 \rightarrow x/y$; vēlāk $x/y/z$ + orientācija.

Ko pasniedzējam uzsvērt: Pārbaudīt vienkāršus leņķus, kuru rezultātu var aptuveni paredzēt ar galvu.

Inverse kinematics — no vēlamās pozas uz locītavu konfigurāciju

Inverse kinematics risina pretējo uzdevumu: zinot vēlamo gala efektora pozu, atrast locītavu vērtības. Risinājums var nebūt unikāls, var nebūt sasniedzams vai atrasties pie singularitātes.

2DOF piemērā ir iespējams analītisks risinājums ar elbow-up/elbow-down konfigurācijām.

Sarežģītiem robotiem bieži izmanto skaitliskas metodes.

Tehniskais piemērs: Mērķis $x=0.55, y=0.25 \rightarrow$ iespējamie $q1/q2$.

Ko pasniedzējam uzsvērt: Studentam jāiemācās pārbaudīt risinājumu ar forward kinematics.

Homogēnās transformācijas un DH — universitātes pāreja

4×4 homogēnā transformācijas matrica vienā struktūrā apvieno rotāciju un translāciju. Vairāku posmu robotā transformācijas var reizināt, lai iegūtu gala efektora pozu attiecībā pret bāzi.

Denavit–Hartenberg konvencija ir sistemātisks veids, kā parametrizēt secīgu locītavu/posmu ģeometriju. Tā ir laba universitātes tēma pēc tam, kad studenti intuitīvi saprot frames un 2DOF kinemātiku.

Tehniskais piemērs: $T_{base\_tool} = T_{01} \cdot T_{12} \cdot \dots \cdot T_{n-1,n}$.

Ko pasniedzējam uzsvērt: Matricu reizināšanas secība nav komutatīva; transformāciju secībai ir fiziska nozīme.

Jacobian, singularitāte un ātrumi

Jacobian sasaista locītavu ātrumus ar gala efektora lineāro/leņķisko ātrumu. Tas ļauj analizēt, kā locītavu kustības ietekmē tool kustību un spēju kustēties noteiktā virzienā.

Singularitātē robots zaudē noteiktu kustības spēju vai nepieciešamie locītavu ātrumi var kļūt ļoti lieli. Ievadkursā pietiek ar ģeometrisku intuīciju un vienkāršu 2DOF piemēru.

Tehniskais piemērs: Izstiepta 2DOF roka var būt singularitātes konfigurācijā.

Ko pasniedzējam uzsvērt: Singularitāte nav programmas kļūda — tā ir robota ģeometrijas īpašība.

11.4. Ieteicamā Python robotikas laboratoriju secība

- 1. Uzzīmēt 2DOF robotu ar matplotlib un mainīt q_1/q_2 .
- 2. Aprēķināt forward kinematics un salīdzināt ar zīmējumu.
- 3. Realizēt 2DOF inverse kinematics ar sasniedzamības pārbaudi.
- 4. Ģenerēt vienkāršu gala efektora trajektoriju un grafikus $x(t)$, $y(t)$, $q_1(t)$, $q_2(t)$.
- 5. Pārnest modeli uz RoboDK/Webots/CoppeliaSim vai ROS 2 simulāciju.
- 6. Universitātes līmenī pāriet uz NumPy matricām, DH un Jacobian.

12. LiDAR, Machine Vision, IMU un sensoru sapludināšana

12.1. LiDAR

- Darbības princips: attāluma noteikšana ar gaismu; 2D scan un 3D point cloud.
- Parametri: range, field of view, angular resolution, scan rate.
- Kļūdas: troksnis, atstarošanās, caurspīdīgas/melnas virsmas, occlusion.
- Programmēšana: polārās koordinātas (r, θ) pārvērst (x, y) , tuvākā šķēršļa meklēšana, filtrēšana, clustering.
- Lietojumi: obstacle detection, mapping, SLAM, droša zona ap robotu.

```
import math
r = 1.8
theta = math.radians(35)
x = r * math.cos(theta)
y = r * math.sin(theta)
```

12.2. Machine Vision / Computer Vision

- Attēls kā pikseļu matrica; RGB un grayscale.

- Thresholding, edge detection, contours, morphology.
- Objekta atrašana, izmēra/pozīcijas noteikšana, defektu kontrole, OCR/QR.
- Robota vadīšana pēc kameras: koordinātu sistēmas, kalibrācija un object pose — paplašināti.
- Python: OpenCV; universitātes paplašinājumā — feature matching, segmentation, deep-learning detectori.

```
import cv2
image = cv2.imread('part.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 100, 200)
```

12.3. IMU

- Accelerometer — lineārais paātrinājums; gyroscope — leņķiskais ātrums; magnetometer — magnētiskais virziens.
- Asis X/Y/Z; roll, pitch, yaw.
- Kalibrācija, bias, noise, drift.
- Integrējot gyro, kļūda ar laiku uzkrājas; accelerometer/magnetometer palīdz koriģēt orientācijas novērtējumu.

12.4. Sensor fusion

Autonomās sistēmās viens sensors reti ir pietiekams. Enkoderi var kļūdīties riteņu slīdēšanas dēļ, IMU driftē, kamera cieš no apgaismojuma, LiDAR — no virsmu īpašībām. Sensor fusion apvieno vairākus avotus, lai iegūtu robustāku stāvokļa novērtējumu.

Sensor fusion piemērs

ENCODERS + IMU + LiDAR + CAMERA → FILTRĒŠANA / KALMAN / EKF → ROBOTA POZAS
NOVĒRTĒJUMS (x, y, θ vai x, y, z, roll, pitch, yaw)

12.5. Robotu uztvere kā programmējama datu ķēde

Attēls. Functional tree — LiDAR, Machine Vision, IMU un sensor fusion.

LiDAR datu programmēšana

2D LiDAR tipiski dod pārus angle + range. Vienkāršākais programmēšanas uzdevums ir pārvērst polārās koordinātas uz x/y, filtrēt nederīgus mērījumus un atrast tuvāko šķērsli noteiktā leņķu sektorā.

3D LiDAR dod point cloud. Tad parādās filtrēšana, downsampling, zemes plaknes atdalīšana, clustering un objektu ģeometrijas novērtēšana. Ievadkursā pietiek ar 2D scan, bet rokasgrāmatā jāparāda attīstības ceļš uz 3D point cloud.

Tehniskais piemērs: front_points = scan[-15°...+15°]; minimum range nosaka priekšējo drošības distanci.

Ko pasniedzējam uzsvērt: LiDAR mērījums nav perfekta patiesība; vienmēr jāfiltrē invalid/inf/nan un jādomā par virsmu īpašībām.

Machine Vision programmēšanas ķēde

Kameras attēls ir skaitļu masīvs. Klasiskā Machine Vision ķēde var būt: attēla iegūšana → apgaismojuma/krāsu normalizācija → threshold/edge → kontūras → mērījums → pass/fail.

Robot guidance gadījumā nepietiek tikai atrast objektu attēlā. Jāzina kameras kalibrācija un transformācija uz robotu koordinātu sistēmu, lai pikseļu informācija kļūtu par fizisku pozu.

Tehniskais piemērs: Detaļas kontūra → centra pikselis → kalibrācija → x/y uz konveijera → robota pick point.

Ko pasniedzējam uzsvērt: Nošķirt “object detection” no “precīza metroloģiska mērījuma”.

IMU programmēšana un orientācijas novērtējums

Gyroscope mēra leņķisko ātrumu; integrējot to laikā, var iegūt orientācijas izmaiņu, taču bias izraisa drift. Accelerometer ilgākā laika logā palīdz noteikt gravitācijas virzienu, bet dinamiskā kustībā satur arī lineāro paātrinājumu.

Tādēļ praksē orientāciju bieži novērtē, apvienojot vairākus sensorus. Complementary filter ir labs mācību piemērs pirms Kalman/EKF.

Tehniskais piemērs: $angle = \alpha * (angle + gyro * dt) + (1 - \alpha) * accel\_angle$.

Ko pasniedzējam uzsvērt: Sampling period dt ir daļa no algoritma, nevis nenozīmīga detaļa.

Sensor fusion, laiks un koordinātu sistēmas

Lai apvienotu sensorus, datiem jāattiecas uz saskaņotu laiku un koordinātu sistēmām. Pat perfekts algoritms var dot sliktu rezultātu, ja kamera un LiDAR nav laika sinhronizēti vai ir nepareizi kalibrēti to savstarpējā transformācija.

Kalman filter un Extended Kalman Filter ir statistiski stāvokļa novērtēšanas algoritmi. Ievadkursā svarīgāk ir saprast predict/update ciklu: modelis prognozē stāvokli, mērījums to koriģē.

Tehniskais piemērs: Wheel odometry + IMU → EKF → pose; LiDAR localization periodiski koriģē drift.

Ko pasniedzējam uzsvērt: Sensor fusion nav “vidējā vērtība no sensoriem”; tas ir modeļa un mērījumu kombinēšanas process.

13. Kā domā autonomi robots?

Šī nodaļa apvieno visu kursu vienā sistēmiskā modelī. Autonomi robots nepārtraukti iegūst datus, interpretē vidi, novērtē savu atrašanās vietu, izvēlas uzvedību, plāno kustību, vada aktuatorus un pārbauda rezultātu.

Autonomā robota cikls

SENSE → PERCEIVE → LOCALIZE/MAP → DECIDE → PLAN → CONTROL → ACT → SENSE ...

13.1. SENSE — ko robots mēra?

LiDAR, kameras, IMU, enkoderi, ToF/ultraskaņa, GPS/GNSS, spēka un griezes momenta sensori. Rezultāts nav “zināšanas”, bet neapstrādāti skaitļi un signāli.

13.2. PERCEIVE — ko šie dati nozīmē?

Attēla apstrāde, point cloud apstrāde, objektu detekcija, segmentācija, clustering, filtrēšana. Mērķis: pārvērst sensoru datus semantiski vai ģeometriski noderīgā informācijā.

13.3. LOCALIZE — kur es atrodos?

Mobilam robotam bieži poza ir (x, y, θ) ; 3D sistēmā — pozīcija un orientācija. Izmanto odometriju, IMU, LiDAR, kameru, GNSS un sensor fusion.

13.4. MAP / SLAM — kā izskatās vide?

Robots izmanto karti vai veido to darbības laikā. SLAM vienlaikus risina lokalizācijas un kartēšanas uzdevumu.

13.5. DECIDE — ko darīt tālāk?

Noteikumu sistēma, finite state machine vai behavior tree izvēlas uzvedību: SEARCH, APPROACH, AVOID, RETURN, CHARGE, ERROR.

13.6. PLAN — kā nokļūt līdz mērķim?

Globālā ceļa plānošana: Dijkstra, A^* . Sarežģītākās telpās: RRT/RRT*. Lokālais plānotājs reaģē uz dinamiskiem šķēršļiem.

13.7. TRAJECTORY — kā kustēties laikā?

Path nosaka ģeometrisku ceļu; trajectory pievieno laiku, ātrumu un paātrinājumu.

13.8. CONTROL — kā izpildīt kustību?

Kinemātika un vadības algoritmi pārvērš trajektoriju motoru komandās. Feedback salīdzina setpoint ar measured value; PID ir klasisks ievads.

13.9. ACT — fiziska iedarbība

DC/BLDC motori, servo, stepper, lineārie aktuatori, pneimatika, manipulatori, dronu propelleri vai ROV thrusters.

```
if battery < 20:
    state = 'RETURN_TO_CHARGER'
elif obstacle_distance < 0.5:
    state = 'AVOID'
elif target_detected:
    state = 'APPROACH'
else:
    state = 'SEARCH'
```

Svarīga atziņa skolēnam

Robots pats “neredz galdu” un “nezina, kur atrodas”. Sensors dod skaitļus. Programmatūra, matemātiskie modeļi un algoritmi no šiem skaitļiem izveido robota pasaules modeli un pieņem lēmumu.

13.10. Functional tree — kā domā autonomi robots?

Attēls. Functional tree — autonoma robota misijas funkciju sadalījums.

13.11. Pilns uztveres-lēmuma-vadības cikls

Autonoma robota “domāšanu” ir lietderīgi skaidrot kā nepārtrauktu stāvokļa novērtēšanas un vadības ciklu. Sensors dod novērojumu, perception iegūst jēgpilnas pazīmes, localization/map nosaka “kur es esmu un kas ir apkārt”, decision izvēlas misijas stāvokli, planner atrod ceļu/trajektoriju, controller pārvērš to aktuatoru komandās, bet nākamie sensora dati parāda, kas faktiski notika.

Odometrija un differential-drive kinemātika

Divriteņu differential-drive robotam kustību nosaka kreisā un labā riteņa ātrumi. Ja abi vienādi, robots kustas taisni; ja atšķiras, robots griežas. No riteņu encoder mērījumiem var integrēt pozas izmaiņu x , y , θ .

Odometrija kļūdas uzkrāj, jo riteņi slīd un ģeometriskie parametri nav perfekti. Tādēļ to koriģē ar IMU, LiDAR, kameru vai GNSS.

Tehniskais piemērs: $v = (v_R + v_L)/2$; $\omega = (v_R - v_L)/\text{wheel\_base}$; pēc dt atjaunina pose.

Ko pasniedzējam uzsvērt: Odometrija ir relative motion estimate, nevis absolūta patiesība.

Path planning pret trajectory planning

Path ir ģeometrisks ceļš konfigurācijas telpā. A^* un Dijkstra var atrast ceļu režģa kartē. Trajectory pievieno laiku, ātrumu un paātrinājumu, lai ceļš kļūtu fiziski izpildāms.

Reāls robots nevar momentā pāriet no nulles uz maksimālo ātrumu vai izpildīt asu pagriezienu bez dinamiskām sekām. Tādēļ plānošana jāsaista ar robota kinemātiskiem/dinamiskiem ierobežojumiem.

Tehniskais piemērs: A^* dod waypoint secību; trajectory generator izlīdzina un piešķir ātrumus.

Ko pasniedzējam uzsvērt: “Atrasts ceļš” vēl nenozīmē “robots spēj pa to droši braukt”.

Replanning un dinamiskie šķēršļi

Autonomā vidē plāns var kļūt nederīgs, jo parādās cilvēks, palete vai cits robots. Sistēmai jāspēj lokāli izvairīties, apstāties vai pārplānot ceļu.

Šeit svarīgs ir prioritāšu princips: drošība un collision avoidance ir augstāk par misijas ātrumu.

Tehniskais piemērs: Nolikta robots apstājas cilvēka priekšā, pēc tam pārplāno, ja šķērslis paliek.

Ko pasniedzējam uzsvērt: Replanning jātestē kā atsevišķs scenārijs.

Recovery un degraded mode

Autonoma sistēma nav pilnvērtīga, ja tai nav definēta rīcība kļūdas gadījumā. Lost localization, sensor timeout, low battery, actuator fault vai network loss ir normāli projektējami stāvokļi.

Recovery var būt atkārtota lokalizācija, atgriešanās drošā punktā, ātruma samazināšana vai kontrolēta apstāšanās.

Tehniskais piemērs: Ja localization confidence nokrīt zem robežas, robots apstājas un mēģina relocalize.

Ko pasniedzējam uzsvērt: Projektā jāvērtē ne tikai “happy path”, bet arī atteices loģika.

13.12. Python laboratorija — autonoma mobilā robota simulācija

- 1. Definēt karti kā 2D režģi un robota pozu (x, y, theta).
- 2. Simulēt LiDAR/attāluma sensoru vienkāršotā modeli.
- 3. Izveidot A* ceļa plānotāju no START līdz GOAL.
- 4. Pārvērst ceļu waypoint secībā un simulēt differential-drive kustību.
- 5. Pievienot obstacle event un pārplānošanu.
- 6. Pievienot IMU/odometrijas troksni un vienkāršu fusion vai korekciju.
- 7. Vizualizēt karti, trajektoriju, plānoto un faktisko pozu ar matplotlib.
- 8. Universitātes variantā pārceļ uz ROS 2 + Gazebo/Webots/CoppeliaSim un izmantot SLAM/Nav2.

14. ROS 2, SLAM un navigācijas sistēmas

ROS 2 nav viena programma, bet robotikas programmatūras ekosistēma un komunikācijas modelis, kas ļauj sadalīt robotu mezglos (nodes) un apmainīties ar ziņojumiem. Šajā kursā obligātais līmenis ir konceptuāls; paplašinātajā/universitātes līmenī iespējami praktiski laboratorijas darbi simulācijā.

Jēdziens	Nozīme	Piemērs
Node	Atsevišķs process/komponents	LiDAR driver, camera node, controller.
Topic	Nepārtraukta datu plūsma	/scan, /imu, /cmd_vel.
Publisher/Subscriber	Datu publicētājs/saņēmējs	LiDAR publicē /scan; SLAM to abonē.
Service	Pieprasījums-atbilde	Reset map, trigger action.
TF	Koordinātu sistēmu transformācijas	map → odom → base_link → lidar.
URDF	Robota ģeometriskais modelis	Links, joints, sensori.
Nav2	ROS 2 navigācijas steks	Localization, planning, control.
SLAM	Karte + lokalizācija	LiDAR vai visual SLAM.

ROS 2 datu plūsmas piemērs

LiDAR node → /scan → SLAM node → /map; IMU/odometry → localization/EKF; goal → planner → controller → /cmd_vel → robot base

14.1. ROS 2 domāšanas modelis: sadalīta robotikas sistēma

ROS 2 palīdz sadalīt robotu komponentēs, kas sazinās ar standartizētiem interfeisiem. Tas ļauj sensoru draiveri, lokalizāciju, planner, controller un lietotāja interfeisu attīstīt un testēt atsevišķi. Kursa obligātajā līmenī pietiek ar arhitektūras izpratni; praktiska programmēšana ir paplašinātais/universitātes līmenis.

Nodes, topics, services un actions

Node ir atsevišķa funkcionāla programmas komponente. Topic ir nepārtraukta ziņojumu plūsma publisher/subscriber modelī. Service ir īss request/response, bet action ir piemērots ilgākam uzdevumam ar feedback un iespēju atcelt.

Šis sadalījums palīdz sasaistīt ROS 2 ar agrāko functional tree: viena funkcija var tikt realizēta vienā vai vairākos noded, bet interfeisi definē datu plūsmu.

Tehniskais piemērs: /scan topic, /cmd_vel topic, navigation action.

Ko pasniedzējam uzsvērt: Node robežām jāseko atbildībām, nevis patvaļīgam failu sadalījumam.

TF un koordinātu transformāciju koks

TF uztur koordinātu sistēmu savstarpējās transformācijas laikā. Tipiska ķēde: map → odom → base_link → lidar/camera.

Ja transformācijas nav korektas, sensora dati ģeometriski nonāk nepareizā vietā un navigācija/SLAM kļūst nekorekta.

Tehniskais piemērs: LiDAR scan jāzina, kur sensors atrodas attiecībā pret base_link.

Ko pasniedzējam uzsvērt: ROS 2 robotikā frame kļūdas bieži izskatās kā “SLAM problēma”, lai gan sakne ir transformācijā.

SLAM, localization un Nav2

SLAM veido karti un novērtē robota pozu, kad karte vēl nav zināma. Localization režīmā robots izmanto jau sagatavotu karti. Nav2 apvieno planner, controller, costmaps, behavior tree un recovery komponentes autonomai navigācijai.

Universitātes laboratorijā studenti var salīdzināt odometriju ar SLAM/localization rezultātu un analizēt sensoru/tf ietekmi.

Tehniskais piemērs: goal pose → planner → controller → cmd_vel; obstacle → costmap → replanning.

Ko pasniedzējam uzsvērt: Nav2 nav “viens A algoritms”; tā ir integrēta navigācijas sistēma.*

14.2. ROS 2 Python/C++ sadalījuma piemērs

Labs noslēguma vai universitātes projekts var izmantot Python perception/planning mezglam un C++ low-level control vai augstas frekvences komponentēm. Tas studentam praktiski parāda, ka viena robotu sistēma var apvienot vairākas valodas ar kopīgiem ziņojumu interfeisiem.

15. Noslēguma patstāvīgais projekts

Pēdējie aptuveni trīs mēneši paredzēti integrētam projektam. Pamata prasība: audzēknis demonstrē algoritmisko domāšanu un izmanto gan Python, gan C++ vismaz vienā būtiskā projekta komponentē. SQL, IoT, vizualizācija vai robotika var būt izvēles paplašinājums atkarībā no pieejamā aprīkojuma.

Posms	Rezultāts
1. Problēma un ideja	Īss problēmas apraksts un lietotājs/tehniskais process.
2. Prasības	Funkcionālās prasības un 2–5 pieņemšanas kritēriji.
3. Arhitektūra	Blokshēma: ievades, programmas moduļi, datubāze/tīkls, izejas.
4. Algoritms	Pseudokods un galveno stāvokļu/logikas diagramma.
5. Python realizācija	Datu apstrāde, simulācija, vadības logika vai vizualizācija.
6. C++ realizācija	Tā paša algoritma daļa vai aparatūrai tuvā komponente.
7. Integrācija	SQL/IoT/robotikas komponente pēc projekta vajadzības.
8. Testēšana	Normāli, robežvērtību un atteices scenāriji.
9. Dokumentācija	README, arhitektūra, koda skaidrojums, testu rezultāti.
10. Aizstāvēšana	Demonstrācija, jautājumi, secinājumi un uzlabojumu plāns.

15.1. Ieteicamie projektu temati

- Viedās telpas temperatūras/mitruma monitorings ar SQL un dashboard.
- Mini ražošanas līnijas simulators ar brāķa uzskaiti.
- IoT laikapstākļu stacija.
- Autonoma mobilā robota simulācija ar A* un šķēršļu izvairīšanos.
- LiDAR datu vizualizācija un šķēršļu noteikšana.
- Machine Vision detaļas esamības/pozīcijas kontrole.
- 2DOF/3DOF manipulatora kinemātikas simulators Python.
- Viedās noliktavas robota stāvokļu mašīna ar uzdevumu izpildi.

15.2. Functional tree kā obligāts projekta projektēšanas artefakts

Noslēguma projektā functional tree tiek izmantots pirms detalizētas programmatūras arhitektūras. Saknes mezgls formulē projekta galveno funkciju. Pirmajā līmenī funkcija tiek sadalīta lielās atbildībās, bet nākamajos līmeņos — pārbaudāmās apakšfunkcijās. Koks nedrīkst būt komponentu saraksts; tas apraksta, KO sistēmai jādara. Komponentu izvēle tiek veikta nākamajā arhitektūras solī.

15.3. Nodarbības 25–36 — noslēguma projekta pilns 12 nedēļu scenārijs

Nodarbība 25 — Problēma, lietotājs un prasības

Nodarbības mērķis: Formulēt projekta problēmu; definēt 4–8 funkcionālās un 2–4 nefunkcionālās prasības; katrai galvenajai prasībai pievienot mērāmu acceptance criterion.

Rezultāts: vienas lapas projekta charter. Pasniedzējs pārbauda, vai prasības apraksta “ko”, nevis tikai “kā”.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 26 — Functional tree, arhitektūra un testu stratēģija

Nodarbības mērķis: Izveidot functional tree; komponentu arhitektūras shēmu; identificēt ievades/izejas, datu formātus un kļūdu stāvokļus.

Rezultāts: functional tree + block diagram + sākotnējais testu saraksts.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 27 — Python realizācija I — kodols

Nodarbības mērķis: Realizēt projekta galveno algoritmu vai simulāciju Python; sadalīt funkcijās/moduļos; sagatavot reprezentatīvus testa datus.

Pasniedzējs veic code walkthrough, nevis tikai skatās gala rezultātu.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 28 — Python realizācija II — dati, testi un vizualizācija

Nodarbibas mērķis: Pievienot datu validāciju, kļūdu apstrādi, grafiku/SQL/JSON pēc projekta vajadzības; izpildīt robežvērtību testus.

Rezultāts: stabila Python komponente un dokumentēti testi.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 29 — C++ realizācija I — algoritma pārnese vai low-level komponente

Nodarbibas mērķis: Izvēlēties projekta daļu, kur C++ ir jāpilds: vadība, embedded loģika, datu apstrāde vai algoritma salīdzinājums.

Rezultāts: kompilējama C++ komponente ar skaidru interfeisu.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 30 — C++ realizācija II — funkcijas/OOP un testi

Nodarbibas mērķis: Refaktorēt funkcijās/klasēs, pievienot validāciju un testus; salīdzināt ar Python versiju pēc lasāmības, tipu kontroles vai veiktspējas.

Rezultāts: C++ komponentes testa pierādījumi.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 31 — IoT/robotikas/sensora integrācija

Nodarbibas mērķis: Pēc projekta: MQTT/HTTP, SQLite/cloud, sensoru simulācija, kinemātika, LiDAR/IMU/vision vai ROS 2 simulācijas komponente.

Rezultāts: integrācijas demonstrācija ar dokumentētu datu plūsmu.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.

- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 32 — Testēšana, validācija un atteices scenāriji

Nodarbibas mērķis: Izpildīt normal, boundary, invalid input, timeout/network loss/sensor fault scenārijus atbilstoši projektam.

Rezultāts: testu protokols Expected/Actual/Pass-Fail un noviržu saraksts.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 33 — Dokumentācija un reproducējamība

Nodarbibas mērķis: README, instalēšanas/palaišanas instrukcija, functional tree, arhitektūra, datu formāti, koda struktūra, zināmie ierobežojumi.

Rezultāts: cits students spēj palaist projektu pēc dokumentācijas.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 34 — Pilnveidošana un tehniskais review

Nodarbibas mērķis: Labot atrastās kļūdas, vienkāršot koda struktūru, noņemt dublēšanos, uzlabot nosaukumus un atkārtot kritiskos testus.

Pasniedzējs veic design review: prasība → funkcija → kods → tests.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 35 — Aizstāvēšana I — demonstrācija

Nodarbibas mērķis: Demonstrēt sistēmu pret acceptance criteria; izskaidrot vismaz vienu Python un vienu C++ komponenti; parādīt kļūdas scenāriju.

Vērtē pierādījumu kvalitāti, ne tikai prezentācijas vizuālo noformējumu.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

Nodarbība 36 — Aizstāvēšana II, refleksija un profesionālās attīstības ceļš

Nodarbibas mērķis: Pabeigt aizstāvēšanas; salīdzināt Python/C++; izvērtēt arhitektūras lēmumus; definēt nākamo projekta/zināšanu attīstības soli.

Rezultāts: Īsa tehniskā retrospektīva un individuāls turpmākās izaugsmes plāns.

90 minūšu scenārijs:

- 10 min — statusa pārskats un šīs nedēļas mērķis.
- 15 min — pasniedzēja mini lekcija/demonstrācija.
- 50 min — patstāvīgs projekta darbs ar individuālām konsultācijām.
- 10 min — peer/design review vai testa demonstrācija.
- 5 min — darba žurnāls, nākamais solis un commit/versijas fiksēšana.

15.4. Noslēguma projekta obligātie artefakti

- Problēmas apraksts un lietotājs/tehniskais process.
- Funkcionālās un nefunkcionālās prasības ar acceptance criteria.
- Functional tree un sistēmas arhitektūras/datu plūsmas shēma.
- Algoritms: pseidokods, blokshēma vai state machine atbilstoši projektam.
- Python komponente ar izskaidrojamu kodu.
- C++ komponente ar izskaidrojamu kodu.
- Integrācijas komponente pēc vajadzības: SQL/IoT/robotika/sensori/simulācija.
- Testu protokols ar normāliem, robežvērtību un atteices gadījumiem.
- README / lietotāja instrukcija / zināmie ierobežojumi.
- Demonstrācija un tehniskā aizstāvēšana.

16. Universitātes līmeņa paplašinājums

Modulis	Tēmas
Datu struktūras	Array/list, stack, queue, set, hash map, linked list, tree, graph.

Modulis	Tēmas
Algoritmu analīze	$O(1)$, $O(n)$, $O(\log n)$, $O(n^2)$; linear/binary search; sorting.
OOP	Encapsulation, inheritance, polymorphism, interfaces/abstraction.
Atmiņa un C++	Pointers, references, stack/heap, dynamic memory, RAII ideja.
Datoru arhitektūra	CPU, RAM, cache, registers, instruction, binary/hex, compiler.
Programmatūras inženierija	Requirements, design, implementation, test, deployment, maintenance.
Matemātika robotikai	Vektori, matricas, koordinātu transformācijas, trigonometrija, statistikas pamati.
Vadības teorija	Dinamiskā sistēma, feedback, PID, diskretizācijas intuīcija.
Robotika	DH, Jacobian, singularities, trajectory planning, dynamics ievads.
Autonomās sistēmas	Kalman/EKF, SLAM, A*/RRT, behavior trees, ROS 2, Nav2.

16.1. Kā no šī kursa izveidot universitātes ievadkursu

Universitātes variantā nav nepieciešams mainīt kursa pamatstāstu. Tiek paaugstināts matemātiskais dziļums, programmatūras kvalitātes prasības un projektu sarežģītība. Zemāk minētie moduļi var kļūt par atsevišķām 90–180 min lekcijām/laboratorijām.

Algoritmu analīze un datu struktūras

Big-O intuīcija un formalizācija; linear/binary search; sorting; stack, queue, hash map, tree, graph; izvēles trade-off.

C++ atmiņas modelis

Pointers, references, stack/heap, object lifetime, RAI, smart pointers, undefined behavior pamatjēga.

Programmatūras inženierija

Requirements traceability, architecture, interfaces, Git workflow, code review, unit/integration/system testing, CI jēdziens.

Lineārā algebra robotikai

Vektori, matricas, rotācijas, homogēnās transformācijas, frame chains, quaternions ievads.

Manipulatoru kinemātika

DH, forward/inverse kinematics, Jacobian, singularities, workspace, trajectory generation.

Mobilā robotika

Differential drive, odometry, motion models, path tracking, localization.

Stāvokļa novērtēšana

Probability intuition, Gaussian, Kalman filter, EKF, sensor models un covariance.

Computer Vision / Machine Vision

Camera model, calibration, projective geometry ievads, features, pose estimation, segmentation/detection.

LiDAR un SLAM

Point clouds, ICP ideja, occupancy grids, scan matching, graph-based SLAM intuīcija.

Vadības teorija

Dinamiskie modeļi, sampling, PID, stability intuition, state-space ievads.

ROS 2

Nodes/topics/services/actions, TF, URDF, launch, rosbag, QoS, SLAM/Nav2 laboratorijas.

Autonomās sistēmas

Behavior trees, global/local planning, recovery, safety, system verification and validation.

17. Skolēna/studentu rokasgrāmatas kodols

Šī paša dokumenta teorijas nodaļas paredzētas arī kā rokasgrāmatas kodols, kuru audzēknis var izmantot pēc kursa. Katrā turpmākajā versijā nodaļām tiks pievienotas “Atceries”, “Tipiskā kļūda”, “Praktiskais piemērs”, “Profesionālajā darbā” un “Universitātes līmenis” ailes.

Nodaļa	Ko saglabāt kā atsauces materiālu
Algoritmi	Vadības struktūras, blokshēmu simboli, pseidokoda piemēri.
Python	Sintakses minimums, kļūdu tipi, faili/CSV/JSON, bibliotēkas.
C++	Programmas karkass, tipi, references/pointers paplašināti.
SQL	Biežākie vaicājumi un datu modelēšanas minimums.
IoT	Arhitektūras shēma, MQTT/HTTP, drošības principi.
Mehatronika	Sensor–controller–actuator–feedback ķēde.
Robotika	Frames, kinematics, sensors, planning, control.
Autonoms robots	Sense–Perceive–Localize–Plan–Control cikls.
Profesionālā izaugsme	Git, Linux, API, ROS 2, simulācija, tālākie mācību ceļi.

17.1. Rokasgrāmatas vienotais ieraksta formāts katrai tēmai

Lai dokuments būtu izmantojams gan mācību laikā, gan vēlāk profesionālajā izaugsmē, katra būtiska tēma turpmāk tiek veidota pēc vienota modeļa:

- Definīcija — īsa, precīza atsauces forma.
- Kāpēc tas vajadzīgs — saikne ar reālu sistēmu.
- Kā tas darbojas — padziļināta teorija.
- Tehniskais piemērs — automatizācija, mehatronika, robotika vai dati.
- Kods/pseudokods — minimāls, bet izpildāms piemērs.
- Tipiskā kļūda — biežākais nepareizais priekšstats.
- Pārbaudi sevi — 2–5 jautājumi vai mazi uzdevumi.
- Profesionālajā darbā — kur šo jēdzienu sastop.
- Universitātes līmenis — nākamais matemātiskais/tehniskais dziļums.
- Functional tree / sistēmas shēma — ja tēma apraksta funkciju kopumu.

18. Pārbaudes darbu un projektu vērtēšana

18.1. Algoritmu pārbaudes darba struktūra

Daļa	Svars	Kritērijs
Teorija	20 %	Jēdzieni un izpratne.
Algoritma analīze	20 %	Spēj izsekot izpildei un noteikt rezultātu.
Blokshēma	30 %	Pareizi simboli, plūsma, nosacījumi un cikli.
Patstāvīgs algoritms	30 %	Loģika, robežgadījumi, tehniskā pamatotība.

18.2. Projekta rubrika

Kritērijs	Punkti	Augsta līmeņa pazīmes
Problēmas/prasību definēšana	10	Skaidra problēma, mērāmi pieņemšanas kritēriji.
Algoritms un arhitektūra	20	Loģisks sadalījums, datu plūsma, kļūdu stāvokļi.
Python realizācija	15	Strukturēts, saprotams, testējams kods.
C++ realizācija	15	Pareizi tipi, funkcijas/OOP pēc vajadzības, skaidra loģika.
Integrācija	10	SQL/IoT/robotika izmantota jēgpilni.

Kritērijs	Punkti	Augsta līmeņa pazīmes
Testēšana	15	Normāli, robežvērtību un atteices testi ar rezultātiem.
Dokumentācija	10	README, shēmas, koda un rezultātu skaidrojums.
Aizstāvēšana	5	Spēj pamatot lēmumus un izskaidrot savu kodu.

18.3. Ieteicamie praktiskie laboratorijas darbi visa kursa laikā

L1 — Algoritms bez programmēšanas

Automātisko durvju, lifta vai konveijera functional tree + blokshēma + 5 testa scenāriji.

L2 — Python mērījumu analizators

Input vai CSV dati; validācija; min/max/avg; alarm; testa tabula.

L3 — Python debugging

Speciāli bojāts kods; atrast syntax/runtime/logic kļūdas un uzrakstīt root-cause skaidrojumu.

L4 — C++ tehniska izvēle

AUTO/MANUAL/STOP, interlock, cikls, funkcijas un kļūdu stāvoklis.

L5 — C++ Sensor klase

Datu tips, validācija, alarm limit, metode update(), tests ar vairākiem objektiem.

L6 — Python + SQLite

Simulēt mērījumus, INSERT datubāzē, SELECT alarmus, eksportēt kopsavilkumu.

L7 — MQTT/IoT simulācija

Definēt topics, JSON payload, savienojuma zuduma uzvedību un edge/cloud atbildības.

L8 — Mehatronikas feedback simulācija

ON/OFF ar histerēzi; pēc izvēles PID; grafiks setpoint/process value/control output.

L9 — 2DOF robot kinematics

Forward + inverse kinematics, sasniedzamība, matplotlib vizualizācija.

L10 — LiDAR scan processing

Polar→Cartesian, invalid filtrēšana, nearest obstacle un drošības sektors.

L11 — Machine Vision

Threshold/edges/contours un objekta centra/izmēra noteikšana sagatavotā attēlā.

L12 — IMU fusion

Gyro integrācija, drift demonstrācija, complementary filter ar simulētiem datiem.

L13 — A* mobilais robots

Grid map, path planning, waypoint visualization un dinamiska šķēršļa replanning.

L14 — ROS 2 simulācija (paplašināti)

/scan, /odom, TF, SLAM vai Nav2 datu plūsmas novērošana simulatorā.

19. Terminoloģijas minimums

Termins	Īsa nozīme
Algoritms	Precīzi definēta darbību secība problēmas risināšanai.
Mainīgais	Nosaukta vērtība, ko programma var izmantot un mainīt.
Boolean	Loģisks datu tips: true/false.
Cikls	Atkārtota komandu izpilde.
Funkcija	Atkārtoti izmantojams programmas darbību bloks.
Compiler	Programma, kas avota kodu pārvērš izpildāmā/mašīnкодā.
Datubāze	Strukturēta datu glabātuve.
SQL	Valoda relāciju datubāzu datu definēšanai un vaicāšanai.
API	Noteikts veids, kā programmas savstarpēji apmainās ar funkcijām/datiem.
IoT	Tīklam pieslēgtas fiziskas ierīces, kas mēra, apstrādā vai ietekmē vidi.
MQTT	Viegls publish/subscribe ziņojumapmaiņas protokols, bieži IoT.
Sensor	Elements, kas fizisku lielumu pārvērš izmantojamā signālā.
Actuator	Elements, kas vadības signālu pārvērš fiziskā darbībā.
Feedback	Izmērīts procesa rezultāts, kas atgriežas kontrolierī.
Kinemātika	Kustības ģeometrijas apraksts, neanalizējot spēkus.
LiDAR	Attāluma/vides mērīšana ar gaismas palīdzību.
IMU	Inerciālā mērvienība ar akselerometru un žiroskopu, bieži arī magnetometru.
Machine Vision	Attēlu izmantošana automatizētai kontrolei un mērīšanai.
Odometrija	Robota kustības/pozās novērtējums no kustības sensoriem.

Termins	Īsa nozīme
SLAM	Vienlaicīga lokalizācija un kartes veidošana.
ROS 2	Robotikas programmatūras ekosistēma un komunikācijas infrastruktūra.
Path planning	Ceļa atrašana no sākuma līdz mērķim.
Trajectory	Kustības ceļš ar laika, ātruma un paātrinājuma informāciju.
Sensor fusion	Vairāku sensoru datu apvienošana robustākam novērtējumam.

20. [ARHĪVS] Izstrādes statuss un nākamie posmi

Versija 0.5 turpina vienoto kursa dokumentu. Papildus iepriekšējās versijas kursa saturam pievienots pilns laboratoriju izsniegšanas komplekts: skolēna darba lapas, pasniedzēja skaidrojumi, atsaucis risinājumi, testēšanas protokoli, vērtēšanas kritēriji un projekta inženiertehniskās dokumentācijas veidnes. Dokumentu var izmantot gan vienas 90 min nodarbības vadīšanai, gan ilgtermiņa profesionāla portfolio veidošanai.

- Pabeigts: 36 nodarbību gada karkass un detalizētas mācību tēmas.
- Pabeigts v0.4: skolēnu darba lapu komplekts ar atbildēm un A/B/C diferenciaciju.
- Pabeigts v0.4: algoritmu, Python, C++ un integrācijas pārbaudes darbi ar vērtēšanas atslēgām.
- Pabeigts v0.4: Python/C++/SQL/IoT/robotikas ātrās atsaucis lapas.
- Pabeigts v0.4: detalizētas laboratorijas Python, SQLite, MQTT, LiDAR, OpenCV, IMU, kinemātikai un A*.
- Paplašināts v0.4: universitātes pārejas temati un pasniedzēja pašmācības ceļš.
- Pabeigts v0.5: L1–L14 skolēna darba lapas, pasniedzēja risinājumi, testēšanas protokoli, projektu inženiertehniskās dokumentācijas veidnes un portfolio/pašvērtējuma sadaļa.
- Nākamais pielāgošanas solis: konkrētās iestādes prasītie kontaktstundu, patstāvīgā darba, kvalifikācijas un literatūras lauki.

Versiju princips

Turpmāk nevajag veidot atsevišķus fragmentu failus. Ja rodas jauna tēma vai labojums, to ievieto atbilstošajā nodaļā un palielina dokumenta versiju (0.2, 0.3 ... 1.0).

21. Skolēnu darba lapas un diferencētie A/B/C uzdevumi

Šī sadaļa paredzēta drukāšanai vai izsniegšanai digitālā formā. Darba lapas veidotas tā, lai skolēns vispirms formulētu problēmu un algoritmu, tikai pēc tam pārietu uz kodu. Atbildes atrodas uzreiz pēc katras darba lapas pasniedzēja vajadzībām; skolēnu izdrukā tās var izņemt.

Darba lapu izmantošanas princips

A līmenis pārbauda minimālo jēdzienu izpratni. B līmenis prasa izmantot zināšanas tehniskā situācijā. C līmenis paredz robežgadījumus, kļūdu analīzi, sistēmas drošību vai algoritma optimizāciju.

DL1 — Algoritms, ievade, apstrāde un izvade

Skolēna uzdevumi:

1. Apraksti ar saviem vārdiem, kas ir algoritms.
2. Automātiskajām durvīm nosauc vismaz 3 ievades, 2 apstrādes lēmumus un 2 izvades.
3. Uzraksti 8–12 soļu algoritmu automātisko durvju darbībai.
4. Atzīmē vienu robežgadījumu, kas var radīt nepareizu darbību.

Atbildes / pasniedzēja atslēga:

- Algoritms — galīga, precīzi definētu darbību secība konkrēta mērķa sasniegšanai.
- Ievades piemēri: kustības sensors, drošības sensors, gala slēdzis. Apstrāde: noteikt vai drīkst atvērt/aizvērt. Izvades: motors, indikators/signalizācija.
- Soļu secībai jāietver gaidīšana, cilvēka noteikšana, atvēršana, gala pozīcija, drošības pārbaude, aizvēršana un atgriešanās gaidīšanas režīmā.
- Robežgadījums: cilvēks paliek ailē; sensors nedarbojas; durvis nerasniedz gala slēdzi.

DL2 — Zarošanās un Boolean loģika

Skolēna uzdevumi:

1. Ieraksti patiesības tabulu A AND B un A OR B.
2. Izveido nosacījumu: motors drīkst startēt tikai tad, ja durvis aizvērtas, E-stop nav aktivizēts un drošības relejs ir OK.
3. Izskaidro atšķirību starp AND un OR drošības interlock piemērā.
4. Izveido pseidokodu ar NORMAL, WARNING un ALARM temperatūras stāvokļiem.

Atbildes / pasniedzēja atslēga:

- AND ir patiess tikai tad, ja abi nosacījumi patiesi; OR — ja patiess vismaz viens.
- DoorClosed AND NOT EmergencyStop AND SafetyRelayOK.

- AND parasti izmanto, ja jāizpildās visiem drošības priekšnosacījumiem; OR — ja pietiek ar vienu no vairākiem trauksmes iemesliem.
- Piemērs: IF T>=80 ALARM; ELSE IF T>=60 WARNING; ELSE NORMAL.

DL3 — Blokshēma un testa dati

Skolēna uzdevumi:

1. Uzzīmē blokshēmu ventilatora vadībai: ieslēgt virs 30 °C, izslēgt zem 26 °C.
2. Norādi, kur blokshēmā ir ievade, lēmums un izvade.
3. Izveido testa tabulu T = 24, 26, 29, 30, 31, 80.
4. Kāpēc vērtība tieši 30 °C ir svarīga testa robežvērtība?

Atbildes / pasniedzēja atslēga:

- Jābūt sākumam, temperatūras nolasīšanai, lēmumam, ventilatora stāvoklim un atkārtošanās cilpai.
- Ievade — temperatūra; lēmums — salīdzinājums; izvade — fan ON/OFF.
- Pie katras vērtības jānorāda sagaidāmais stāvoklis un jāņem vērā histerēzes stāvokļa saglabāšana 26...30 °C intervālā.
- Tā ir prasības robeža; kļūdas bieži rodas ar > pret >=.

DL4 — Cikli, timeout un watchdog

Skolēna uzdevumi:

1. Kad izmantotu FOR, bet kad WHILE?
2. Uzraksti pseidokodu, kas nolasa 10 temperatūras mērījumus un aprēķina vidējo.
3. Uzraksti pseidokodu “gaidi līdz sensors READY”, bet ar 5 s timeout.
4. Izskaidro, kāpēc bezgalīgs cikls tehniskā sistēmā var būt bīstams.

Atbildes / pasniedzēja atslēga:

- FOR — zināms atkārtojumu skaits; WHILE — atkārtojums atkarīgs no nosacījuma.
- sum=0; FOR i=1..10: sum+=T; avg=sum/10.
- start_time; WHILE NOT ready AND elapsed<5s: read sensor; pēc cikla — READY vai TIMEOUT kļūda.
- Var bloķēt citas funkcijas, aizkavēt drošības reakciju, pārkarst aktuators vai sistēma “iestrēgt”.

DL5 — Python: ievade, tipi un aprēķins

Skolēna uzdevumi:

1. Uzraksti Python programmu, kas ievada part_count un reject_count un aprēķina brāķa procentu.
2. Pievieno validāciju, lai reject_count nevarētu būt lielāks par part_count.
3. Kas notiks, ja part_count = 0?

4. Izveido 4 testa gadījumus.

Atbildes / pasniedzēja atslēga:

- Pamats: `int(input())`, aprēķins `reject_count/part_count*100`.
- IF `reject_count<0 OR part_count<0 OR reject_count>part_count`: kļūda.
- Dalīšana ar nulli — jāapstrādā pirms aprēķina.
- Normāls, 0 brāķa, 100% brāķa, `part_count=0`, nederīga negatīva ievade.

DL6 — Python: nosacījumi un stāvokļi

Skolēna uzdevumi:

1. Izveido funkciju `classify_temperature(t)`, kas atgriež `NORMAL/WARNING/ALARM`.
2. Paplašini ar sensora kļūdu, ja `t < -50` vai `t > 150`.
3. Izskaidro `elif` nozīmi.
4. Uzraksti vienu testu katram stāvoklim.

Atbildes / pasniedzēja atslēga:

- Funkcijā vispirms validācija, tad `alarm/warning/normal` robežas.
- Sensora kļūda jāpārbauda pirms procesa robežām.
- `elif` novērš visu nosacījumu neatkarīgu izpildi un veido savstarpēji izslēdzošu izvēli.
- Jāiekļauj arī tiešās robežvērtības.

DL7 — Python: cikli, saraksti un debugging

Skolēna uzdevumi:

1. Dots saraksts `[22.1, 22.4, 85.0, 22.3, -999]`. Atrodi `min`, `max` un vidējo derīgajiem datiem.
2. Pieņem, ka `-999` ir sensora kļūdas kods. Filtrē to.
3. `85.0` ir ārpus procesa robežas `>80`. Atsevišķi saskaiti `alarm` mērījumus.
4. Uzraksti, kā ar `print` vai `debugger` noteikt, kur kļūst nepareizs sum.

Atbildes / pasniedzēja atslēga:

- Derīgie procesa dati: `22.1, 22.4, 85.0, 22.3`; `-999` jāatmet kā `invalid`.
- Vidējais jāaprēķina tikai pēc validācijas politikas; var diskutēt, vai `alarm` vērtība ir derīgs mērījums, bet ārpus procesa robežas.
- `Alarm_count=1`, ja `85.0` tiek uzskatīts par derīgu sensoru mērījumu.
- Novērot mainīgā vērtību katrā iterācijā/`breakpoint`.

DL8 — C++ tipi, const un precizitāte

Skolēna uzdevumi:

1. Izvēlies tipu: detaļu skaits, temperatūra, alarma stāvoklis, ierīces nosaukums.
2. Kāpēc MAX_TEMPERATURE labāk deklarēt const?
3. Kas var notikt ar integer overflow?
4. Uzraksti C++ deklarācijas.

Atbildes / pasniedzēja atslēga:

- int/unsigned vai fiksēta platuma vesels tips; double; bool; std::string.
- Konfigurācijas robežu nejauši nedrīkst mainīt programmas izpildes laikā.
- Vērtība var pāriet ārpus tipa diapazona un radīt nepareizu rezultātu.
- Piemērs: const double MAX_TEMPERATURE_C = 80.0; bool alarm_active=false;

DL9 — C++ funkcijas un moduļi

Skolēna uzdevumi:

1. Sadaliet “sensoru monitoringu” funkcijās.
2. Uzraksti funkcijas parakstu bool isValidTemperature(double t).
3. Uzraksti funkciju double average(const double values[], int n) konceptuālā līmenī.
4. Nosauc priekšrocību, ko funkcijas dod testēšanai.

Atbildes / pasniedzēja atslēga:

- readSensor, validate, classify, log, updateOutput u.c.
- bool isValidTemperature(double t);
- Jāpārbauda n>0; ciklā saskaitīt un dalīt ar n.
- Katru funkciju var testēt izolēti ar zināmām ievadēm.

DL10 — SQL tehniskajiem mērījumiem

Skolēna uzdevumi:

1. Izveido tabulas struktūru measurements ar id, timestamp, sensor_id, value, unit, status.
2. Uzraksti INSERT vienam mērījumam.
3. Uzraksti SELECT visiem ALARM ierakstiem.
4. Uzraksti SELECT pēdējiem 10 ierakstiem.

Atbildes / pasniedzēja atslēga:

- CREATE TABLE ar atbilstošiem INTEGER/TEXT/REAL tipiem.
- INSERT INTO measurements (...) VALUES (...).
- SELECT * FROM measurements WHERE status='ALARM';
- SELECT * FROM measurements ORDER BY timestamp DESC LIMIT 10;

DL11 — IoT, MQTT un JSON

Skolēna uzdevumi:

1. Izveido topic temperatūras sensoram telpā 101.
2. Izveido JSON payload ar sensor_id, value, unit, timestamp un quality.
3. Kurš ir publisher un kurš subscriber?
4. Ko sistēmai darīt, ja internets pazūd 10 minūtes?

Atbildes / pasniedzēja atslēga:

- Piemērs: factory/room101/temperature.
- {"sensor_id":"T101","value":24.6,"unit":"C","timestamp":"...","quality":"GOOD"}.
- Sensors/edge ierīce publicē; datu glabātājs/dashboard abonē.
- Lokāli buferēt datus, norādīt connection state, pēc savienojuma atjaunošanas droši nosūtīt vai sinhronizēt.

DL12 — Robots: kinemātika, LiDAR un IMU

Skolēna uzdevumi:

1. 2DOF robotam paskaidro, ko nozīmē forward kinematics.
2. LiDAR dod $r=2$ m pie 30° . Uzraksti formulas x, y aprēķinam.
3. Kāpēc IMU gyro orientācija ar laiku driftē?
4. Kāpēc autonomam robotam apvieno LiDAR, IMU un enkoderus?

Atbildes / pasniedzēja atslēga:

- No locītavu leņķiem aprēķina gala efektora pozīciju/pozu.
- $x=r \cos(\theta)$, $y=r \sin(\theta)$.
- Neliela gyro bias kļūda tiek integrēta laikā.
- Sensoru kļūdas ir atšķirīgas; sensor fusion dod robustāku stāvokļa novērtējumu.

21.13. A/B/C diferencēto uzdevumu banka

Tēma	A līmenis — pamati	B līmenis — pielietojums	C līmenis — padziļināti
Algoritmi	Apraksti algoritmu 6–8 soļos un nosauc I/O.	Izveido blokshēmu ar vienu zarošanos un vienu ciklu.	Pievieno timeout, kļūdu režīmu un argumentē drošības prioritāti.

Tēma	A līmenis — pamati	B līmenis — pielietojums	C līmenis — padziļināti
Python	Ievade, aprēķins, print.	Funkcija + validācija + saraksta apstrāde.	Testi, exception apstrāde, datu klase/modulis un logika robežgadījumiem.
C++	Mainīgie, cin/cout, if.	Funkcijas, masīvs/vector, switch, validācija.	Klase sensoram, const, references, vienību drošība un testējama arhitektūra.
SQL	INSERT un vienkāršs SELECT.	WHERE, ORDER BY, agregācija.	Normalizācija, indeksu ideja, transakciju/integritāte s kļūdu scenāriji.
IoT	Nosauc sensor→device→cloud ķēdi.	MQTT topic + JSON payload + subscriber.	Offline buffering, QoS izvēles pamatojums, timestamp/quality un drošības modelis.
Mehatronika	Nosauc sensoru un aktuatoru.	Izveido closed-loop blokshēmu.	Histerēze/PID, interlock, fail-safe un atteices analīze.
Robotika	Nosauc SENSE→PLAN→ACT posmus.	Aprēķini 2DOF forward kinematics un LiDAR x/y.	Sensor fusion, SLAM, replanning un degraded mode scenārijs.

22. Pilni pārbaudes darbi un vērtēšanas atslēgas

Pārbaudes darbi paredzēti 60–90 min atkarībā no grupas sagatavotības. Punktu skaits var tikt pārveidots iestādes vērtēšanas sistēmā. Zem katra darba pievienota pasniedzēja atslēga un minimālie vērtēšanas kritēriji.

22.1. Pārbaudes darbs — Algoritmu pamati

Ieteicamais laiks: 80 min. Atļauts izmantot tukšu blokshēmas simbolu lapu bez atbildēm.

A daļa: teorija — 10 p.

1. Definē algoritmu un nosauc četras tā īpašības.
2. Izskaidro ievade → apstrāde → izvade modeli.

3. Nosauc atšķirību starp FOR un WHILE.

4. Ko tehniskā sistēmā nozīmē timeout?

B daļa: algoritma analīze — 10 p.

1. Dots pseidokods: nolasīt T; ja $T > 80 \rightarrow \text{ALARM}$, citādi RUN. Nosaki rezultātu $T=79,80,81$ un atrod robežnosacījuma risku.

2. Atrodi kļūdu algoritmā, kur motors tiek ieslēgts, bet nav neviena STOP nosacījuma.

C daļa: blokshēma — 15 p.

1. Izveido blokshēmu sūknim: sūknis drīkst darboties, ja līmenis zem HIGH, E-stop nav aktīvs un sensors ir derīgs; pie sensora kļūdas — ALARM un STOP.

D daļa: patstāvīgs algoritms — 15 p.

1. Izveido 10–15 soļu algoritmu konveijeram, kas saskaita detaļas, apstājas pie 100 detaļām un nekavējoties apstājas, ja aktivizēts E-stop. Iekļauj ievades, izvades un vismaz vienu timeout.

Kopā: 50 punkti.

Vērtēšanas atslēga

- Teorijā pilns punkts par precīzu jēdzienu, daļējs par pareizu būtību bez terminoloģijas.
- Robežvērtības analīzē jānorāda, ka $T=80$ neaktivizē ALARM, ja izmantots $>$; jāargumentē, vai prasība paredz $>=$.
- Blokshēmā drošības nosacījumiem jābūt pirms sūkņa ieslēgšanas un jābūt skaidram STOP ceļam.
- Patstāvīgajā algoritmā maksimālo punktu iegūst par loģisku stāvokļu secību, drošības prioritāti un testējamiem nosacījumiem.

22.2. Pārbaudes darbs — Python

Ieteicamais laiks: 90 min. Var izmantot Python dokumentācijas/atsauces lapu, bet ne gatavu risinājumu.

A daļa: koda lasīšana — 10 p.

1. Nosaki programmas izvadi dotam if/elif piemēram.

2. Nosaki, cik reizes izpildīsies dotais for un while cikls.

3. Atrodi divas kļūdas kodā ar input() un skaitlisku aprēķinu.

B daļa: funkcija un validācija — 15 p.

1. Uzraksti classify_temperature(t): SENSOR_ERROR ārpus $[-50,150]$, ALARM $>=80$, WARNING $>=60$, citādi NORMAL.

C daļa: datu kopa — 15 p.

1. Dots saraksts ar 12 temperatūrām un vienu None. Aprēķini derīgo vidējo, min, max un ALARM skaitu.

D daļa: tehnisks mini uzdevums — 10 p.

1. Uzraksti programmu, kas atkārtoti pieņem mērījumus līdz ievadei q, validē datus un beigās izvada kopsavilkumu.

Kopā: 50 punkti.

Vērtēšanas atslēga

- Jāvērtē ne tikai gala rezultāts, bet datu validācijas secība un robežvērtības.
- None/nederīgu vērtību nedrīkst iekļaut vidējā.
- Programmai jābeidzas kontrolēti un nedrīkst avarēt pie nederīgas teksta ievades, ja paredzēta exception apstrāde.
- Papildu punkts var tikt piešķirts par funkciju sadalījumu un skaidriem mainīgo nosaukumiem, nepārsniedzot darba kopējo maksimumu.

22.3. Pārbaudes darbs — C++

Ieteicamais laiks: 90 min. Vērtē kompilējamu struktūru, tipu izvēli un algoritma korektumu.

A daļa: tipi un sintakse — 12 p.

1. Izvēlies tipu četriem tehniskiem datiem un pamato.
2. Izlabo 5 sintakses/tipu kļūdas dotā C++ fragmentā.
3. Izskaidro const nozīmi.

B daļa: zarošanās un cikli — 12 p.

1. Uzraksti while izvēlni START/STOP/STATUS/EXIT ar switch.
2. Pievieno drošības nosacījumu, kas START aizliedz pie alarm_active=true.

C daļa: funkcijas un masīvi — 18 p.

1. Izveido isValid(double t), classify(double t) un average(...) funkciju.
2. Aprēķini no masīva derīgo vidējo un alarmu skaitu.

D daļa: OOP/debugging — 18 p.

1. Izveido klases Sensor minimālo interfeisu ar id, value, valid un update().
2. Dots kods ar kļūdu: vidējais tiek dalīts ar visa masīva izmēru, lai gan nederīgie dati izlaisti. Izskaidro un izlabo.

Kopā: 60 punkti.

Vērtēšanas atslēga

- Kompilējamībai ir nozīme, bet par nelielu sintakses kļūdu nezaudē visu algoritma punktu daļu.
- Tipu izvēlei jāatbilst datu nozīmei un precizitātei.
- Vidējā aprēķinam dalītājam jābūt derīgo mērījumu skaitam.
- OOP uzdevumā vērtē atbildības skaidrību, nevis sarežģītu inheritance izmantošanu.

22.4. Pārbaudes darbs — SQL, IoT un sistēmu integrācija

Ieteicamais laiks: 75 min. Galvenais mērķis — izprast datu ķēdi un spēt izsekot vienam mērījumam no sensora līdz datubāzei.

A daļa: SQL — 15 p.

1. Izveido measurements tabulu.
2. Uzraksti INSERT.
3. Atlasiet pēdējos 20 ALARM mērījumus konkrētam sensoram.

B daļa: JSON/MQTT — 10 p.

1. Izveido topic un JSON payload temperatūras sensoram.
2. Nosauc publisher, broker un subscriber lomas.

C daļa: arhitektūra — 15 p.

1. Uzzīmē sensors → edge → network → broker/API → DB → dashboard ķēdi un pieraksti katra bloka atbildību.

D daļa: atteice un drošība — 10 p.

1. Apraksti sistēmas uzvedību, ja pazūd internets, bet lokālajai vadībai jāturpina darboties.
2. Nosauc vismaz trīs datu kvalitātes/kiberdrošības riskus.

Kopā: 50 punkti.

Vērtēšanas atslēga

- SQL sintaksē pieļauj nelielas dialekta atšķirības, ja loģika pareiza.
- Arhitektūrā lokāla drošības vadība nedrīkst būt atkarīga no mākoņa pieejamības.
- Atteices risinājumā vērtē buffering, timestamp, quality/status un savienojuma atjaunošanas loģiku.
- Drošības riski var ietvert vāju autentifikāciju, nešifrētu transportu, neuzticamus timestamp, neatļautu komandu izpildi u.c.

23. Ātrās atsauces lapas: Python, C++, SQL, IoT un robotika

Šīs lapas paredzētas kā ilgtermiņa rokasgrāmatas kodols. Tās var izdrukāt atsevišķi vai atļaut izmantot praktiskajos darbos, lai vērtētu problēmas risināšanu, nevis sintakses iegaumēšanu.

23.1. Python ātrā atsauce

```
# Izvade un ievade
print("Temperature:", temperature)
text = input("Ievadi vērtību: ")
value = float(text)

# Nosacījums
if value >= 80:
    status = "ALARM"
elif value >= 60:
    status = "WARNING"
else:
    status = "NORMAL"

# Cikli
for i in range(10):
    print(i)

while running:
    # atkārtota procesa loģika
    break

# Funkcija
def classify_temperature(t):
    if t < -50 or t > 150:
        return "SENSOR_ERROR"
    if t >= 80:
        return "ALARM"
    return "NORMAL"

# Saraksts
values = [21.4, 22.0, 23.1]
avg = sum(values) / len(values)
```

- Pirms float()/int() konversijas apsver nederīgu teksta ievadi.
- Vienmēr pārbaudi dalītāju pret nulli.
- Robežvērtības testē atsevišķi: tieši zem, tieši uz un tieši virs robežas.
- Tehniskos datus glabā arī mērvienību, timestamp un quality/status, ja tas ir svarīgi.

23.2. C++ ātrā atsauce

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

const double MAX_TEMPERATURE_C = 80.0;

bool isValid(double t) {
    return t >= -50.0 && t <= 150.0;
}

string classify(double t) {
    if (!isValid(t)) return "SENSOR_ERROR";
    if (t >= 80.0) return "ALARM";
    if (t >= 60.0) return "WARNING";
    return "NORMAL";
}

int main() {
    vector<double> values{21.0, 22.3, 81.0};
    for (double t : values) {
        cout << t << " -> " << classify(t) << "\n";
    }
    return 0;
}
```

- Izvēlies tipu pēc datu nozīmes un diapazona.
- const/constexpr izmanto konfigurācijas robežām, kuras nevajag nejauši mainīt.
- Funkcijām dod vienu skaidru atbildību.
- Atšķir kompilēšanas kļūdu, izpildlaika kļūdu un loģikas kļūdu.

23.3. SQL ātrā atsauce

```
CREATE TABLE measurements (  
  id INTEGER PRIMARY KEY,  
  timestamp TEXT NOT NULL,  
  sensor_id TEXT NOT NULL,  
  value REAL,  
  unit TEXT,  
  status TEXT  
);  
  
INSERT INTO measurements(timestamp, sensor_id, value, unit, status)  
VALUES ('2026-09-10T12:30:00', 'T101', 24.6, 'C', 'NORMAL');  
  
SELECT *  
FROM measurements  
WHERE sensor_id = 'T101' AND status = 'ALARM'  
ORDER BY timestamp DESC  
LIMIT 20;  
  
SELECT sensor_id, AVG(value), MIN(value), MAX(value)  
FROM measurements  
GROUP BY sensor_id;
```

- Tabula apraksta entītiiju/datu kopu; rinda — vienu ierakstu; kolonna — atribūtu.
- Primārā atslēga identificē ierakstu.
- WHERE filtrē, ORDER BY kārtot, GROUP BY grupē, agregācijas funkcijas apkopo.
- Ražošanas/IoT datos timestamp un sensor_id bieži ir tikpat svarīgi kā pati value.

23.4. IoT / MQTT / HTTP / JSON ātrā atsauce

Topic:
factory/room101/temperature

JSON payload:
{
 "sensor_id": "T101",
 "value": 24.6,
 "unit": "C",
 "timestamp": "2026-09-10T12:30:00Z",
 "quality": "GOOD"
}

MQTT:
Publisher -> Broker -> Subscriber

HTTP/REST piemērs:
POST /api/measurements
GET /api/measurements?sensor_id=T101

- MQTT ir ziņapmaiņas modelis; HTTP/REST parasti ir pieprasījums/atbilde.
- Cloud nedrīkst būt vienīgais kritiskas drošības vadības slānis.
- Plāno offline režīmu, bufferi, atkārtotu nosūtīšanu un dublikātu novēršanu.
- Autentifikācija, šifrēts transports, piekļuves tiesības un auditējama konfigurācija ir profesionālas sistēmas daļa.

23.5. Robotikas matemātikas un uztveres ātrā atsauce

Leņķis grādos -> radiānos:

$$\text{theta\_rad} = \text{theta\_deg} * \pi / 180$$

Polar -> Cartesian:

$$x = r * \cos(\text{theta})$$

$$y = r * \sin(\text{theta})$$

2DOF forward kinematics:

$$x = L1 * \cos(t1) + L2 * \cos(t1+t2)$$

$$y = L1 * \sin(t1) + L2 * \sin(t1+t2)$$

Differential drive (pamata ideja):

$$v = (vR + vL) / 2$$

$$\omega = (vR - vL) / \text{wheel\_base}$$

Autonoma robota cikls:

SENSE -> PERCEIVE -> LOCALIZE/MAP -> DECIDE -> PLAN -> CONTROL -> ACT

- Vienmēr norādi koordinātu sistēmu un mērvienības.
- Sensoru dati ir saistīti ar laiku — timestamp neatbilstība var sabojāt sensor fusion.
- LiDAR: attālums + leņķis; kamera: pikseļi; IMU: paātrinājums un leņķiskais ātrums.
- Path ir ģeometrisks ceļš; trajectory pievieno laiku, ātrumu un paātrinājumu.

23.6. Debugging un testēšanas kontroles saraksts

- Vai prasība ir uzrakstīta viennozīmīgi?
- Vai ievades datu tipi un mērvienības ir zināmi?
- Vai testētas robežvērtības?
- Vai apstrādāta nederīga ievade un sensora kļūda?
- Vai ciklam ir droša iziešana/timeout, ja tas nepieciešams?
- Vai programma saglabā/izvada pietiekamu diagnostiku?
- Vai funkcijas var testēt atsevišķi?
- Vai fizikāla sistēma kļūmes gadījumā pāriet drošā stāvoklī?
- Vai rezultāts ir reproducējams ar saglabātiem testa datiem?

24. Laboratorijas darbu rokasgrāmata

Laboratorijas ir veidotas pieaugošā sarežģītībā. Daļai darbu pietiek ar datoru un simulētiem datiem; aparāturu var pievienot vēlāk, nemainot algoritma būtību. Katram darbam ir mērķis, priekšzināšanas, uzdevumi, pārbaudes punkti, atskaites minimums un paplašinājums.

Drošības piezīme

Ja laboratorija tiek pārcelta no simulācijas uz reālu motoru, releju, robotu vai citu aparāturu, jāievēro attiecīgās laboratorijas elektro- un mehāniskās drošības instrukcijas. Programmatūras STOP nav uzskatāms par vienīgo fiziskās drošības līdzekli.

L1 — Functional tree un algoritma dekompozīcija

Mērķis: No tehniskas prasības izveidot funkcionālu sadalījumu, pseidokodu un testu sarakstu.

Priekšzināšanas: Algoritma jēdziens, ievade/izvade, zarošanās.

Darba gaita:

1. Izvēlies sistēmu: automātiskās durvis, konveijers vai ventilācijas vadība.
2. Uzraksti vienu galveno funkciju darbības vārda formā.
3. Sadaliet to 4–6 galvenajās funkcijās un katru 2–4 apakšfunkcijās.
4. Pie katras gala funkcijas pieraksti, kā var pārbaudīt, ka tā izpildīta.
5. No viena zara izveido pseidokodu un blokshēmu.

Pārbaudes punkti:

- Funkcijas apraksta “ko sistēmai jādara”, nevis konkrētu komponentu nosaukumus.
- Kokā nav pazudušas drošības/atteices funkcijas.
- Testi ir novērojami un izmērāmi.

Atskaites minimums:

- Functional tree.
- Pseidokods vienam zaram.
- 3 testa gadījumi un 1 robežgadījums.

Paplašinājums:

- Pievieno fail-safe vai degraded mode zaru.

L2 — Python tehnisko datu klasifikators

Mērķis: Uzrakstīt testējamu Python funkciju mērījumu validācijai un statusa noteikšanai.

Priekšzināšanas: Python mainīgie, funkcijas, if/elif/else.

Darba gaita:

1. Izveido `classify_temperature(t)`.
2. Ārpus `[-50,150]` atgriez `SENSOR_ERROR`; `>=80` `ALARM`; `>=60` `WARNING`; pārējais `NORMAL`.
3. Izveido sarakstu ar vismaz 12 testu vērtībām, iekļaujot robežas.
4. Izvada tabulu `value → status`.
5. Aprēķina katra statusa skaitu.

Sākuma kods / references piemērs:

```
def classify_temperature(t):
    if t < -50 or t > 150:
        return "SENSOR_ERROR"
    if t >= 80:
        return "ALARM"
    if t >= 60:
        return "WARNING"
    return "NORMAL"

values = [-51, -50, 20, 59.9, 60, 79.9, 80, 150, 151]
```

Pārbaudes punkti:

- -50 un 150 ir derīgas robežvērtības pēc dotās prasības.
- 60 un 80 jātestē tieši uz robežas.
- Nederīgs sensors netiek klasificēts kā procesa `ALARM`.

Atskaites minimums:

- Kods.
- Testu tabula.
- Īss secinājums par robežvērtībām.

Paplašinājums:

- Pārveido robežas konfigurējamās konstantes.
- Uzraksti `pytest` stila testu funkcijas.

L3 — Python datu loggera simulācija

Mērķis: Simulēt periodiskus mērījumus, validēt, uzkrāt un izveidot kopsavilkumu.

Priekšzināšanas: Python cikli, saraksti, time/random pamati.

Darba gaita:

1. Ģenerē 60 temperatūras vērtības.
2. Katram mērījumam pievieno timestamp un statusu.
3. Atsevišķi uzglabā nederīgos mērījumus.
4. Aprēķini min/max/avg derīgajiem datiem un ALARM skaitu.
5. Saglabā CSV failā.

Sākuma kods / references piemērs:

```
import random
from datetime import datetime
import csv

rows = []
for i in range(60):
    value = round(random.uniform(20, 85), 2)
    rows.append({
        "timestamp": datetime.now().isoformat(timespec="seconds"),
        "value": value
    })
```

Pārbaudes punkti:

- Vidējais neietver nederīgus mērījumus.
- CSV satur kolonnas un mērvienību/statusu.
- Timestamp ir reproducējami interpretējams.

Atskaites minimums:

- CSV paraugs.
- Kopsavilkuma tabula.
- Apraksts, kā loggeris uzvestos pie sensora kļūdas.

Paplašinājums:

- Pievieno 5 s periodu un timeout simulāciju.

L4 — C++ vadības stāvokļu programma

Mērķis: Realizēt vienkāršu iekārtas state machine C++.

Priekšzināšanas: C++ if/switch/while, bool, funkcijas.

Darba gaita:

1. Definē stāvokļus IDLE, RUNNING, ALARM, STOPPED.
2. Izveido izvēlni START/STOP/STATUS/FAULT/RESET/EXIT.
3. START atļauts tikai, ja nav alarma.
4. FAULT nekavējoties pāriet ALARM.
5. RESET drīkst atgriezt IDLE tikai pēc kļūdas nosacījuma noņemšanas.

Sākuma kods / references piemērs:

```
enum class State { IDLE, RUNNING, ALARM, STOPPED };

State state = State::IDLE;
bool faultActive = false;

// switch(command) { ... }
```

Pārbaudes punkti:

- Nav neatļautas pārejas ALARM→RUNNING bez reset.
- EXIT kontrolēti beidz ciklu.
- Stāvokļu nosaukumi atspoguļo sistēmas nozīmi.

Atskaites minimums:

- State transition tabula.
- C++ kods.
- Vismaz 6 pāreju testi.

Paplašinājums:

- Izveido atsevišķu funkciju handleCommand(...) un logEvent(...).

L5 — C++ Sensor klase un datu kvalitāte

Mērķis: Izveidot vienkāršu klasi, kas kapsulē mērījumu, identifikatoru un validāciju.

Priekšzināšanas: C++ class/object, funkcijas, const.

Darba gaita:

1. Izveido class Sensor ar id, value, valid.
2. Pievieno update(double newValue).

3. Pievieno isValidRange() vai validācijas logiku update().
4. Izveido divus sensoru objektus un vairākus mērījumus.
5. Izvada tikai derīgos mērījumus ar sensoru ID.

Sākuma kods / references piemērs:

```
class Sensor {
private:
    std::string id;
    double value{0.0};
    bool valid{false};
public:
    Sensor(const std::string& sensorId) : id(sensorId) {}
    void update(double newValue) {
        value = newValue;
        valid = (newValue >= -50.0 && newValue <= 150.0);
    }
    bool isValid() const { return valid; }
    double getValue() const { return value; }
};
```

Pārbaudes punkti:

- Privātie dati netiek mainīti patvaļīgi no ārpusē.
- const getter nemaina objektu.
- Validācijas politika ir skaidra.

Atskaites minimums:

- Klases diagramma vienkāršotā formā.
- Kods.
- Testu tabula.

Paplašinājums:

- Pievieno timestamp/status vai atvasinātu TemperatureSensor klasi tikai kā paplašinājumu, nevis obligāti.

L6 — Python + SQLite tehnisko mērījumu datubāze

Mērķis: No Python izveidot SQLite DB, ierakstīt mērījumus un izpildīt analītiskus SELECT.

Priekšzināšanas: Python, SQL pamati.

Darba gaita:

1. Izveido measurements tabulu.
2. Ģenerē vismaz 100 simulētus mērījumus un INSERT datubāzē.
3. Atlasi visus ALARM.
4. Atlasi vidējo/min/max pa sensor_id.
5. Izvada pēdējos 10 ierakstus.

Sākuma kods / references piemērs:

```
import sqlite3

con = sqlite3.connect("measurements.db")
cur = con.cursor()
cur.execute("""
CREATE TABLE IF NOT EXISTS measurements(
    id INTEGER PRIMARY KEY,
    timestamp TEXT NOT NULL,
    sensor_id TEXT NOT NULL,
    value REAL,
    unit TEXT,
    status TEXT
)
""")
con.commit()
```

Pārbaudes punkti:

- Parametrizēti INSERT ir labāki par SQL virknes salīmēšanu.
- Datubāzes savienojums tiek aizvērts vai izmantots context manager.
- Statuss atbilst tai pašai klasifikācijas loģikai, kas izmantota Python blokā.

Atskaites minimums:

- DB shēma.
- 3 SELECT vaicājumi un rezultāti.
- Secinājums par datubāzes priekšrocībām pret tikai CSV.

Paplašinājums:

- Pievieno indeksu sensor_id/timestamp un salīdzini vaicājuma ideju uz lielāka datu apjoma.

L7 — MQTT/IoT datu plūsmas simulācija

Mērķis: Projektēt un simulēt publisher→broker→subscriber datu ceļu ar JSON payload.

Priekšzināšanas: IoT arhitektūra, JSON, Python pamati.

Darba gaita:

1. Definē topic hierarhiju vismaz 3 sensoriem.
2. Definē JSON shēmu: sensor_id, timestamp, value, unit, quality.
3. Publisher pusē ģenerē datus.
4. Subscriber pusē validē payload un sagatavo INSERT DB.
5. Apraksti offline scenāriju un dublikātu novēršanas principu.

Sākuma kods / references piemērs:

```
payload = {  
    "sensor_id": "T101",  
    "timestamp": "2026-09-10T12:30:00Z",  
    "value": 24.6,  
    "unit": "C",  
    "quality": "GOOD"  
}  
# JSON serializācija: json.dumps(payload)
```

Pārbaudes punkti:

- Topic nosaukumi ir strukturēti un paredzami.
- Payload satur pietiekamu kontekstu, lai mērījumu interpretētu vēlāk.
- Lokālā vadība nav atkarīga no broker/cloud pieejamības.

Atskaites minimums:

- Arhitektūras shēma.
- Topic saraksts un JSON piemērs.
- Atteices scenārijs.

Paplašinājums:

- Ja pieejams lokāls MQTT brokeris, realizē reālu publish/subscribe ar paho-mqtt.

L8 — Mehatronikas feedback un histerēzes simulācija

Mērķis: Modelēt closed-loop temperatūras vadību un saprast, kā histerēze novērš biežu pārslēgšanos.

Priekšzināšanas: Python cikli, sistēmas input/process/output.

Darba gaita:

1. Izveido procesa temperatūras vienkāršu modeli.
2. Ja $T > 30$, cooling ON; ja $T < 27$, cooling OFF; starp robežām saglabā iepriekšējo stāvokli.
3. Simulē vismaz 200 laika soļus.
4. Uzzīmē temperatūru un vadības signālu.
5. Salīdzini ar vadību bez histerēzes.

Sākuma kods / references piemērs:

```
setpoint_high = 30.0
setpoint_low = 27.0
cooling = False

if temperature > setpoint_high:
    cooling = True
elif temperature < setpoint_low:
    cooling = False
# citādi cooling saglabā iepriekšējo stāvokli
```

Pārbaudes punkti:

- Starp 27...30 °C nav nepārtrauktas pārslēgšanās.
- Modeli ir skaidrs, kā cooling ietekmē temperatūras izmaiņu.
- Grafikā var redzēt kontroles rezultātu.

Atskaites minimums:

- Modeļa apraksts.
- Grafiks.
- Secinājums par histerēzes ieguvumu.

Paplašinājums:

- Pievieno vienkāršu P vai PID regulatoru un salīdzini.

L9 — 2DOF robotu kinemātika Python

Mērķis: Aprēķināt un vizualizēt manipulatora gala efektora pozīciju un atrisināt vienkāršu inverse kinematics.

Priekšzināšanas: Trigonometriskās funkcijas, Python math/NumPy pamati.

Darba gaita:

1. Definē L1, L2, theta1, theta2.
2. Aprēķini x,y ar forward kinematics.
3. Uzzīmē abu posmu stāvokli.
4. Dotam mērķim x,y pārbaudi sasniedzamību.
5. Aprēķini abas iespējamās elbow-up/elbow-down konfigurācijas, ja tās eksistē.

Sākuma kods / references piemērs:

```
import math

L1, L2 = 0.5, 0.3
t1 = math.radians(30)
t2 = math.radians(45)

x1 = L1 * math.cos(t1)
y1 = L1 * math.sin(t1)
x2 = x1 + L2 * math.cos(t1 + t2)
y2 = y1 + L2 * math.sin(t1 + t2)
print(x2, y2)
```

Pārbaudes punkti:

- Grādi tiek pārveidoti radiānos.
- Sasniedzamība: $|L1-L2| \leq r \leq L1+L2$.
- Atšķir joint space no Cartesian space.

Atskaites minimums:

- Formulas.
- Kods un vizualizācija.
- Vismaz 3 konfigurāciju rezultāti.

Paplašinājums:

- Izveido homogēnās transformācijas matricas vai DH tabulu.

L10 — LiDAR scan processing Python

Mērķis: No simulēta 2D LiDAR skenējuma izveidot punktus, filtrēt nederīgus datus un noteikt drošības sektoru.

Priekšzināšanas: Trigonometrija, Python saraksti/NumPy.

Darba gaita:

1. Izveido angles un distances datu kopu.
2. Atmet distances ≤ 0 , NaN vai virs sensora pieņemtajā diapazonā.
3. Pārveido polārās koordinātas x,y.
4. Atrodi tuvāko objektu priekšējā sektorā $-30^\circ \dots +30^\circ$.
5. Izvada STOP, ja tuvākais objekts < 0.7 m; citādi CLEAR.

Sākuma kods / references piemērs:

```
import math

points = []
for angle_deg, r in zip(angles, distances):
    if r is None or r <= 0 or r > 12.0:
        continue
    a = math.radians(angle_deg)
    x = r * math.cos(a)
    y = r * math.sin(a)
    points.append((x, y))
```

Pārbaudes punkti:

- Leņķu zīmes un koordinātu sistēma ir dokumentēta.
- Invalid dati netiek interpretēti kā šķērslis 0 m attālumā.
- Drošības sektors atšķiras no visa 360° minimuma.

Atskaites minimums:

- Functional tree.
- Punktu grafiks.
- Nearest obstacle un STOP/CLEAR rezultāts.

Paplašinājums:

- Pievieno vienkāršu clustering pēc blakus punktu attāluma.

L11 — Machine Vision ar OpenCV

Mērķis: Apstrādāt sagatavotu attēlu un noteikt vienkārša objekta kontūru, centru un izmēru pikseļos.

Priekšzināšanas: Python, attēla pamatjēdzieni.

Functional tree — laboratorijas sistēmas funkciju sadalījums.

Darba gaita:

1. Ielādē attēlu.
2. Pārveido grayscale.
3. Lieto threshold vai Canny atkarībā no attēla.
4. Atrodi kontūras.
5. Izvēlies atbilstošo kontūru pēc laukuma un aprēķini centru/bounding box.

Sākuma kods / references piemērs:

```
import cv2

img = cv2.imread("part.png")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, binary = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

for c in contours:
    area = cv2.contourArea(c)
    if area > 500:
        x, y, w, h = cv2.boundingRect(c)
        print(x, y, w, h)
```

Pārbaudes punkti:

- Attēla koordinātu sākumpunkts parasti ir augšējā kreisajā stūrī.
- Pikseļu izmērs nav automātiski mm — vajadzīga kalibrācija.
- Threshold ir jutīgs pret apgaismojumu; dokumentē izmantoto metodi.

Atskaītes minimums:

- Functional tree.
- Oriģinālais un apstrādātais attēls.
- Objekta centra/izmēra rezultāts.

Paplašinājums:

- Pievieno px→mm kalibrāciju ar zināmu etalonu vai kameru kalibrācijas ievadu.

L12 — IMU drift un complementary filter

Mērķis: Parādīt gyro integrācijas driftu un apvienot gyro ar accelerometer vienkāršā complementary filter.

Priekšzināšanas: Python, leņķi, diskrets laika solis.

Functional tree — laboratorijas sistēmas funkciju sadalījums.

Darba gaita:

1. Izveido simulētu gyro_rate un accel_angle datu sēriju.
2. Integrē gyro: $\text{angle} += \text{rate} * \text{dt}$.
3. Novēro drift, pievienojot mazu bias.
4. Apvieno ar accelerometer: $\text{angle} = \alpha * (\text{angle} + \text{gyro} * \text{dt}) + (1 - \alpha) * \text{accel\_angle}$.
5. Salīdzini grafikus.

Sākuma kods / references piemērs:

```
alpha = 0.98
angle = 0.0

for gyro_rate, accel_angle in samples:
    angle = alpha * (angle + gyro_rate * dt) + (1-alpha) * accel_angle
    fused.append(angle)
```

Pārbaudes punkti:

- dt ir sekundēs un atbilst datu paraugu biežumam.
- Gyro bias tiek integrēts un rada driftu.
- Accelerometer nav ideāls dinamiskas kustības laikā — filtrs ir kompromiss.

Atskaites minimums:

- Functional tree.
- 3 grafiki: gyro, accel, fused.
- Secinājums par sensor fusion.

Paplašinājums:

- Universitātes līmenī formulē stāvokļa/novērojuma modeli Kalman filtra ievadam.

L13 — A* ceļa plānošana mobilam robotam

Mērķis: Atrast ceļu režģa kartē un demonstrēt replanning pēc jauna šķēršļa.

Priekšzināšanas: Python datu struktūras, heuristikas ideja.

Functional tree — laboratorijas sistēmas funkciju sadalījums.

Darba gaita:

1. Definē 2D grid ar start, goal un obstacles.
2. Implementē vai izmanto pasniedzēja sagatavi A* algoritmam.
3. Vizualizē atrasto path.
4. Pievieno jaunu šķērslī uz path.
5. Palaid A* vēlreiz un salīdzini jauno ceļu.

Sākuma kods / references piemērs:

```
# f(n) = g(n) + h(n)
# g — izmaksas no start līdz n
# h — heuristika no n līdz goal

def heuristic(a, b):
    return abs(a[0]-b[0]) + abs(a[1]-b[1])
```

Pārbaudes punkti:

- Heuristika nedrīkst sistemātiski pārvērtēt izmaksas, ja grib optimālu klasisku A*.
- Path planning nav tas pats, kas motora vadība.
- Ja kartē nav ceļa, algoritmam jāatgriež kontrolēts NO_PATH.

Atskaites minimums:

- Functional tree.
- Sākotnējā un pārplānotā karte.
- Ceļa garumi un secinājums.

Paplašinājums:

- Salīdzini Dijkstra (h=0) un A*.

L14 — ROS 2 datu plūsmas un autonomās navigācijas analīze

Mērķis: Izprast ROS 2 mezglu, topic un TF lomas autonomā robota arhitektūrā, izmantojot simulatoru vai sagatavotus ekrānattēlus/datus.

Priekšzināšanas: Robotikas arhitektūra, LiDAR/odometry pamati.

Functional tree — laboratorijas sistēmas funkciju sadalījums.

Darba gaita:

1. Identificē sensoru nodes un to topics (/scan, /imu, /odom).
2. Izveido vienkāršu node-topic diagrammu.
3. Izskaidro TF saites map→odom→base_link→sensor frame.
4. Norādi, kur arhitektūrā darbojas SLAM/localization un planner/controller.
5. Analizē vienu atteici: pazūd /scan vai /odom.

Sākuma kods / references piemērs:

```
# ROS 2 Python konceptuāls subscriber fragments
# rclpy Node abonē sensor_msgs/LaserScan topic /scan
# callback funkcija saņem jaunu mērījumu un atjauno uztveres stāvokli.
```

Pārbaudes punkti:

- Topic ir datu kanāls, node — izpildāma programmatūras komponente.
- TF apraksta koordinātu sistēmu attiecības, nevis tikai “vēl vienu topic”.
- SLAM un localization nav vienlaikus identiska loma: viens var veidot karti, otrs pozicionēties zināmā kartē.

Atskaites minimums:

- Functional tree.
- Node/topic/TF diagramma.
- Atteices scenārijs un paredzamā drošā uzvedība.

Paplašinājums:

- Ja pieejams simulators: novēro ros2 topic list/echo, rqt_graph, TF tree un Nav2 statusus.

25. Pasniedzēja/universitātes padziļinājuma pielikums

Šī sadaļa paredzēta diviem mērķiem: (1) pasniedzējam pašam sistemātiski padziļināt zināšanas, un (2) to pašu kursa materiālu pacelt līdz universitātes ievadlīmenim, nemainot pamatkursa loģiku. Zemāk minētās tēmas nav obligāti jāiekļauj profesionālās vidusskolas vērtēšanā.

25.1. Algoritmu analīze un datu struktūras

- $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$ intuitīvs un formāls skaidrojums.
- Array/vector, linked list, stack, queue, set, map/hash map, tree un graph.
- Linear/binary search, insertion/selection/bubble sort; pēc tam merge/quick sort.
- Uzdevums: izmērīt izpildes laiku dažādiem n un salīdzināt ar teorētisku augšanas tendenci.

25.2. C++ atmiņas modelis

- Stack pret heap; automātiska un dinamiska dzīves ilguma ideja.
- Pointer, reference, const correctness, RAI.
- `std::vector`/`std::string` priekšrocības pret manuālu raw array/memory vadību iesācēju kursā.
- Paplašināti: smart pointers un ownership.

25.3. Lineārā algebra robotikai

- Vektors, matrica, matricas reizināšana.
- Rotācijas matricas 2D/3D.
- Homogēnā transformācija un frame composition.
- DH parametri un manipulatora ķēdes transformācijas.

25.4. Jacobian un diferenciālā kinemātika

- Locītavu ātrumi un gala efektora lineārais/leņķiskais ātrums.
- Jacobian dimensija un ģeometriskā nozīme.
- Singularitātes un vadības problēmas to tuvumā.

25.5. Vadības teorijas pāreja

- Dinamiska sistēma, setpoint, process variable, error.
- P, PI, PID un to kvalitatīvā ietekme.
- Overshoot, settling time, steady-state error.
- Diskrēts regulators un sample time.

25.6. Stāvokļa novērtēšana

- Trokšņa un nenoteiktības modelis.
- Kalman filtra prediction/update intuīcija.
- EKF nepieciešamība nelineārai robotikas sistēmai.
- Odometry + IMU + GNSS/LiDAR fusion konteksts.

25.7. SLAM un navigācija

- Pose graph / feature / scan matching jēdzienu karte.
- Loop closure nozīme.

- Global localization pret local tracking.
- Global planner, local planner/controller, collision checking un recovery.

25.8. Machine Vision padziļināti

- Kameras iekšējie parametri, distortion un calibration.
- Perspective projection un px→world koordinātes.
- Feature detection/matching, segmentation un object detection jēdzieni.
- Machine Vision mērījumu nenoteiktība un apgaismojuma ietekme.

25.9. Programmatūras inženierija

- Requirements → design → implementation → test → deployment → maintenance.
- Git branches, pull request/review un issue tracking.
- Unit/integration/system/acceptance testu līmeņi.
- CI/CD jēdziens un reproducējama vide.

25.10. Industriālā automatizācija un PLC pāreja

- PLC scan cycle.
- IEC 61131-3 valodas: LD, FBD, ST u.c.
- Interlock, permissive, alarm, sequence/state machine.
- OPC UA/MQTT/SCADA/MES jēdzienu savstarpējā vieta sistēmā.

25.11. Pasniedzēja pašmācības secība 12 moduļos

1. Python datu apstrāde + NumPy + matplotlib.
2. C++ modernās valodas pamati un std::vector/std::string.
3. Git/GitHub un reproducējams projekts.
4. SQL/SQLite + REST/MQTT datu integrācija.
5. Elektronikas un mikrokontrolleru I/O, ADC, PWM.
6. Mehatronikas feedback un PID simulācija.
7. Lineārā algebra un 2D/3D transformācijas.
8. Manipulatora forward/inverse kinematics + DH.
9. LiDAR, IMU, camera datu apstrāde.
10. Odometry, sensor fusion un Kalman/EKF pamati.
11. ROS 2 nodes/topics/TF + simulators.
12. SLAM, Nav2 un autonoma robota integrācijas projekts.

26. Kursa apstiprināšanas pielikums un ieviešanas kontrolsaraksts

Šī sadaļa ir veidota kā aizpildāms pamats kursa iesniegšanai profesionālajā izglītībā vai pielāgošanai universitātes studiju kursa aprakstam. Precīzie lauki jāpielāgo konkrētās izglītības iestādes iekšējai veidlapai; šeit nav pieņemts, ka visām iestādēm ir vienādas prasības.

Lauks	Ieteicamais saturs / aizpildāmais pamats
Kursa nosaukums	Programmēšanas un datorizētu sistēmu pamati
Mērķauditorija	Audzēkņi pēc 9. klases tehniskajā/profesionālajā izglītībā; pielāgojams universitātes ievadkursam.
Kontaktstundu modelis	1 × 90 min nedēļā; plānā 36 nodarbības.
Pamatbloki	Algoritmi; Python; C++; SQL; IoT/mākoņdati; mehatronika; robotika; autonomās sistēmas; projekts.
Mācību metodes	Īsa teorija, demonstrācija, vadīts praktiskais darbs, individuāls uzdevums, laboratorija, projekta darbs, aizstāvēšana.
Vērtēšana	Pārbaudes darbi + praktiskie/laboratorijas darbi + noslēguma projekts; procentu svarus pielāgo iestāde.
Nepieciešamā programmatūra	Python vide, C++ kompilators/IDE, SQLite; paplašināti OpenCV, ROS 2/simulators, MQTT brokeris.
Aparatūra	Pamatkursam nav obligāta. Paplašināti: mikrokontroleris, sensori, IMU, LiDAR/kamera vai simulators.
Patstāvīgais darbs	Uzdevumu pabeigšana, projekta dokumentēšana, testu sagatavošana, refleksija un profesionālās attīstības materiālu apguve.
Kursa rezultāts	Spēja no prasības izveidot algoritmu, realizēt programmu, apstrādāt datus, testēt, dokumentēt un izskaidrot datorizētas/mehatroniskas sistēmas pamatdarbību.

26.1. Sasniedzamo rezultātu kontrolsaraksts

- [] Skolēns spēj formulēt algoritmu un izvēlēties secību, zarošanos un ciklu.
- [] Skolēns spēj izveidot blokshēmu/pseudokodu un testēt robežgadījumus.
- [] Skolēns spēj realizēt vienkāršu tehnisku uzdevumu Python.
- [] Skolēns spēj realizēt algoritmu C++ un izvēlēties datu tipus/funkcijas.
- [] Skolēns saprot SQL tabulu, INSERT/SELECT un tehnisko mērījumu datu modeli.

- [] Skolēns spēj izskaidrot IoT datu ceļu, MQTT/HTTP lomu un offline scenāriju.
- [] Skolēns spēj identificēt sensoru, aktuatoru, kontrolieri un feedback mehatroniskā sistēmā.
- [] Skolēns spēj izskaidrot robota SENSE→PERCEIVE→LOCALIZE/MAP→PLAN→CONTROL→ACT ciklu.
- [] Skolēns demonstrē kinemātikas, LiDAR, IMU vai Machine Vision pamatuzdevumu ar Python/simulāciju.
- [] Skolēns spēj izstrādāt un aizstāvēt projektu ar functional tree, kodu, testiem un dokumentāciju.

26.2. Ieviešanas kontrolsaraksts pasniedzējam

- [] Apstiprināts kontaktstundu skaits un semestru kalendārs.
- [] Noteikta Python un C++ darba vide iestādes datoros.
- [] Pārbaudītas instalācijas tiesības SQLite/OpenCV/ROS 2 vai sagatavota alternatīva simulācija.
- [] Sagatavotas skolēnu mapes/repozitoriji un failu nosaukumu standarts.
- [] Noteikts AI rīku izmantošanas noteikums un prasība izskaidrot iesniegto kodu.
- [] Pirms aparātūras laboratorijām apstiprinātas drošības instrukcijas.
- [] Izvēlēta vērtēšanas skala un pārbaudes darbu procentu svars.
- [] Noslēguma projekta sākumā apstiprinātas prasības, functional tree un testu kritēriji.
- [] Veidota versiju kontrole kursa materiālam; labojumi tiek veikti vienā pamatdokumentā.

26.3. Literatūras un resursu sadaļas struktūra

Konkrētus izdevumus un saites ieteicams ievietot pēc iestādes prasībām un regulāri pārskatīt. Kursa bibliogrāfiju lietderīgi sadalīt:

- Algoritmu un datorzinātnes pamati.
- Python oficiālā dokumentācija un mācību materiāli.
- C++ valodas standarta/atsauces materiāli.
- SQL/SQLite un datubāzu pamati.
- MQTT/HTTP/REST un IoT arhitektūra.
- Mehatronika un vadības sistēmas.
- Robotu kinemātika, perception, SLAM un mobile robotics.
- ROS 2 oficiālā dokumentācija un simulatoru dokumentācija.
- Kiberdrošības un drošas industriālās sistēmas principi.

27. Laboratoriju skolēna darba lapas un pasniedzēja risinājumi

Šī nodaļa pārvērš 24. nodaļā aprakstītās laboratorijas par praktiski izsniedzamu materiālu. Katra laboratorija sastāv no skolēna/studenta darba lapas un pasniedzēja versijas. Darba lapās apzināti izmantota inženiertehniska testēšanas pieeja: prasība → realizācija → testa gadījums → expected → actual → PASS/FAIL → secinājums.

Vienotais laboratorijas darba princips

1) pirms koda nosaka sistēmas funkciju; 2) definē ievades un izejas; 3) izveido algoritmu vai functional tree; 4) realizē; 5) testus definē pirms rezultāta; 6) dokumentē kļūdas un labojumus; 7) aizstāv lēmumus, ne tikai parāda gala programmu.

27.1. L1 — Functional tree un algoritma dekompozīcija

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: iemācīties sadalīt tehnisku sistēmu funkcijās, pirms tiek rakstīts kods.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- definēt sistēmas galveno funkciju darbības vārda formā;
- sadalīt galveno funkciju 4–7 apakšfunkcijās;
- katrai apakšfunkcijai noteikt ievades, izejas un vismaz vienu pārbaudes kritēriju;
- pārvērst vienu koka zaru pseidokodā.

Ko pasniedzējam stāstīt pirms darba

Funkcija nav komponente: Functional tree apraksta, ko sistēmai jādara, nevis uzreiz nosauc detaļas. “Noteikt šķērsli” ir funkcija; “LiDAR” ir viens iespējamais risinājums.

Dekompozīcija: Sarežģītu sistēmu sadala līdz līmenim, kur funkciju var realizēt, testēt un piešķirt modulim.

Traceability: Vēlāk katru funkciju sasaista ar prasību, kodu un testu. Tas novērš situāciju, kur projektā ir “skaists kods”, bet nav pierādāms, ko tas izpilda.

Skolēna/studenta uzdevums

- Izvēlies sistēmu: automātiskas durvis, temperatūras kontrole, mobilais robots vai pasniedzēja dots piemērs.
- Uzraksti vienu galveno funkciju formā “To / Veikt ...”.
- Izveido 4–7 pirmā līmeņa apakšfunkcijas.
- Vismaz divas apakšfunkcijas sadali vēl vienā līmenī.
- Katrai gala funkcijai pieraksti ievadi, izeju un pārbaudāmu kritēriju.
- Izvēlies vienu zaru un uzraksti pseidokodu.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	Galvenā funkcija ir viena un apraksta sistēmas mērķi	Ir viena skaidra galvenā funkcija bez komponentes nosaukuma.			
2	Vismaz 4 pirmā līmeņa funkcijas	Koks satur vismaz 4 loģiski atšķirīgas funkcijas.			
3	Vismaz 2 zari sadalīti dziļāk	Divām funkcijām ir pārbaudāmas apakšfunkcijas.			
4	Funkcija sasaistīta ar testu	Vismaz vienai gala funkcijai definēts izmērāms PASS nosacījums.			

Analīzes jautājumi

1. Kāpēc “uzstādīt ultraskaņas sensoru” nav laba funkcijas definīcija?
2. Kurā līmenī dekompozīcija kļūst pārāk smalka?
3. Kā functional tree palīdz sadalīt darbu komandā?
4. Kā no funkcijas izveidot testu?

Tipiskās kļūdas un diagnostika

- komponentes tiek jauktas ar funkcijām;
- vienā mezglā apvienotas vairākas darbības (“mērīt un kontrolēt un saglabāt”);
- apakšfunkcijas neatbalsta vecākfunkciju;
- nav iespējams noteikt, vai funkcija izpildīta.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Galvenā funkcija	4	Skaidra, viena, mērķorientēta.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
FUNCTION ControlAutomaticDoor
  READ presence_sensor
  IF presence_sensor = TRUE THEN
    COMMAND open_door
    VERIFY open_limit_switch
  END IF
  WHILE doorway_occupied = TRUE
    KEEP door_open
  END WHILE
  COMMAND close_door
  VERIFY closed_limit_switch
END FUNCTION
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Labs koks nav “pareizā atbilde”, bet argumentēts sistēmas funkcionālais modelis.
- Pirms detaļu izvēles studentam jāspēj aprakstīt funkcijas tehnoloģiski neitrāli.
- Augstskolas līmenī šo darbu var sasaistīt ar requirements engineering un V-model traceability.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtoto testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.2. L2 — Python tehnisko datu klasifikators

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: realizēt prasību kā testējamu Python funkciju ar robežvērtību un nederīgu datu apstrādi.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- izmantot funkciju, if/elif/else un datu validāciju;
- atšķirt INPUT_ERROR no SENSOR_ERROR;
- izveidot robežvērtību testus;
- pamatot nosacījumu secību.

Ko pasniedzējam stāstīt pirms darba

Prasība pirms koda: Vispirms jāvienojas par diapazoniem. Ja tie nav definēti, divi pareizi uzrakstīti kodi var dot atšķirīgus rezultātus.

Robežvērtības: Visvairāk kļūdu rodas pie 59.9/60, 79.9/80 un pie sensora diapazona robežām.

Validācijas slāņi: Teksts, ko nevar pārvērst skaitlī, ir ievades formāta kļūda. Skaitlis 500 var būt sintaktiski derīgs skaitlis, bet nederīgs sensora mērījums.

Skolēna/studenta uzdevums

1. Pieraksti klasifikācijas prasības tabulā pirms programmēšanas.
2. Izveido `classify_temperature(t)`.
3. Pievieno lietotāja ievades apstrādi ar `try/except`.
4. Izpildi vismaz 8 testus, tostarp robežvērtības.
5. Pievieno vienu savu testu, kas pārbauda tipisku kļūdu.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	t=25	NORMAL			
2	t=60	WARNING			
3	t=79.9	WARNING			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
4	t=80	ALARM			
5	t=-50	NORMAL			
6	t=150	ALARM			
7	t=151	SENSOR_ERROR			
8	ievade="abc"	INPUT_ERROR			

Analīzes jautājumi

1. Kāpēc sensoru diapazona validāciju veic pirms WARNING/ALARM loģikas?
2. Kāda būtu atšķirība starp >80 un >=80?
3. Kā šo funkciju izmantotu reālā datu loggerī?

Tipiskās kļūdas un diagnostika

- float(input()) bez try/except;
- robežvērtības saliktas nepareizā secībā;
- sensors ārpus diapazona tiek kļūdaini klasificēts kā ALARM;
- testi pārbauda tikai "ērtas" vidus vērtības.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Prasību interpretācija	5	Pareizi diapazoni un robežas.
Kods	8	Funkcija, validācija, skaidri nosaukumi.
Testi	8	Robežvērtības un kļūdainas ievades.
Analīze	4	Students pamato secību un kļūdu tipus.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
def classify_temperature(t):
    if t < -50 or t > 150:
        return "SENSOR_ERROR"
    if t >= 80:
        return "ALARM"
    if t >= 60:
        return "WARNING"
    return "NORMAL"

raw = input("Ievadi temperatūru °C: ")
try:
    value = float(raw)
    print(classify_temperature(value))
except ValueError:
    print("INPUT_ERROR")
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Lieciet studentam vispirms uzrakstīt testa tabulu, tikai tad kodu.
- Demonstrējiet, ka prasības formulējums “virs 80” un “80 vai vairāk” nav viens un tas pats.
- Universitātes līmenī var ieviest pytest parametrizētos testus.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtotu testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.3. L3 — Python datu loggera simulācija

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: izveidot mērījumu datu plūsmu ar laika zīmogu, kvalitātes statusu, CSV saglabāšanu un kopsavilkumu.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- izveidot strukturētu mērījuma ierakstu;
- atšķirt mērījuma vērtību no kvalitātes statusa;
- saglabāt datus CSV;
- aprēķināt statistiku tikai no derīgiem datiem.

Ko pasniedzējam stāstīt pirms darba

Mērījums nav tikai skaitlis: Profesionālā sistēmā mērījumu raksturo vērtība, mērvienība, laiks, avots un kvalitāte.

Laika zīmogs: Bez timestamp nevar uzticami salīdzināt notikumus no dažādām sistēmām.

Datu integritāte: Nederīgus datus nevajag slepeni dzēst; tiem var saglabāt BAD kvalitāti, lai vēsture būtu izsekojama.

Skolēna/studenta uzdevums

- Izveido vismaz 10 mērījumu sarakstu; iekļauj vienu None un vienu ārpus diapazona.
- Katram ierakstam piešķir timestamp un GOOD/BAD statusu.
- Saglabā CSV ar galveni.
- Aprēķini count, avg, min, max no GOOD datiem.
- Atver CSV un pārbaudi, vai kolonnas ir saprotamas citam lietotājam.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	10 ieraksti, visi derīgi	CSV ir 10 datu rindas; valid_count=10.			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
2	Viens None	Ieraksts saglabāts ar BAD; tas nav iekļauts vidējā.			
3	Vērtība 999	Ieraksts BAD; nav iekļauts statistikā.			
4	CSV atvēršana	Ir timestamp, value_C un quality kolonnas.			

Analīzes jautājumi

1. Kāpēc kvalitātes statuss ir svarīgāks par vienkāršu nederīga ieraksta dzēšanu?
2. Kādas problēmas rada dažādas laika zonas?
3. Ko mainītu, ja dati pienāktu reizi sekundē visu gadu?

Tipiskās kļūdas un diagnostika

- vidējais tiek dalīts ar visu ierakstu skaitu, nevis derīgo skaitu;
- CSV nav mērvienības vai saprotamas galvenes;
- timestamp tiek ģenerēts vienreiz un atkārtots visām rindām;
- BAD dati pazūd bez pēdām.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Datu struktūra	6	Timestamp, vērtība, kvalitāte.
CSV	5	Pareiza galvene un ieraksti.
Statistika	7	Aprēķins tikai no derīgiem datiem.
Testēšana/analīze	7	Nederīgu datu scenāriji un secinājumi.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
import csv
from datetime import datetime

measurements = [21.5, 22.1, 23.0, 80.0, None, 24.2]

def quality(v):
    if v is None:
        return "BAD"
    if v < -50 or v > 150:
        return "BAD"
    return "GOOD"

with open("measurements.csv", "w", newline="", encoding="utf-8") as f:
    w = csv.writer(f)
    w.writerow(["timestamp", "value_C", "quality"])
    for v in measurements:
        w.writerow([datetime.now().isoformat(timespec="seconds"), v, quality(v)])

valid = [v for v in measurements if v is not None and -50 <= v <= 150]
print("count=", len(valid), "avg=", sum(valid)/len(valid), "min=", min(valid), "max=", max(valid))
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Paskaidrojiet saikni ar industriāliem historian/logger principiem.
- Ja laiks atļauj, studenti lai pievieno sensor_id un status kolonnas.
- Augstskolas versijā var pāriet uz pandas un time-series datu analīzi.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtu testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.4. L4 — C++ vadības stāvokļu programma

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: izveidot determinētu state machine, kas neļauj neatļautas pārejas.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- definēt stāvokļus ar enum class;
- izveidot atļauto pāreju loģiku;
- piešķirt ALARM augstāku prioritāti par normālu darbību;
- testēt neatļautas komandas.

Ko pasniedzējam stāstīt pirms darba

State machine: Sistēmas uzvedību apraksta ne tikai komanda, bet arī pašreizējais stāvoklis.

Drošības prioritāte: ALARM/E-stop tipa signālam jāspēj pārtraukt normālo secību.

Determinisms: Pie vienādas ievades un stāvokļa sistēmai jānonāk paredzamā nākamajā stāvoklī.

Skolēna/studenta uzdevums

- Uzzīmē pāreju diagrammu IDLE–RUNNING–ALARM.
- Definē enum class State.
- Izveido komandu ciklu.
- START atļauj tikai no IDLE bez aktīva alarma.
- ALARM jāspēj aktivizēt no jebkura normāla stāvokļa.
- RESET atļauj tikai no ALARM uz IDLE.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	IDLE + START	RUNNING			
2	RUNNING + START	Paliek RUNNING / komanda noraidīta			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
3	RUNNING + ALARM	ALARM			
4	ALARM + START	Paliek ALARM			
5	ALARM + RESET	IDLE			

Analīzes jautājumi

1. Kāpēc ar vienu bool running nepietiek sarežģītākai sistēmai?
2. Kas notiktu, ja būtu stāvoklis MAINTENANCE?
3. Kā atšķirt command no state?

Tipiskās kļūdas un diagnostika

- komandas tieši maina izejas bez stāvokļa pārbaudes;
- RESET ļauj uzreiz sākt kustību;
- ALARM nav prioritārs;
- nav dokumentētu neatļauto pāreju.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Stāvokļu modelis	6	Skaidri stāvokļi un pārejas.
C++ realizācija	8	enum class, cikls, nosacījumi.
Drošības loģika	6	ALARM prioritāte, RESET.
Testi	5	Atļautās un neatļautās pārejas.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
#include <iostream>
#include <string>
using namespace std;

enum class State { IDLE, RUNNING, ALARM };

int main(){
    State state = State::IDLE;
    bool alarm_active = false;
    string cmd;
    while(true){
        cout << "Command START/STOP/ALARM/RESET/EXIT: ";
        cin >> cmd;
        if(cmd == "EXIT") break;
        if(cmd == "ALARM"){ alarm_active = true; state = State::ALARM; }
        else if(cmd == "RESET" && alarm_active){ alarm_active = false; state = State::IDLE; }
        else if(cmd == "START" && state == State::IDLE && !alarm_active){ state = State::RUNNING; }
        else if(cmd == "STOP" && state == State::RUNNING){ state = State::IDLE; }
        cout << "State=" << (state==State::IDLE?"IDLE":state==State::RUNNING?"RUNNING":"ALARM") << "\n";
    }
}
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Pirms koda lieciet studentiem aizpildīt pāreju tabulu.
- Parādiet, ka state machine ir tilts uz PLC sekvencēm, robotu behavior/state loģiku un embedded vadību.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtoto testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.5. L5 — C++ Sensor klase un datu kvalitāte

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: modelēt tehnisku komponenti kā klasi ar skaidru atbildību un validētu stāvokli.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- izveidot class ar private datiem un public metodēm;
- izmantot konstruktoru;
- atdalīt sensora identitāti, vērtību un validitāti;
- pamatot encapsulation nozīmi.

Ko pasniedzējam stāstīt pirms darba

Objekts kā tehnisks modelis: Klase var attēlot sensoru, motoru, ventilatoru vai citu sistēmas komponenti ar datiem un uzvedību.

Encapsulation: Iekšējo stāvokli maina caur kontrolētu interfeisu, nevis patvaļīgi no jebkuras programmas vietas.

Validitāte: Objektam jā saglabā ne tikai vērtība, bet arī informācija, vai tai var uzticēties.

Skolēna/studenta uzdevums

1. Izveido Sensor klasi ar id, value, valid.
2. Pievieno konstruktoru sensor_id inicializācijai.
3. Izveido update(v), kas nosaka validitāti.
4. Izveido getter metodes.
5. Testē ar normālu, robežvērtību un nederīgu mērījumu.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	update(25)	valid=true, value=25			
2	update(-50)	valid=true			
3	update(150)	valid=true			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
4	update(151)	valid=false			

Analīzes jautājumi

1. Kāpēc value nevajadzētu būt public?
2. Vai nederīga vērtība jāsaglabā? Kādi ir plusi/mīnusi?
3. Kā klasi paplašinātu ar timestamp?

Tipiskās kļūdas un diagnostika

- visi lauki public un nav kontrolētas validācijas;
- isValid() pārreķina ar citu diapazonu nekā update();
- konstruktors neglabā id;
- students OOP izmanto tikai sintaksei, bet nav skaidras klases atbildības.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Klases struktūra	7	Private dati, public interfeiss.
Validācija	6	Korektas robežas un status.
Testi	6	Normālas un nederīgas vērtības.
Pamatojums	6	Students izskaidro encapsulation un atbildību.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
#include <iostream>
#include <string>
using namespace std;

class Sensor {
private:
    string id;
    double value{0.0};
    bool valid{false};
public:
    Sensor(string sensorId): id(sensorId) {}
    void update(double v){
        value = v;
        valid = (v >= -50.0 && v <= 150.0);
    }
    bool isValid() const { return valid; }
    double getValue() const { return value; }
    string getId() const { return id; }
};

int main(){
    Sensor t1("T-101");
    t1.update(24.6);
    cout << t1.getId() << " " << t1.getValue() << " valid=" << t1.isValid() << "\n";
}
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Šī ir laba vieta izskaidrot “single responsibility” intuitīvi.
- Paplašinājumā var ieviest inheritance tikai pēc tam, kad viena klase ir saprotama.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtoto testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.6. L6 — Python + SQLite tehnisko mērījumu datubāze

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: saglabāt strukturētus mērījumus SQL datubāzē un atlasīt tos ar vaicājumiem.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- izveidot tabulu ar primāro atslēgu;
- izmantot parametrizētu INSERT;
- izpildīt SELECT ar WHERE/ORDER BY/LIMIT;
- saglabāt sensor_id, vērtību, kvalitāti un laiku.

Ko pasniedzējam stāstīt pirms darba

Datubāze pret CSV: CSV ir vienkāršs fails; SQL dod strukturētu vaicājumu valodu, indeksus, integritātes iespējas un labāku vairāku tabulu modeli.

Primārā atslēga: Katram ierakstam vajag unikālu identifikatoru, lai uz to varētu atsaukties.

Parametrizēti vaicājumi: Datus nedrīkst līmēt SQL tekstā; parametri samazina kļūdas un SQL injection risku.

Skolēna/studenta uzdevums

1. Izveido measurements.db.
2. Izveido measurements tabulu.
3. Ieraksti vismaz 8 mērījumus diviem sensor_id.
4. Atlasiet pēdējos 5 ierakstus.
5. Atlasiet tikai GOOD mērījumus ar value ≥ 60 .
6. Izmaini vienu vērtību un atkārto vaicājumus.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	DB izveide	Fails eksistē; tabula measurement s ir pieejama.			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
2	8 INSERT	SELECT COUNT(*) atgriež vismaz 8.			
3	WHERE quality="GOOD"	Rezultātā nav BAD ierakstu.			
4	ORDER BY id DESC LIMIT 5	Atgriež 5 jaunākos ierakstus dīlstošā secībā.			

Analīzes jautājumi

1. Kāpēc tehniskajos datos vajadzīgs sensor_id?
2. Kad būtu nepieciešama atsevišķa sensors tabula?
3. Kā novērst datubāzes bojāšanos, ja programma pārtrūkst rakstīšanas brīdī?

Tipiskās kļūdas un diagnostika

- nav commit();
- SQL tiek būvēts ar f-string no lietotāja ievades;
- timestamp nav saglabāts;
- vienā kolonnā sajauc vērtību un mērvienību kā tekstu.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Datu modelis	6	Jēgpilnas kolonnas un primārā atslēga.
SQL/Python	8	CREATE, parametrizēts INSERT, SELECT.
Vaicājumi	6	WHERE/ORDER BY/LIMIT.
Testi/analīze	5	Students pārbauda un izskaidro rezultātus.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
import sqlite3

con = sqlite3.connect("measurements.db")
cur = con.cursor()
cur.execute("""CREATE TABLE IF NOT EXISTS measurements(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    sensor_id TEXT NOT NULL,
    value REAL,
    quality TEXT NOT NULL,
    ts TEXT NOT NULL
)""")
cur.execute("INSERT INTO measurements(sensor_id,value,quality,ts) VALUES(?,?,?,datetime('now'))",
    ("T-101", 24.6, "GOOD"))
con.commit()
for row in cur.execute("SELECT sensor_id,value,quality,ts FROM measurements ORDER BY id DESC LIMIT 10"):
    print(row)
con.close()
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Parādiet transaction ideju: izmaiņas kļūst pastāvīgas pēc commit.
- Universitātes līmenī var pievienot normalizāciju, indeksus un transaction isolation.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtoto testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.7. L7 — MQTT/IoT datu plūsmas simulācija

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: izprast publisher–broker–subscriber arhitektūru un izveidot jēgpilnu topic/payload datu līgumu.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- atšķirt topic no payload;
- izveidot JSON mērījuma ziņojumu;
- aprakstīt publisher, broker un subscriber lomas;
- izskaidrot, kā sistēma uzvedas savienojuma zuduma laikā.

Ko pasniedzējam stāstīt pirms darba

MQTT modelis: Publisher nesūta ziņu tieši konkrētam klientam; tas publicē topic, broker izplata abonentiem.

Datu līgums: Payload struktūra ir interfeiss starp sistēmām. Mainot lauku nosaukumus, var salauzt subscriber.

Lokālā autonomija: Drošības vadībai nevajadzētu būt atkarīgai no mākoņa vai brokera pieejamības; IoT biežāk izmanto monitorēšanai, koordinācijai un datu analīzei.

Skolēna/studenta uzdevums

1. Uzzīmē publisher → broker → subscriber datu plūsmu.
2. Izveido topic trīs sensoriem.
3. Izveido JSON payload ar id, value, unit, quality, timestamp.
4. Python kodā serializē un deserializē JSON.
5. Apraksti scenāriju: internets/broker nav pieejams 5 minūtes.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	GOOD payload	Subscriber spēj nolasīt sensor_id, value, quality, timestamp.			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
2	BAD quality	Ziņa tiek saglabāta ar BAD; nav izmantota lokālai vadībai.			
3	Nezināms JSON lauks	Subscriber ignorē vai dokumentē; programma neavarē.			
4	Broker offline	Lokālā vadība turpinās; telemetrija tiek buferēta vai atzīmēta kā nenosūtīta.			

Analīzes jautājumi

1. Kāpēc topic nedrīkst saturēt pašu mērījuma vērtību?
2. Kas ir svarīgāks: "latest value" vai pilna vēsture?
3. Kur sistēmā vajadzētu glabāt neizsūtītas ziņas?

Tipiskās kļūdas un diagnostika

- payload nav timestamp vai quality;
- topic struktūra nav konsekventa;
- cloud tiek izmantots kā vienīgā drošības vadības vieta;
- nav definēta offline uzvedība.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Arhitektūra	6	Pareizas MQTT lomas.
JSON/topic	7	Jēgpilna un konsekventa datu struktūra.
Python demonstrācija	5	Serialize/deserialize un validācija.

Kritērijs	Punkti	Ko vērtē
Offline analīze	7	Lokāla autonomija, buffering, status.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
import json
from datetime import datetime

def make_message(sensor_id, value, quality):
    return {
        "sensor_id": sensor_id,
        "value": value,
        "unit": "degC",
        "quality": quality,
        "timestamp": datetime.utcnow().isoformat(timespec="seconds") + "Z"
    }

topic = "factory/line1/temperature/T-101"
payload = json.dumps(make_message("T-101", 24.6, "GOOD"))
print("PUBLISH", topic, payload)

# Subscriber-side simulation
received = json.loads(payload)
if received["quality"] == "GOOD":
    print("STORE", received["sensor_id"], received["value"])
else:
    print("STORE WITH BAD QUALITY / DO NOT USE FOR CONTROL")
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Ja ir pieejams brokeris, kodu var paplašināt ar paho-mqtt; pamata mācību rezultāts tomēr ir arhitektūra, nevis konkrēta bibliotēka.
- Uzsvērt topic namespace dizainu un backward compatibility.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtoto testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.8. L8 — Mehatronikas feedback un histerēzes simulācija

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: modelēt vienkāršu closed-loop vadību un novērst izejas biežu pārslēgšanos ar histerēzi.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- atšķirt setpoint/threshold, measured value un actuator state;
- izskaidrot feedback cilpu;
- realizēt divus sliekšņus ON/OFF;
- salīdzināt ar vienu sliekšni.

Ko pasniedzējam stāstīt pirms darba

Closed-loop: Sensors mēra procesa rezultātu, kontrolleris pieņem lēmumu un aktrsators ietekmē procesu.

Histerēze: Ar vienu sliekšni un trokšņainu mērījumu izeja var “čakarēties” ON/OFF. Divi sliekšņi ievieš stabilu zonu.

PID konteksts: Histerēze nav PID, bet tā ir laba pāreja uz vadības teoriju: mērījums, kļūda, darbība, dinamika.

Skolēna/studenta uzdevums

1. Izveido temperatūru secību, kas šķērso 30 °C un 25 °C.
2. Realizē ventilatoru ar vienu sliekšni 30 °C un pieraksti pārslēgšanos.
3. Realizē histerēzi: ON pie ≥ 30 , OFF pie ≤ 25 .
4. Salīdzini rezultātus.
5. Papildu: pievieno sensora troksni ± 0.5 °C.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	T=29, fan OFF	OFF			
2	T=31, fan OFF	Pāriet ON			
3	T=28, fan ON	Paliek ON			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
4	T=25, fan ON	Pāriet OFF			

Analīzes jautājumi

1. Kāpēc pie 28 °C ventilators var būt gan ON, gan OFF atkarībā no iepriekšējā stāvokļa?
2. Kā histerēze maina sistēmas “atmiņu”?
3. Kad labāk izmantot PID, nevis ON/OFF vadību?

Tipiskās kļūdas un diagnostika

- OFF nosacījums uzrakstīts kā $t < 30$, tāpēc histerēzes nav;
- netiek saglabāts iepriekšējais fan_on stāvoklis;
- sliekšņi sajaukti vietām;
- students ignorē procesa inerci.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Feedback izpratne	5	Pareizi identificē sensor/controller/actuator/process.
Histerēzes kods	8	Divi sliekšņi un stāvokļa atmiņa.
Testi	6	Pārslēgšanās abos virzienos.
Analīze	6	Salīdzina vienu sliekšni ar histerēzi.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
measurements = [25, 27, 29, 31, 29, 27, 25, 23, 24, 28]
fan_on = False
ON_LIMIT = 30.0
OFF_LIMIT = 25.0

for t in measurements:
    if not fan_on and t >= ON_LIMIT:
        fan_on = True
    elif fan_on and t <= OFF_LIMIT:
        fan_on = False
    print(f"T={t:4.1f} C fan={'ON' if fan_on else 'OFF'}")
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Šis ir labs brīdis ieviest stateful control: lēmums atkarīgs no pašreizējā mērījuma un iepriekšējā stāvokļa.
- Augstskolas līmenī var modelēt first-order thermal process un PID.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtoti testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.9. L9 — 2DOF robotu kinemātika Python

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: aprēķināt un vizualizēt plakana 2DOF manipulatora forward kinematics.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- izmantot leņķus radiānos;
- aprēķināt locītavas un gala efektora koordinātes;
- izskaidrot lokālo un globālo leņķu summēšanu;
- vizualizēt robotu koordinātu plaknē.

Ko pasniedzējam stāstīt pirms darba

Forward kinematics: Doti locītavu stāvokļi q , jāatrod gala efektora poza.

Koordinātu sistēmas: Otrā posma orientācija globālajā sistēmā ir q_1+q_2 , ja q_2 definēts relatīvi pret pirmo posmu.

Radians: Python math trigonometriskās funkcijas izmanto radiānus; grādi jāpārveido.

Skolēna/studenta uzdevums

1. Uzzīmē 2DOF robotu un pieraksti L_1 , L_2 , q_1 , q_2 .
2. Aprēķini pirmās locītavas beigu punktu.
3. Aprēķini gala efektoru.
4. Pārbaudi ar $q_1=0^\circ$, $q_2=0^\circ$ — robotam jābūt pilnībā izstieptam pa $+x$.
5. Vizualizē vismaz 4 konfigurācijas.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	$q_1=0^\circ$, $q_2=0^\circ$	$x=L_1+L_2$, $y=0$			
2	$q_1=90^\circ$, $q_2=0^\circ$	$x\approx 0$, $y=L_1+L_2$			
3	$q_1=0^\circ$, $q_2=90^\circ$	$x=L_1$, $y=L_2$			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
4	L1=0.5,L2=0.3, q1=30°,q2=45°	Rezultāts sakrīt ar neatkarīgu manuālu aprēķinu pie noapaļošanas.			

Analīzes jautājumi

1. Kāpēc q2 formulā parādās q1+q2?
2. Kas ir workspace?
3. Vai vienam x,y punktam inverse kinematics var būt vairāki risinājumi?

Tipiskās kļūdas un diagnostika

- grādi ievadīti tieši cos/sin;
- otrā posma formulā lietots tikai q2;
- x/y asis grafikā nav vienādā mērogā;
- students nejauca link length ar joint angle.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Matemātika	8	Pareizas formulas un radiāni.
Python	6	Aprēķins un strukturēti mainīgie.
Vizualizācija	5	Pareizs ģeometrisks attēlojums.
Testi/analīze	6	Speciālie stāvokļi un workspace izpratne.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
import math
import matplotlib.pyplot as plt

L1, L2 = 0.50, 0.30
q1 = math.radians(30)
q2 = math.radians(45)

x1 = L1 * math.cos(q1)
y1 = L1 * math.sin(q1)
x2 = x1 + L2 * math.cos(q1 + q2)
y2 = y1 + L2 * math.sin(q1 + q2)

print(f"End effector: x={x2:.3f}, y={y2:.3f}")
plt.plot([0,x1,x2],[0,y1,y2], marker="o")
plt.axis("equal"); plt.grid(True); plt.xlabel("x [m]"); plt.ylabel("y [m]")
plt.show()
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Pirms koda izrēķiniet vienu konfigurāciju uz tāfeles.
- Paplašinājumā ieviesiet inverse kinematics, transformāciju matricas un DH.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtu testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.10. L10 — LiDAR scan processing Python

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: apstrādāt 2D LiDAR skenējumu no polarām koordinātēm līdz šķēršļa lēmumam.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- pārveidot r, θ uz x, y ;
- filtrēt nederīgus attālumus;
- noteikt tuvāko objektu un priekšējo drošības sektoru;
- atšķirt “datu apstrādi” no “vadības lēmuma”.

Ko pasniedzējam stāstīt pirms darba

LiDAR skenējums: Katrs punkts satur leņķi un attālumu sensora koordinātu sistēmā.

Polar→Cartesian: $x=r \cos\theta$, $y=r \sin\theta$ ļauj punktus zīmēt un kombinēt ar karti.

Drošības sektors: Autonomam robotam bieži nav svarīgs absolūti tuvākais punkts, bet tuvākais punkts kustības koridorā.

Skolēna/studenta uzdevums

1. Izveido vismaz 20 leņķu/attālumu pārus.
2. Atmet $r < 0.1$ un $r > 8.0$.
3. Pārveido derīgos punktus x, y .
4. Atrodi globāli tuvāko objektu.
5. Atrodi tuvāko objektu priekšējā sektorā $\pm 20^\circ$.
6. Izvada STOP, ja priekšējais attālums < 0.7 m.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	Punkts $r=0.05$	Tiek atmests kā nederīgs.			
2	Priekšā 0.55 m	STOP			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
3	Sānos 0.4 m pie 60°	Globāli nearest=0.4, bet priekšējais sektors var palikt GO.			
4	Nav punktu priekšējā sektorā	front_min=inf vai ekvivalents; GO ar skaidru politiku.			

Analīzes jautājumi

1. Kāpēc “nearest point anywhere” nav pietiekams kustības lēmumam?
2. Kā sensora leņķiska izšķirtspēja ietekmē šauru objektu noteikšanu?
3. Ko nozīmē sensor frame un base_link frame?

Tipiskās kļūdas un diagnostika

- grādi nav pārveidoti radiānos;
- 0 vai None attālums tiek izmantots kā reāls šķērslis bez kvalitātes politikas;
- STOP loģika sajaukta ar punktu filtrēšanas funkciju;
- netiek ņemts vērā robota platums.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Datu filtrēšana	5	Nederīgie punkti tiek korekti apstrādāti.
Koordinātu pārveide	7	Polar→Cartesian.
Šķēršļa loģika	7	Priekšējais sektors un sliekšnis.
Analīze	6	Frame, izšķirtspēja, robota izmērs.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
import math

angles_deg = [-60,-30,0,30,60]
distances_m = [2.0, 1.4, 0.55, 1.2, 2.2]
MIN_R, MAX_R = 0.10, 8.0
points=[]
for deg,r in zip(angles_deg,distances_m):
    if not (MIN_R <= r <= MAX_R):
        continue
    a=math.radians(deg)
    x=r*math.cos(a); y=r*math.sin(a)
    points.append((x,y,r,deg))

nearest=min(points,key=lambda p:p[2])
front=[p for p in points if abs(p[3]) <= 20]
front_min=min((p[2] for p in front), default=float("inf"))
print("nearest=",nearest)
print("front_min=",front_min, "STOP" if front_min < 0.7 else "GO")
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Skolēniem vizuāli uzzīmējiet polaros starus.
- Paplašinājumā pievienojiet matplotlib scatter, clustering un transformāciju uz robota frame.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtu testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.11. L11 — Machine Vision ar OpenCV

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: izveidot determinētu attēla apstrādes ķēdi objekta atrašanai un rezultāta dokumentēšanai.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- izskaidrot image acquisition → preprocessing → segmentation → feature extraction → decision;
- izmantojot grayscale, blur, threshold un contours;
- aprēķināt objekta centru/izmēru;
- saglabāt vizuālu pierādījumu.

Ko pasniedzējam stāstīt pirms darba

Machine Vision vs “AI”: Daudzi rūpnieciski uzdevumi neprasa neironu tīklu; stabils apgaismojums un determinēti attēla algoritmi bieži ir vienkāršāki un audītējami.

Threshold: Sliekšnis sadala pikseļus klasēs; rezultāts ļoti atkarīgs no apgaismojuma.

Kalibrācija: Pikseļu koordinātes nav automātiski milimetri; izmēriem vajadzīgs mērogs/kameras kalibrācija.

Skolēna/studenta uzdevums

- Izvēlies vienkāršu augsta kontrasta detaļas attēlu.
- Ielādē un pārbaudi img is not None.
- Pārveido grayscale un blur.
- Izveido binary mask.
- Atrodi kontūras un filtrē pēc laukuma.
- Aprēķini bounding box un centru.
- Saglabā result.png ar anotācijām.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	Fails neeksistē	Kontrolēta FileNotFound Error / skaidra kļūda.			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
2	Viens liels objekts	Atrod 1 atbilstošu kontūru un centru.			
3	Mazs troksnis <500 px	Tiek ignorēts.			
4	Apgaismojums mainās	Students konstatē, ka fiksēts threshold var kļūt nestabils.			

Analīzes jautājumi

1. Kāpēc blur pirms threshold var palīdzēt?
2. Kā no pikseļu centra iegūt robota koordinātes?
3. Kādi ir Machine Vision validācijas riski ražošanā?

Tipiskās kļūdas un diagnostika

- nav pārbaudīts img=None;
- threshold virziens (BINARY/BINARY_INV) neatbilst attēlam;
- visi mazie trokšņi tiek uzskatīti par objektiem;
- pixel coordinates interpretētas kā mm bez kalibrācijas.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Apstrādes ķēde	6	Loģiska secība.
OpenCV realizācija	7	Attēls, maska, contours.
Rezultāta mērījums	5	Centrs/izmērs un anotācija.
Validācija	7	Fails, troksnis, apgaismojums, kalibrācija.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
import cv2

img = cv2.imread("part.jpg")
if img is None:
    raise FileNotFoundError("part.jpg not found")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (5,5), 0)
_, mask = cv2.threshold(blur, 120, 255, cv2.THRESH_BINARY)
contours,_ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    area = cv2.contourArea(cnt)
    if area < 500:
        continue
    x,y,w,h = cv2.boundingRect(cnt)
    cx,cy = x+w//2, y+h//2
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
    cv2.circle(img,(cx,cy),4,(0,0,255),-1)
    print("object", "area=",area,"center=", (cx,cy))

cv2.imwrite("result.png", img)
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Izmantojiet vienu kontrolētu attēlu, lai nodarbības mērķis būtu algoritma ķēde, nevis cīņa ar kameru.
- Paplašinājumā: adaptive threshold, morphology, camera calibration, pose estimation, object detection.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārto testu; nedrīkst "pielāgot" expected faktiskajam rezultātam pēc fakta.

27.12. L12 — IMU drift un complementary filter

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: izprast, kā gyro integrācija rada drift un kā divu sensoru informāciju var sapludināt.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- interpretēt gyro kā leņķisko ātrumu;
- izmantot dt integrācijā;
- izskaidrot accelerometer un gyro stiprās/vājās puses;
- realizēt complementary filter.

Ko pasniedzējam stāstīt pirms darba

Gyro integrācija: Leņķis tiek iegūts integrējot ātrumu laikā; neliels bias ar laiku uzkrājas.

Accelerometer: Ilgtermiņā var palīdzēt pret drift, bet kustībā tas mēra arī lineāru paātrinājumu un ir trokšņains.

Sensor fusion: Mērķis nav akli vidējot sensorus, bet apvienot to informāciju, izmantojot zināmu dinamiku un uzticamību.

Skolēna/studenta uzdevums

1. Izveido konstantu gyro signālu un integrē 5–10 s.
2. Pievieno gyro bias, piemēram +0.5 deg/s, un novēro drift.
3. Izveido accel_angle signālu ar troksni.
4. Realizē complementary filter ar alpha 0.98.
5. Salīdzini gyro-only, accel-only un fused rezultātu.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	gyro=0, bias=0	Leņķis paliek nemainīgs.			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
2	gyro bias +0.5 deg/s 10 s	Gyro-only kļūda pieaug apmēram par 5°.			
3	alpha=1.0	Fused kļūst par gyro-only integrāciju.			
4	alpha=0.0	Fused seko accel_angle.			

Analīzes jautājumi

1. Kāpēc precīzs dt ir kritisks?
2. Kāpēc magnetometru ne vienmēr var uzticami izmantot telpās?
3. Kā Kalman filter atšķiras konceptuāli no fiksēta alpha?

Tipiskās kļūdas un diagnostika

- dt pieņemts, bet reālajā sistēmā cikls nav konstants;
- grādi un radiāni sajaukti;
- accel angle tiek uzskatīts par perfektu kustības laikā;
- filter inicializācija nav definēta.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
IMU jēdzieni	6	Gyro, accel, drift, bias.
Integrācija	6	Pareizs dt un vienības.
Filter	7	Pareiza complementary formula.
Analīze	6	Salīdzina sensoru īpašības.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
import math

dt = 0.01
alpha = 0.98
angle = 0.0
samples = [
    # gyro_deg_s, accel_angle_deg
    (10.0, 0.2), (10.0, 0.4), (9.5, 0.7), (9.8, 1.0), (10.2, 1.2),
]
for gyro, accel_angle in samples:
    gyro_angle = angle + gyro*dt
    angle = alpha*gyro_angle + (1-alpha)*accel_angle
    print(f'gyro_pred={gyro_angle:.3f} fused={angle:.3f}')
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Sāciet ar vienkāršu 1D leņķi, nevis uzreiz 3D quaternion.
- Universitātes pāreja: covariance, process/measurement model, EKF.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtu testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.13. L13 — A* ceļa plānošana mobilam robotam

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90–120 min. Mērķis: atrast ceļu occupancy grid kartē un izprast izmaksas, heuristiku un replanning nepieciešamību.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- atšķirt map, start, goal un obstacle;
- izskaidrot g , h un $f=g+h$;
- rekonstruēt ceļu ar parent saitēm;
- pamatot, kā mainīga vide izraisa replanning.

Ko pasniedzējam stāstīt pirms darba

A*: Algoritms izvēlas daudzsološos mezglus pēc līdzšinējās izmaksas g un novērtējuma h līdz mērķim.

Heuristika: Manhattan distance ir piemērota 4-neighbor grid bez diagonāles; nepiemērota heuristika var mainīt garantijas.

Plāns nav vadība: A* dod ceļu/waypoints; zemāka līmeņa kontrollerim vēl jāvada robots fiziski pa šo ceļu.

Skolēna/studenta uzdevums

1. Izveido 10×10 grid ar šķēršļiem.
2. Realizē kaimiņu ģenerēšanu.
3. Realizē g un Manhattan h .
4. Atrodi ceļu no start līdz goal.
5. Vizualizē ceļu tekstā vai matplotlib.
6. Pievieno jaunu šķērslī uz atrastā ceļa un palaid replanning.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	Start=Goal	Ceļš satur tikai start/goal punktu.			

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
2	Mērķis sasniedzams	Atgriež nepārtrauktu ceļu bez šķēršļu šūnām.			
3	Mērķis pilnībā bloķēts	Atgriež None / skaidru no-path statusu.			
4	Jauns šķērslis uz ceļa	Pēc replanning ceļš mainās vai tiek ziņots no-path.			

Analīzes jautājumi

1. Kāpēc A* ceļš vēl nav trajektorija?
2. Kas notiek, ja robots ir lielāks par vienu grid šūnu?
3. Kāda atšķirība starp global planner un local obstacle avoidance?

Tipiskās kļūdas un diagnostika

- nav pārbaudītas kartes robežas;
- parent atjaunošana nav saistīta ar labāku g;
- šķēršļu šūnas netiek izlaistas;
- A* tiek uzskatīts par reāllaika motora kontrolleri.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
A* logika	8	g/h/f un open set.
Ceļa rekonstrukcija	5	Parent ķēde.
Robustums	6	No-path, robežas, šķēršļi.
Robotikas interpretācija	6	Plāns vs trajektorija/kontrole/replanning.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
import heapq

def astar(grid, start, goal):
    H,W=len(grid),len(grid[0])
    def h(a,b): return abs(a[0]-b[0])+abs(a[1]-b[1])
    open_heap=[(0,start)]
    g={start:0}; parent={start:None}
    while open_heap:
        _,cur=heapq.heappop(open_heap)
        if cur==goal:
            path=[]
            while cur is not None:
                path.append(cur); cur=parent[cur]
            return path[::-1]
        for dr,dc in [(1,0),(-1,0),(0,1),(0,-1)]:
            nxt=(cur[0]+dr,cur[1]+dc)
            if not (0<=nxt[0]<H and 0<=nxt[1]<W): continue
            if grid[nxt[0]][nxt[1]]==1: continue
            ng=g[cur]+1
            if ng < g.get(nxt,float('inf')):
                g[nxt]=ng; parent[nxt]=cur
                heapq.heappush(open_heap,(ng+h(nxt,goal),nxt))
    return None

grid=[[0,0,0,0,0],[0,1,1,1,0],[0,0,0,0,0],[0,1,0,1,0],[0,0,0,0,0]]
print(astar(grid,(0,0),(4,4)))
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Vispirms demonstrēriet algoritmu uz maza 5×5 režģa ar roku.
- Paplašinājumā: diagonāles, weighted cost map, Dijkstra salīdzinājums, RRT, dynamic replanning.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtu testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

27.14. L14 — ROS 2 datu plūsmas un autonomās navigācijas analīze

LABORATORIJAS DARBA KARTE

Ieteicamais laiks: 90 min. Mērķis: saprast sadalītas robotikas programmatūras arhitektūru un izsekot datu plūsmai sensor → localization/SLAM → planner → controller.

Vārds, uzvārds		Grupa	
Datums		Darba Nr. / variants	

Sasniedzamie rezultāti

- atšķirt node, topic, message, service/action;
- izskaidrot TF frame koku;
- uzzīmēt /scan, /imu, /odom, /map, /cmd_vel datu plūsmu;
- noteikt, kuri bloki ir perception, localization, planning un control.

Ko pasniedzējam stāstīt pirms darba

ROS 2 nav “robotu algoritms”: Tas ir programmatūras ekosistēmas un komunikācijas modelis, kur funkcijas sadalītas mezglos.

TF: Sensoru dati ir jāpildina tikai zinot koordinātu sistēmas un transformācijas.

Nav2/SLAM: SLAM veido/izmanto karti un pozu; navigācijas steks plāno un vada. Robota aparatūras draiveri ir vēl cits slānis.

Skolēna/studenta uzdevums

1. Uzzīmē vismaz 8 node arhitektūru autonomam mobilam robotam.
2. Katram node pieraksti publicētos/abonētos topic.
3. Izveido TF koku map → odom → base_link → lidar_link / imu_link.
4. Atzīmē, kur rodas /cmd_vel.
5. Apraksti sensor failure: LiDAR pazūd 2 s. Kā sistēma to konstatē un kādā režīmā pāriet?
6. Papildu: realizē minimālu rclpy publisher/subscriber, ja vide pieejama.

Testēšanas protokols

Faktisko rezultātu un PASS/FAIL aizpilda darba izpildītājs. PASS drīkst atzīmēt tikai tad, ja faktiskais rezultāts atbilst iepriekš definētajam paredzētajam rezultātam.

Nr.	Testa nosacījums	Expected / paredzētais	Actual / faktiskais	PASS/FAIL	Komentārs
1	/scan pazūd	Navigācija nedrīkst turpināt aklu kustību bez definētas degraded/safe politikas.			
2	Nepareizs TF	Dati var būt skaitliski derīgi, bet telpiski nepareizi; jādiagnosticē transformācija.			
3	/cmd_vel ir 0	Robots stāv, ja hardware interface pareizi interpretē komandu.			
4	Node restart	Pārējās sistēmas daļas nedrīkst nekontrolēti sabrukt; jābūt startup/recovery loģikai.			

Analīzes jautājumi

1. Kāpēc map→odom un odom→base_link ir atšķirīgi transformāciju slāņi?
2. Kāpēc ROS 2 node sadalījums palīdz testēšanai?
3. Ko nozīmē QoS sensoru datiem?
4. Kuri dati jālogē, lai pēc incidenta varētu rekonstruēt robota lēmumu?

Tipiskās kļūdas un diagnostika

- topic nosaukumi tiek zīmēti bez publisher/subscriber virziena;
- TF tiek jaukts ar sensora mērījumu;
- SLAM tiek uzskatīts par obstacle avoidance;
- nav failure/recovery stāvokļa.

Vērtēšana

Kritērijs	Punkti	Ko vērtē
Node/topic arhitektūra	7	Pareiza datu plūsma.
TF	6	Sakārtots frame koks.
Autonomās navigācijas slāņi	6	Perception/localization/ planning/control.
Failure analīze	6	Droša reakcija, diagnostika, recovery.

PASNIEDZĒJA MATERIĀLS — NEIZSNIEGT KĀ NEAIZPILDĪTU DARBA LAPU

Šajā daļā dots sagaidāmais risinājuma virziens, atbildes un tehniskie akcenti. Skolēna risinājumam nav obligāti jābūt identiskam, ja tas izpilda prasības un testus.

Atsauces risinājums / demonstrācijas kods:

```
# Konceptuāls ROS 2 Python publisher fragments (rclpy vide)
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32

class TemperaturePublisher(Node):
    def __init__(self):
        super().__init__('temperature_publisher')
        self.pub = self.create_publisher(Float32, '/temperature', 10)
        self.timer = self.create_timer(1.0, self.publish_value)
    def publish_value(self):
        msg = Float32(); msg.data = 24.6
        self.pub.publish(msg)

def main():
    rclpy.init(); node=TemperaturePublisher()
    rclpy.spin(node); node.destroy_node(); rclpy.shutdown()
```

Pasniedzēja skaidrojums un sagaidāmie secinājumi

- Šīs laboratorijas mērķis ir arhitektūra; ROS 2 instalācija nav obligāts priekšnosacījums pamata līmenim.
- Augstskolas līmenī pievienojiet QoS, lifecycle nodes, rosbag2, Nav2 behavior tree, EKF un launch konfigurācijas.

Sagaidāmais testēšanas princips: vispirms definē expected, tikai pēc programmas palaišanas ieraksta actual. Ja tests neiziet, students pieraksta novērojumu, labo cēloni un atkārtu testu; nedrīkst “pielāgot” expected faktiskajam rezultātam pēc fakta.

28. Projekta inženiertehniskās dokumentācijas veidnes

Šīs veidnes paredzētas noslēguma projektam un padziļinātiem laboratorijas darbiem. To mērķis ir iemācīt studentam pierādīt, ka risinājums atbilst prasībām. Veidnes veidotas tehniskai programmatūras/mehatronikas/robotikas sistēmai, bet tās var vienkāršot koledžas līmenim vai paplašināt universitātes darbam.

28.1. Prasību specifikācijas veidne

Prasības formulēšanas princips

Prasībai jābūt viennozīmīgai un pārbaudāmai. Ieteicamā forma: “Sistēmai jā...” + izmērāms nosacījums. Izvairīties no vārdiem “ātri”, “ērti”, “pietiekami precīzi”, ja nav skaitliska kritērija.

ID	Prasība	Tips	Prioritāte	Verifikācijas metode	Acceptance criterion
REQ-001	Sistēmai jāaptur mobilais robots, ja priekšējā sektorā šķērslis atrodas tuvāk par 0.70 m.	Functional/ Safety	Must	Test	Pie testiem 0.69 m → STOP; 0.71 m → neaktivizē šo stop nosacījumu.
REQ-002	Katram saglabātajam mērījumam jābūt timestamp, sensor_id un quality statusam.	Data	Must	Inspection + Test	DB ieraksts satur visus laukus 100 % testa ierakstu.

28.2. Prasību-funkciju-koda-testu izsekojamības matrica

Traceability matrix ļauj izsekot, vai katrai prasībai ir projektēšanas funkcija, realizācija un pierādījums. Tas ir īpaši vērtīgi projektos, kuros programmatūra vada fizisku procesu.

Prasība	Functional tree mezgls	Programmas modulis/funkcija	Testa ID	Rezultāts	Piezīme
REQ-001	Detect obstacle → Evaluate distance	safety.check_front_distance()	TC-04	PASS	Robežvērtības pārbaudītas.
REQ-002	Store measurement → Validate record	db.store_measurement()	TC-09	PASS	BAD quality saglabājas ar statusu.

28.3. Interfeisa / datu līguma veidne

Interface ID: IF-MQTT-001

Producer: ESP32_T101

Consumer: Python_DataLogger

Transport: MQTT

Topic: factory/line1/temperature/T-101

Payload schema:

```
{  
  "sensor_id": "string",  
  "value": "number|null",  
  "unit": "degC",  
  "quality": "GOOD|BAD|UNCERTAIN",  
  "timestamp": "ISO-8601 UTC"  
}
```

Failure behavior: if broker unavailable, buffer locally; do not stop local safety control.

28.4. Testa plāna un testa protokola veidne

Test ID	Prasība	Sākuma stāvoklis	Ievade/darbība	Expected	Actual	PASS/FAIL	Pierādījums

28.5. Atteiču scenāriju un riska žurnāls

ID	Atteice/ risks	Cēlonis	Sekas	Detekcija	Reakcija	Pārbaude
F-01	LiDAR dati pazūd	Savienojum s/sensora atteice	Nav uzticamas priekšējās distances	Timeout kopš pēdējā /scan	STOP vai degradēts drošs režīms	Atvienot datu avotu un pārbaudīt reakcijas laiku.
F-02	DB nav pieejama	Diska/tīkla problēma	Telemetrija netiek saglabāta	Write exception / health check	Buferēt lokāli; vadība turpinās	Simulēt write failure.

28.6. Koda pārskata (code review) kontrolsaraksts

- Mainīgo un funkciju nosaukumi apraksta nozīmi, nevis tikai x, a, temp1 bez konteksta.
- I/O un validācija atdalīta no biznesa/vadības loģikas.
- Robežvērtības un kļūdu stāvokļi nav “maģiski skaitļi” vairākās vietās.
- Funkcijām ir viena skaidra atbildība.
- Kļūdas netiek klusi ignorētas; ir status/log/exception politika.
- Drošības funkcijām ir augstāka prioritāte par normālu secību.
- Testi pārbauda normālo, robežvērtību un atteices scenāriju.
- Komentāri izskaidro “kāpēc”, nevis atkārto acīmredzamu “ko dara rinda”.
- Nav sensitīvu paroļu/API key kodā.
- README paskaidro palaišanu, ievades, izejas un zināmos ierobežojumus.

28.7. Projekta aizstāvēšanas tehniskā struktūra

1. Problēma un lietotājs — ko sistēma risina?
2. Prasības — 3–8 svarīgākās izmērāmās prasības.

3. Functional tree — kā sistēmas funkcija sadalīta apakšfunkcijās.
4. Arhitektūra — sensori, programmatūra, datu plūsma, aktuatori/cloud/DB.
5. Galvenais algoritms — state machine, kinemātika, planner vai datu apstrāde.
6. Realizācija — Python/C++/SQL/IoT/robotikas komponentes.
7. Testēšana — vismaz 5 testa gadījumi, tostarp robežvērtība un atteice.
8. Rezultāti — expected pret actual, demonstrācija.
9. Ierobežojumi un riski — ko sistēma vēl neprot?
10. Nākamais solis — kā projektu attīstītu profesionālā vai universitātes līmenī.

29. Mācību procesa pierādījumi, pašvērtējums un portfolio

Lai kurss būtu izmantojams arī profesionālās izaugsmes un universitātes kontekstā, studentam jāuzkrāj ne tikai atzīmes, bet arī pierādījumi par spēju analizēt, projektēt, programmēt, testēt un pamatot tehnisku sistēmu. Šo sadaļu var izmantot kā semestra portfolio struktūru.

29.1. Minimālais portfolio saturs

- 1 functional tree ar skaidriem testējamiem gala mezgliem;
- 1 blokshēma un atbilstošs pseidokods;
- 1 Python programma ar testiem un kļūdu apstrādi;
- 1 C++ programma ar funkcijām vai klasi;
- 1 SQL datubāzes piemērs un vismaz 3 vaicājumi;
- 1 IoT/JSON/MQTT datu līgums vai simulācija;
- 1 mehatronikas feedback simulācija;
- 1 robotikas darbs (kinemātika, LiDAR, Vision, IMU vai path planning);
- noslēguma projekta prasības, arhitektūra, testu protokols un demonstrācijas pierādījumi;
- 1 refleksija par kļūdu, kas sākotnēji netika pamanīta, un kā tā tika diagnosticēta.

29.2. Pašvērtējuma matrica

Kompetence	1 — vēl vajag palīdzību	2 — izpildu pēc parauga	3 — izpildu patstāvīgi	4 — varu pamatot/mācīt citam
Algoritma dekompozīcija				
Python				
C++				
SQL/datu modelis				
IoT/datu plūsma				
Testēšana/debugging				
Mehatronikas feedback				
Robotu kinemātika				

Kompetence	1 — vēl vajag palīdzību	2 — izpildu pēc parauga	3 — izpildu patstāvīgi	4 — varu pamatot/mācīt citam
LiDAR/Vision/IMU				
Autonomā robota arhitektūra				
Tehniskā dokumentācija				
Aizstāvēšana un argumentācija				

29.3. Pasniedzēja refleksijas lauks pēc katras 90 min nodarbības

Īss cikls kursa uzlabošanai

Pēc nodarbības pieraksta 3 lietas: 1) ko grupa saprata ātri; 2) kur bija sistemātiska kļūda; 3) ko nākamajā reizē mainīt teorijā, piemērā vai uzdevuma secībā. Šādi kursa dokuments var attīstīties no reālas mācīšanas pieredzes.

Nodarbība / datums	
Kas nostrādāja labi?	
Kur skolēni iestrēga?	
Tipiskā kļūda / nepareizais priekšstats	
Ko mainīt nākamajā reizē?	
Papildu ideja / profesionāls piemērs	

29.4. Nākamais dokumenta attīstības slānis

- pilni “ko stāstīt” scenāriji visām 36 nodarbībām ar pasniedzēja runas struktūru un demonstrāciju secību;
- vizuālie materiāli: blokhēmas, datu plūsmas, state machine, PID/feedback, kinemātikas un ROS 2 shēmas;

- atsevišķa skolēna rokasgrāmatas izdrukas versija bez pasniedzēja atbildēm;
- konkrētai izglītības iestādei pielāgota kursa apstiprināšanas forma ar kontaktstundām un patstāvīgā darba apjomu;
- fiziskās laboratorijas komplekta specifikācija: ESP32/Arduino, sensori, motors/driver, IMU, 2D LiDAR vai simulācijas alternatīva;
- universitātes papildmoduļi ar lineāro algebru, DH/Jacobian, Kalman/EKF, vadības teoriju un ROS 2 praktiku.

Dokumenta darba versija 0.6 — vienotā kursa izstrāde turpināma tajā pašā failā.

30. Paplašinātā mācību grāmatas teorija

Šī nodaļa paplašina iepriekšējo kursa saturu līdz rokasgrāmatas un universitātes ievadlīmeņa mācību grāmatas dziļumam. Katrs temats ir veidots tā, lai pasniedzējs var izvēlēties, cik dziļi to pasniegt: pamata līmenī svarīga ir sistēmas ideja un praktiskais uzdevums; padziļinātajā līmenī — mehānisms, robežgadījumi un testēšana; universitātes līmenī — matemātiskais modelis, arhitektūra un profesionāla argumentācija.

Trīs dziļuma līmeņi

Pamata līmenis: saprast un pielietot. Padziļinātais profesionālais līmenis: analizēt, diagnosticēt un integrēt. Universitātes līmenis: modelēt, pamatot, salīdzināt un projektēt.

30.1. Python — no algoritma līdz strukturētai inženiertehniskai programmai

Programmas izpildes modelis

Python avota kodu izpilda Python interpretators. Mācību līmenī svarīgākais nav terminoloģiska strīda par “interpretētu” vai “kompilētu” valodu niansēm, bet izpratne, ka avota teksts tiek pārvērsts izpildāmā formā un izpildīts noteiktā secībā. Kļūdas var rasties pirms programmas izpildes, izpildes laikā vai loģikā, kad programma darbojas, bet rezultāts ir nepareizs.

Dinamiska tipēšana un datu objekti

Python mainīgā nosaukums ir saite uz objektu. Tas ļauj ātri rakstīt programmas, tomēr tehniskā kodā prasa disciplīnu: skaidrus nosaukumus, validāciju, dokumentētas mērvienības un paredzamu interfeisu. Īpaši sensoru datus jāatšķir “nav mērījuma”, nulle un kļūdaina vērtība.

Mutability un blakusefekti

Saraksti un vārdnīcas ir maināmi objekti. Ja viena un tā pati datu struktūra tiek nodota vairākām funkcijām, izmaiņas vienā vietā var ietekmēt citu programmas daļu. Tas ir labs iemesls mācīt funkciju atbildību, kopēšanu, nemaināmus parametrus un testus.

Funkcijas un moduļi

Funkcija sadala problēmu nelielā, testējamā vienībā. Modulis grupē saistītas funkcijas, konstantes un klases. Tehniskā projektā atsevišķi moduļi var būt sensoru nolasīšanai, validācijai, vadības loģikai, datubāzei un lietotāja interfeisam.

Izņēmumi un kļūdu apstrāde

try/except nav paredzēts kļūdu “paslēpšanai”. Profesionālā pieejā izņēmums tiek apstrādāts tur, kur programma spēj pieņemt jēgpilnu lēmumu: atkārtot operāciju, pāriet drošā režīmā, ierakstīt kļūdu žurnālā vai pārtraukt procesu ar skaidru paziņojumu.

Faili, CSV un JSON

CSV ir ērts tabulāriem datiem; JSON ir piemērots strukturētai datu apmaiņai. Tehniskā sistēmā jādefinē lauku nosaukumi, mērvienības, timestamp formāts, versija un rīcība, ja lauks trūkst.

OOP inženiertehniskā kontekstā

Klase ir noderīga, ja objektam ir stāvoklis un ar to saistīta uzvedība. Sensor klasē var apvienot kalibrācijas parametrus, statusu un read() metodi; motoram — ātrumu, enable stāvokli un drošības ierobežojumus. OOP nav pašmērķis — tā jāizmanto, ja tā samazina sarežģītību.

```
from dataclasses import dataclass
from datetime import datetime

@dataclass
class Measurement:
    timestamp: datetime
    temperature_c: float
    quality_ok: bool

def validate_temperature(value: float) -> bool:
    return -40.0 <= value <= 125.0

def process_measurement(value: float) -> Measurement:
    return Measurement(
        timestamp=datetime.now(),
        temperature_c=value,
        quality_ok=validate_temperature(value),
    )
```

Ko pasniedzējam uzsvērt

Laba Python programma nav “īss kods par katru cenu”. Inženiertehniskā vidē svarīgāki ir skaidri datu līgumi, validācija, kļūdu apstrāde, testējamība un iespēja pēc incidenta saprast, kas notika.

30.2. C++ — tipi, atmiņa un programmas dzīves cikls

Preprocessing, compilation un linking

C++ programmas izveidē avota faili tiek priekšapstrādāti, kompilēti objektu failos un saistīti izpildāmā programmā. Šī ķēde palīdz saprast kļūdu tipus: sintakses/kompilācijas kļūda, linking kļūda un runtime kļūda.

Statiskā tipēšana

Mainīgā tips ir zināms kompilācijas laikā. Tas ļauj kompilatoram atklāt daļu kļūdu agrāk un ir īpaši vērtīgi iegultās un vadības sistēmās, kur atmiņa un datu formāts ir jākontrolē.

Stack un heap

Lokāliem mainīgajiem un funkciju izsaukumiem tipiski izmanto stack; dinamiskai atmiņai — heap. Pamata kursā pietiek saprast dzīves ilgumu. Universitātes limenī jārunā par ownership, lifetime, dangling pointer un memory leak.

Pointer un reference

Pointer glabā adresi un var būt null; reference ir cita objekta aizstājvārds. Tie kļūst svarīgi, strādājot ar aparatūras reģistriem, lielām datu struktūrām, draiveriem un bibliotēku API.

RAII

Resource Acquisition Is Initialization princips sasaista resursa dzīves ciklu ar objekta dzīves ciklu. Tas palīdz droši pārvaldīt failus, mutex, atmiņu un citus resursus bez manuālas “atbrīvošanas” katrā iespējamā programmas zarā.

STL

std::vector, std::array, std::string, std::map un algoritmi ļauj izmantot pārbaudītas datu struktūras. Studentam jāsaprot izvēles pamatojums: vai izmērs ir fiksēts, vai nepieciešama meklēšana, vai svarīga iterācijas secība.

Embedded un real-time apsvērumi

Mikrokontrolerī atmiņa, laiks un enerģija ir ierobežoti. Real-time nenozīmē “ļoti ātri”; tas nozīmē, ka rezultātam jābūt pieejamam noteiktā termiņā. Tāpēc jāvērtē worst-case execution time, bloķējošas operācijas un dinamiskas alokācijas riski.

```
#include <array>
#include <cstdint>

class MotorController {
public:
    void setTargetRpm(double rpm) { targetRpm_ = rpm; }
    double error(double measuredRpm) const {
        return targetRpm_ - measuredRpm;
    }
private:
    double targetRpm_{0.0};
};
```

Diskusijas jautājums

Kāpēc vadības sistēmā reizēm izvēlas C++ komponenti motora kontrolei, bet Python — datu analīzei vai eksperimentu automatizēšanai? Atbildei jābalstās prasībās, nevis “viena valoda ir labāka”.

30.3. SQL un datu inženierijas pamati tehniskām sistēmām

Relāciju modelis

Relāciju datubāze glabā datus tabulās ar definētiem laukiem un saistībām. Tehniskā projektā atsevišķi objekti var būt device, sensor, measurement, alarm un maintenance_event.

Primary key un foreign key

Primārā atslēga unikāli identificē ierakstu; ārējā atslēga sasaista ierakstu ar citu tabulu. Tas ļauj izvairīties no nekontrolētas teksta dublēšanas un saglabāt datu integritāti.

Normalizācija

Normalizācijas mērķis ir samazināt nevajadzīgu dublēšanu un anomālijas. Mācību līmenī pietiek ar jautājumu: “Vai vienu faktu glabājam vienā vietā?” Reālā analītiskā reizēm apzināti denormalizē, ja to prasa veikspēja.

JOIN

JOIN savieno informāciju no vairākām tabulām. Tas ir svarīgs solis no “SQL sintakses” uz datu modeli: students sāk domāt par objektu attiecībām.

Indeksi

Indekss paātrina meklēšanu, bet aizņem vietu un palielina rakstīšanas izmaksas. Laika rindām tipisks indekss ir pēc device_id un timestamp.

Transakcijas

Transakcija nodrošina, ka saistīta izmaiņu grupa tiek izpildīta kā viena loģiska vienība. Tas ir būtiski, ja, piemēram, jāreģistrē gan mērījums, gan tam piesaistīts alarms.

Time-series dati

Sensoru dati bieži ir laika rindas. Jādomā par paraugu biežumu, timestamp avotu, laika zonu, datu retināšanu, arhivēšanu un to, cik ilgi jā saglabā izejas dati.

```
CREATE TABLE device (  
    id INTEGER PRIMARY KEY,  
    name TEXT NOT NULL  
);  
  
CREATE TABLE measurement (  
    id INTEGER PRIMARY KEY,  
    device_id INTEGER NOT NULL,  
    ts TEXT NOT NULL,  
    value REAL NOT NULL,  
    FOREIGN KEY (device_id) REFERENCES device(id)  
);  
  
SELECT d.name, AVG(m.value) AS avg_value  
FROM device d  
JOIN measurement m ON m.device_id = d.id  
WHERE m.ts >= '2026-01-01T00:00:00'  
GROUP BY d.name;
```

30.4. IoT un mākoņdatu arhitektūra

Device-edge-cloud modelis

Ierīce iegūst datus; edge slānis var filtrēt, buferēt vai pieņemt lokālus lēmumus; cloud/servera slānis glabā, analizē un izplata informāciju. Drošības kritiska vadība parasti nedrīkst būt atkarīga tikai no interneta savienojuma.

MQTT

MQTT izmanto publish/subscribe modeli. Publisher nosūta ziņu topic; broker to izplata abonentiem. Students jāiemāca projektēt topic nosaukumus, payload shēmu, timestamp un device identitāti.

QoS

MQTT QoS nosaka piegādes garantijas līmeni. Augstāka garantija maksā ar lielāku protokola slodzi un iespējamu dublikātu apstrādes nepieciešamību. Jāizvēlas pēc datu nozīmīguma.

HTTP/REST

HTTP pieprasījumi ir ērti konfigurācijai, API un retākiem datu pieprasījumiem. REST interfeisā resursiem ir URI, tiek izmantotas metodes, piemēram, GET/POST/PUT/DELETE, un atbildes statusa kodi.

Offline un buffering

Reālai IoT sistēmai jāparedz tīkla zudums. Edge ierīce var buferēt datus lokāli, saglabāt secības numuru un pēc savienojuma atjaunošanas nosūtīt iztrūkstošos ierakstus.

Laiks

Ja vairāku sensoru dati jāapvieno, timestamp precizitāte kļūst kritiska. Jāzina, kura ierīce piešķir laiku, kā tiek sinhronizēti pulksteņi un vai sistēma izmanto UTC.

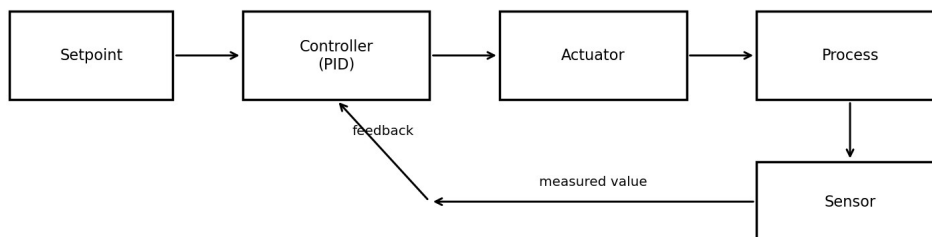
Kiberdrošība

Minimālais komplekts: autentifikācija, autorizācija, šifrēts transports, secrets neglabāt avota kodā, ierīču identitāte, atjaunināšanas politika un žurnāli. Drošības apsvērumi jāievieš jau arhitektūras posmā.

Slānis	Tipiska atbildība	Ko mācīt studentam
Device	Sensors, mikrokontroleris, lokāla I/O	mērījums, validācija, kļūdas kods
Edge	buferis, filtrēšana, lokāla loģika	offline režīms, datu kvalitāte
Transport	MQTT/HTTP/TCP	topic/API, QoS, timeout
Server/Cloud	glabāšana, API, analīze	SQL, dashboard, alarms
Application	lietotāja lēmums	vizualizācija, auditējams notikums

30.5. Mehatronika — no fiziskā signāla līdz vadības lēmumam

Slēgtās vadības cilpas pamatstruktūra



30.1. attēls. Slēgtā vadības cilpa: setpoint → controller → actuator → process → sensor → feedback.

Mērījumu ķēde

Fizikāls lielums tiek pārveidots elektriskā signālā, kondicionēts, digitalizēts un interpretēts programmatūrā. Katrā posmā rodas kļūdas: sensoru tolerance, troksnis, kvantēšana, nobīde, mērvienību konversija.

Sampling un Nyquist intuīcija

Digitālā sistēma neredz nepārtrauktu signālu — tā iegūst paraugus. Ja mērām pārāk reti, varam zaudēt būtisku dinamiku vai iegūt maldinošu rezultātu. Universitātes līmenī šo sasaista ar Nyquist-Shannon principu.

Aktuators un jaudas interfeiss

Programmas GPIO signāls parasti nespēj tieši darbināt motoru. Starp kontrolleri un aktuatoru ir driver, relejs, H-bridge, invertors vai cits jaudas posms. Programmatūras komanda tāpēc ir tikai viena daļa no enerģijas ķēdes.

PWM

Pulse Width Modulation maina vidējo enerģijas padevi, kontrolējot duty cycle. To plaši izmanto DC motoru, LED un dažādu jaudas slodžu vadībā.

Feedback

Atgriezeniskā saite ļauj salīdzināt vēlamu stāvokli ar faktisko un koriģēt vadību. Bez feedback sistēma nevar zināt, vai motors patiešām sasniedza vēlamu ātrumu.

Histerēze

ON/OFF regulatoram histerēze novērš biežu pārslēgšanos ap vienu robežvērtību. Piemēram, ventilators ieslēdzas virs 30 °C, bet izslēdzas tikai zem 27 °C.

Drošības slānis

E-stop, safety relay, guard interlock un limit switch nav vienkārši “parasti if”. Drošības funkciju arhitektūra jāatdala no normālas procesa vadības un jāprojektē atbilstoši riskam.

31. Universitātes robotikas lekciju modulis

Šis modulis paredzēts kā nākamais līmenis pēc kursa pamatdaļas. To var izmantot pasniedzēja pašmācībai, izvēles lekcijām profesionālajā izglītībā vai kā universitātes ievadkursa kodolu. Katrā apakšnodaļā ir teorija, aprēķina piemērs un Python realizācijas virziens.

31.1. Vektori, matricas un koordinātu sistēmas robotikā

Robots darbojas ar vienu un to pašu fizisko punktu dažādās koordinātu sistēmās: world/map, base, joint/link, tool, camera, LiDAR un IMU. Tāpēc robotikā būtiska prasme ir pārveidot koordinātas no viena frame uz citu.

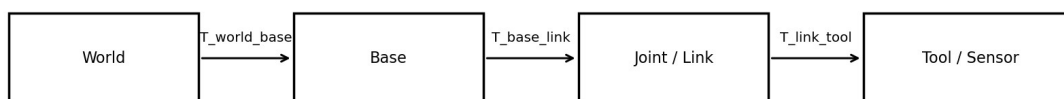
$$\mathbf{p} = [x, y, z]^T$$

Punkta pozīcijas vektors trīsdimensiju telpā.

$$\mathbf{p}_A = \mathbf{R}_{AB} \cdot \mathbf{p}_B + \mathbf{t}_{AB}$$

Punkts no B koordinātu sistēmas izteikts A koordinātu sistēmā.

Koordinātu sistēmu un transformāciju ķēde



31.1. attēls. Transformāciju ķēde starp world, base, link un tool/sensor frame.

Ko stāstīt auditorijai

Koordinātu transformācija nav tikai matemātika. Ja LiDAR ir uzstādīts 20 cm priekšā robota centra un pagriezts par 5°, tā mērījumu nedrīkst tieši interpretēt base frame bez transformācijas.

31.2. Homogēnās transformācijas

Homogēnā 4×4 matrica apvieno rotāciju un translāciju vienā operācijā. Tas ļauj ērti reizināt secīgas transformācijas un veidot kinemātisku ķēdi.

$$T = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{\text{world\_tool}} = T_{\text{world\_base}} @ T_{\text{base\_link1}} @ T_{\text{link1\_link2}} @ T_{\text{link2\_tool}}$$

Svarīga ideja: matricu reizināšanas kārtība nav patvaļīga. Transformācija jāveido tādā secībā, kādā pārvietojamies pa koordinātu sistēmu ķēdi.

31.3. Denavit–Hartenberg (DH) parametri

DH konvencija apraksta manipulatora secīgo locītavu ģeometriju ar četriem parametriem. Tā ir sistematiska metode, lai no mehāniskās struktūras iegūtu transformāciju matricas.

Parametrs	Nozīme	Tipisks jautājums
a _i	link length	cik tālu pa x asi līdz nākamajai asij?
alpha _i	link twist	par cik pagriežas locītavu asu orientācija?
d _i	link offset	cik tālu pa z asi?
theta _i	joint angle	rotācijas locītavas mainīgais leņķis

Pasniedzēja metodika

Sākumā neļaut studentam mehāniski aizpildīt DH tabulu. Vispirms katrai locītavai uzzīmē z asi, tad izvēlas x asi, un tikai pēc tam lasa parametrus. Lielākā daļa kļūdu ir frame izvēlē, nevis matricu reizināšanā.

31.4. Forward un inverse kinematics ar Python

Forward kinematics atbild uz jautājumu: “Ja locītavu stāvokļi ir zināmi, kur atrodas gala efektors?”

Inverse kinematics — “Kādi locītavu stāvokļi nepieciešami, lai gala efektors nonāktu vēlamajā pozā?”

$$x = L1 \cos(\theta_1) + L2 \cos(\theta_1 + \theta_2)$$

$$y = L1 \sin(\theta_1) + L2 \sin(\theta_1 + \theta_2)$$

$$c2 = (x^2 + y^2 - L1^2 - L2^2) / (2 L1 L2)$$

```
import math
```

```
def fk_2dof(L1, L2, th1, th2):  
    x = L1*math.cos(th1) + L2*math.cos(th1+th2)  
    y = L1*math.sin(th1) + L2*math.sin(th1+th2)  
    return x, y
```

```
def ik_2dof(L1, L2, x, y):  
    c2 = (x*x + y*y - L1*L1 - L2*L2)/(2*L1*L2)  
    if abs(c2) > 1:  
        raise ValueError('Target outside workspace')  
    s2 = math.sqrt(max(0.0, 1-c2*c2))  
    th2 = math.atan2(s2, c2)  
    th1 = math.atan2(y, x) - math.atan2(L2*s2, L1+L2*c2)  
    return th1, th2
```

Universitātes diskusija

Vienam mērķim var būt vairākas IK konfigurācijas (piemēram, elbow-up un elbow-down), dažiem mērķiem — neviena. Tādēļ inverse kinematics nav tikai formulas ievietošana; nepieciešama workspace un konfigurācijas izvēles loģika.

31.5. Jacobian, ātrumi un singularitātes

Jacobian matrica sasaista locītavu ātrumus ar gala efektora lineāro/leņķisko ātrumu. 2DOF plaknes manipulatoram tā iegūstama, diferencējot x un y pēc locītavu leņķiem.

$$J = \begin{bmatrix} -L1 \cdot \sin(t1) - L2 \cdot \sin(t1+t2), & -L2 \cdot \sin(t1+t2) \\ L1 \cdot \cos(t1) + L2 \cdot \cos(t1+t2), & L2 \cdot \cos(t1+t2) \end{bmatrix}$$

$$[x\_dot, y\_dot]^T = J \cdot [\theta1\_dot, \theta2\_dot]^T$$

Singularitātē Jacobian zaudē rangu. Fiziski tas nozīmē, ka noteiktā virzienā gala efektora kustību nevar realizēt ar galīgiem locītavu ātrumiem vai sistēma zaudē vienu neatkarīgu kustības virzienu.

31.6. Mobilā robota differential-drive kinemātika un odometrija

$$v = (v\_R + v\_L) / 2$$

$$\omega = (v\_R - v\_L) / b$$

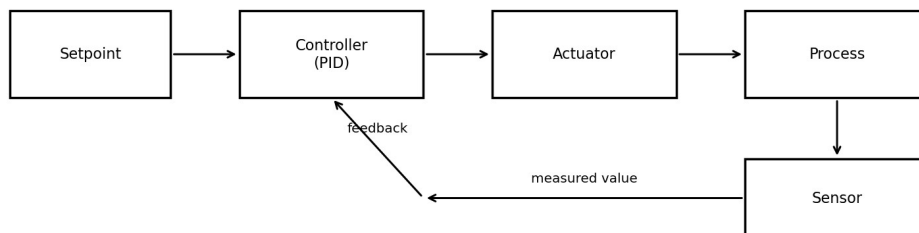
b — attālums starp riteniem.

$$\dot{x} = v \cos(\theta), \quad \dot{y} = v \sin(\theta), \quad \dot{\theta} = \omega$$

Odometrija integrē riteņu kustību laikā. Tā ir relatīva pozas novērtēšana un kļūda uzkrājas riteņu slīdēšanas, diametra kļūdas un grīdas nevienmērības dēļ. Tāpēc autonomā robotā odometriju bieži koriģē ar IMU, LiDAR vai kameru.

31.7. PID vadības teorijas ievads

Slēgtās vadības cilpas pamatstruktūra



31.2. attēls. PID darbojas slēgtā cilpā, izmantojot kļūdu starp setpoint un mērījumu.

$$e(t) = r(t) - y(t)$$

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t) dt + K_d \cdot de(t)/dt$$

Daļa	Ko tā “redz”	Ieguvums	Tipisks risks
P	pašreizējo kļūdu	ātra reakcija	svārstības / statiska kļūda
I	uzkrāto kļūdu	novērš statisku kļūdu	integrators windup
D	kļūdas izmaiņu ātrumu	slāpē un prognozē tendenci	jūtība pret troksni

```

error = setpoint - measured
integral += error * dt
derivative = (error - previous_error) / dt
output = Kp*error + Ki*integral + Kd*derivative
previous_error = error

```

Diskrētā programmā obligāti jādomā par dt , output saturation, integral windup, sensora troksni un to, kas notiek, ja mērījums nav derīgs.

31.8. IMU, orientācija un sensor fusion

Akselerometrs mēra specifisko spēku, žiroskops — leņķisko ātrumu, magnetometrs — magnētiskā lauka virzienu. Viena sensora interpretācija nav pietiekama visos apstākļos: žiroskops driftē, akselerometrs ir jutīgs pret kustības paātrinājumu, magnetometrs — pret magnētiskiem traucējumiem.

$$\text{angle} = \alpha \cdot (\text{angle} + \text{gyro\_rate} \cdot dt) + (1 - \alpha) \cdot \text{acc\_angle}$$

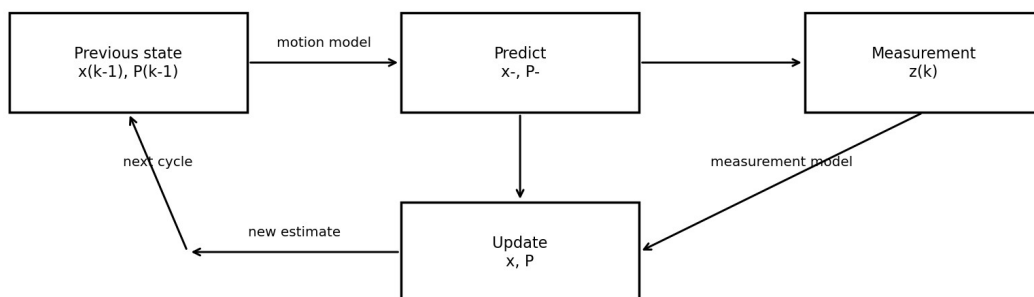
Vienkāršota complementary filter ideja vienai asij.

Ko uzsvērt

Sensor fusion nav “vairāk sensoru = automātiski labāk”. Katram sensoram jāzina trokšņa modelis, koordinātu sistēma, laiks, kalibrācija un atteices pazīmes.

31.9. Kalman filtrs un Extended Kalman Filter (EKF)

Kalman filtra predict-update cikls



31.3. attēls. Stāvokļa novērtējums sastāv no predict un update soļā.

Kalman filtrs kombinē sistēmas dinamisko modeli ar mērījumiem, ņemot vērā nenoteiktību. Tas nevis vienkārši “vidējo sensorus”, bet uztur stāvokļa novērtējumu un tā kovarianci. EKF izmanto nelineārām sistēmām, linearizējot modeli ap pašreizējo stāvokli.

Predict:

$$\begin{aligned} \hat{x}_{k^-} &= F \hat{x}_{k-1} + B u_k \\ P_{k^-} &= F P_{k-1} F^T + Q \end{aligned}$$

Update:

$$\begin{aligned} y_k &= z_k - H \hat{x}_{k^-} \\ S_k &= H P_{k^-} H^T + R \\ K_k &= P_{k^-} H^T S_k^{-1} \\ \hat{x}_k &= \hat{x}_{k^-} + K_k y_k \\ P_k &= (I - K_k H) P_{k^-} \end{aligned}$$

Universitātes līmenis

Svarīgākais nav no galvas iemācīties formulas, bet saprast F, H, Q un R nozīmi: kustības modelis, mērījuma modelis, procesa nenoteiktība un mērījuma nenoteiktība.

31.10. SLAM — lokalizācija un kartēšana vienlaikus

SLAM risina savstarpēji saistītu problēmu: lai izveidotu karti, jāzina sensora poza; lai noteiktu pozu, nepieciešama karte vai atkārtoti novērojami vides elementi. Praktiskās sistēmas izmanto odometriju, IMU, LiDAR vai kameru, datu asociāciju un optimizāciju.

- Front-end: iegūst sensoru pazīmes vai scan matching attiecības starp mērījumiem.
- Back-end: optimizē pozu/karte grafiku, samazinot uzkrāto kļūdu.
- Loop closure: atpazīst, ka robots atgriezies iepriekš apmeklētā vietā, un korigē kartes driftu.
- Failure modes: atkārtotā vide, maz pazīmju, kustīgi objekti, slikta sensora kalibrācija, nepareizi timestamp.

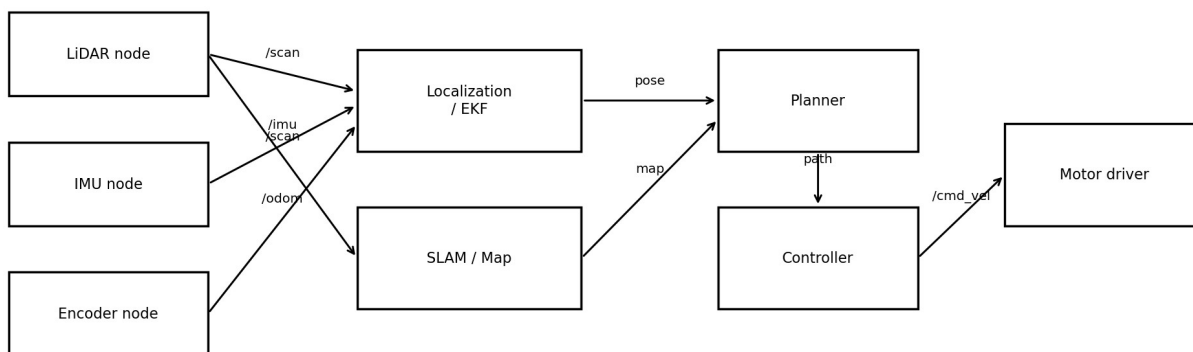
31.11. Path planning, trajectory planning un lokālā izvairīšanās

Path planning meklē ģeometrisku ceļu; trajectory planning pievieno laiku, ātrumu un paātrinājumu; lokālais planners/controller reaģē uz tuvākās apkārtnes dinamiku. Šos jēdzienus jānošķir, lai students nesauc katru šķēršļu apiešanu par A* vai SLAM.

Metode	Telpa	Stiprā puse	Ierobežojums
Dijkstra	grafi/režģi	garantē īsāko ceļu ar nenegatīvām izmaksām	meklē daudz mezglu
A*	grafi/režģi	heuristika paātrina meklēšanu	atkarīgs no kartes/heuristikas
RRT	nepārtraukta konfigurāciju telpa	labs sarežģītai daudzdimensiju telpai	ceļš nav automātiski gluds/optimāls
RRT*	nepārtraukta telpa	asimptotiski uzlabo ceļu	vairāk aprēķinu

31.12. ROS 2 sistēmu arhitektūra

ROS 2 autonomās navigācijas konceptuālā arhitektūra



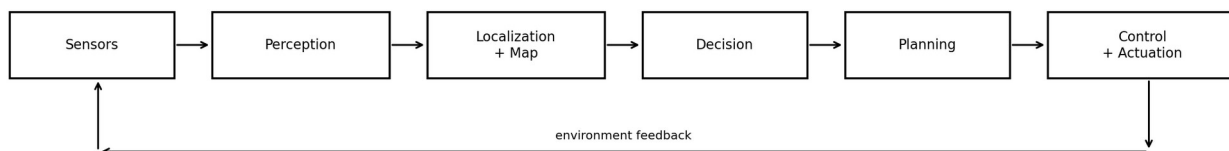
31.4. attēls. Konceptuālā ROS 2 navigācijas sistēmas node/topic arhitektūra.

ROS 2 ļauj sadalīt robotu neatkarīgās programmatūras komponentēs. Node veic konkrētu atbildību; topic pārraida straumējošus datus; service apkalpo īsu pieprasījumu/atbildi; action paredzēta ilgākam uzdevumam ar feedback un atcelšanas iespēju. TF uztur koordinātu sistēmu transformācijas laikā.

- Perception nodes: kamera, LiDAR, detektori, filtrēšana.
- Localization nodes: odometrija, IMU fusion, map matching.
- Planning nodes: global path, local trajectory.
- Control nodes: /cmd_vel vai locītavu komandas.
- Hardware interface: motora draiveri, enkoderi, safety I/O.
- Logging/diagnostics: timestamp, status, frequency, dropped messages, resource usage.

31.13. Kā projektēt autonoma robota “domāšanas” arhitektūru

Autonoma robota uztveres-lēmuma-vadības cikls



31.5. attēls. Autonoma robota cikls no sensora līdz fiziskai darbībai un atgriezeniskajai saitei.

Autonomija nav viena AI funkcija. Tā ir vairāku slāņu sadarbība ar skaidri definētām atbildībām, datu kvalitātes kritērijiem, prioritātēm un recovery režīmiem. Drošības slānim jāspēj pārtraukt normālo misiju, ja rodas kritisks stāvoklis.

Slānis	Galvenais jautājums	Piemērs
Sense	Ko es izmēru?	LiDAR range, IMU gyro, encoder ticks
Perceive	Ko tas nozīmē?	šķērslis, līnija, objekts
Localize	Kur es esmu?	x,y,theta + covariance
Decide	Ko darīt?	continue, stop, return-to-home
Plan	Kā tur nokļūt?	A*, RRT, trajectory
Control	Kā izsekot komandai?	PID, kinematic controller
Safety	Vai drīkst turpināt?	E-stop, watchdog, limit checks

32. Aprēķinu piemēri un formulas

Šīs sadaļas mērķis ir dot gatavus piemērus, kurus pasniedzējs var izrēķināt uz tāfeles, pēc tam atkārtot Python vidē un visbeidzot dot studentam modificētu variantu patstāvīgai izpildei.

32.1. 2DOF forward kinematics

1. Pieņemam $L1 = 0,50$ m, $L2 = 0,30$ m, $\theta_1 = 30^\circ$, $\theta_2 = 45^\circ$.
2. Pārvēršam leņķus radiānos: $\theta_1 \approx 0,524$ rad, $\theta_2 \approx 0,785$ rad.
3. $x = 0,50 \cos(30^\circ) + 0,30 \cos(75^\circ) \approx 0,511$ m.
4. $y = 0,50 \sin(30^\circ) + 0,30 \sin(75^\circ) \approx 0,540$ m.
5. Rezultāts: gala efektors atrodas aptuveni (0,511; 0,540) m.

32.2. Differential-drive ātrums

1. Kreisais ritenis $v_L = 0,30$ m/s, labais $v_R = 0,50$ m/s, riteņu bāze $b = 0,40$ m.
2. Lineārais ātrums $v = (0,50 + 0,30)/2 = 0,40$ m/s.
3. Leņķiskais ātrums $\omega = (0,50 - 0,30)/0,40 = 0,50$ rad/s.
4. Interpretācija: robots kustas uz priekšu un griežas pa kreisi.

32.3. Enkoderu ceļa aprēķins

1. Riteņa diametrs 0,10 m, enkoders 500 impulsi/apgriezienā, saskaitīti 1250 impulsi.
2. Apgriezieni $= 1250 / 500 = 2,5$.
3. Riteņa apkārtmērs $= \pi \cdot 0,10 \approx 0,314$ m.
4. Ceļš $\approx 2,5 \cdot 0,314 = 0,785$ m.

32.4. Viena PID soļa aprēķins

1. Setpoint = 100 rpm, measured = 92 rpm, $K_p=0,8$, $K_i=0,2$, $K_d=0,05$, $dt=0,1$ s.
2. Kļūda $e = 8$ rpm. Pieņemam iepriekšējo kļūdu 10 rpm un iepriekšējo integrāli 3,0.
3. Integral $= 3,0 + 8 \cdot 0,1 = 3,8$.
4. Derivative $= (8-10)/0,1 = -20$ rpm/s.
5. $u = 0,8 \cdot 8 + 0,2 \cdot 3,8 + 0,05 \cdot (-20) = 6,16$.

32.5. LiDAR polārās koordinātas uz x,y

1. Mērījums $r=2,0$ m, leņķis $\theta=30^\circ$.
2. $x = r \cos(\theta) = 2 \cdot 0,866 = 1,732$ m.
3. $y = r \sin(\theta) = 2 \cdot 0,5 = 1,0$ m.
4. Punkts sensora koordinātu sistēmā ir aptuveni (1,732; 1,0) m.

32.6. Complementary filter

1. Iepriekšējais leņķis 10° , gyro rate = $5^\circ/\text{s}$, $dt=0,02\text{ s}$, acc angle = $9,5^\circ$, $\alpha=0,98$.
2. Gyro prognoze = $10 + 5 \cdot 0,02 = 10,1^\circ$.
3. Filtrs = $0,98 \cdot 10,1 + 0,02 \cdot 9,5 \approx 10,088^\circ$.
4. Interpretācija: īstermiņā dominē žiroskops, ilgtermiņā akselerometrs koriģē driftu.

32.7. Sampling frekvences izvēle

1. Pieņemam, ka interesējošā procesa augstākā būtiskā frekvence ir 20 Hz.
2. Teorētiskais minimums bez aliasing ir virs 40 Hz.
3. Praktiskā sistēmā izvēlamies rezervi, piemēram, 100–200 Hz, un izmantojam atbilstošu filtrēšanu.
4. Pasniedzējam jāuzsver: sampling rate izvēli nosaka process, nevis “jo sensors spēj 1 kHz”.

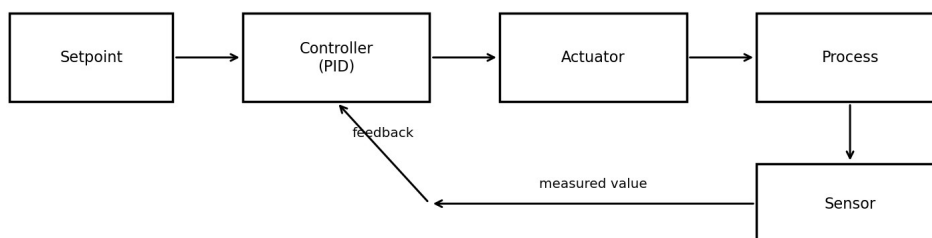
32.8. SQL agregācija laika logā

1. Uzdevums: aprēķināt sensora vidējo temperatūru pēdējās stundas laikā.
2. Jāfiltrē pēc sensor/device identitātes un timestamp, pēc tam jālieto AVG(value).
3. Paplašinājums: vienlaikus MIN, MAX, COUNT un kvalitātes filtra nosacījums.

33. Vizuālie sistēmu modeļi un arhitektūras shēmas

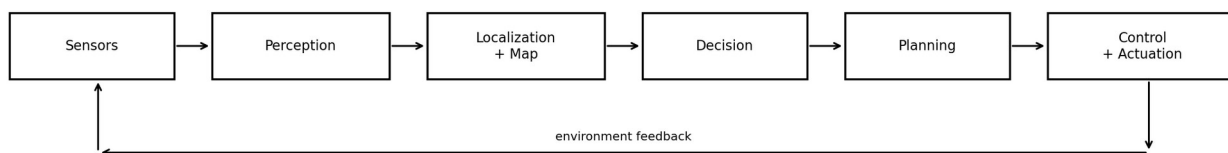
Vizuālais modelis palīdz pirms programmēšanas vienoties par sistēmas funkcijām, datu plūsmu un robežām. Šīs shēmas paredzētas lekciju demonstrācijām un studentu projektu dokumentācijai.

Slēgtās vadības cilpas pamatstruktūra



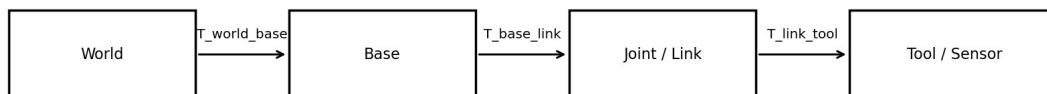
33.1. Slēgtā vadības cilpa.

Autonoma robota uztveres-lēmuma-vadības cikls



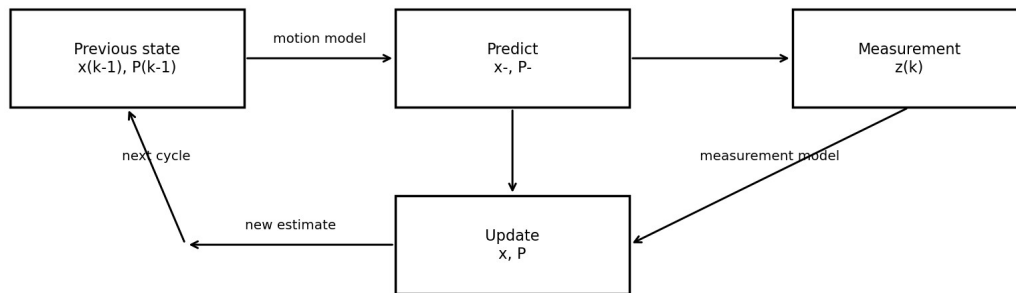
33.2. Autonoma robota uztveres-lēmuma-vadības arhitektūra.

Koordinātu sistēmu un transformāciju ķēde



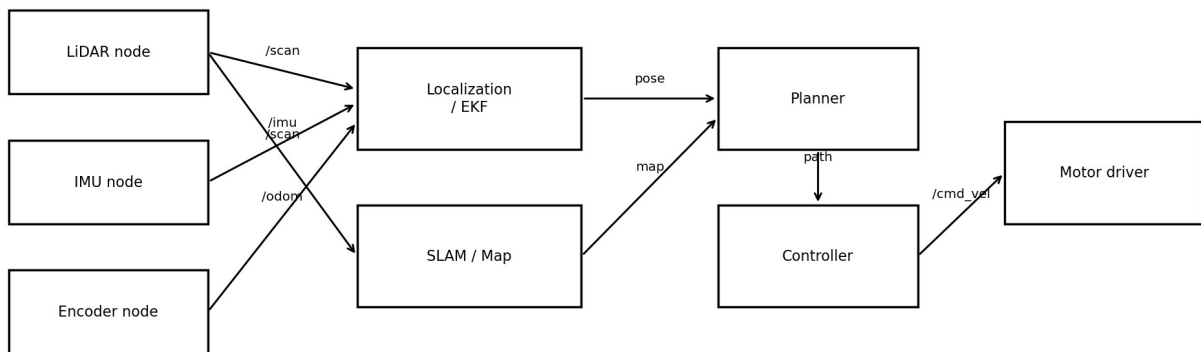
33.3. Koordinātu transformāciju ķēde.

Kalman filtra predict-update cikls



33.4. Kalman filtra predict-update struktūra.

ROS 2 autonomās navigācijas konceptuālā arhitektūra



33.5. ROS 2 autonomās navigācijas node/topic modelis.

33.6. Kā studentam pašam veidot sistēmas shēmu

1. Nosaki sistēmas robežu: kas ir iekšā un kas ārpus projekta?
2. Nosaki fiziskās ievades un izejas: sensori, operatora komandas, aktuatori.
3. Sadalīt programmatūru atbildībās: acquisition, validation, state estimation, decision, control, logging.
4. Uz katras bultas pieraksti datu saturu, vienības, virzienu un atjaunošanas biežumu.
5. Atzīmē safety un diagnostic ceļus atsevišķi no normālās darbības.
6. Pārbaudi, vai katra functional tree gala funkcija ir redzama arhitektūrā un tai ir tests.

34. Pasniedzēja profesionālās pašmācības modulis

Šis ceļš paredzēts pasniedzējam, kurš vēlas no kursa satura izaugt līdz spējai praktiski realizēt robotikas un autonomo sistēmu piemērus. Katrs posms beidzas ar taustāmu artefaktu, kuru vēlāk var izmantot lekcijā.

Posms	Tēma	Jāapgūst	Portfolio rezultāts
1	Python numeriskā programmēšana	NumPy, matplotlib, datu faili	2DOF FK/IK notebook
2	C++ strukturēts kods	classes, STL, memory/lifetime	Sensor + controller mini bibliotēka
3	Linux/Git	shell, process, SSH, Git workflow	repo ar README un reproducējamu palaišanu
4	Mikrokontrolleri	GPIO, ADC, PWM, timer, serial	sensors + PWM motor simulator/hardware
5	Vadības sistēmas	feedback, PID, sampling	motora/temperatūras PID simulācija
6	Robotu kinemātika	frames, transforms, DH, Jacobian	manipulatora vizualizācija Python
7	Mobilā robotika	odometry, differential drive	2D robota simulators
8	LiDAR/Vision/IMU	point data, OpenCV, orientation	perception datu pipeline
9	State estimation	complementary, Kalman/EKF	IMU+odometry fusion simulācija
10	Planning	A*, RRT, trajectory	grid planner + replanning
11	ROS 2	node/topic/TF/launch/logging	vairāku node simulācija
12	SLAM + autonomous navigation	map, localization, Nav stack	pilns simulēts autonomous mission demo

34.1. Pašmācības projekta princips

Katru jauno tēmu ieteicams mācīties vienā un tajā pašā ciklā: (1) teorija; (2) minimāls aprēķins ar roku; (3) Python/C++ simulācija; (4) testa gadījumi; (5) vizualizācija; (6) īss tehniskais skaidrojums saviem vārdiem. Ja nevar izskaidrot rezultātu bez koda, tēma vēl nav pietiekami apgūta.

35. 36 nodarbību pasniedzēja runas scenāriju karte

Šī karte nav pilns lekcijas teksts, bet detalizēts runas “mugurkauls”, lai katrai 90 min nodarbībai būtu skaidrs ievads, centrālais skaidrojums, demonstrācija un profesionālā saikne. To var izmantot arī kā pamatu slaidu veidošanai.

Nodarbība	Tēma	Centrālais vēstījums	Demonstrācijas enkurs
1	Algoritms	Cilvēks daudz ko izdara intuitīvi; datoram katrs lēmums jāformalizē.	Automātiskās durvis: sensors → lēmums → motors → gala sledzis.
2	Vadības struktūras	Secība, zarošanās un atkārtotās veido programmas vadības plūsmu.	Termostats un ciklisks sensora nolasījums.
3	Blokshēmas	Blokshēma ir izpildāms domāšanas modelis, nevis dekorācija.	Iziet blokshēmu ar trim testa datu kopām.
4	Mainīgie un dati	Datiem ir nozīme, tips, vienība, derīguma robežas un kvalitāte.	Temperatūra 25,0 bez °C un timestamp nav pilns mērījums.
5	Boolean loģika	Drošības un procesa lēmumi bieži ir vairāku nosacījumu kombinācija.	DoorClosed AND SafetyOK AND NOT EStop.
6	Cikli	Cikls ir periodiskas sistēmas pamatmehānisms, bet tam vajag izejas nosacījumu.	WHILE + timeout + watchdog.
7	Dekompozīcija	Sarežģīta sistēma kļūst vadāma, sadalot to atbildībās.	Functional tree un moduļu I/O.
8	Algoritmu pārbaude	Pārbaude vērtē spēju domāt, ne tikai atcerēties definīciju.	No prasības līdz blokshēmai un testa gadījumiem.
9	Python ievads	Python ir veids, kā jau zināmo algoritmu padarīt izpildāmu.	input → validate → calculate → print.
10	Python if	Nosacījumu secība ietekmē rezultātu; robežvērtības jādefinē precīzi.	Alarm limeņi un datu validācija.
11	Python cikli	Atkārtotu datu apstrādi apvieno ar sarakstiem, agregāciju un debugging.	10 sensoru paraugi → min/max/avg.
12	Python projekts	Projektā svarīgs ir ceļš prasība → kods → tests → demonstrācija.	Mini tehniska sistēma.
13	C++ struktūra	Kompilators dod agrīnu atgriezenisko saiti par tipiem un programmas struktūru.	source → compile → link → run.
14	C++ tipi	Tipa izvēle ir inženiertehniskais lēmums par diapazonu un precizitāti.	int pret double mērījumiem.
15	C++ state logic	switch un enum palīdz skaidri attēlot sistēmas stāvokļus.	IDLE/RUN/FAULT state machine.
16	C++ cikli	Vadības cilpai jābūt paredzamai un droši pārtraucamai.	periodiska nolasīšana un stop nosacījums.

Nodarbība	Tēma	Centrālais vēstījums	Demonstrācijas enkurs
17	Funkcijas	Viena funkcija — viena skaidra atbilde.	readSensor, validate, decide, actuate.
18	Masīvi	Datu kopa ļauj analizēt procesu laikā, nevis vienu mērījumu.	mērījumu logs un filtrs.
19	OOP	Klase ir noderīga, ja komponentei ir stāvoklis un uzvedība.	Sensor class ar calibration/status.
20	C++ pārbaude	Testēšana ietver normālo, robežvērtību un atteices scenāriju.	bojāts sensors un invalid input.
21	SQL	Datubāze glabā strukturētu procesa vēsturi un saistības.	device + measurement tabulas.
22	CSV/JSON/SQLite	Datu formāts ir līgums starp komponentēm.	JSON payload + SQLite ieraksts.
23	IoT	IoT ir visa datu ķēde, nevis tikai Wi-Fi sensors.	device → broker/API → DB → dashboard.
24	Integrācija	Integrācijā svarīgi interfeisi, laiks, kļūdas un traceability.	sensors simulācija līdz SQL.
25	Projekta prasības	Labā prasība ir pārbaudāma un izmērāma.	“sistēmai jāreaģē < 200 ms”.
26	Functional tree	Funkcija apraksta ko sistēma dara, nevis kāda detaļa nopirkta.	mission → perception/planning/control.
27	Python kodols	Sāc ar minimālu vertikālu funkciju, nevis visu sistēmu.	viens end-to-end datu ceļš.
28	Dati un testi	Expected definē pirms Actual; citādi tests kļūst par rezultāta attaisnošanu.	PASS/FAIL protokols.
29	C++ low-level	Pārnes to komponenti, kur nepieciešama tipu/laika/aparatūras kontrole.	controller vai device interface.
30	C++ OOP/testi	Sadalīt komponenti tā, lai to var testēt bez visa robota.	mock sensor / controller class.
31	Integrācija	Sensoriem ir koordinātu sistēma, timestamp un kvalitātes statuss.	LiDAR/IMU/MQTT integrācija.
32	Validācija	Atteices scenārijs ir tikpat svarīgs kā normālais scenārijs.	sensor timeout → degraded/safe mode.
33	Dokumentācija	Dokuments pierāda, kāpēc sistēma dara to, ko dara.	requirements <-> functions <-> code <-> tests.
34	Technical review	Review nav tikai kļūdu meklēšana; tas pārbauda argumentāciju un riskus.	architecture/code/test review.
35	Aizstāvēšana I	Demonstrācijā vispirms parāda prasību un tikai tad rezultātu.	test case evidence.

Nodarbība	Tēma	Centrālais vēstījums	Demonstrācijas enkurs
36	Aizstāvēšana II	Profesionālā izaugsme sākas ar spēju nosaukt ierobežojumus un nākamo soli.	portfolio + learning roadmap.

35.1. Izvērstis runas scenārijs — Algoritms un precīza instrukcija

Ievads (5–10 min): Sāc ar šķietami vienkāršu komandu “atver durvis”. Pajautā, ko cilvēks izdarīs automātiski: pamanīs rokturi, sapratīs virzienu, novērtēs šķērsli. Dators nevienu no šiem pieņēmumiem neveic pats.

Teorija (20 min): Definē algoritmu kā galīgu un nepārprotamu darbību secību. Izskaidro diskretumu, noteiktību, galīgumu, rezultativitāti un pielietojamību. Katru īpašību sasaisti ar automātisko durvju piemēru.

Sistēmu domāšana (15 min): Uz tāfeles uzraksti Input → Processing → Output. Input: motion sensor, safety beam, limit switch. Processing: state/decision logic. Output: motor open/close, alarm/status.

Kļūdas piemērs (10 min): Izņem no algoritma gala slēdzi un jautā, kā motors zina, kad apstāties. No tā pārej uz faktu, ka programmatūras nepilnība var kļūt par fizisku risku.

Prakse (35 min): Studenti apraksta automātiskās durvis 8–12 soļos, pēc tam pāros mēģina atrast viens otra algoritmā nenodefinētus pieņēmumus.

35.9. Izvērstis runas scenārijs — Python kā algoritma realizācija

Ievads: Atgādinā, ka Python nemaina algoritmu. Tā dod sintaksi, ar kuru dators var izpildīt jau izplānoto secību.

Demonstrācija: uzraksti vienu Input → Processing → Output piemēru pseidokodā un blakus Python. Parādi input() atgrieztā teksta konversiju uz float un paskaidro, kāpēc datu tips ir svarīgs.

Tehniskais piemērs: ievadīta sensora vērtība, validācijas robeža, aprēķins un statusa izvade. Pievieno kļūdainu ievadi un demonstrē, kas notiek bez validācijas.

Debugging: speciāli izveido NameError un ValueError. Studentiem jāpasaka, vai kļūda ir sintaksē, izpildes laikā vai loģikā.

Prakse: neliela programma ar diviem ievades datiem, mērvienību un vienu validācijas nosacījumu.

35.21. Izvērstis runas scenārijs — SQL un tehnisko datu vēsture

Ievads: Parādi vienu sensora skaitli un jautā, vai ar to pietiek, lai pēc nedēļas izmeklētu procesa problēmu. Pievieno timestamp, sensor_id, unit un quality status.

Modelis: paskaidro tabulu, rindu, kolonnu, primary key un foreign key. Uz tāfeles uzzīmē device 1→N measurement.

SQL: INSERT saglabā faktu; SELECT atbild uz jautājumu; WHERE filtrē; ORDER BY parāda secību; aggregate funkcijas palīdz analizēt periodu.

Profesionālā saikne: datubāzes vērtība ir ne tikai “saglabāt datus”, bet nodrošināt izsekojamību, audītējamu vēsturi un reproducējamu analīzi.

Prakse: izveidot tabulu, ievietot 10 mērījumus un uzrakstīt trīs jautājumus, uz kuriem datubāzei jāatbild.

35.26. Izvērstis runas scenārijs — Functional tree un sistēmas arhitektūra

Ievads: Parādi projektu “autonoms robots nogādā detaļu no A uz B”. Pirmais jautājums nav “kādu LiDAR pirkt?”, bet “kādas funkcijas sistēmai obligāti jāveic?”.

Functional tree: root funkciju sadali perception, localization, decision, planning, control, safety. Tad katru zaru sadali līdz funkcijām, kuras var pārbaudīt.

Arhitektūra: tikai pēc funkcijām izvēlas programmatūras moduļus un aparatūru. Viena funkcija var tikt realizēta ar dažādām tehnoloģijām.

Traceability: katrai prasībai jābūt funkcionālajā kokā, realizācijā un testā. Ja mezgls nav testējams, tas vēl ir pārāk abstrakts.

Prakse: studentu projektiem izveidot root + vismaz 4 galvenos zarus + 2 līmeņus zem tiem.

35.31. Izvērstis runas scenārijs — Sensoru integrācija un autonomais robots

Ievads: Autonoms robots “redz” tikai datu plūsmas — attālumus, pikselus, paātrinājumus, encoder ticks. Programmatūra no tiem veido vides un sava stāvokļa novērtējumu.

Sensoru metadati: katram mērījumam vajag frame, timestamp, mērvienību, kvalitāti un kalibrācijas kontekstu. Divus sensorus nevar korekti apvienot, ja nav zināms, kur un kad tie mērija.

Pipeline: LiDAR/IMU/encoder → perception/state estimation → localization/map → planner → controller. Uzsver, ka kļūda augšējā slānī izplatās uz lēmumu.

Atteice: simulē LiDAR timeout. Kā sistēma to konstatē? Vai var turpināt ar samazinātu ātrumu, vai jāapstājas? Tas ir degraded/recovery mode jautājums.

Prakse: studentam uzzīmēt autonomā robota datu plūsmu ar vismaz 8 blokiem un katrai bultai pierakstīt datu tipu/frekvenci.

36. [ARHĪVS] Versijas 0.6 izstrādes statuss un nākamie soļi

Versijā 0.6 vienotajam dokumentam pievienots mācību grāmatas padziļinājuma slānis, universitātes robotikas lekciju modulis, aprēķinu piemēri, vizuālās arhitektūras shēmas, pasniedzēja pašmācības ceļš un 36 nodarbību runas scenāriju karte.

Komponente	v0.6 statuss	Nākamais attīstības līmenis
Algoritmi/Python/C++/SQL/IoT	plaša teorija + darbi + testi	atsevišķa skolēna tīrā versija
Mehatronika	signālu ķēde, feedback, PWM, PID, safety	fiziskā laboratorijas komplekta montāžas shēmas
Robotika	kinemātika, LiDAR, Vision, IMU, planning	vairāk simulācijas projektu un 3D kinemātika
Universitātes modulis	DH, Jacobian, PID, Kalman/EKF, SLAM, ROS 2	lekciju slaidi un semināru uzdevumi
Pasniedzēja materiāls	runas karte + 5 izvērsti scenāriji	pilni izvērsti scenāriji visām 36 nodarbībām
Dokumentācija	requirements/traceability/test/risk veidnes	formāla kursa apstiprināšanas versija konkrētai iestādei

Saglabājamais princips

Viss saturs tiek turpināts vienā dokumentā. Ja tiek izveidota skolēna tīrā versija vai slaidi, tie tiek ģenerēti no šī pamatdokumenta, nevis uzturēti kā neatkarīga, savstarpēji nesaskaņota satura versija.

Dokumenta darba versija 0.6 — vienotā kursa izstrāde turpināma tajā pašā failā.

37. Pilnie 36 nodarbību pasniedzēja runas scenāriji

Šīs nodaļas mērķis ir padarīt gada plānu tieši izmantojamu auditorijā. Katrā scenārijā ir 90 minūšu struktūra, pasniedzēja stāstījuma kodols, jautājumi auditorijai, demonstrācija, tipiskie pārpratumi un stundas noslēgums. Teksts nav jālasa vārds vārdā; tas ir pasniedzēja drošības tīkls, lai stundas laikā nepazustu tehniskā doma.

37.1. Nodarbība 1 — Kas ir algoritms un kā domā dators?

Galvenā doma: Algoritms nav programmēšanas valoda. Tas ir precīzs risinājuma apraksts, kuru vēlāk var realizēt Python, C++, PLC vai citā vidē.

Ko tieši stāstīt auditorijai

Sāc ar jautājumu: “Ko nozīmē datoram pateikt — atver durvis?” Ļauj audzēkņiem nosaukt 3–4 soļus un apzināti uzdod neērtus papildjautājumus: cik tālu atvērt, kad apstāties, ko darīt, ja cilvēks paliek ailē.

Izskaidro galīgumu, noteiktību, diskretumu un rezultativitāti. Uzsvēr, ka tehniskā sistēmā neprecizitāte var kļūt par drošības problēmu, nevis tikai “bugu”.

Uz tāfeles uzzīmē Ievade → Apstrāde → Izvade. Automātiskajām durvīm ievade ir sensori, apstrāde ir vadības loģika, izvade ir motora komanda un indikācija.

Pārej uz domu: programmēšana sākas pirms koda. Vispirms jāspēj nosaukt problēmu, ievades, stāvokļus un sagaidāmo rezultātu.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Vai dators pats “saprot”, ko mēs gribam?

- Kas notiek, ja algoritmā nav apstāšanās nosacījuma?
- Kāda ir atšķirība starp instrukciju cilvēkam un instrukciju PLC?

Demonstrācija / tāfeles piemērs

Demonstrē automātisko durvju 8–12 soļu algoritmu un kopā atrod vienu trūkstošu drošības nosacījumu.

Tipiskie pārpratumi, ko koriģēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.2. Nodarbība 2 — Secība, zarošanās un atkārtošanās

Galvenā doma: Gandrīz jebkuras programmas vadības plūsmu var uzbūvēt no secības, izvēles un atkārtošanās.

Ko tieši stāstīt auditorijai

Atgādinī iepriekšējo stundu un pajautā, kas automātisko durvju algoritma vienkārši notiek secīgi, kas ir lēmums un kas atkārtojas.

Secību skaidro kā darbības, kur katra seko iepriekšējai. Zarošanos skaidro kā jautājumu ar vismaz diviem iespējamiem turpinājumiem. Ciklu — kā atkārtotu novērtēšanu vai darbību.

Tehniskā piemēra pamatā izmanto termostatu: nolasīt temperatūru, salīdzināt ar robežu, ieslēgt/izslēgt aktuatoru, pēc laika atkārtot.

Uzsvēr, ka “if” un “while” vēl nav konkrētas valodas komandas — tās ir algoritmiskas idejas.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kad labāk izmantot ciklu, nevis kopēt vienu darbību 100 reizes?
- Vai nosacījumam vienmēr ir tikai JĀ/NĒ?
- Kas notiek, ja cikla nosacījums nekad nekļūst nepatiess?

Demonstrācija / tāfeles piemērs

Kopīgi izveido pseidokodu “ventilators ieslēdzas virs 30 °C un darbojas, kamēr temperatūra nav zem 27 °C”.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.3. Nodarbība 3 — Blokhēmas

Galvenā doma: Blokhēma ir vadības plūsmas modelis, kas ļauj redzēt algoritmu pirms sintakses un pirms programmas izpildes.

Ko tieši stāstīt auditorijai

Parādi simbolus: sākums/beigas, process, ievade/izvade, lēmums, plūsmas bultas. Neļauj simbolus uztvert kā dekorāciju — katram ir semantiska nozīme.

Demonstrē, kā vienu un to pašu pseidokodu pārveidot blokhēmā. Pēc tam “izspēlē” shēmu ar testa vērtībām kā dators.

Ievies trace jēdzienu: izvēlamies konkrētu ievadi un soli pa solim pierakstām, pa kuru ceļu algoritms iet.

Uzsvēr, ka laba blokhēma atklāj nepilnības: neatgrieztas cilpas, nepārbaudītus stāvokļus, neesošu stop nosacījumu.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kur shēmā rodas lēmums?
- Ko nozīmē bulta, kas atgriežas iepriekšējā blokā?

- Kā pārbaudīt shēmu bez programmas?

Demonstrācija / tāfeles piemērs

Uzzīmē motoru aizsardzības blokshēmu ar temperatūras robežu un testa datiem 65, 79, 80, 81 °C.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.4. Nodarbība 4 — Mainīgie un datu tipi

Galvenā doma: Mainīgais ir nosaukts datu elements; datu tips nosaka, kādu informāciju glabājam un kādas darbības ar to ir jēgpilnas.

Ko tieši stāstīt auditorijai

Sāc ar sensora mērījumu: “temperatūra = 24,6”. Pajautā, kas šajā pierakstā ir nosaukums, vērtība, mērvienība un tips.

Izskaidro integer, float, Boolean un string, tehniskajiem piemēriem izmantojot detaļu skaitu, temperatūru, drošības stāvokli un iekārtas ID.

Atšķir datu tipu no mērvienības. Float nepasaka, vai skaitlis ir °C, bar vai mm. Profesionālā sistēmā mērvienība ir daļa no datu nozīmes.

Ievies validāciju: ne katrs skaitlis ir ticams mērījums. Ja temperatūras sensors dod 900 °C telpā, kodam nevajag akli ticēt.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Vai 1 un True ir viens un tas pats?
- Kāpēc “25.0” kā teksts nav tas pats, kas 25.0 kā skaitlis?

- Kas ir nederīgs mērijums?

Demonstrācija / tāfeles piemērs

Dod četru sensoru vērtības un lūdz noteikt piemērotu datu tipu un validācijas robežu.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.5. Nodarbība 5 — Loģika un nosacījumi

Galvenā doma: Boolean loģika ļauj no vairākiem stāvokļiem izveidot drošu lēmumu.

Ko tieši stāstīt auditorijai

Atkārto salīdzināšanas operatorus un uzreiz saisti tos ar fizisku nozīmi: >, <, >=, <=, ==, !=.

AND skaidro kā “visi obligātie nosacījumi”, OR — “vismaz viens derīgs ceļš”, NOT — stāvokļa inversija.

Izveido drošības interlock piemēru: motors drīkst startēt tikai tad, ja durvis aizvērtas AND avārijas stop nav aktīvs AND drošības relejs OK.

Parādi patiesības tabulu un pēc tam prasi auditorijai pašiem atrast kombināciju, kurā starts jābloķē.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc OR drošības ķēdē var būt bīstams, ja izmantots nepareizā vietā?
- Ko nozīmē fail-safe loģika?
- Kā testēt visus nosacījuma zarus?

Demonstrācija / tāfeles piemērs

Kopīgi izveido interlock izteiksmi trīs sensoriem un 8 iespējamo kombināciju tabulu.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.6. Nodarbība 6 — Cikli un atkārtota sistēmas darbība

Galvenā doma: Cikli modelē atkārtotu darbu; tehniskā sistēmā svarīgi ir arī apstāšanās nosacījums, laiks un atteices gadījumi.

Ko tieši stāstīt auditorijai

FOR skaidro ar zināmu mērījumu skaitu: paņemt 10 temperatūras paraugus. WHILE — ar procesu, kura beigas nosaka stāvoklis: pildīt tvertni, kamēr līmenis nav sasniegts.

Ievies bezgalīga cikla risku. Tehniskā procesā sensors var sabojāties, tāpēc “gaidīt kamēr sensors = TRUE” bez timeout var būt bīstami.

Paskaidro timeout un watchdog kā aizsardzības mehānismus. Watchdog nav “gudrs” — tas tikai secina, ka programma vairs neuzvedas gaidīti.

Demonstrē cikla skaitītāju, stāvokļa maiņu un žurnāla ierakstu katrā iterācijā.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kad FOR ir dabiskāks par WHILE?
- Ko darīt, ja sensors nekad nedod mērķa stāvokli?
- Vai katram ciklam vajag delay?

Demonstrācija / tāfeles piemērs

Uz tāfeles izveido tvertnes pildīšanas algoritmu ar maksimums 30 s timeout un avārijas stāvokli.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.7. Nodarbība 7 — Dekompozīcija, moduļi un sistēmas domāšana

Galvenā doma: Sarežģītu sistēmu neveido kā vienu milzīgu algoritmu; to sadala funkcijās ar skaidru atbildību un interfeisiem.

Ko tieši stāstīt auditorijai

Parādi lielu problēmu “autonoms noliktavas robots” un pajautā, vai iespējams to uzreiz ieprogrammēt. Pēc tam sadali uztverē, lokalizācijā, plānošanā, kustībā un drošībā.

Ievies “viena atbildība” principu: modulim jābūt saprotamam un testējamam atsevišķi.

Ievades/izejas definē kā līgumu. Sensoru modulis izvada attālumu; kustības modulis saņem mērķa ātrumu; žurnāls saņem notikumu.

Saisti dekompozīciju ar vēlākām funkcijām, klasēm, API, ROS 2 nodes un Functional Tree.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kā saprast, ka modulis ir pārāk liels?
- Kas notiek, ja divi moduļi slepeni maina vienus un tos pašus datus?
- Kāpēc interfeiss ir tikpat svarīgs kā algoritms?

Demonstrācija / tāfeles piemērs

Kopīgi izveido Functional Tree viedai temperatūras kontroles sistēmai.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.8. Nodarbība 8 — Pārbaudes darbs: algoritmi

Galvenā doma: Pārbaudes darba mērķis ir novērtēt domāšanu un izsekojamību, nevis tikai terminu iegaumēšanu.

Ko tieši stāstīt auditorijai

Pirms darba skaidri nosauc četras daļas un punktu sadalījumu. Uzsvēr, ka blokshēmā vērtē plūsmas loģiku, nevis māksliniecisku noformējumu.

Atgādinī, ka testa dati ir daļa no risinājuma. Pareiza shēma bez pārbaudes var slēpt robežgadījumu kļūdu.

Darba laikā nepalīdzi ar risinājumu, bet drīkstī precizēt uzdevuma formulējumu, ja jautājums ir par teksta nozīmi.

Pēc darba veic 10 min refleksiju: kurā vietā bija jādomā visvairāk un kā kļūdu varētu atklāt agrāk.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kā pierādīt, ka algoritms strādā?
- Kāpēc viens veiksmīgs piemērs nav pietiekams?

Demonstrācija / tāfeles piemērs

Pēc iesniegšanas parādi vienu anonimizētu kļūdainu zarošanās piemēru un kopā atrod kļūdu.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.9. Nodarbība 9 — Python vide, programmas izpilde un datu plūsma

Galvenā doma: Python ir veids, kā iepriekš apgūto algoritmu padarīt izpildāmu; programmēšanas valoda nav algoritma aizvietotājs.

Ko tieši stāstīt auditorijai

Sāc ar vienkāršu print un input, bet uzreiz izseko datu plūsmu: lietotājs → string → pārveidošana → aprēķins → rezultāts.

Paskaidro interpretētās izpildes ideju bez pārspīlētas teorijas. Svarīgi ir saprast, ka kļūda parādās konkrētā izpildes vietā.

Parādi type() un apzināti izveido kļūdu, mēģinot saskaitīt tekstu ar skaitli. Izmanto traceback kā diagnostikas informāciju.

Tehniskais piemērs: ievadīt mērījumu, pārbaudīt diapazonu un aprēķināt novirzi no setpoint.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc input() rezultāts ir teksts?
- Ko dod mainīgā nosaukums?
- Kā mēs varam pierādīt, ka pārveidošana notika pareizi?

Demonstrācija / tāfeles piemērs

Uzraksti 10 rindu temperatūras novirzes programmu un kopā izsekojiet katra mainīgā vērtību.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.10. Nodarbība 10 — Python nosacījumi, Boolean loģika un datu validācija

Galvenā doma: Nosacījumi pārvērš mērījumu lēmumā; profesionālā sistēmā pirms lēmuma jānovērtē arī datu ticamība.

Ko tieši stāstīt auditorijai

Parādi if/elif/else kā to pašu zarošanos no blokshēmām. Uzsver atkāpes sintaktisko nozīmi Python.

Ievies validācijas secību: vai dati pārveidojami → vai diapazonā → tikai tad procesa loģika.

Parādi “normal / warning / alarm” klasifikāciju un robežvērtību testus tieši uz robežas.

Izskaidro, ka pārāk sarežģīts nosacījums ir signāls dekompozīcijai: nosauc Boolean apakšnosacījumus.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāda atšķirība starp ≥ 30 un > 30 ?
- Kāpēc robežvērtības jātestē atsevišķi?
- Ko darīt ar None vai nederīgu tekstu?

Demonstrācija / tāfeles piemērs

Izveido temperatūras stāvokļa klasifikatoru ar validāciju un četriem testa gadījumiem.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.11. Nodarbība 11 — Python cikli, datu kopas, debugging un testēšanas domāšana

Galvenā doma: Cikls kļūst profesionāls tad, kad mēs kontrolējam stāvokli, kļūdas un pārbaudes rezultātus, nevis tikai atkārtojam kodu.

Ko tieši stāstīt auditorijai

Ar sarakstu modelē 10 sensora mērījumus. FOR ciklā aprēķini min, max, sum un vidējo vērtību.

Pēc tam apzināti ievieto kļūdu: nepareizs skaitītājs, nepareizi inicializēta summa vai nepareiza robeža.

Māci debugging secību: reproducē → lokalizē → izvirzi hipotēzi → maini vienu lietu → atkārtoti testu.

Paskaidro, ka print-debugging ir sākums, bet vēlāk izmantos debugger, logus un unit testus.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Athbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc “it works on my data” nav testēšana?
- Kas ir robežgadījums?
- Kā saprast, vai kļūda ir datos vai algoritmā?

Demonstrācija / tāfeles piemērs

Dod sabojātu vidējās vērtības programmu ar 3 kļūdām un lūdz sistemātiski tās atrast.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.12. Nodarbība 12 — Python mini projekts — no prasības līdz demonstrācijai

Galvenā doma: Mini projekts noslēdz pirmo loku: prasība → algoritms → kods → tests → demonstrācija.

Ko tieši stāstīt auditorijai

Pirms programmēšanas prasi vienu teikumu “Ko sistēma dara?” un 3–5 funkcionālās prasības.

Prasi uzrakstīt pseidokodu vai blokshēmu. Ja audzēknis uzreiz sāk kodēt, atgriez viņu pie prasības un testa.

Darba laikā akcentē mazu iterāciju: vispirms strādājošs minimums, pēc tam validācija un paplašinājumi.

Aizstāvēšanā vērtē spēju izskaidrot kodu, nevis tikai skaistu rezultātu.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāds ir minimālais strādājošais risinājums?
- Kādi 3 testi pierāda, ka projekts dara prasīto?
- Ko tu mainītu otrajā versijā?

Demonstrācija / tāfeles piemērs

Demonstrē projekta README minimumu: mērķis, palaišana, ievades, piemērs, testi.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.13. Nodarbība 13 — C++ programmas struktūra, kompilācija un pirmā programma

Galvenā doma: C++ realizē to pašu algoritmisko domāšanu, bet prasa precīzāk definēt tipus un programmas struktūru.

Ko tieši stāstīt auditorijai

Salīdzini Python faila palaišanu ar C++ ceļu source → compile → link → executable.

Izskaidro #include, main(), semikolu un cout. Neiedziļinies visos valodas izņēmumos pirmajā stundā.

Apzināti parādi kompilācijas kļūdu un izskaidro, ka kompilators ir agrīns kvalitātes filtrs.

Pārnes vienu Python temperatūras aprēķinu uz C++ un salīdzini tikai sintaksi, algoritmu atstājot nemainīgu.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Ko kompilators zina pirms programmas palaišanas?
- Kāpēc C++ jānorāda tips?
- Kāda ir atšķirība starp compile-time un runtime kļūdu?

Demonstrācija / tāfeles piemērs

Pārnes 8–10 rindu Python programmu uz C++ un atzīmē, kur nepieciešama papildu informācija.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.14. Nodarbība 14 — C++ datu tipi, operatori, const un precizitāte

Galvenā doma: C++ tips ir ligums par datu interpretāciju; const palīdz izteikt to, kam programmas laikā nav jāklūst citādam.

Ko tieši stāstīt auditorijai

Salīdzini int, float, double, bool un string ar iepriekšējo Python pieredzi.

Demonstrē integer division un precizitātes pārsteigumu. Saisti to ar mērījumu datiem un aprēķinu uzticamību.

const skaidro ar fizisku parametru: riteņa diametrs, konfigurācijas koeficients, maksimālā atļautā temperatūra.

Uzsvēr vienības komentāros/nosaukumos, piemēram, distance_m vai temperature_c.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kad int ir nepareiza izvēle?
- Kāpēc 1/2 var nedot 0.5?
- Ko const pasaka nākamajam programmētājam?

Demonstrācija / tāfeles piemērs

Aprēķini ceļu no enkodera impulsiem, izmantojot const riteņa diametru un impulsus uz apgriezianu.

Tipiskie pārpratumi, ko koriģēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.15. Nodarbība 15 — C++ zarošanās, switch un sistēmas stāvokļi

Galvenā doma: C++ nosacījumi nav jauna loģika; būtiska jaunā ideja ir skaidri modelēt sistēmas stāvokļus.

Ko tieši stāstīt auditorijai

Atkārtoti if/else un pēc tam parādi switch komandām vai režīmiem.

Ievies enum class stāvokļiem IDLE, RUNNING, ALARM, STOPPED. Tas ir drošāks modelis par “magic numbers”.

Parādi, ka state machine pāreju nosaka notikums un nosacījums, nevis nejauša koda secība.

Tehniskais piemērs: iekārtas Start/Stop/Reset ar drošības interlock.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Athbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc bool motorOn nav pietiekams sarežģītai iekārtai?
- Kas ir neatļauta stāvokļu pāreja?
- Kā testēt state machine?

Demonstrācija / tāfeles piemērs

Uzzīmē četru stāvokļu diagrammu un pēc tam realizē pāreju tabulu pseidokodā.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.16. Nodarbība 16 — C++ cikli, izvēlnes un atkārtota procesa vadība

Galvenā doma: Ciklu mērķis nav vienkārši atkārtot; tie uztur sistēmas dzīves ciklu un apstrādā notikumus.

Ko tieši stāstīt auditorijai

Parādi while kā vienkāršu vadības cilpu un for mērījumu datu apstrādei.

Izveido termināļa izvēlni Start/Stop/Status/Exit un sasaisti ar stāvokļa mainīgo.

Uzsvēr, ka īstā iegultā sistēmā bloķējošs while vai delay var traucēt citiem uzdevumiem.

Ievies polling un event-driven atšķirību tikai konceptuāli.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kas notiek ar citiem sensoriem, ja kods 5 sekundes “gul”?
- Kad izvēlne ir state machine?
- Kā ciklam pievienot drošu iziešanu?

Demonstrācija / tāfeles piemērs

Demonstrē vadības cilpu ar komandu nolasīšanu un timeout statusa izvadīšanai.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.17. Nodarbība 17 — Funkcijas, parametri un programmas dekompozīcija

Galvenā doma: Funkcija padara dekompozīciju izpildāmu un testējamu.

Ko tieši stāstīt auditorijai

Atgriezies pie Functional Tree un parādi, kā viena apakšfunkcija kļūst par C++ funkciju.

Izskaidro input parametru, return vērtību un lokālo mainīgo. Piemēri: isTemperatureValid(), calculateDistance(), setMotorSpeed().

Pārrunā pass-by-value un tikai intuitīvi iepazīstini ar reference, ja nepieciešams mainīt objektu.

Uzsvēr funkcijas nosaukumu kā dokumentāciju: nosaukumam jāpasaka, ko tā dara.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kādu funkciju ir grūti testēt?
- Cik lielai jābūt funkcijai?
- Kāpēc globālie mainīgie sarežģī testēšanu?

Demonstrācija / tāfeles piemērs

Sadaliet vienu 40 rindu programmu 4 nosauktās funkcijās un izveidojiet katrai vienu testu.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.18. Nodarbība 18 — Masīvi, mērījumu datu apstrāde un algoritmi

Galvenā doma: Datu kopa ļauj analizēt procesa uzvedību laikā, ne tikai vienu mirkļa vērtību.

Ko tieši stāstīt auditorijai

Parādi masīvu kā fiksētas struktūras datu kopu un salīdzini ar Python list.

Implementē min, max, average ar ciklu, nepaļaujoties uz gatavu funkciju, lai redz algoritmu.

Ievies out-of-bounds risku un indeksus 0..N-1.

Saisti ar slidoša loga ideju: pēdējie N mērījumi trokšņa mazināšanai.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc pēdējais indekss ir N-1?
- Kas notiek, ja rakstām ārpus masīva robežām?
- Kā datu logs palīdz sensoru filtrēšanā?

Demonstrācija / tāfeles piemērs

Aprēķini 10 temperatūras mērījumu vidējo, min, max un alarm_count.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.19. Nodarbība 19 — OOP ievads: class, object un tehniska komponente

Galvenā doma: OOP palīdz vienā vietā apvienot komponentes datus un darbības, īpaši sistēmās ar vairākiem līdzīgiem sensoriem vai aktuatoriem.

Ko tieši stāstīt auditorijai

Sāc ar “Sensor” kā reālu komponenti: tam ir ID, pēdējā vērtība, derīguma stāvoklis un read/validate metodes.

Izskaidro class kā tipu un object kā konkrētu instanci. Nevajag pirmajā stundā mācīt visu mantojumu/polimorfismu.

Encapsulation skaidro kā iespēju paslēpt nekorektu stāvokļu veidošanu.

Parādi divus Sensor objektus ar vienādu klasi, bet atšķirīgiem datiem.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc neglabāt visu atsevišķos globālos mainīgajos?
- Kas ir objekta stāvoklis?
- Kā klase palīdz testēt sensoru validāciju?

Demonstrācija / tāfeles piemērs

Izveido vienkāršu Sensor klasi ar name, value, valid un metodi update().

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.20. Nodarbība 20 — C++ pārbaudes darbs un debugging

Galvenā doma: Šajā posmā vērtē spēju lasīt, labot un strukturēt C++ kodu, nevis tikai rakstīt no nulles.

Ko tieši stāstīt auditorijai

Pirms darba atgādināni atšķirību starp kompilācijas, runtime un loģikas kļūdu.

Debugging uzdevumā prasi studentam pierakstīt ne tikai labojumu, bet arī simptomu un cēloni.

Vērtē arī to, vai risinājums saglabā prasības un testus.

Pēc darba izrunā vienu kļūdu, ko kompilators neatradis, piemēram, nepareizu robežnosacījumu.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kura kļūda ir visbīstamākā — sintakses vai loģikas?
- Kā testēšana palīdz, ja programma kompilējas?

Demonstrācija / tāfeles piemērs

Dod 15–20 rindu programmu ar sintakses, tipa un loģikas kļūdu maisījumu.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.21. Nodarbība 21 — SQL pamati tehnisko mērījumu datiem

Galvenā doma: Datubāze saglabā procesa vēsturi strukturēti, lai vēlāk varētu pierādīt, analizēt un salīdzināt notikumus.

Ko tieši stāstīt auditorijai

Sāc ar jautājumu: “Kur paliek sensora dati pēc programmas aizvēršanas?” Pārej uz persistent storage.

Izskaidro tabulu, rindu, kolonnu, primāro atslēgu un datu tipu.

INSERT ir jauns notikums/mērījums, SELECT — jautājums datiem, WHERE — filtrs, ORDER BY — secība.

Tehniskā tabula measurements: id, timestamp, sensor_id, value, unit, quality.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc timestamp ir obligāts tehniskiem datiem?
- Kāpēc sensor_id labāk par vienkāršu “temperature”?
- Kas notiek, ja vienība nav saglabāta?

Demonstrācija / tāfeles piemērs

Izveido 10 mērījumus un ar SELECT atrodi visus alarm līmeņa ierakstus.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.22. Nodarbība 22 — CSV, JSON un Python + SQLite

Galvenā doma: CSV, JSON un SQL nav konkurenti; tie ir dažādi datu glabāšanas un apmaiņas līmeņi.

Ko tieši stāstīt auditorijai

Salīdzini CSV kā vienkāršu tabulāru failu, JSON kā strukturētu ziņojumu un SQLite kā vaicājamu datubāzi.

Parādi Python sqlite3 savienojumu, parameterized INSERT un SELECT.

Uzsvēr, ka SQL vaicājumus nevajag būvēt ar nekontrolētu string salikšanu.

Demonstrē vienu sensora JSON ziņojumu un tā saglabāšanu datubāzē.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc JSON ir ērts IoT ziņojumiem?
- Kad CSV ir pietiekams?
- Kāpēc datubāzei vajag shēmu?

Demonstrācija / tāfeles piemērs

Python skripts nolasa JSON mērījumu, validē laukus un ieraksta SQLite.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.23. Nodarbība 23 — IoT, MQTT, HTTP un mākoņdatu arhitektūra

Galvenā doma: IoT ir datu un komandu ceļš starp fizisku ierīci un programmatūras pakalpojumiem; protokols ir tikai viena daļa no sistēmas.

Ko tieši stāstīt auditorijai

Uzzīmē device → network → broker/API → service → database → dashboard.

MQTT skaidro ar publisher/topic/subscriber un broker. HTTP — ar request/response un endpoint.

Ievies QoS, offline un buffering ideju: internets nav garantēti pieejams.

Drošības minimums: autentifikācija, šifrēts transports, slepeno atslēgu neglabāšana publiskā kodā.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kas notiek, ja Wi-Fi pazūd uz 10 minūtēm?
- Kad MQTT ir ērtāks par HTTP?
- Kur glabāt ierīces identitāti?

Demonstrācija / tāfeles piemērs

Izspēlē MQTT datu plūsmu uz tāfeles ar trim topic: telemetry, status, command.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.24. Nodarbība 24 — Sistēmu integrācijas mini darbs — sensors → kods → DB → dashboard

Galvenā doma: Integrācija pārbauda ne tikai katru komponenti, bet līgumus starp komponentēm.

Ko tieši stāstīt auditorijai

Pirms darba definē datu līgumu: obligātie lauki, tipi, vienības, timestamp un kļūdu stāvokļi.

Sadaliet sistēmu trīs daļās: datu ģenerators, glabāšana, vizualizācija.

Testējiet “laimīgo ceļu” un vismaz vienu atteici: nederīgs JSON, DB kļūda, pazudis lauks.

Uzsvēr traceability: prasība → komponente → tests → pierādījums.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kur integrācijas kļūda visbiežāk slēpjas?
- Ko nozīmē backward compatibility datu formātam?
- Kā saglabāt neapstrādātus datus?

Demonstrācija / tāfeles piemērs

Mini projekta demo: Python ģenerē mērījumu, SQLite saglabā, matplotlib parāda laika grafiku.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.25. Nodarbība 25 — Problēma, lietotājs un prasības

Galvenā doma: Labs projekts sākas ar problēmas formulējumu un pārbaudāmām prasībām, nevis ar izvēlēto tehnoloģiju.

Ko tieši stāstīt auditorijai

Aizliedz pirmajās 20 minūtēs izvēlēties bibliotēku vai detaļu. Vispirms: lietotājs, problēma, robežas, success criteria.

Funkcionāla prasība pasaka, ko sistēma dara; nefunkcionāla — cik ātri, precīzi, droši, uzticami.

Prasībai jābūt testējamai. “Sistēmai jābūt labai” nav prasība.

Noslēgumā katrai prasībai pievieno verifikācijas metodi: test, inspection, analysis, demonstration.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Athbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kā pierādīsi, ka prasība izpildīta?
- Vai prasība satur slēptu risinājumu?
- Kas ir ārpus projekta robežām?

Demonstrācija / tāfeles piemērs

Pārveido trīs sliktas prasības par testējamām prasībām.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.26. Nodarbība 26 — Functional Tree, arhitektūra un testu stratēģija

Galvenā doma: Functional Tree atbild uz jautājumu “ko sistēmai jādara?”, pirms mēs nosakām, ar kādu detaļu vai klasi to realizēt.

Ko tieši stāstīt auditorijai

Sāc ar projekta top-level funkciju. Sadaliet 4–7 galvenajās funkcijās un tālāk līdz testējamām apakšfunkcijām.

Tikai pēc funkcijām veido fizisko/programmatūras arhitektūru: sensors, controller, DB, UI.

Katram lapas mezglam pajautā: kā mēs pierādīsim, ka šī funkcija strādā?

Izveido sākotnējo traceability matricu prasība <-> funkcija <-> komponente <-> tests.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Athbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Vai Functional Tree drīkst saturēt konkrētu detaļas modeli?
- Kad dekompozīcija ir pietiekami dziļa?
- Kā koks palīdz sadalīt komandas darbu?

Demonstrācija / tāfeles piemērs

Izveido top-level koku “To monitor a room and generate an alarm when environmental limits are exceeded”.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.27. Nodarbība 27 — Python realizācija I — kodols

Galvenā doma: Pirmais projekta programmatūras mērķis ir mazs, testējams kodols, nevis pilna gala sistēma.

Ko tieši stāstīt auditorijai

Izvēlies vienu kritisko datu plūsmu un realizē to no ievades līdz rezultātam.

Funkcijas veido pēc Functional Tree. Katras funkcijas input/output jābūt saprotamam.

Ievies konfigurāciju atsevišķi no loģikas: robežas, ceļi, ID nav jāizkaisa pa kodu.

Pēc katras jaunas funkcijas izpildi vismaz vienu testu.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kas ir projekta “thin slice”?
- Ko kodā var aizstāt ar simulētu sensoru?
- Kāpēc konfigurācija nav biznesa loģika?

Demonstrācija / tāfeles piemērs

Pasniedzējs demonstrē “fake sensor” funkciju, kas ļauj testēt bez aparatūras.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.28. Nodarbība 28 — Python realizācija II — dati, testi un vizualizācija

Galvenā doma: Otrais slānis padara kodolu novērojamu: dati tiek saglabāti, testēti un parādīti.

Ko tieši stāstīt auditorijai

Ievies logging ar timestamp un stāvokli. Žurnāls nav tikai print.

Saglabā mērījumus CSV vai SQLite un izveido vienu grafiku, kas palīdz novērtēt sistēmas uzvedību.

Raksti testus robežvērtībām un nederīgiem datiem.

Paskaidro, ka vizualizācija nav tikai skaistums — tā ir diagnostikas un verifikācijas metode.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Ko grafiks atklāj, ko nevar redzēt vienā skaitlī?
- Kāda ir atšķirība starp log un measurement data?
- Ko obligāti pievienot timestamp?

Demonstrācija / tāfeles piemērs

Parādi testa datu sēriju ar vienu anomāliju un prasi to atrast grafikā.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.29. Nodarbība 29 — C++ realizācija I — algoritma pārnese vai low-level komponente

Galvenā doma: Pārnēsot no Python uz C++, svarīgi saglabāt funkcionalitāti un testus, nevis mehāniski pārrakstīt sintaksi.

Ko tieši stāstīt auditorijai

Izvēlies komponenti, kur C++ dod jēgu: mikrokontroleris, motoru vadība, ātra apstrāde vai stingri tipi.

Fiksē I/O līgumu pirms pārnese. Ja Python un C++ ievade vienāda, rezultātam jābūt salīdzināmam.

Izmanto const, enum un funkcijas, lai kods atspoguļotu sistēmas modeli.

Salīdzini testu rezultātus abās valodās.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc ne visu projektu rakstīt C++?
- Kā pierādīt, ka pārnese nav mainījusi algoritmu?
- Kur C++ stingrā tipēšana palīdz?

Demonstrācija / tāfeles piemērs

Izvēlies vienu Python funkciju un izveido C++ ekvivalentu ar identisku testu tabulu.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.30. Nodarbība 30 — C++ realizācija II — funkcijas/OOP un testi

Galvenā doma: C++ projekta otrajā iterācijā strukturējam komponentes un ieviešam testējamus interfeisus.

Ko tieši stāstīt auditorijai

Komponentēm ar stāvokli izmanto klasi: MotorController, Encoder, SensorFilter.

Nošķir hardware access no decision logic, lai lēmumu var testēt arī bez īstas aparatūras.

Pievieno kļūdu stāvokļus un drošus defaultus.

Veic code review pēc nosaukumiem, konstantes, robežpārbaudēm un atbildībām.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kāpēc hardware abstraction ir noderīga?
- Kas jānotiek, ja sensors nav pieejams?
- Ko nozīmē safe default?

Demonstrācija / tāfeles piemērs

Pārveido “motor speed = sensor reading” uz divām klasēm ar validācijas slāni.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.31. Nodarbība 31 — IoT/robotikas/sensora integrācija

Galvenā doma: Šajā etapā programmatūra pirmo reizi sastop reālu laiku, troksni, savienojuma kļūdas un fiziskus ierobežojumus.

Ko tieši stāstīt auditorijai

Pirms savienošanas definē voltāžas, interfeisus, datu ātrumu un kopīgu GND.

Sensoram pievieno statusu: valid/invalid/stale, ne tikai value.

IoT plūsmā definē device_id, timestamp un message version.

Robotikā pārbaudi frame un units: m pret mm, deg pret rad ir klasiskas integrācijas kļūdas.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kas notiek, ja sensora dati ir 2 sekundes veci?
- Kur glabāt kalibrācijas koeficientu?
- Kā noteikt, vai kļūda ir aparatūrā vai kodā?

Demonstrācija / tāfeles piemērs

Demonstrē sensora “health” lauku kopā ar mērījumu un timestamp.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.32. Nodarbība 32 — Testēšana, validācija un atteices scenāriji

Galvenā doma: Inženiertehniski pabeigts projekts nav tas, kas vienreiz nostrādā; tas ir risinājums ar pierādāmu uzvedību normālos un kļūdu apstākļos.

Ko tieši stāstīt auditorijai

Atšķir verification — vai uzbūvējām atbilstoši prasībai — no validation — vai risinām pareizo problēmu.

Veido testu matricu normal, boundary, invalid, timeout, communication loss, restart.

Atteices scenārijam definē detect → contain → recover → log.

Prasi Actual un evidence, ne tikai PASS/FAIL.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Ko nozīmē “test passed” bez pierādījuma?
- Kas notiek pēc restart?
- Kā sistēmai jāuzvedas pie sakaru zuduma?

Demonstrācija / tāfeles piemērs

Kopā izveido 6 testa gadījumus vienam sensoru/aktuatora moduļim.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.33. Nodarbība 33 — Dokumentācija un reproducējamība

Galvenā doma: Dokumentācija ļauj citam cilvēkam atkārtot, pārbaudīt un uzturēt risinājumu.

Ko tieši stāstīt auditorijai

README minimums: problēma, arhitektūra, prasības, palaišana, konfigurācija, testi, zināmie ierobežojumi.

Versijas: atzīmē izmantoto kodu, bibliotēkas, aparatūras revīziju un datu formātu.

Pievieno pieslēgumu tabulu, nevis paļaujies uz fotogrāfiju vien.

Paskaidro, ka reproducējamība ir profesionālās uzticamības pazīme.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Vai cits students var projektu palaist bez autora klātbūtnes?
- Kuru informāciju obligāti vajag pēc sešiem mēnešiem?
- Kā dokumentēt kalibrāciju?

Demonstrācija / tāfeles piemērs

Dod sliktu README un liec auditorijai nosaukt trūkstošos elementus.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.34. Nodarbība 34 — Pilnveidošana un tehniskais review

Galvenā doma: Review mērķis nav atrast vainīgo; tas ir sistemātiski atrast riskus pirms gala demonstrācijas.

Ko tieši stāstīt auditorijai

Pārbaudi traceability: vai katrai prasībai ir risinājums un tests.

Pārbaudi arhitektūru: vai nav vienas komponentes ar pārāk daudz atbildībām.

Pārbaudi kodu: hard-coded vērtības, nekontrolēti exception, neizmantotas funkcijas, dubults kods.

Pārbaudi demonstrāciju: vai ir drošs fallback, ja aparatūra uzvedas citādi.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kādu risku tu labotu vispirms?
- Ko var izņemt, lai risinājums kļūtu vienkāršāks?
- Kāds ir projekta tehniskais parāds?

Demonstrācija / tāfeles piemērs

Veic 15 min “peer review” ar vienotu kontrolsarakstu.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.35. Nodarbība 35 — Aizstāvēšana I — demonstrācija

Galvenā doma: Tehniska demonstrācija pierāda funkciju ar iepriekš definētiem kritērijiem, nevis tikai rāda ekrānu.

Ko tieši stāstīt auditorijai

Sāc ar problēmu un vienu arhitektūras shēmu, pēc tam demonstrē 2–3 galvenās prasības.

Parādi testa pierādījumu, ne tikai kodu.

Ja demonstrācija neizdodas, studentam jāprot diagnosticēt un paskaidrot, ko pārbaudītu pirmo.

Jautā par trade-off: kāpēc izvēlēts šis sensors, protokols vai algoritms.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Kura prasība ir vissvarīgākā?
- Kāds bija grūtākais atteices scenārijs?
- Ko pierāda tavs tests?

Demonstrācija / tāfeles piemērs

Izmanto 5 min tehniskās prezentācijas struktūru: problem → design → demo → evidence → limitations.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

37.36. Nodarbība 36 — Aizstāvēšana II, refleksija un profesionālās attīstības ceļš

Galvenā doma: Noslēgums sasaista projekta pieredzi ar nākamo profesionālās izaugsmes soli.

Ko tieši stāstīt auditorijai

Pēc aizstāvēšanas prasi refleksiju trīs līmeņos: tehniskā prasme, darba metode, nākamais izaicinājums.

Parādi, kā tas pats pamats turpinās automatizācijā, embedded, IoT, robotikā, datu inženierijā un universitātes studijās.

Audzēknim jāizvēlas viens portfolio artefakts un jāpaskaidro, ko tas pierāda.

Noslēdz ar domu: valodas mainīsies, bet prasību, algoritmu, sistēmu, testēšanas un drošības domāšana paliek.

90 minūšu ritms

Laiks	Pasniedzēja darbība	Audzēkņu darbība
0–10 min	Aktivizē iepriekšējās zināšanas, uzdod ievadjautājumu un parādi tehnisko problēmu.	Atbild, formulē sākotnējo pieņēmumu.
10–30 min	Izskaidro pamatjēdzienus un terminus ar vienu tehnisku piemēru.	Pieraksta jēdzienus, uzdod precizējošus jautājumus.
30–50 min	Veic demonstrāciju vai kopīgu algoritma/koda analīzi.	Prognozē rezultātu un seko izpildes soļiem.
50–75 min	Dod vadītu vai patstāvīgu uzdevumu; konsultē, nepasakot risinājumu.	Izstrādā risinājumu un testē.
75–85 min	Kopīgi analizē 1–2 risinājumus un tipisku kļūdu.	Salīdzina pieejas, labo kļūdas.
85–90 min	Noslēdz ar 3 galvenajām atziņām un nākamās stundas saikni.	Formulē vienu apgūto ideju un vienu jautājumu.

Jautājumi auditorijai

- Ko tu tagad proti izskaidrot, ko gada sākumā neprati?
- Kuru tēmu turpinātu 20 stundas dziļāk?
- Kādu projektu tu varētu uzbūvēt nākamajā līmenī?

Demonstrācija / tāfeles piemērs

Izveido individuālu 3 mēnešu turpmākās mācīšanās ceļu ar vienu praktisku projektu.

Tipiskie pārpratumi, ko korigēt

- Audzēknis mēģina atcerēties sintaksi, nepaskaidrojot algoritmu. Atgriez pie ievades, stāvokļa, lēmuma un rezultāta.
- Tiek pārbaudīts tikai viens “pareizais” piemērs. Prasi normālu, robežvērtības un kļūdas gadījumu.
- Tehniskais termins tiek lietots bez vienības, datu avota vai konteksta. Prasi precizējumu.

Stundas noslēguma frāze

“Šodienas tēma nav atsevišķa komanda vai formula — tā ir viena daļa no ķēdes problēma → modelis → realizācija → tests → pierādījums.”

38. Fiziskās laboratorijas: Arduino, ESP32, sensori, motori un robots

Šis bloks paredzēts pārejai no simulācijas uz fizisku sistēmu. Pieslēgumu piemēri ir mācību konfigurācijas, nevis universāli pinouti visām moduļu revīzijām. Pirms laboratorijas pasniezējs pārbauda konkrētās plates un breakout moduļa dokumentāciju, barošanas spriegumu, loģikas līmeni un maksimālo strāvu. Motori netiek baroti no mikrokontrollera GPIO.

38.1. Drošības un darba organizācijas minimums

- Savienojumus veic ar atslēgtu barošanu. Pēc pieslēgšanas vizuāli pārbauda VCC/GND un tikai tad pieslēdz USB vai ārējo barošanas avotu.
- Pirmajās motoru laboratorijās izmanto laboratorijas barošanas avotu ar strāvas ierobežojumu vai drošu bateriju turētāju; LiPo nav obligāts mācību sākumam.
- Mikrokontrollera GPIO nedrīkst tieši darbināt DC motoru, releja spoli vai citu lielas strāvas slodzi. Izmanto draiveri/MOSFET un vajadzīgo aizsardzību.
- ESP32 klasiskās plates loģika ir 3,3 V; 5 V signālu uz GPIO nepievada bez atbilstošas līmeņa saskaņošanas.
- Motora barošanas un loģikas zemējums parasti ir kopīgs, ja draivera shēma to paredz; pārbauda izmantotā moduļa dokumentāciju.
- Pirms rotējošu detaļu testa robots tiek pacelts tā, lai riteņi nevarētu negaidīti aizbraukt no galda. Garus matus, vadus un brīvu apģērbu tur prom no kustīgām daļām.
- LiDAR/ToF optiskos sensorus izmanto atbilstoši ražotāja drošības norādēm; sensoru neatver un nepārveido optisko mezglu.

38.2. Ieteicamais bāzes laboratorijas komplekts

Komponente	Daudzums uz darba vietu	Loma kursā
Arduino UNO R3 vai ekvivalents ATmega328P mācību dēlis	1	Digitālā/analogā I/O, vienkārša C++/Arduino vide
ESP32-WROOM-32 bāzeta development board	1	Wi-Fi, I2C, PWM, IoT un mobilā robota kontrolleris
Breadboard, vadi, 220–330 Ω rezistori, 4.7–10 kΩ rezistori, LED, pogas	1 komplekts	Droši pamatpieslēgumi
10 kΩ potenciometrs	1	ADC un signālu skalēšana
VL53L0X breakout, kas ir saderīgs ar 3,3 V loģiku	1	Time-of-Flight attāluma mērījums
IMU breakout (piem., MPU-6050 saderīgs modulis)	1	Paātrinājums, leņķiskais ātrums, orientācijas filtrēšana

Komponente	Daudzums uz darba vietu	Loma kursā
TB6612FNG dual DC motor driver breakout	1	Divu DC motoru virziens un PWM
2 × DC gear motor ar kvadrātūras enkoderi	2	Odometrija un closed-loop ātruma vadība
2WD šasija + brīvais ritenis	1	Mobilā robota integrācija
RPLIDAR A1 ar ražotāja USB/UART adapteri	1 uz 2–4 darba vietām	2D LiDAR datu apstrāde un kartēšanas ievads
USB kamera vai datora kamera	1	Machine Vision/OpenCV
5–9 V laboratorijas barošanas avots ar strāvas ierobežošanu	1	Droša motoru barošana un diagnostika
Multimetrs	1	Sprieguma, nepārtrauktības un polaritātes pārbaude

38.3. H1 — Arduino UNO — digitālā ievade/izvade: LED un poga

Mērķis: Iemācīt active-low pogas loģiku, digitālu I/O un to, kā fiziskais signāls kļūst par Boolean stāvokli.

Aprīkojums

- Arduino UNO R3
- LED
- 220–330 Ω rezistors
- poga
- breadboard
- vadi

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
UNO D8	Rezistors → LED anods	LED vadības izeja
LED katods	GND	Atgriezes ceļš
UNO D2	Poga → GND	Digitālā ievade ar INPUT_PULLUP

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseidokoda kodols

```
const int LED_PIN = 8;
const int BUTTON_PIN = 2;

void setup() {
  pinMode(LED_PIN, OUTPUT);
```

```

pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  bool pressed = (digitalRead(BUTTON_PIN) == LOW);
  digitalWrite(LED_PIN, pressed ? HIGH : LOW);
}

```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Poga nav nospiesta	LED izslēgts		
Poga nospiesta	LED ieslēgts		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.4. H2 — Arduino UNO — analogais mērījums, ADC un seriālais žurnāls

Mērķis: Parādīt sampling, ADC koda vērtību, skalēšanu un to, ka mērījums nav tikai “skaitlis”, bet signāla interpretācija.

Aprīkojums

- Arduino UNO R3
- 10 kΩ potenciometrs
- breadboard
- vadi

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
UNO 5V	Potenciometra ārējā kāja	Barošana
UNO GND	Potenciometra otra ārējā kāja	Zeme
UNO A0	Potenciometra vidējā kāja	Analogā ievade

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseidokoda kodols

```
const int ADC_PIN = A0;

void setup() {
  Serial.begin(115200);
}

void loop() {
  int raw = analogRead(ADC_PIN);
  float voltage = raw * (5.0f / 1023.0f);
```

```

Serial.print(raw);
Serial.print(",");
Serial.println(voltage, 3);
delay(100);
}

```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Potenciometrs pie GND	ADC tuvu 0, spriegums tuvu 0 V		
Potenciometrs vidū	ADC aptuveni pusdiapazonā		
Potenciometrs pie 5 V	ADC tuvu pilnam diapazonam		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.5. H3 — ESP32 + VL53L0X — I2C attāluma sensors

Mērķis: I2C sensoru lasīšana, mērījuma kvalitāte, timeout un validācijas robežas. GPIO21/22 šeit ir kursa izvēlēta konvencija; ESP32 I2C pinus var konfigurēt.

Aprikojums

- ESP32-WROOM-32 development board
- VL53L0X breakout, pārbaudīts 3,3 V loģikai
- vadi

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
ESP32 3V3	VL53L0X VCC/VIN atbilstoši breakout specifikācijai	Barošana
ESP32 GND	VL53L0X GND	Zeme
ESP32 GPIO21	VL53L0X SDA	I2C dati — laboratorijas konvencija
ESP32 GPIO22	VL53L0X SCL	I2C pulkstenis — laboratorijas konvencija

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseudokoda kodols

```
// Bibliotēkas API ir atkarīgs no izmantotā VL53L0X breakout.  
// Pseudokods laboratorijai:  
setup_i2c(SDA=21, SCL=22);  
sensor.begin();
```

```

loop:
  distance_mm = sensor.read();
  if sensor.timeout():
    status = TIMEOUT;
  else if distance_mm < MIN_VALID or distance_mm > MAX_VALID:
    status = INVALID;
  else:
    status = VALID;
  publish(distance_mm, status);

```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Objekts 0.3–0.5 m priekšā	Stabils derīgs attālums		
Objekts izņemts ārpus diapazona	Invalid/timeout atbilstoši bibliotēkas rezultātam		
Sensors atvienots	Programma nedrīkst “uzkārties”; jāreģistrē kļūda		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.6. H4 — ESP32 + IMU — paātrinājums, žiroskops un drift

Mērķis: Atšķirt accelerometer un gyroscope mērījumus; parādīt bias, drift, kalibrācijas un filtra nepieciešamību.

Aprīkojums

- ESP32 development board
- IMU breakout (MPU-6050 saderīgs vai cits I2C IMU)
- vadi

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
ESP32 3V3	IMU VCC, ja breakout paredz 3,3 V	Barošana
ESP32 GND	IMU GND	Zeme
ESP32 GPIO21	IMU SDA	I2C dati
ESP32 GPIO22	IMU SCL	I2C pulkstenis

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseidokoda kodols

```
// Pseidokods neatkarīgs no konkrētās IMU bibliotēkas
read ax, ay, az, gx, gy, gz
gyro_angle += (gx - gyro_bias) * dt
acc_angle = atan2(ay, az) * 180.0 / PI
alpha = 0.98
angle = alpha * gyro_angle + (1-alpha) * acc_angle
```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
IMU nekustīga uz galda	Gyro tuvu 0 pēc bias korekcijas		
Lēns sasvērums	Acc angle mainās paredzamā virzienā		
30 s nekustīga	Tīrs gyro integrālis driftē vairāk nekā complementary filter rezultāts		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.7. H5 — ESP32 + TB6612FNG — DC motora virziens un PWM

Mērķis: Parādīt, ka mikrokontrolle dod tikai vadības signālu; jaudas ceļš iet caur motoru draiveri. Sākt ar mazu duty cycle un paceltu riteni.

Aprīkojums

- ESP32 development board
- TB6612FNG breakout
- DC gear motor
- ārējs 5–9 V avots ar strāvas limitu

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
ESP32 3V3	TB6612 VCC	Loģikas barošana
Ārējais +5...9 V	TB6612 VM	Motora barošana
Kopīgs GND	ESP32 GND + TB6612 GND + barošanas GND	Atsauce
GPIO18	AIN1	Virziens A
GPIO19	AIN2	Virziens A
GPIO25	PWMA	PWM A
GPIO23	STBY	Draivera enable/standby
A01/A02	Motora spaiļes	DC motors

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.

6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseidokoda kodols

```
// Arduino-ESP32 stila pseidokods, API versija var atšķirties
const int IN1=18, IN2=19, PWM=25, STBY=23;

void setMotor(int speedCmd) { // -100..100
  speedCmd = constrain(speedCmd, -100, 100);
  digitalWrite(STBY, HIGH);
  digitalWrite(IN1, speedCmd >= 0 ? HIGH : LOW);
  digitalWrite(IN2, speedCmd >= 0 ? LOW : HIGH);
  int duty = map(abs(speedCmd), 0, 100, 0, 255);
  writePwm(PWM, duty);
}
```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
0 %	Motors apstāties		
30 % uz priekšu	Lēna rotācija uz priekšu		
-30 %	Rotācija pretējā virzienā		
STBY LOW	Motors nedarbojas		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.8. H6 — Kvadrātūras enkoders — impulsi, virziens un attālums

Mērķis: Skaitīt impulsus un noteikt virzienu no A/B fāzes; pēc tam pārveidot impulsus uz apgriezieniem un lineāru ceļu.

Aprīkojums

- ESP32
- DC gear motor ar 3,3 V saderīgu enkodera izeju
- TB6612FNG
- multimetrs/osciloskops, ja pieejams

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
Encoder A	GPIO32	Kvadrātūras kanāls A
Encoder B	GPIO33	Kvadrātūras kanāls B
Encoder GND	GND	Zeme
Encoder VCC	Atbilstoši enkodera specifikācijai	Barošana

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseudokoda kodols

```
volatile long ticks = 0;
```

```
void IRAM_ATTR onAO {  
  bool a = digitalRead(32);  
  bool b = digitalRead(33);  
  ticks += (a == b) ? 1 : -1;
```

```
}
```

```
// distance_m = ticks / ticks_per_rev * PI * wheel_diameter_m;
```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Ritenis 1 apgriezīnu uz priekšu	Ticks palielinās par kalibrēto skaitu		
1 apgrieziens atpakaļ	Ticks samazinās		
5 apgriezieni	Aprēķinātais ceļš ir tuvs $5 \times$ riteņa apkārtmēram		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.9. H7 — Closed-loop motora ātruma vadība ar enkoderi

Mērķis: Pāriet no open-loop “PWM = 40 %” uz closed-loop “mērķis = 120 rpm”; mērīt, aprēķināt kļūdu un koriģēt vadību.

Aprīkojums

- ESP32
- TB6612FNG
- DC gear motor ar enkoderi
- ārējs barošanas avots

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
Izmanto H5 + H6 pieslēgumus	—	PWM + enkoders vienā sistēmā

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseidokoda kodols

```
target_rpm = 120
every 100 ms:
    measured_rpm = ticks_delta / ticks_per_rev * 600.0
    error = target_rpm - measured_rpm
    pwm = clamp(pwm + Kp * error, 0, 255)
    setMotorPwm(pwm)
```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Ritenis brīvi	RPM nostabilizējas ap mērķi		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.10. H8 — 2WD mobilais robots — divi motori, enkoderi un differential drive

Mērķis: Savienot motora ātrumus ar robota lineāro un leņķisko ātrumu. Pirms grīdas testa pārbaudīt riteņu virzienu, stop un avārijas komandēšanu.

Aprīkojums

- ESP32
- TB6612FNG
- 2 motori ar enkoderiem
- 2WD šasija
- barošanas avots

Pieslēgumu tabula / elektroskāmes ekvivalents

No	Uz	Piezīme
Motor A	TB6612 A kanāls	Kreisais ritenis
Motor B	TB6612 B kanāls	Labais ritenis
Left encoder A/B	GPIO32/GPIO33	Kreisā riteņa odometrija
Right encoder A/B	GPIO16/GPIO17	Labā riteņa odometrija

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseudokoda kodols

```
v = (v_right + v_left) / 2.0
omega = (v_right - v_left) / wheel_base

x += v * cos(theta) * dt
```

```

y += v * sin(theta) * dt
theta += omega * dt

```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
$v_L = v_R > 0$	Robots brauc taisni		
$v_L = 0, v_R > 0$	Robots griežas pa kreisi lokā		
$v_L = -v_R$	Robots aptuveni griežas uz vietas		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.11. H9 — ESP32 IoT telemetrija — sensoru dati uz MQTT

Mērķis: Datu ziņojumam definēt version, device_id, timestamp, value, unit, quality. Atvienot tīklu un novērot atkopšanos.

Aprikojums

- ESP32
- jebkurš iepriekšējais sensors
- Wi-Fi tīkls
- MQTT brokeris mācību vidē

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
Sensora interfeiss	ESP32	Mērījums
ESP32 Wi-Fi	Brokeris	MQTT transports

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseudokoda kodols

```
{
  "schema": 1,
  "device_id": "robot-01",
  "ts_ms": 1710000000000,
  "distance_mm": 438,
  "quality": "valid"
}
```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Brokeris pieejams	Ziņojumi redzami topic telemetry/robot-01		
Wi-Fi izslēgts	Ierīce neuzkaras; statusā redzama sakaru kļūda		
Wi-Fi atjaunots	Savienojums atkopjas un telemetrija turpinās		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.12. H10 — RPLIDAR A1 + Python — 2D skenējums un polārā karte

Mērķis: Pirmajā LiDAR laboratorijā nevada motoru un barošanu ar paštaisītu shēmu; izmanto ražotāja adapteri. Apstrādā angle, distance, quality un veido x/y punktus.

Aprīkojums

- RPLIDAR A1
- ražotāja USB/UART adapteris
- PC ar Python

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
RPLIDAR A1	Ražotāja USB/UART adapteris	Barošana + seriālais interfeiss
USB adapteris	PC	Datu plūsma uz Python

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseidokoda kodols

```
for angle_deg, distance_m, quality in scan:
    if quality < Q_MIN or not (0.15 <= distance_m <= MAX_RANGE):
        continue
    a = radians(angle_deg)
    x = distance_m * cos(a)
    y = distance_m * sin(a)
    points.append((x, y))
```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Siena 1–2 m attālumā	Punktu mākonī redzama aptuveni taisna struktūra		
Objekts pārvietots	Skenējumā mainās atbilstošais sektors		
Nederīgi punkti filtrēti	Kartē mazāk acīmredzamu outlieru		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.13. H11 — Machine Vision — USB kamera + Python/OpenCV

Mērķis: Noņemt attēlu, pārveidot grayscale, threshold/edges, atrast kontūras un sasaistīt pikseļu koordinātes ar robota lēmumu.

Aprīkojums

- PC
- USB kamera vai iebūvēta kamera
- Python + OpenCV

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
Kamera	PC USB/iebūvēts interfeiss	Attēlu plūsma

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseidokoda kodols

```
frame = camera.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 80, 160)
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Labs apgaismojums	Objekta kontūra stabila		

Testa gadījums	Expected	Actual	PASS/FAIL
Vājš apgaismojums	Rezultāts pasliktinās — jāpārrunā preprocessing		
Objekts pārvietots	Centroid koordinātes mainās paredzami		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.14. H12 — Integrēts autonoma mini robota prototips

Mērķis: Integrēt state machine: IDLE → FORWARD → OBSTACLE_STOP → TURN → FORWARD.
Drošībai noteikt maksimālo ātrumu un “stop on stale sensor”.

Aprīkojums

- ESP32
- TB6612FNG
- 2 motori ar enkoderiem
- VL53L0X vai cits priekšējais attāluma sensors
- IMU
- 2WD šasija

Pieslēgumu tabula / elektroslēmas ekvivalents

No	Uz	Piezīme
Motori + enkoderi	Kā H8	Kustība un odometrija
VL53L0X	I2C 21/22 kopnē	Šķērslis priekšā
IMU	Tajā pašā I2C kopnē, ja adreses nekonfliktē	Orientācijas dinamika
ESP32	Wi-Fi pēc izvēles	Telemetrija

Ko stāstīt pirms pieslēgšanas

Vispirms nošķir signāla ceļu no jaudas ceļa. Nosauc sprieguma līmeni un paskaidro, ko katrs vads dara. Pirms barošanas pieslēgšanas students ar shēmu salīdzina VCC, GND un signālu pinus; motoru laboratorijās pārbauda ārējā barošanas avota strāvas limitu.

Darba gaita

1. Ar izslēgtu barošanu saliec pieslēgumu pēc tabulas un pārbaudi polaritāti.
2. Veic “smoke test” ar minimālo funkcionalitāti: vai plate startējas, vai sensors/izvade ir sasniedzama.
3. Palaid programmu ar drošiem sākuma parametriem un reģistrē pirmo mērījumu/reakciju.
4. Izpildi testa tabulu, ierakstot Actual rezultātu un PASS/FAIL.
5. Izprovocē vismaz vienu kontrolētu kļūdas gadījumu un apraksti sistēmas reakciju.
6. Saglabā kodu, pieslēgumu tabulu, testa pierādījumu un īsu secinājumu portfolio.

Koda/pseidokoda kodols

```
state = IDLE
loop:
  read_sensors()
```

```

if sensor_stale or emergency:
    stop(); state = FAULT
elif state == FORWARD and distance_mm < 350:
    stop(); state = TURN
elif state == TURN and turn_complete():
    state = FORWARD
execute_state(state)

```

Testēšanas protokols

Testa gadījums	Expected	Actual	PASS/FAIL
Sensors derīgs, ceļš brīvs	Robots brauc ar ierobežotu ātrumu		
Šķērslis < 0.35 m	Robots apstājas un pāriet izvairīšanās režīmā		
Sensora dati novecojuši	Robots apstājas drošā stāvoklī		

Paplašinājums A/B/C līmenim

- A līmenis: panākt stabilu pamatfunkciju un korekti aizpildīt testa protokolu.
- B līmenis: pievienot datu validāciju, statusu un kļūdas apstrādi.
- C līmenis: pievienot mērījumu žurnālu/telemetriju, automātisku testu vai closed-loop uzvedību.

38.15. Fizisko laboratoriju progresijas karte

Posms	Laboratorijas	Galvenā profesionālā prasme
I/O un signāli	H1–H2	No fiziska signāla uz programmas mainīgo; pamata diagnostika
Digitālie sensori	H3–H4	I2C, datu kvalitāte, timeout, kalibrācija
Aktuatori	H5	Jaudas interfeiss, PWM, virziens, drošs starts
Atgriezeniskā saite	H6–H7	Enkodērs, ātrums, closed-loop un vienkāršs regulators
Mobilā robotika	H8	Differential-drive kinemātika un odometrija

Posms	Laboratorijas	Galvenā profesionālā prasme
Savienotās sistēmas	H9	MQTT, telemetrija, sakaru atteice
Perception	H10–H11	LiDAR un Machine Vision datu apstrāde
Autonomija	H12	State machine, sensor fusion princips, safe stop, integrācijas tests

38.16. Fiziskās laboratorijas vērtēšanas rubrika

Kritērijs	0–1	2	3
Drošs pieslēgums	Būtiskas kļūdas / nepieciešama iejaukšanās	Pamatā korekts ar nelielām kļūdām	Patstāvīgi pārbaudīts VCC/GND/signāli un drošības riski
Funkcionalitāte	Nedarbojas un nav diagnostikas	Pamatfunkcija darbojas	Darbojas pamatfunkcija + kļūdas/robežgadījums
Testēšana	Nav testa pierādījumu	Aizpildīti pamatgadījumi	Normal + boundary/fault ar Actual un secinājumu
Kods	Nestrukturēts / nesaprotams	Sadalīts pamatfunkcijās	Skaidri interfeisi, konstantes, validācija, stāvokļi
Dokumentācija	Nav reproducējama	Pieslēgums + īss apraksts	Pieslēgums, versijas, testi, secinājumi un zināmie ierobežojumi

39. Tehnisko avotu un aparatūras drošības piezīmes

Šīs versijas fizisko laboratoriju tehniskie parametri pārbaudīti pret ražotāju primārajiem avotiem 18.08.2026. Konkrētās breakout plates var pievienot regulatorus, līmeņu pārveidotājus vai citus elementus, tāpēc pirms pieslēgšanas vienmēr pārbauda tieši izmantotā moduļa dokumentāciju.

Avots	Kursā izmantotais fakts / ierobežojums
Arduino — UNO R3 oficiālā dokumentācija	ATmega328P bāze, 14 digitālie I/O, 6 PWM izejas, 6 analogās ievades, 16 MHz; izmantots H1–H2 kontekstam.
Espressif — ESP-IDF I2C dokumentācija	ESP32 I2C ir konfigurējams, izmanto SDA/SCL open-drain līnijas un atbalsta Standard/Fast mode; GPIO21/22 dokumentā ir kursa laboratorijas konvencija, nevis universāls noteikums.
Espressif — ESP-IDF LEDC/PWM dokumentācija	LEDC perifērija ģenerē PWM, un frekvence ir saistīta ar duty resolution; izmantots motoru PWM skaidrojumam.
STMicroelectronics — VL53L0X	Time-of-Flight sensors ar I2C interfeisu; ražotājs norāda absolūtā attāluma mērījumu līdz 2 m un Class 1 lāzera ierīci.
Toshiba — TB6612FNG	Divkanālu brushed DC motor driver, CW/CCW/brake/stop/standby funkcijas; ražotāja lapā norādīts 1.2 A vidējais output current un 3.2 A peak absolute maximum kontekstā.
SLAMTEC — RPLIDAR A1	360° 2D skeneris; ražotāja specifikācija norāda 0.15–12 m tipisko mērījuma diapazonu, 5 V sistēmas spriegumu un UART serial 3.3 V līmeņa izeju. Pirmajā laboratorijā izmantot ražotāja adapteri.

Aparatūras avoti šeit ir kursa drošības un arhitektūras pamatojums, nevis aizvietošanas konkrētā komplekta datasheet. Ja mācību iestāde iegādājas citas revīzijas vai analogus, pasniedzējs izveido “approved lab hardware” sarakstu ar pārbaudītu pinout un barošanas prasībām.

40. [ARHĪVS] Versijas 0.7 izstrādes statuss

- Pievienoti pilni izvērsti pasniedzēja runas scenāriji visām 36×90 min nodarbībām.
- Katram scenārijam pievienota galvenā doma, 90 minūšu ritms, auditorijas jautājumi, demonstrācija, tipiskie pārpratumi un noslēguma formulējums.
- Pievienots fizisko laboratoriju drošības minimums un ieteicamais aparatūras komplekts.
- Izstrādātas 12 fiziskās laboratorijas no Arduino digitālās I/O līdz integrētam autonomam 2WD robotam.
- Fiziskajām laboratorijām pievienotas pieslēgumu tabulas, darba gaitas, koda/pseidokoda kodoli, testa protokoli un A/B/C paplašinājumi.
- Pievienota fizisko laboratoriju progresijas karte un vienota vērtēšanas rubrika.
- Pievienotas ražotāju primāro tehnisko avotu un aparatūras drošības piezīmes.

Nākamais attīstības slānis: studentu izdrukājamās H1–H12 darba lapas ar tukšiem mērījumu/testu laukiem; detalizētas C++/ESP32 etalona programmas; 2WD robota mehāniskās kalibrācijas procedūra; ROS 2 pārejas laboratorija no mikrokontrollera telemetrijas uz /odom, /imu un /scan datu plūsmu.

41. Izdrukājamās H1-H12 fizisko laboratoriju darba lapas

Šī nodaļa paredzēta tiešai izdrukai un izmantošanai laboratorijā. Katra darba lapa veidota pēc vienotas inženiertehniskas loģikas: mērķis → drošības pārbaude → pieslēgums → sākotnējie parametri → mērījumi → testēšana → kļūdu analīze → secinājumi. Skolēna/studentu daļā apzināti atstāti tukši lauki, lai rezultāti tiktu dokumentēti, nevis tikai demonstrēti uz ekrāna.

Drošības princips: Darba lapas drīkst kopēt kā atsevišķas lapas. Pasniedzējs pirms laboratorijas norāda konkrētās plates revīziju, izmantotos sensoru moduļus, pieļaujamo barošanas spriegumu un laboratorijas lokālos drošības noteikumus.

41.2. H1 — Arduino digitālā/analogā I/O un PWM

Mērķis: Nolasīt analogu signālu, pārveidot to programmā un vadīt izeju ar PWM.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- Arduino UNO vai ekvivalents
- potenciometrs vai analogs sensors
- LED + 220–330 Ω rezistors
- USB kabelis

B. Pirms darba — drošības un gatavības pārbaude

- [] Analogo ieeju nebarot virs plates atļautā sprieguma.
- [] LED obligāti lietot ar strāvas ierobežošanas rezistoru.
- [] Pirms pārspraušanas atvienot USB/barošanu, ja pastāv īsslēguma risks.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
A0 mērījums	ADC raw	0...1023		
PWM izeja	duty	0...255		
LED reakcija	vizuāli	vienmērīga		

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kā ADC skaitlis saistīts ar fizisko spriegumu?

Atbilde: _____

2. Kāpēc analogWrite uz UNO nav īsta analogā sprieguma izeja?

Atbilde: _____

3. Kas notiek, ja sensors dod vērtību ārpus paredzētā diapazona?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.3. H2 — ESP32 I2C skenēšana un sensora adreses atrašana

Mērķis: Pārbaudīt I2C kopni un identificēt pieslēgtu ierīču adreses.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- ESP32
- I2C sensors/modulis
- savienojuma vadi
- USB kabelis

B. Pirms darba — drošības un gatavības pārbaude

- [] Pārbaudīt sensora logikas spriegumu; ESP32 GPIO ir 3,3 V logika.
- [] SDA/SCL kopnei nepieciešami pull-up rezistori; bieži tie jau ir uz moduļa.
- [] Nepievienot 5 V signālu tieši ESP32 GPIO, ja modulis to neierobežo.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
SDA pin	GPIO			
SCL pin	GPIO			
Atrasta adrese	hex			

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Skenēšanas stabilitāte	atkārtojumi	3		

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kāpēc vienā I2C kopnē var būt vairākas ierīces?

Atbilde: _____

2. Kas notiek, ja divām ierīcēm ir viena un tā pati adrese?

Atbilde: _____

3. Kā atšķirt vadu problēmu no programmas problēmas?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.4. H3 — VL53L0X/ToF attāluma sensors

Mērķis: Nolasīt attālumu, filtrēt mērījumu un pieņemt vienkāršu drošības lēmumu.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- ESP32 vai Arduino
- VL53L0X modulis
- lineāls/mērlente
- šķērslis ar dažādu virsmas krāsu

B. Pirms darba — drošības un gatavības pārbaude

- [] Pārbaudīt moduļa breakout plates barošanas prasības.
- [] Mērījuma kļūda atkarīga no virsmas, leņķa un apkārtējās gaismas.
- [] Robotu testā pirmo reizi motorus turēt atslēgtus.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Mērķis 1	mm	100		
Mērķis 2	mm	300		
Mērķis 3	mm	500		

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
STOP sliekšnis	mm			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kāpēc sensora mērījums nav absolūti precīzs?

Atbilde: _____

2. Kā izvēlēties STOP sliekšni, ņemot vērā bremzēšanas ceļu?

Atbilde: _____

3. Kā samazināt nejaušu viena mērījuma ietekmi?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.5. H4 — IMU: paātrinājums, žiroskops un orientācijas novērtējums

Mērķis: Nolasīt IMU datus, noteikt bias/drift un aprēķināt vienkāršu orientācijas novērtējumu.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- ESP32
- MPU6050 vai ekvivalenta IMU
- USB kabelis
- stabila horizontāla virsma

B. Pirms darba — drošības un gatavības pārbaude

- [] Moduli nostiprināt, lai testa laikā tas nekristu uz vadiem.
- [] Reģistrēt asu orientāciju; “X/Y/Z” ir sensora koordinātu sistēma.
- [] Kalibrāciju veikt nekustīgam sensoram.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
ax	g			
ay	g			
az	g			

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
gx bias	deg/s			
gy bias	deg/s			
gz bias	deg/s			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kāpēc žiroskopa integrācija rada drift?

Atbilde: _____

2. Kāpēc accelerometer nav labs ātrai rotācijai?

Atbilde: _____

3. Ko nozīmē sensor fusion?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.6. H5 — TB6612FNG un DC motora vadība ar ESP32

Mērķis: Droši vadīt motora virzienu un ātrumu ar H-bridge un PWM.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- ESP32
- TB6612FNG modulis
- DC motors
- atsevišķs motora barošanas avots
- kopējais GND

B. Pirms darba — drošības un gatavības pārbaude

- [] Motora strāvu nebarot no ESP32 3,3 V tapas.
- [] Loģikas un motora barošanu savienot saskaņā ar moduļa specifikāciju; GND jābūt kopējam.
- [] Pirmo testu veikt ar riteni paceltu no galda.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
PWM 25%	rpm/novērojums			
PWM 50%	rpm/novērojums			

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
PWM 75%	rpm/novērojums			
Reverse	OK/FAIL			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kāpēc motors nesāk griezties pie ļoti maza PWM?

Atbilde: _____

2. Kas ir back-EMF?

Atbilde: _____

3. Kāpēc programmatūras STOP nav tas pats, kas drošības E-stop?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.7. H6 — Kvadrātūras enkoders

Mērķis: Nolasīt enkodera impulsus, noteikt virzienu un aprēķināt riteņa veikto attālumu.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- ESP32
- motors ar enkoderi vai atsevišķs enkoders
- mērlente
- stends/robots

B. Pirms darba — drošības un gatavības pārbaude

[] Enkodera signālu spriegumam jābūt savietojamam ar ESP32.

[] Izvēlēties GPIO, kas atbalsta nepieciešamo ieeju un interrupt.

[] Mehānisko riteni testēt tā, lai tas nevar aizķert vadus.

[] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
CPR/PPR	ticks/rev			
Riteņa diametrs	mm			
1 apgrieziens	ticks			

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Aprēķinātais ceļš	mm			
Nomērītais ceļš	mm			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Ar ko PPR atšķiras no CPR dažādās datu lapās?

Atbilde: _____

2. Kāpēc interrupt kodam jābūt īsam?

Atbilde: _____

3. Kā riteņa slīdēšana ietekmē odometriju?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.8. H7 — Closed-loop motora ātruma vadība

Mērķis: Izmantot enkodera atgriezenisko saiti, lai uzturētu noteiktu motora ātrumu.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- ESP32
- TB6612FNG
- DC motors ar enkoderi
- barošanas avots

B. Pirms darba — drošības un gatavības pārbaude

- [] Pirms PID testēšanas pārbaudīt, ka atvērtās cilpas motora vadība un enkoders strādā atsevišķi.
- [] Sākt ar maziem koeficientiem un drošu RPM mērķi.
- [] Ieviest PWM ierobežojumu un timeout.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Setpoint	rpm			
Kp				
Ki				

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Kd				
Overshoot	%			
Settling time	s			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Ko dara P, I un D daļa?

Atbilde: _____

2. Kas ir integratora windup?

Atbilde: _____

3. Kāpēc sampling periods ietekmē regulatoru?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.9. H8 — 2WD differential-drive mobilais robots

Mērķis: Aprēķināt abu riteņu attālumus, robota pozu $x/y/\theta$ un vadīt taisnu kustību/pagriezienu.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- 2WD šasija
- ESP32
- TB6612FNG
- 2 motori ar enkoderiem
- akumulators/barošanas avots

B. Pirms darba — drošības un gatavības pārbaude

- [] Robotu pirmajās pārbaudēs novietot uz statīva.
- [] Nodrošināt brīvu testa zonu un mehānisku iespēju nekavējoties atslēgt motoru barošanu.
- [] Pirms kustības pārbaudīt kreisā/labā motora polaritāti un enkoderu zīmes.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Riteņa rādiuss r	m			
Track width b	m			

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
CPR L/R	ticks/rev			
3 m tests	m			
360° tests	deg			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kāpēc taisnā kustībā robots var novirzīties?

Atbilde: _____

2. Kā b kļūda ietekmē pagrieziena leņķi?

Atbilde: _____

3. Kāpēc enkoderu odometrijas kļūda ar laiku uzkrājas?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.10. H9 — ESP32 IoT telemetrija ar MQTT

Mērķis: Nosūtīt sensoru/robota telemetriju strukturētā ziņojumā un apstrādāt savienojuma zudumu.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- ESP32
- Wi-Fi tīkls laboratorijā
- MQTT brokeris
- viens sensors vai simulēti dati

B. Pirms darba — drošības un gatavības pārbaude

- ☐ Neievietot publiski lietotājpārdus/parole kodā.
- ☐ Laboratorijas brokeri konfigurēt tikai mācību tīklā.
- ☐ Vadības komandām paredzēt autentifikāciju un drošu noklusējuma stāvokli.
- ☐ Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Topic				
Publish period	ms			

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Payload	JSON			
Reconnect test	PASS/FAIL			
Offline behavior	apraksts			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Ar ko MQTT atšķiras no HTTP request/response?

Atbilde: _____

2. Ko nozīmē QoS?

Atbilde: _____

3. Kas robotam jādara, ja sakari pazūd?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.11. H10 — RPLIDAR 2D skenējums un polārā karte

Mērķis: Iegūt 2D lāzerskeni, filtrēt nederīgus punktus un noteikt tuvāko šķērslī sektora.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- RPLIDAR A1 vai ekvivalents
- dators
- USB adapteris
- Python vide

B. Pirms darba — drošības un gatavības pārbaude

- [] Rotējošu LiDAR mehāniku neaiztikt darbības laikā.
- [] Kabeli novietot tā, lai tas nevar ieķerties rotējošajā daļā.
- [] Lāzera sensoru izmantot atbilstoši ražotāja drošības klasifikācijai.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Scan points	skaits			
Min range	m			
Front sector min	m			

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Invalid points	skaitis			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kāpēc LiDAR mēra polārās koordinātās?

Atbilde: _____

2. Kas ir occlusion?

Atbilde: _____

3. Kāpēc jāfiltrē 0/inf/ārpus diapazona vērtības?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.12. H11 — Machine Vision ar OpenCV

Mērķis: Iegūt kameras attēlu, veikt priekšapstrādi un atrast objektu/contour.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- USB kamera
- dators
- Python + OpenCV
- kontrastējošs testa objekts

B. Pirms darba — drošības un gatavības pārbaude

- [] Kameras attēlos neiekļaut personas bez nepieciešamības/piekrišanas.
- [] Fiksēt apgaismojuma apstākļus, lai tests būtu atkārtojams.
- [] Atšķirt demonstrācijas threshold no robustas industriālās machine vision sistēmas.
- [] Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Resolution	px			
Threshold				
Contours	skaits			

Mērijums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Largest area	px ²			
Detection	PASS/FAIL			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kāpēc apgaismojums maina rezultātu?

Atbilde: _____

2. Ko dod blur pirms threshold/edge detection?

Atbilde: _____

3. Kāda atšķirība starp klasisku machine vision un ML object detection?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

41.13. H12 — Integrēts autonoma mini robota prototips

Mērķis: Apvienot kustību, enkoderus, attāluma uztveri/IMU un lēmumu loģiku vienā droši testējamā sistēmā.

Students	Grupa	Datums	Darba vieta

A. Aprīkojums

- 2WD robots
- ESP32
- motora draiveris
- enkoderi
- IMU
- ToF/LiDAR pēc pieejamības

B. Pirms darba — drošības un gatavības pārbaude

- ☐ Izmantot fizisku motora barošanas atslēgšanu.
- ☐ Katru funkciju testēt atsevišķi pirms integrācijas.
- ☐ Programmatūrā ieviest sensoru timeout, komandas timeout un drošu STOP stāvokli.
- ☐ Pieslēgumu pārbaudīja pasniedzējs / atbildīgā persona: _____

C. Sākotnējie parametri un pieslēgums

Parametrs / signāls	Vērtība / GPIO / adrese	Piezīmes
Barošanas spriegums		
GND kopīgs?		
Galvenais ievades signāls		
Galvenais izejas signāls		
Programmas versija / commit		

D. Darba gaita

1. Uzzīmē vai papildini funkcionālo ķēdi: ievade → apstrāde → izvade → feedback (ja attiecināms).
2. Pirms ieslēgšanas pārbaudi vadu savienojumus un barošanas polaritāti.
3. Palaid minimālo diagnostikas programmu un pierādi, ka katra komponente strādā atsevišķi.
4. Palaid laboratorijas galveno programmu un saglabā mērījumu rezultātus.
5. Izpildi vismaz vienu normālu, vienu robežgadījuma un vienu kļūmes testu.
6. Veic vienu mērķtiecīgu izmaiņu kodā/parametrā un apraksti, kā mainījās rezultāts.

E. Mērījumu tabula

Mērījums	Vienība	Mērķis / references vērtība	Faktiskā vērtība	Novirze / komentārs
Start-up self-test	PASS/FAIL			
Straight 2 m	error cm			
Obstacle stop	distance m			
Sensor timeout	STOP?			
Recovery	PASS/FAIL			

F. Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normāla darbība				
Apakšējā robeža				
Augšējā robeža				
Sensora/komunikācijas kļūme				
Atkārtots tests pēc izmaiņas				

G. Analīzes jautājumi

1. Kādas funkcijas ir safety-critical?

Atbilde: _____

2. Kā functional tree palīdz atrast testējamās funkcijas?

Atbilde: _____

3. Kādi sensori nepieciešami, lai no “reaktīva” robota pārietu uz autonomu navigāciju?

Atbilde: _____

H. Secinājumi un pašvērtējums

Kas izdevās: _____

Galvenā kļūda / problēma: _____

Kā problēma tika diagnosticēta: _____

Ko nākamreiz darītu citādi: _____

Pašvērtējums (1–10): _____ Pasniedzēja vērtējums: _____

42. Etalona programmas: Arduino, ESP32, Python un IoT

Šīs programmas paredzētas kā pasniedzēja etalons un skolēna/studentu salīdzināšanas materiāls pēc patstāvīga mēģinājuma. Tās nav jāizmanto kā vienīgais iespējamais risinājums. Pinout un bibliotēku versijas vienmēr jāsalīdzina ar konkrēto laboratorijas aparāturu.

Versiju piezīme: ESP32 piemēros izmantota aktuālā Arduino-ESP32 LEDC pieeja `ledcAttach(pin, frequency, resolution)` un `ledcWrite(pin, duty)`. I2C piemērā Generic ESP32 tipiskie pini ir SDA=21 un SCL=22, taču konkrētai platei tie var atšķirties.

42.1. H1 etalons — Arduino ADC → PWM

```
const int POT_PIN = A0;
const int LED_PIN = 9; // UNO PWM pin

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  int raw = analogRead(POT_PIN); // 0...1023 on classic UNO
  int duty = map(raw, 0, 1023, 0, 255); // scale to PWM duty
  duty = constrain(duty, 0, 255);
  analogWrite(LED_PIN, duty);

  Serial.print("adc=");
  Serial.print(raw);
  Serial.print(", pwm=");
  Serial.println(duty);
  delay(50);
}
```

Pasniedzējam: uzsveriet, ka PWM nav nepārtraukts analogs spriegums. Slodze saņem impulsus ar mainītu duty cycle; mehāniskās un elektriskās sistēmas bieži šo vidējo enerģiju “izfiltrē”.

42.2. H2 etalons — ESP32 I2C scanner

```
#include <Arduino.h>
#include <Wire.h>

constexpr int SDA_PIN = 21;
constexpr int SCL_PIN = 22;

void setup() {
  Serial.begin(115200);
  Wire.begin(SDA_PIN, SCL_PIN);
  delay(500);
}

void loop() {
  int found = 0;
  Serial.println("I2C scan start");

  for (uint8_t address = 1; address < 127; ++address) {
    Wire.beginTransmission(address);
    uint8_t error = Wire.endTransmission();
  }
```

```

if (error == 0) {
    Serial.printf("Found: 0x%02X\n", address);
    found++;
}
}

Serial.printf("Devices found: %d\n\n", found);
delay(3000);
}

```

42.3. H3 etalons — VL53L0X attālums un drošības sliekšnis

Nepieciešama bibliotēka: Adafruit VL53L0X vai cita saderīga bibliotēka. Pirms darba pasniedzējs fiksē izmantotās bibliotēkas nosaukumu un versiju.

```

#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_VL53L0X.h>

Adafruit_VL53L0X lox;
constexpr uint16_t STOP_MM = 250;

void setup() {
    Serial.begin(115200);
    Wire.begin();
    if (!lox.begin()) {
        Serial.println("VL53L0X init failed");
        while (true) delay(1000);
    }
}

void loop() {
    VL53L0X_RangingMeasurementData_t m;
    lox.rangingTest(&m, false);

    if (m.RangeStatus != 4) {
        uint16_t d = m.RangeMilliMeter;
        Serial.print("distance_mm=");
        Serial.print(d);
        Serial.print(", state=");
        Serial.println(d < STOP_MM ? "STOP" : "CLEAR");
    } else {
        Serial.println("distance=INVALID, state=SAFE_STOP");
    }
    delay(100);
}

```

42.4. H4 etalons — MPU6050 bez ārējas bibliotēkas: raw I2C dati

```

#include <Arduino.h>
#include <Wire.h>

constexpr uint8_t MPU = 0x68;

void writeReg(uint8_t reg, uint8_t value) {
    Wire.beginTransmission(MPU);
    Wire.write(reg);
    Wire.write(value);
    Wire.endTransmission(true);
}

```



```

}

int16_t read160 {
    return (int16_t)((Wire.read() << 8) | Wire.read());
}

void setup() {
    Serial.begin(115200);
    Wire.begin(21, 22);
    writeReg(0x6B, 0x00); // wake from sleep
    delay(100);
}

void loop() {
    Wire.beginTransaction(MPU);
    Wire.write(0x3B);      // ACCEL_XOUT_H
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, (uint8_t)14, (uint8_t)true);

    int16_t axRaw = read160;
    int16_t ayRaw = read160;
    int16_t azRaw = read160;
    int16_t tempRaw = read160;
    int16_t gxRaw = read160;
    int16_t gyRaw = read160;
    int16_t gzRaw = read160;

    // Default ranges after reset: accelerometer  $\pm 2$  g, gyro  $\pm 250$  deg/s.
    float ax = axRaw / 16384.0f;
    float ay = ayRaw / 16384.0f;
    float az = azRaw / 16384.0f;
    float gx = gxRaw / 131.0f;
    float gy = gyRaw / 131.0f;
    float gz = gzRaw / 131.0f;

    Serial.printf("a[g]=%.3f,%.3f,%.3f g[deg/s]=%.2f,%.2f,%.2f\n",
        ax, ay, az, gx, gy, gz);
    delay(20);
}

```

Svarīgi: Sensitivitātes koeficienti der tikai tad, ja reģistri nav pārkonfigurēti uz citu mērījumu diapazonu. Konkrētam sensoram vienmēr pārbauda datu lapu.

42.5. H5 etalons — ESP32 + TB6612FNG viena motora vadība

```

#include <Arduino.h>

constexpr int PWMA = 25;
constexpr int AIN1 = 26;
constexpr int AIN2 = 27;
constexpr int STBY = 33;
constexpr int PWM_FREQ = 20000;
constexpr int PWM_BITS = 8;

void setMotor(int command) {
    command = constrain(command, -255, 255);
    digitalWrite(STBY, HIGH);

    if (command > 0) {

```

```

    digitalWrite(AIN1, HIGH);
    digitalWrite(AIN2, LOW);
    ledcWrite(PWMA, command);
} else if (command < 0) {
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, HIGH);
    ledcWrite(PWMA, -command);
} else {
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, LOW);
    ledcWrite(PWMA, 0);
}
}

void setup() {
    pinMode(AIN1, OUTPUT);
    pinMode(AIN2, OUTPUT);
    pinMode(STBY, OUTPUT);
    ledcAttach(PWMA, PWM_FREQ, PWM_BITS);
    setMotor(0);
}

void loop() {
    setMotor(100); delay(2000);
    setMotor(0); delay(1000);
    setMotor(-100); delay(2000);
    setMotor(0); delay(2000);
}

```

42.6. H6 etalons — enkodera interrupt un attāluma aprēķins

```

#include <Arduino.h>

constexpr int ENC_A = 32;
constexpr int ENC_B = 33;
constexpr float WHEEL_DIAMETER_M = 0.065f;
constexpr long TICKS_PER_REV = 600; // must match your counting method

volatile long ticks = 0;

void IRAM_ATTR onA() {
    bool a = digitalRead(ENC_A);
    bool b = digitalRead(ENC_B);
    ticks += (a == b) ? 1 : -1;
}

void setup() {
    Serial.begin(115200);
    pinMode(ENC_A, INPUT_PULLUP);
    pinMode(ENC_B, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(ENC_A), onA, CHANGE);
}

void loop() {
    noInterrupts();
    long t = ticks;
    interrupts();

    float distancePerTick = PI * WHEEL_DIAMETER_M / TICKS_PER_REV;

```

```

float distance = t * distancePerTick;
Serial.printf("ticks=%ld,distance_m=%.4f\n", t, distance);
delay(100);
}

```

42.7. H7 etalons — PI/PID tipa motora ātruma regulators

```

#include <Arduino.h>

constexpr int PWMA = 25, AIN1 = 26, AIN2 = 27, STBY = 33;
constexpr int ENC_A = 32, ENC_B = 34;
constexpr long TICKS_PER_REV = 600;
constexpr float DT = 0.05f; // 50 ms

volatile long ticks = 0;
float targetRpm = 80.0f;
float kp = 1.2f, ki = 0.5f, kd = 0.02f;
float integral = 0.0f, prevError = 0.0f;

void IRAM_ATTR onA() {
    bool a = digitalRead(ENC_A);
    bool b = digitalRead(ENC_B);
    ticks += (a == b) ? 1 : -1;
}

void motorForward(int pwm) {
    digitalWrite(STBY, HIGH);
    digitalWrite(AIN1, HIGH); digitalWrite(AIN2, LOW);
    ledcWrite(PWMA, constrain(pwm, 0, 255));
}

void setup() {
    Serial.begin(115200);
    pinMode(AIN1, OUTPUT); pinMode(AIN2, OUTPUT); pinMode(STBY, OUTPUT);
    pinMode(ENC_A, INPUT_PULLUP); pinMode(ENC_B, INPUT); // external pull-up if needed
    attachInterrupt(digitalPinToInterrupt(ENC_A), onA, CHANGE);
    ledcAttach(PWMA, 20000, 8);
}

void loop() {
    static unsigned long lastMs = millis();
    static long prevTicks = 0;
    if (millis() - lastMs < 50) return;
    lastMs += 50;

    noInterrupts(); long nowTicks = ticks; interrupts();
    long delta = nowTicks - prevTicks; prevTicks = nowTicks;
    float rpm = (delta / (float)TICKS_PER_REV) * (60.0f / DT);

    float error = targetRpm - rpm;
    integral += error * DT;
    integral = constrain(integral, -100.0f, 100.0f); // simple anti-windup
    float derivative = (error - prevError) / DT;
    prevError = error;

    float u = kp*error + ki*integral + kd*derivative;
    int pwm = constrain((int)u, 0, 255);
    motorForward(pwm);
}

```

```

Serial.printf("sp=%.1f,rpm=%.1f,pwm=%d,e=%.1f\n", targetRpm, rpm, pwm, error);
}

```

42.8. H8 etalons — 2WD enkoderu odoometrijas kodols

```

#include <Arduino.h>

constexpr float R = 0.0325f;    // wheel radius [m]
constexpr float B = 0.155f;    // wheel track [m]
constexpr long TICKS_REV = 600; // per wheel, for chosen decode method

volatile long ticksL = 0, ticksR = 0;
constexpr int LA=18, LB=19, RA=21, RB=22;

void IRAM_ATTR isrL(){ ticksL += (digitalRead(LA)==digitalRead(LB)) ? 1 : -1; }
void IRAM_ATTR isrR(){ ticksR += (digitalRead(RA)==digitalRead(RB)) ? 1 : -1; }

float x=0, y=0, th=0;

void setup(){
  Serial.begin(115200);
  pinMode(LA,INPUT_PULLUP); pinMode(LB,INPUT_PULLUP);
  pinMode(RA,INPUT_PULLUP); pinMode(RB,INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(LA), isrL, CHANGE);
  attachInterrupt(digitalPinToInterrupt(RA), isrR, CHANGE);
}

void loop(){
  static long oldL=0, oldR=0;
  static unsigned long oldMs=millis();
  if(millis()-oldMs < 50) return;
  float dt=(millis()-oldMs)/1000.0f; oldMs=millis();

  noInterrupts(); long l=ticksL, r=ticksR; interrupts();
  long dTL=l-oldL, dTR=r-oldR; oldL=l; oldR=r;

  float metersPerTick = 2.0f*PI*R/TICKS_REV;
  float dL=dTL*metersPerTick;
  float dR=dTR*metersPerTick;
  float dS=(dR+dL)/2.0f;
  float dTh=(dR-dL)/B;

  float thMid=th+dTh/2.0f;
  x += dS*cos(thMid);
  y += dS*sin(thMid);
  th += dTh;
  while(th > PI) th -= 2*PI;
  while(th < -PI) th += 2*PI;

  float v=dS/dt, w=dTh/dt;
  Serial.printf("x=%.4f,y=%.4f,theta=%.4f,v=%.3f,w=%.3f\n",x,y,th,v,w);
}

```

Šis kodols apzināti nevada motorus; tas izolē odoometrijas problēmu. H8 pilnā laboratorijā motoru vadību pievieno tikai pēc tam, kad enkoderu zīmes un skaitīšanas koeficienti ir pierādīti.

42.9. H9 etalons — ESP32 MQTT JSON telemetrija

Nepieciešama PubSubClient vai cita MQTT klienta bibliotēka. Piemērā paroles ir tikai vietturi; reālā darbā tās nedrīkst glabāt publiskā repozitorijā.

```
#include <Arduino.h>
#include <WiFi.h>
#include <PubSubClient.h>

const char* WIFI_SSID = "LAB_WIFI";
const char* WIFI_PASS = "CHANGE_ME";
const char* MQTT_HOST = "192.168.1.10";
const int MQTT_PORT = 1883;

WiFiClient net;
PubSubClient mqtt(net);

void ensureConnections(){
  if(WiFi.status() != WL_CONNECTED){
    WiFi.begin(WIFI_SSID, WIFI_PASS);
    while(WiFi.status() != WL_CONNECTED) delay(500);
  }
  if(!mqtt.connected()){
    while(!mqtt.connected()){
      String id = "robot-" + String((uint32_t)ESP.getEfuseMac(), HEX);
      if(!mqtt.connect(id.c_str())) delay(1000);
    }
  }
}

void setup(){
  Serial.begin(115200);
  mqtt.setServer(MQTT_HOST, MQTT_PORT);
}

void loop(){
  ensureConnections();
  mqtt.loop();

  static uint32_t last = 0;
  if(millis() - last >= 1000){
    last = millis();
    float temperature = 23.4f; // replace with sensor
    float battery = 7.8f;
    char payload[160];
    snprintf(payload, sizeof(payload),
      "{\"ts_ms\":%lu,\"temperature_c\":%.2f,\"battery_v\":%.2f}",
      millis(), temperature, battery);
    mqtt.publish("lab/robot01/telemetry", payload);
  }
}
```

42.10. H10 etalons — Python RPLIDAR sektora analīze

```
from math import radians
from rplidar import RPLidar

PORT = '/dev/ttyUSB0' # Windows example: 'COM5'
FRONT_HALF_ANGLE = 20.0
```

```

STOP_M = 0.50

lidar = RPLidar(PORT)
try:
    for scan in lidar.iter_scans():
        front = []
        for quality, angle_deg, distance_mm in scan:
            if distance_mm <= 0:
                continue
            # front is near 0 deg; handle wrap-around
            a = angle_deg if angle_deg <= 180 else angle_deg - 360
            if abs(a) <= FRONT_HALF_ANGLE:
                front.append(distance_mm / 1000.0)

        if front:
            dmin = min(front)
            state = 'STOP' if dmin < STOP_M else 'CLEAR'
            print(f'front_min={dmin:.3f} m state={state}')
finally:
    lidar.stop()
    lidar.stop_motor()
    lidar.disconnect()

```

42.11. H11 etalons — OpenCV objekta kontūra

```

import cv2

cap = cv2.VideoCapture(0)
while True:
    ok, frame = cap.read()
    if not ok:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    _, mask = cv2.threshold(blur, 120, 255, cv2.THRESH_BINARY)

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
                                   cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        c = max(contours, key=cv2.contourArea)
        area = cv2.contourArea(c)
        if area > 500:
            x, y, w, h = cv2.boundingRect(c)
            cx, cy = x + w // 2, y + h // 2
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            cv2.circle(frame, (cx, cy), 4, (0, 0, 255), -1)
            cv2.putText(frame, f'area={area:.0f}', (x, y-8),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

    cv2.imshow('frame', frame)
    cv2.imshow('mask', mask)
    if cv2.waitKey(1) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()

```

42.12. H12 integrācijas etalons — drošības stāvokļu mašīna

```
enum class State { INIT, IDLE, DRIVE, OBSTACLE_STOP, SENSOR_FAULT, ESTOP };
State state = State::INIT;

bool sensorsHealthy();
bool obstacleDetected();
bool estopActive();
void motorsStop();
void driveForward();

void controlStep() {
    if (estopActive()) {
        state = State::ESTOP;
        motorsStop();
        return;
    }
    if (!sensorsHealthy()) {
        state = State::SENSOR_FAULT;
        motorsStop();
        return;
    }
}

switch (state) {
    case State::INIT:
        motorsStop();
        state = State::IDLE;
        break;
    case State::IDLE:
        // external start condition -> DRIVE
        break;
    case State::DRIVE:
        if (obstacleDetected()) {
            motorsStop();
            state = State::OBSTACLE_STOP;
        } else {
            driveForward();
        }
        break;
    case State::OBSTACLE_STOP:
        motorsStop();
        if (!obstacleDetected()) state = State::DRIVE;
        break;
    case State::SENSOR_FAULT:
    case State::ESTOP:
        motorsStop();
        break;
}
}
```

Profesionālā robeža: Programmatūras state machine ir mācību modelis un neaizstāj sertificētu aparatūras drošības ķēdi. Reālai iekārtai E-stop un safety funkcijas projektē atbilstoši piemērojamajiem drošības standartiem un riska novērtējumam.

43. 2WD mobilā robota mehāniskā un odometrijas kalibrēšana

Šīs nodaļas mērķis ir pāriet no “robots kustas” uz “robots kustību mēra pietiekami atkārtojami, lai rezultātu varētu izmantot lokalizācijā, sensor fusion un ROS 2 navigācijā”. Differential-drive odometrija ir ļoti jutīga pret riteņa efektīvo rādiusu, track width, enkodera skaitīšanas koeficientu, riteņu slidēšanu un mehānisko asimetriju.

43.1. Pamatparametri un vienādojumi

Apzīmējums	Nozīme	Vienība
r_L, r_R	kreisā/labā riteņa efektīvais rādiuss	m
b	attālums starp riteņu kontakta līniju centriem (track width)	m
N	enkodera ticks uz vienu riteņa apgrieziena izvēlētajā dekodēšanas režīmā	ticks/rev
$\Delta n_L, \Delta n_R$	ticks izmaiņa par laika soli	ticks
$\Delta s_L, \Delta s_R$	kreisā/labā riteņa ceļš	m
Δs	robota centra lineārais pārvietojums	m
$\Delta \theta$	robota orientācijas izmaiņa	rad

Differential-drive odometrijas pamatformulas

```
distance_per_tick_L = 2π r_L / N_L
distance_per_tick_R = 2π r_R / N_R
Δs_L = Δn_L · distance_per_tick_L
Δs_R = Δn_R · distance_per_tick_R
Δs = (Δs_R + Δs_L) / 2
Δθ = (Δs_R - Δs_L) / b
```

43.2. Kalibrēšanas secība

1. Mehāniski pārbauda riteņus, asu brīvkustību, riepu vienādu stāvokli un baterijas uzlādi.
2. Nosaka enkodera faktisko skaitīšanas režīmu un ticks/rev, manuāli pagriežot riteni vismaz 5 apgriezienus.
3. Nomēra sākotnējo riteņa diametru un track width. Šīs vērtības ir sākuma pieņēmums, nevis automātiski “patiesā” odometrijas vērtība.
4. Veic taisnās distances testu vairākos atkārtojumos un koriģē efektīvo riteņa rādiusu/scale.
5. Veic 360° pagriežiena testu vairākos atkārtojumos un koriģē track width.
6. Veic atkārtotu taisnās kustības testu, lai pārbaudītu savstarpēju ietekmi.
7. Novērtē kreisās/labās puses asimetriju un, ja vajadzīgs, izmanto atsevišķus r_L un r_R vai motoru feed-forward koeficientus.
8. Saglabā parametrus konfigurācijā ar datumu, platformas identifikatoru un testa rezultātiem.

43.3. Enkodera CPR/TICKS pārbaude

Termini PPR, CPR un “counts per revolution” ražotāju dokumentācijā var tikt lietoti atšķirīgi. Tāpēc kursā gala patiesība ir eksperimentāli noteiktais skaits, ko tieši skaita jūsu programmas interrupt loģika.

Atkātojums	Manuālie riteņa apgriezieni	Ticks sākumā	Ticks beigās	Ticks/rev
1	5			
2	5			
3	5			

Pieņemšanas kritērijs mācību laboratorijai: trīs atkātojumu ticks/rev vērtībām jābūt vienādām vai atšķirībai jābūt izskaidrojamai ar dekodēšanas/elektrisko problēmu, nevis “noapaļošanu”.

43.4. Riteņa efektīvā rādiusa kalibrēšana taisnā kustībā

Ja odometrija rāda D_{odom} , bet fiziski nomērītais ceļš ir D_{phys} , vienkārša skalas korekcija ir:

$$k_r = D_{phys} / D_{odom}$$
$$r_{new} = r_{old} \cdot k_r$$

Piemērs: $r_{old} = 0,0325$ m, odometrija pēc testa rāda 3,10 m, bet fiziski robots nobrauc 3,00 m. Tad $k_r = 3,00/3,10 = 0,9677$ un $r_{new} \approx 0,03145$ m. Šī ir efektīvā kalibrācijas vērtība, kas ietver arī riepas deformāciju un kontakta īpatnības.

Tests	D_{phys} [m]	D_{odom} [m]	k_r	Novirze [%]	Piezīme
1	3.000				
2	3.000				
3	3.000				
4	3.000				
5	3.000				

43.5. Track width b kalibrēšana ar pagriezienu

Ja riteņu ceļa starpība ir pareiza, bet odometrijas lenķis neatbilst fiziskajam lenķim, korigē track width. Ja ar b_{old} odometrija aprēķina θ_{odom} , bet fiziski izmērīts θ_{phys} , var izmantot:

$$b_{new} = b_{old} \cdot (\theta_{odom} / \theta_{phys})$$

Piemērs: $b_{old} = 0,155$ m, odometrija aprēķina 360° , bet fiziskais pagrieziens ir 342° . $b_{new} = 0,155 \times 360/342 \approx 0,1632$ m. Pēc korekcijas testu atkārtoti abos virzienos, jo mehāniska asimetrija vai slīdēšana var dot atšķirīgu rezultātu CW un CCW.

Tests	Virziens	θ_{target} [°]	θ_{phys} [°]	θ_{odom} [°]	b_{calc} [m]
1	CW	360			
2	CCW	360			
3	CW	360			
4	CCW	360			

Tests	Virziens	θ_{target} [°]	θ_{phys} [°]	θ_{odom} [°]	b_calc [m]
5	CW	360			

43.6. Kreisās/labās puses asimetrija

Ja robots uz vienādas PWM komandas konsekventi griežas uz vienu pusi, problēma var būt motoru mehāniskā/elektriskā atšķirība, efektīva riteņa rādiusa atšķirība vai enkodera mēroga kļūda.

Closed-loop ātruma vadība samazina motoru atšķirību ietekmi, bet neatceļ ģeometrisku kļūdu.

Simptoms	Iespējamais cēlonis	Diagnostikas tests	Iespējamā korekcija
Drifts pa kreisi	labais ritenis efektīvi ātrāks/lielāks	salīdzināt ticks un fizisko ceļu katram ritenim	atsevišķs r_L/r_R vai speed loop
Pagriezieni par lielu	b par mazu modeli	360° tests	palielināt b
Pagriezieni par mazu	b par lielu modeli	360° tests	samazināt b
Attālums konsekventi par lielu	r/tick scale par lielu	3 m tests	samazināt r vai distance_per_tick
Rezultāts mainās uz dažādām gridām	slīdēšana/saķere	atkārtot uz vienādas virsmas	fiksēt testa virsmu, sensor fusion

43.7. Kalibrācijas testa protokols un statistika

Studentam jāaprēķina vismaz vidējā kļūda un izkliede. Universitātes līmenī papildina ar standartnovirzi un ticamības/nenoteiktības diskusiju.

```
mean_error = (1/n) Σ e_i
RMSE = sqrt((1/n) Σ e_i^2)
relative_error_% = 100 · (measured - reference) / reference
```

Rādītājs	Pirms kalibrācijas	Pēc kalibrācijas	Kritērijs	PASS/FAIL
3 m distance mean error			≤ 2–3 % (mācību mērķis)	
360° yaw mean error			≤ 3–5 % (mācību mērķis)	
Atkārtojamība			stabila vienā virsmā	
CW vs CCW atšķirība			izskaidrota/ dokumentēta	

43.8. Python palīgriks kalibrācijas datu aprēķinam

```
from statistics import mean, pstdev

def wheel_radius_calibration(r_old, physical_m, odom_m):
    factors = [p/o for p, o in zip(physical_m, odom_m)]
    k = mean(factors)
    return r_old * k, k, pstdev(factors)

def track_width_calibration(b_old, theta_odom_deg, theta_phys_deg):
    factors = [o/p for o, p in zip(theta_odom_deg, theta_phys_deg)]
    k = mean(factors)
    return b_old * k, k, pstdev(factors)

r_new, kr, kr_std = wheel_radius_calibration(
    0.0325,
    [3.000, 3.000, 3.000, 3.000, 3.000],
```

```
[3.10, 3.08, 3.11, 3.09, 3.10]
)
print('r_new=', r_new, 'scale=', kr, 'std=', kr_std)

b_new, kb, kb_std = track_width_calibration(
    0.155,
    [360, 360, 360, 360],
    [342, 345, 343, 344]
)
print('b_new=', b_new, 'scale=', kb, 'std=', kb_std)
```

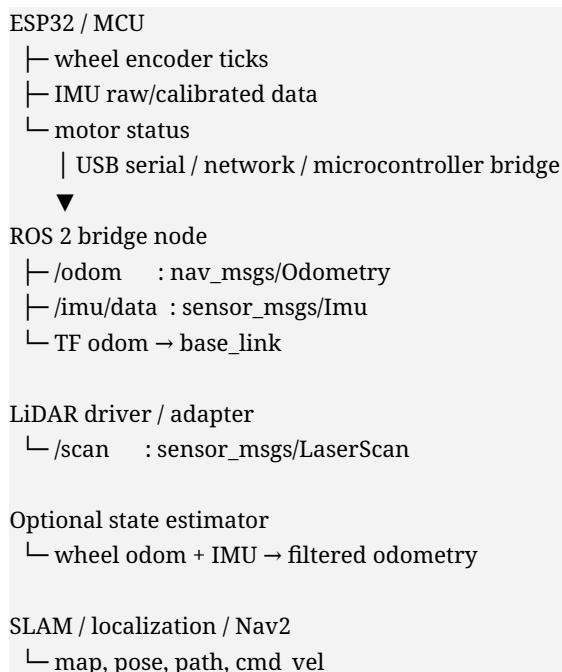
44. [ARHĪVS] Pāreja uz ROS 2: /odom, /imu, /scan un TF

Šī nodaļa sasaista kursa fizisko robotu ar ROS 2 datu modeli. Mērķis nav uzreiz uzstādīt pilnu autonomo navigāciju, bet standartizēt robota datus tā, lai dažādi mezgli varētu savstarpēji sadarboties: enkoderu odometrija kļūst par nav_msgs/Odometry, IMU dati par sensor_msgs/Imu, bet 2D LiDAR skenējums par sensor_msgs/LaserScan. Koordinātu sistēmu saites tiek publicētas ar TF.

Versiju princips: ROS 2 pakotņu API un distro laika gaitā mainās. Šajā dokumentā mācām stabilos konceptus un standarta message tipus; pirms konkrētas instalācijas pasniedzējs pārbauda izmantotā ROS 2 distributīva dokumentāciju.

44.1. Datu arhitektūra no ESP32 līdz ROS 2

Mācību platformas ROS 2 datu plūsma



44.2. Topic un frame līgums

Elements	Tips / frame	Nozīme	Kas publicē
/odom	nav_msgs/Odometry	robota lokālās pozas un ātruma novērtējums	wheel odometry / estimator
/imu/data	sensor_msgs/Imu	orientācija, angular velocity, linear acceleration	IMU driver/bridge
/scan	sensor_msgs/LaserScan	2D lāzerskenis lidar frame	LiDAR driver
/cmd_vel	geometry_msgs/Twist	vēlamais lineārais/leņķiskais ātrums	teleop/Nav2/controller
odom → base_link	dynamic TF	lokālā odometrijas koordinātu saite	odom/estimator
base_link → imu_link	static TF	IMU novietojums robotā	robot description/static broadcaster
base_link → laser_frame	static TF	LiDAR novietojums robotā	robot description/static broadcaster

Elements	Tips / frame	Nozīme	Kas publicē
map → odom	dynamic TF	globālās lokalizācijas korekcija	SLAM/localization

44.3. /odom — ko nozīmē nav_msgs/Odometry

Odometry satur pose un twist kopā ar covariance. Pose tiek interpretēta header.frame_id koordinātu sistēmā, bet twist attiecas uz child_frame_id. Mācību 2WD robotam tipiska izvēle ir header.frame_id="odom" un child_frame_id="base_link".

Lauks	Mācību platformas saturs
header.stamp	mērījuma/novērtējuma laiks
header.frame_id	odom
child_frame_id	base_link
pose.pose.position.x/y	integrēta 2WD odometrija
pose.pose.orientation	theta pārveidots quaternion
twist.twist.linear.x	$v = \Delta s / \Delta t$
twist.twist.angular.z	$\omega = \Delta \theta / \Delta t$
pose/twist covariance	nenoteiktības apraksts; nulles "tāpēc ka nezinām" nav laba profesionāla prakse

44.4. ROS 2 Python odometrijas publisher + TF

Piemērs pieņem, ka no MCU/serial bridge jau ir iegūtas x, y, theta, v un w vērtības. Lai kods būtu viegli testējams bez aparātūras, šeit izmantots vienkāršs iekšējs simulēts stāvoklis; laboratorijā update daļu aizstāj ar seriālā porta parseri.

```
import math
import rclpy
from rclpy.node import Node
from nav_msgs.msg import Odometry
from geometry_msgs.msg import TransformStamped
from tf2_ros import TransformBroadcaster

def yaw_to_quaternion(yaw):
    return (0.0, 0.0, math.sin(yaw/2.0), math.cos(yaw/2.0))

class OdomPublisher(Node):
    def __init__(self):
        super().__init__('training_odom_publisher')
        self.pub = self.create_publisher(Odometry, '/odom', 20)
        self.tf = TransformBroadcaster(self)
        self.timer = self.create_timer(0.05, self.step)
        self.x = self.y = self.th = 0.0
        self.v = 0.20
        self.w = 0.10
        self.last = self.get_clock().now()

    def step(self):
        now = self.get_clock().now()
        dt = (now - self.last).nanoseconds * 1e-9
        self.last = now

        # Replace this block with wheel odometry received from MCU.
        self.x += self.v * math.cos(self.th) * dt
```

```

self.y += self.v * math.sin(self.th) * dt
self.th += self.w * dt

qx, qy, qz, qw = yaw_to_quaternion(self.th)

msg = Odometry()
msg.header.stamp = now.to_msg()
msg.header.frame_id = 'odom'
msg.child_frame_id = 'base_link'
msg.pose.pose.position.x = self.x
msg.pose.pose.position.y = self.y
msg.pose.pose.orientation.x = qx
msg.pose.pose.orientation.y = qy
msg.pose.pose.orientation.z = qz
msg.pose.pose.orientation.w = qw
msg.twist.twist.linear.x = self.v
msg.twist.twist.angular.z = self.w

# Example covariance for a planar training robot.
# Replace with values justified by real calibration/experiments.
msg.pose.covariance[0] = 0.02 # x
msg.pose.covariance[7] = 0.02 # y
msg.pose.covariance[14] = 1e6 # z not estimated
msg.pose.covariance[21] = 1e6 # roll not estimated
msg.pose.covariance[28] = 1e6 # pitch not estimated
msg.pose.covariance[35] = 0.05 # yaw
msg.twist.covariance[0] = 0.04 # vx
msg.twist.covariance[7] = 1e6 # vy not estimated
msg.twist.covariance[14] = 1e6 # vz not estimated
msg.twist.covariance[21] = 1e6 # wx not estimated
msg.twist.covariance[28] = 1e6 # wy not estimated
msg.twist.covariance[35] = 0.08 # wz
self.pub.publish(msg)

t = TransformStamped()
t.header.stamp = msg.header.stamp
t.header.frame_id = 'odom'
t.child_frame_id = 'base_link'
t.transform.translation.x = self.x
t.transform.translation.y = self.y
t.transform.rotation = msg.pose.pose.orientation
self.tf.sendTransform(t)

def main():
    rclpy.init()
    node = OdomPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()

```

44.5. /imu/data — sensor_msgs/Imu

IMU message paredz orientāciju (quaternion), angular velocity un linear acceleration, kā arī covariance matricas. Ja orientation netiek aprēķināta MCU pusē, profesionāli jānorāda, ka šis lauks nav pieejams, nevis jāizdomā “0,0,0,1” kā it kā precīzs mērījums.

IMU dati	Ko publicēt	Vienība ROS 2
Gyro X/Y/Z	angular_velocity	rad/s
Accelerometer X/Y/Z	linear_acceleration	m/s ²
Orientation (ja ir fusion)	orientation quaternion	bezizmēra
Nenoteiktība	*_covariance	variances/covariances

Pārveidošana no deg/s uz rad/s: $\omega_{\text{rad}} = \omega_{\text{deg}} \times \pi/180$. Pārveidošana no g uz m/s²: $a = a_g \times 9,80665$.

44.6. /scan — sensor_msgs/LaserScan

LaserScan reprezentē vienu plakanā lāzera skenējumu. Būtiskie lauki ir angle_min, angle_max, angle_increment, range_min, range_max un ranges masīvs. Robota sensoru koordinātu sistēmai jābūt definētai ar frame_id, piemēram, laser_frame.

Lauks	Nozīme
header.frame_id	sensora koordinātu sistēma, piem. laser_frame
angle_min / angle_max	skenējuma leņķu robežas [rad]
angle_increment	leņķa solis starp ranges elementiem [rad]
scan_time / time_increment	laika informācija par skenējumu
range_min / range_max	derīgā attāluma robežas [m]
ranges[]	attālumi metros atbilstošajos leņķos
intensities[]	intensitāte, ja sensors to nodrošina

44.7. Vienkāršs LaserScan publisher demonstrācijai

```
import math
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan

class ScanDemo(Node):
    def __init__(self):
        super().__init__('scan_demo')
        self.pub = self.create_publisher(LaserScan, '/scan', 10)
        self.timer = self.create_timer(0.1, self.publish_scan)

    def publish_scan(self):
        n = 360
        msg = LaserScan()
        msg.header.stamp = self.get_clock().now().to_msg()
        msg.header.frame_id = 'laser_frame'
        msg.angle_min = -math.pi
        msg.angle_max = math.pi
        msg.angle_increment = 2.0 * math.pi / n
        msg.range_min = 0.12
        msg.range_max = 12.0
        msg.scan_time = 0.1
        msg.time_increment = msg.scan_time / n

        ranges = [4.0] * n
        # Synthetic obstacle in front around 0 rad.
        for deg in range(-10, 11):
            idx = (deg + 180) % 360
            ranges[idx] = 0.7
        msg.ranges = ranges
```

```

self.pub.publish(msg)

def main():
    rclpy.init()
    node = ScanDemo()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()

```

Šis piemērs ir tikai message un RViz plūsmas pārbaudei. Reālam LiDAR izmanto sensora ROS 2 driveri vai adapteri, kas nodrošina korektu timing, frame un fiziskā sensora ranges.

44.8. TF koks un statistiskie sensora transformi

Tipiska 2WD mācību robota frame struktūra

```

map
├── odom
│   ├── base_link
│   │   ├── imu_link
│   │   ├── laser_frame
│   │   └── camera_link

```

map→odom parasti nodrošina SLAM/lokalizācija; odom→base_link nodrošina odometrija vai stāvokļa novērtētājs; base_link→sensor frames ir statistiska mehāniskā ģeometrija. Nepareizs sensora TF var radīt šķietami “sliktu SLAM”, lai gan pats LiDAR mēra pareizi.

44.9. No raw encoder + IMU uz filtrētu stāvokli

```

wheel encoders → wheel odometry ─┐
                                │→ state estimator → filtered odometry
IMU ───────────────────────────┘

filtered odometry + /scan + TF → SLAM / localization → map→odom

```

State estimator mērķis nav “izlabot visu”, bet matemātiski apvienot mērījumus ar to nenoteiktību. Ja wheel odometry slīd un IMU driftē, abiem jābūt korekti kalibrētiem un covariance jāatspoguļo reālistiska uzticamība.

44.10. Laboratorija R1 — ROS 2 odometry pipeline

Mērķis: no H8 kalibrētā robota datiem izveidot `/odom` un TF `odom→base\_link`, pārbaudīt RViz un pierādīt, ka frame kustas atbilstoši fiziskajam robotam.

1. Palaid robotu uz statīva un pārbaudi enkoderu zīmes.
2. Palaid MCU seriālo telemetriju ar timestamp/ticksL/ticksR vai x/y/theta/v/w.
3. ROS 2 datorā palaid bridge node un publicē `/odom`.
4. Pārbaudi `ros2 topic echo /odom` un publicēšanas frekvenci.
5. Pārbaudi TF koku un RViz Fixed Frame = odom.
6. Veic 2 m taisno testu un 360° pagriezienu; salīdzini RViz ar fizisko rezultātu.

7. Reģistrē kļūdu un norādi, vai problēma rodas sensorā, kalibrācijā, parserī, formulā vai TF.

Test case	Expected	Actual	PASS/FAIL	Komentārs
Robot stationary	$v \approx 0$, pose stable			
Forward 2 m	x changes ~2 m			
360° turn	theta returns near start modulo 2π			
TF available	odom→base_link exists			
Encoder unplugged	fault/timeout detected			

44.11. Laboratorija R2 — IMU un frame pārbaude

1. Pārbaudi imu_link asu virzienus pret robota base_link.
2. Nekustīgam robotam reģistrē gyro bias un accelerometer mērījumus.
3. Pagriez robotu ap Z asi un pārbaudi angular_velocity.z zīmi.
4. Pārlicinies, ka vienības ir rad/s un m/s².
5. Vizualizē IMU/TF un dokumentē jebkuru asu maiņu vai montāžas rotāciju.

44.12. Laboratorija R3 — /scan un šķēršļu uztvere

1. Vispirms palaid sintētisko LaserScan publisher un pārlicinies, ka RViz redz šķērslī.
2. Pēc tam aizvieto to ar reālā LiDAR drivera /scan.
3. Pārbaudi range_min/range_max un invalid vērtības.
4. Pārbaudi base_link→laser_frame transformu.
5. Novieto objektu 0,5 m priekšā un salīdzini fizisko mērījumu ar /scan sektora minimumu.

44.13. ROS 2 diagnostikas komandu minimums

```
ros2 topic list
ros2 topic info /odom
ros2 topic echo /odom --once
ros2 topic hz /odom
ros2 topic echo /imu/data --once
ros2 topic hz /scan
ros2 node list
ros2 node info <node_name>
ros2 run tf2_ros tf2_echo odom base_link
ros2 run tf2_tools view_frames
```

44.14. Biežākās kļūdas pārejā uz ROS 2

Simptoms	Tipisks cēlonis	Ko pārbaudīt
RViz rāda robotu “sānis”	nepareizs frame/axis vai yaw quaternion	base_link asis, TF, yaw zīme
/odom kustas, robots stāv	enkodera troksnis / ticks parser	raw ticks, velocity deadband
LiDAR karte rotē kopā nepareizi	nepareizs laser_frame TF	sensora montāžas yaw
SLAM “dubulto sienas”	odometrijas kļūda, timing, TF	timestamps, calibration, scan rate

Simptoms	Tipisks cēlonis	Ko pārbaudīt
IMU padara fusion sliktāku	nepareizas vienības/axis/covariance	rad/s, m/s ² , frame, bias
Topic ir, bet patērētājs nesaņem	QoS vai nosaukuma/namespace problēma	topic info, QoS, remap

44.15. Avoti un versiju pārbaude

Šīs nodaļas standarta message tipu un TF principi saskaņoti ar ROS 2 oficiālo dokumentāciju: sensor_msgs (Imu, LaserScan), nav_msgs (Odometry) un tf2_ros TransformBroadcaster. ESP32 LEDC un I2C piemēros izmantota Espressif Arduino-ESP32 oficiālās dokumentācijas aktuālā API pieeja. Pirms laboratorijas pasniedzējs fiksē konkrēto ROS 2 distributīvu, Arduino-ESP32 core un trešo pušu bibliotēku versijas.

Bloks	v0.8 statuss	Nākamais attīstības slānis
H1–H12 darba lapas	izstrādātas izdrukai	atsevišķas pasniedzēja aizpildītās etalona lapas
Arduino/ESP32 kodi	etalona kodi pievienoti	vienota Git repozitorija struktūra un unit/integration testi
2WD kalibrācija	pilna metodika + protokoli	automātiska CSV analīze un grafiki
ROS 2 /odom	message + TF + laboratorija	reāls serial parser/microcontroller bridge
ROS 2 /imu	datu modelis + laboratorija	EKF konfigurācijas laboratorija
ROS 2 /scan	datu modelis + demo + laboratorija	reāla LiDAR drivera integrācija
Autonomā navigācija	arhitektūra sagatavota	SLAM Toolbox/Nav2 praktiskā secība, recovery behavior, costmaps
Universitātes līmenis	matemātiskie pamati jau dokumentā	kontroles laboratorijas, EKF/SLAM datu analīze, eksāmena banka
Turpinājums: Nākamā loģiskā dokumenta versija: v0.9 ar ROS 2/SLAM/Nav2 praktisko laboratoriju komplektu, robot_localization EKF konfigurāciju, bag datu analīzi, Git/GitHub kursa darba plūsmu un pilnu eksāmena/jautājumu banku koledžas un universitātes līmenim.		

45. Stāvokļa novērtēšana: EKF un robot_localization

Šis nodaļas mērķis ir pāriet no atsevišķiem sensoriem uz vienotu robota stāvokļa novērtējumu. Riteņu enkoderi labi raksturo relatīvo kustību, bet slīd; IMU labi mēra īstermiņa rotācijas dinamiku, bet tai ir bias un drift; ārējie pozīcijas avoti var būt lēnāki vai periodiski nepieejami. Stāvokļa novērtētāja uzdevums ir apvienot mērījumus, ņemot vērā to nenoteiktību, nevis vienkārši “izvēlēties vienu labāko sensoru”.

Profesionālā robotikā svarīgs ir ne tikai novērtējums x , y un yaw , bet arī uzticamības apraksts. Tāpēc ROS ziņojumos pose, twist un IMU dati satur covariance laukus. Ja covariance ir nereālistiska, filtrs var pārlietu uzticēties sliktam sensoram vai ignorēt labu sensoru.

45.1. Robota stāvokļa vektors

Stāvokļa vektors ir mainīgo kopa, ar kuru matemātiski apraksta sistēmu noteiktā laika momentā. Vienkāršam 2D mobilam robotam minimālais stāvoklis bieži ir x , y un yaw . Dinamiskākā modelī pievieno lineāros un leņķiskos ātrumus, bet pilnā 3D modelī arī z , roll, pitch un paātrinājumus.

Stāvokļa vektora piemēri

Vienkāršs 2D stāvoklis:

$$\mathbf{x}_k = [x, y, \theta]^T$$

Paplašināts 2D stāvoklis:

$$\mathbf{x}_k = [x, y, \theta, v_x, v_y, \omega_z]^T$$

3D stāvokļa piemērs:

$$\begin{aligned} &[x, y, z, \text{roll}, \text{pitch}, \text{yaw}, \\ &v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, \\ &a_x, a_y, a_z]^T \end{aligned}$$

Pasniedzējam uzsvērt: stāvoklis nav “visi sensora dati”. Stāvoklis ir tas, ko sistēmai nepieciešams novērtēt un izmantot turpmākai vadībai, lokalizācijai vai plānošanai.

45.2. Kāpēc vajadzīgs filtrs?

- Sensors satur troksni: pat nekustīgs IMU vai attāluma sensors nerāda pilnīgi konstantu vērtību.
- Sensoriem ir sistemātiskas kļūdas: gyro bias, enkodera mēroga kļūda, riteņu diametra atšķirība.
- Sensori darbojas ar atšķirīgu frekvenci un aizturi.
- Daži sensori mēra relatīvu kustību, citi absolūtāku stāvokli.
- Sensori var īslaicīgi pazust, atgriezt nederīgu vērtību vai kļūt neuzticami noteiktos apstākļos.

Filtrs nav paredzēts, lai “izglābtu” nepareizi kalibrētu robotu. Ja enkodera zīmes ir sajauktas, TF ir nepareizs vai IMU ass ir apgriezta, matemātiskais filtrs tikai padarīs kļūdu grūtāk diagnosticējamu. Tāpēc R1–R3 laboratorijās vispirms tiek pierādīta katra sensora plūsma atsevišķi.

45.3. Kalman filtra pamatideja: prognoze un korekcija

Kalman tipa filtru var skaidrot kā atkārtotu divu darbību ciklu. Prognozes soli kustības modelis aprēķina, kur robotam vajadzētu atrasties pēc iepriekšējā stāvokļa un kustības. Korekcijas soli jaunais sensora mērījums korigē prognozi. Abu avotu ietekme ir atkarīga no nenoteiktības.

EKF cikla konceptuālās formulas

1. PREDICT

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k)$$

$$P_k^- = F P_{k-1} F^T + Q$$

2. CORRECT

$$\text{Innovation: } y_k = z_k - h(\hat{x}_k^-)$$

$$S_k = H P_k^- H^T + R$$

$$K_k = P_k^- H^T S_k^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k y_k$$

$$P_k = (I - K_k H) P_k^-$$

Koledžas līmenī nav jāizved katra matrica. Galvenais ir saprast lomas: Q raksturo procesa/modelēšanas nenoteiktību, R – mērījuma nenoteiktību, P – stāvokļa novērtējuma nenoteiktību, bet Kalman gain K nosaka, cik lielā mērā korekcija seko jaunajam mērījumam.

45.4. Kāpēc EKF, nevis parasts Kalman filtrs?

Klasiskais Kalman filtrs pieņem lineāru sistēmas un mērījumu modeli. Robota kinemātikā ir sinuss, kosinuss, leņķu normalizācija un citas nelinearitātes. Extended Kalman Filter (EKF) lokāli linearizē nelineāro modeli ap pašreizējo novērtējumu, izmantojot Jacobian matricas. Tas ļauj Kalman principu izmantot daudzām robotikas sistēmām.

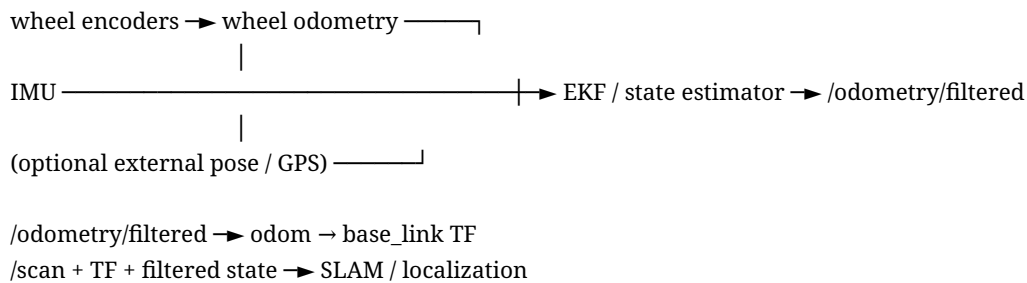
Universitātes līmenī jāspēj izskaidrot, ka linearizācija ir lokāls tuvinājums un ļoti nelineāros gadījumos, nepareizas inicializācijas vai lielu kļūdu apstākļos EKF var strādāt slikti vai diverģēt.

45.5. robot_localization loma ROS 2 sistēmā

robot_localization ir stāvokļa novērtēšanas pakotne, kas paredzēta dažādu pozas, ātruma un IMU avotu sapludināšanai. Mācību platformā tā ir loģisks nākamais solis pēc /odom un /imu/data

sakārtošanas. Filtra izvade var kļūt par stabilāku odometrijas avotu, ko izmanto SLAM un navigācija.

Mācību robota sensor fusion datu plūsma



45.6. 2D konfigurācijas principi

Divriteņu mobilam robotam mācību telpā vispirms jārisina 2D problēma. Tas samazina nezināmo skaitu un ļauj koncentrēties uz x, y, yaw, lineāro kustību uz priekšu un rotāciju ap Z asi. Ja robots fiziski nepārvietojas pa z asi un netiek izmantots roll/pitch navigācijas modelī, filtra 2D režīms ir praktisks sākumpunkts.

Svarīga inženiertehniska izvēle ir neapvienot vienu un to pašu fizisko informāciju vairākas reizes tā, it kā tie būtu neatkarīgi mērījumi. Piemēram, ja wheel odometry pose un twist abi ir aprēķināti no tiem pašiem enkoderiem, mācību konfigurācijā bieži ir saprātīgi filtrā izmantot enkoderu ātruma komponentes, nevis vienlaicīgi piešķirt lielu uzticību gan pose, gan twist no viena avota.

45.7. EKF YAML mācību piemērs

Zemāk redzamais ir mācību konfigurācijas skelets. Tas jāpielāgo tieši instalētajai robot_localization versijai, topic nosaukumiem un sensora kvalitātei. Boolean konfigurācijas vektors robot_localization ROS 2 piemērā atbilst 15 stāvokļa komponentēm: x, y, z, roll, pitch, yaw, v_x, v_y, v_z, v_roll, v_pitch, v_yaw, a_x, a_y, a_z.

robot_localization EKF mācību konfigurācijas skelets

```
ekf_filter_node:
  ros__parameters:
    frequency: 30.0
    sensor_timeout: 0.2
    two_d_mode: true
    publish_tf: true

    map_frame: map
    odom_frame: odom
    base_link_frame: base_link
    world_frame: odom

    odom0: /wheel/odom
    odom0_config: [false, false, false,
                  false, false, false,
                  true, false, false,
                  false, false, true,
                  false, false, false]

    imu0: /imu/data
    imu0_config: [false, false, false,
                 false, false, false,
                 false, false, false,
                 false, false, true,
                 false, false, false]

    imu0_remove_gravitational_acceleration: true
```

Šajā sākuma variantā wheel odometry dod v_x un yaw rate, bet IMU dod yaw rate kā otru neatkarīgu dinamisku mērījumu. Ja IMU orientācija ir kvalitatīvi aprēķināta un covariance ir ticama, vēlāk var izmēģināt arī orientation/yaw sapludināšanu. Ja IMU orientation lauks nav derīgs, to nedrīkst “aizpildīt ar nullēm” un pasniegt kā reālu mērījumu.

45.8. Covariance — sensora uzticamība skaitļos

Covariance matrica apraksta mērījuma nenoteiktību un savstarpējās saistības starp komponentēm. Mācību kursā sākumā pietiek ar diagonāles interpretāciju: mazāka dispersija nozīmē lielāku uzticību attiecīgajam mērījumam, lielāka dispersija – mazāku uzticību.

Piemērs:

$$\sigma_v = 0.05 \text{ m/s} \rightarrow \text{variance} = \sigma^2 = 0.0025$$
$$\sigma_\omega = 0.03 \text{ rad/s} \rightarrow \text{variance} = 0.0009$$

Nav korekti iestatīt ļoti mazu covariance tikai tāpēc,
ka vēlamies, lai filtrs “vairāk klausītu” konkrēto sensoru.
Covariance jāpamato ar mērījumiem.

Labs laboratorijas uzdevums ir nekustīgam robotam 30–60 sekundes reģistrēt IMU un enkoderu datus, aprēķināt standartnovirzi un izmantot to kā sākotnējo covariance novērtējuma pamatu. Pēc tam jāveic dinamiskais tests, jo sensora uzvedība kustībā var atšķirties.

45.9. Innovation un filtra diagnostika

Innovation ir starpība starp sagaidīto mērījumu un faktiski saņemto mērījumu mērījumu telpā. Ja innovation ilgstoši ir liels vienā virzienā, tas var norādīt uz sensoru bias, nepareizu mērogu, laika nobīdi, TF kļūdu vai nepareizu modeli. Filtra “gluds grafiks” nav pierādījums, ka rezultāts ir pareizs.

Diagnostikas jautājumi pasniedzējam un studentam

- Vai sensori izmanto vienas un tās pašas SI mērvienības?
- Vai timestamp ir monotons un pietiekami precīzs?
- Vai frame_id un sensora mehāniskais TF ir pareizi?
- Vai covariance ir reālistiska, nevis patvaļīga?
- Vai wheel odometry mērogs un track width ir kalibrēts pirms filtra?
- Vai datu frekvences ir pietiekamas un vai nav liela aizture?
- Kas notiek, kad viens sensors tiek atvienots?

45.10. R4 laboratorija — wheel odometry + IMU → filtrēta odometrija

Mērķis: izveidot un eksperimentāli novērtēt vienkāršu EKF sensor fusion konfigurāciju. Studentam jāspēj pierādīt, ka filtrs ne tikai publicē datus, bet arī saprast, kā mainās rezultāts, ja maina sensora uzticamību vai izslēdz vienu no avotiem.

1. Atkārto R1 un R2 validāciju: /wheel/odom un /imu/data atsevišķi ir pareizi.
2. Reģistrē 30 s nekustīga robota datus un novērtē gyro Z trokšņa standartnovirzi.
3. Izveido robot_localization konfigurācijas failu ar two_d_mode un diviem ievades avotiem.
4. Palaid filtru un pārbaudi /odometry/filtered publicēšanas frekvenci.
5. Pārbaudi TF: nedrīkst būt divi mezgli, kas vienlaikus publicē pretrunīgu odom→base_link transformu.
6. Veic 2–3 m taisnās kustības testu un 360° pagriezienu.
7. Atkārto testu ar IMU ievadi izslēgtu un salīdzini yaw kļūdu.
8. Atkārto testu ar konservatīvāku/agresīvāku IMU covariance un dokumentē ietekmi.
9. Reģistrē visus topic ar rosbag2 turpmākai analīzei.

Testa punkts	Reference	Wheel odom	Filtered odom	Secinājums
Taisne	2.000 m			
Pagrieziens	360°			

Testa punkts	Reference	Wheel odom	Filtered odom	Secinājums
Nekustīgs 30 s	yaw $\approx$ const			
IMU atvienošana	sistēma turpina ar degradētu kvalitāti			

45.11. Vienkāršs offline 1D Kalman demonstrators Python

Šis nav robot_localization aizstājējs. Tas ir didaktisks eksperiments, lai vizuāli redzētu prediction/correction principu vienā dimensijā.

1D Kalman principa demonstrācija

```
import random

x = 0.0      # state estimate
P = 1.0      # estimate variance
Q = 0.02     # process variance
R = 0.25     # measurement variance

true_x = 0.0
for k in range(30):
    true_x += 1.0
    z = true_x + random.gauss(0.0, R**0.5)

    # predict
    x = x + 1.0
    P = P + Q

    # correct
    K = P / (P + R)
    x = x + K * (z - x)
    P = (1 - K) * P

    print(k, round(z, 3), round(x, 3), round(P, 4))
```

Avoti un versiju piezīme

Šajā nodaļā izmantota oficiālā projekta/ražotāja dokumentācija. ROS 2 pakotņu parametri, launch faili un noklusējuma konfigurācijas var atšķirties starp distribūcijām, tādēļ laboratorijā vienmēr jāpārbauda tieši instalētās versijas dokumentācija un pakotnes piemēra konfigurācija.

- robot_localization projekta ROS 2 ekf.yaml piemēra konfigurācija (cra-ros-pkg/robot_localization).
- ROS 2 message tipi: nav_msgs/Odometry un sensor_msgs/Imu.
- Kursa 44. nodaļas /odom, /imu un TF līgums.

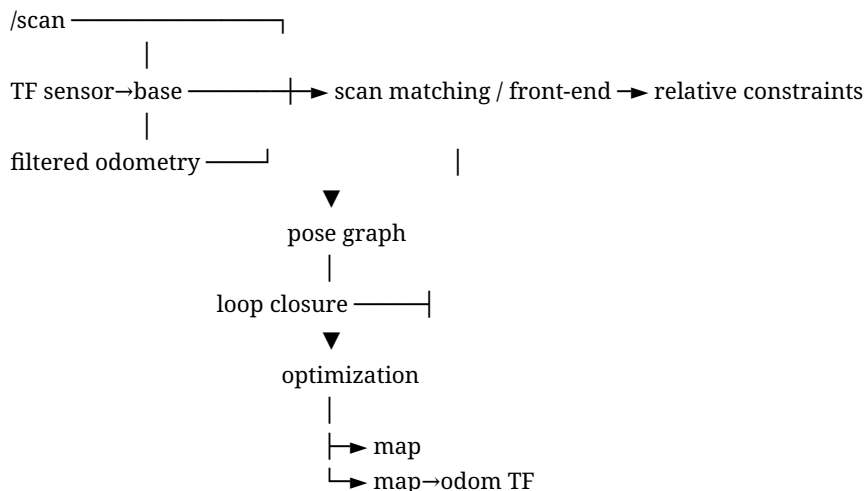
46. SLAM: kartēšana, lokalizācija un slam_toolbox

SLAM — Simultaneous Localization and Mapping — risina savstarpēji saistītu problēmu: robotam jāveido apkārtējās vides karte, vienlaikus novērtējot savu atrašanās vietu šajā kartē. Ja pozīcija ir

nepareiza, punkti kartē tiek novietoti nepareizi; ja karte ir nepareiza, lokalizācija kļūst neuzticama. Tāpēc SLAM ir optimizācijas un datu asociācijas problēma, nevis tikai “LiDAR punktu uzzīmēšana”.

46.1. SLAM informācijas ķēde

2D LiDAR SLAM konceptuālā arhitektūra



46.2. Front-end un back-end

SLAM front-end apstrādā sensoru mērījumus un meklē relatīvas kustības/atbilstības starp skenējumiem vai pazīmēm. 2D LiDAR sistēmā centrālā metode ir scan matching: jaunā lāzera skenējuma ģeometriju salīdzina ar iepriekšējo kartes vai skenējumu informāciju.

Back-end uzkrāj ierobežojumus starp robota pozām un optimizē pose graph. Kad robots atgriežas jau apmeklētā vietā, loop closure var radīt jaunu ierobežojumu, kas koriģē uzkrāto driftu. Rezultātā var mainīties ne tikai pašreizējā poza, bet arī iepriekš izveidotās trajektorijas daļas.

46.3. Scan matching

Scan matching meklē transformāciju starp skenējumiem vai skenējumu un karti. Ja sākuma pozas minējums ir pietiekami labs, algoritms var precizēt translāciju un rotāciju. Ja sākuma kļūda ir pārāk liela, vide ir ļoti simetriska vai LiDAR redz maz struktūras, var rasties nepareiza atbilstība.

Vide, kur SLAM kļūst grūtāks

- Gari vienādi koridori bez unikāliem orientieriem.
- Lieli atvērti laukumi ar maz šķēršļiem.
- Daudz kustīgu cilvēku vai pārvietojamu objektu.
- Stikls, spoguļi vai virsmas ar sarežģītu LiDAR atstarošanu.
- Liela riteņu slīdēšana un strauja rotācija.
- Nepareizs laser_frame TF vai sensora laika nobīde.

46.4. Pose graph un loop closure

Pose graph mezgli reprezentē robota pozas noteiktos laika momentos vai skenējumu mezglus, bet malas – relatīvus ierobežojumus starp tiem. Odometrija dod secīgus kustības ierobežojumus, scan

matching papildina sensora ģeometrijas ierobežojumus, un loop closure pievieno saikni starp attāliem trajektorijas momentiem, kas patiesībā atrodas vienā un tajā pašā vietā.

Universitātes līmenī pose graph optimizāciju var formulēt kā izmaksu funkcijas minimizēšanu, kur katrs mērījuma ierobežojums veido kļūdas terminu ar informācijas/covariance svaru.

46.5. slam_toolbox mācību loma

slam_toolbox ir ROS 2 2D SLAM rīku kopa. Kursā to izmanto kā praktisku tiltu starp teoriju un reālu robotu: /scan, TF un odometrijas dati tiek izmantoti, lai veidotu 2D occupancy grid un pose graph. Projekts atbalsta kartēšanas un lokalizācijas darba plūsmas, kā arī kartes/pose graph saglabāšanu un atkārtotu izmantošanu.

46.6. Pirms SLAM palaišanas jābūt pierādītam

- /scan ir fiziski pareizā mērogā un priekšējais virziens atbilst laser_frame asīm.
- base_link→laser_frame statiskais TF atbilst faktiskajai montāžai.
- odom→base_link ir nepārtraukts un bez pēkšņiem lēcieniem normālā kustībā.
- IMU/odometrijas sensori ir kalibrēti un timestamp nav acīmredzami nepareizs.
- Robota kustības ātrums ir piemērots LiDAR skenēšanas frekvencei.
- Nav divu mezglu, kas vienlaikus publicē vienu un to pašu TF ar atšķirīgu saturu.

46.7. Occupancy grid interpretācija

2D occupancy grid sadala telpu šūnās. Katra šūna reprezentē brīvas, aizņemtas vai nezināmas telpas novērtējumu. Kartes resolution nosaka šūnas fizisko izmēru. Mazāka šūna dod lielāku ģeometrisku detalizāciju, bet palielina atmiņas un aprēķinu prasības un ne vienmēr uzlabo karti, ja sensora precizitāte ir zemāka.

```
#####  
#.....#  # = occupied  
#..R....#  . = free  
#....???#  ? = unknown  
#####
```

46.8. Kvalitatīvas kartēšanas stratēģija

- Sāc ar mazu, ģeometriski saprotamu telpu, nevis visu ēku.
- Brauc vienmērīgi un izvairies no ļoti straujām rotācijām.
- Veido cilpas: atgriezies jau kartētā vietā, lai būtu iespēja loop closure.
- Saglabā rosbag paralēli kartēšanai, lai kļūdas var atkārtot offline.
- Dokumentē robota ātrumu, LiDAR modeli, TF un konfigurācijas versiju.
- Salīdzini kartes dimensijas ar fiziskiem references mērījumiem.

46.9. R5 laboratorija — izveido, novērtē un saglabā karti

Mērķis: ar kalibrētu 2WD robotu izveidot nelielas telpas vai koridora 2D karti un tehniski pamatot kartes kvalitāti. Darba rezultāts nav tikai ekrānuzņēmums; jābūt testa protokolam, rosbag datiem un kļūdu analīzei.

1. Izvēlies telpu ar vismaz 4–6 labi identificējamiem ģeometriskiem orientieriem.

2. Nomēri divus references attālumus ar mērlenti (piem., sienu attālums un koridora platums).
3. Palaid /odometry/filtered, /scan un TF diagnostiku.
4. Palaid slam_toolbox instalētajai distribūcijai atbilstošā mapping režīmā.
5. Reģistrē /scan, /tf, /tf_static, /odom vai /odometry/filtered un /map ar rosbag2.
6. Brauc lēni pa trajektoriju ar vismaz vienu atgriešanos sākuma zonā.
7. Saglabā karti un, ja iespējams, arī pose graph sesiju.
8. Salīdzini kartē izmērītos attālumus ar references mērījumiem.
9. Apraksti vismaz trīs redzamus artefaktus vai pierādi, ka tie nav novēroti.

Kritērijs	Reference	Rezultāts	Novirze	PASS/FAIL
Sienu attālums				
Koridora platums				
Loop closure	trajektorija aizveras			
Dubultās sienas	nav/izskaidrots			
TF stabilitāte	bez lēcieniem			

46.10. Tipiskie SLAM artefakti un diagnoze

Simptoms	Iespējamais cēlonis	Pārbaude
Dubultas/paralēlas sienas	yaw drift, slikts scan matching, nepareizs TF	rosbag replay; salīdzināt odom, scan, TF
Karte lēnām “lokās”	uzkrāts odometrijas drift, nav loop closure	braukt slēgtu cilpu; pārbaudīt track width
Skenējums RViz “peld”	nepareizs timestamp/TF vai kustības distorsija	nekustīgs tests un tf2_echo
Siena ir nepareizā attālumā	LiDAR mērogs vai kartes interpretācija	fizisks references mērījums
Pēc loop closure liels lēciens	iepriekš uzkrāts drift vai kļūdaina atbilstība	analizēt trajektoriju un pose graph

Avoti un versiju piezīme

Šajā nodaļā izmantota oficiālā projekta/ražotāja dokumentācija. ROS 2 pakotņu parametri, launch faili un noklusējuma konfigurācijas var atšķirties starp distribūcijām, tādēļ laboratorijā vienmēr jāpārbauda tieši instalētās versijas dokumentācija un pakotnes piemēra konfigurācija.

- slam_toolbox ROS 2 projekta dokumentācija un README (SteveMacenski/slam_toolbox).
- ROS 2 Nav2 dokumentācijas SLAM un navigācijas integrācijas principi.

47. Nav2: costmaps, plānotāji, kontrolieri un Behavior Trees

Nav2 nav viens “ceļa meklēšanas algoritms”. Tā ir moduļu navigācijas sistēma: robotam vajadzīga lokalizācija un TF, pasaules reprezentācija costmap formā, globālā ceļa plānošana, lokāla kustības vadība, uzvedību koordinācija un kļūmju/recovery mehānismi. Šī arhitektūra ir labs piemērs tam, kā lielu autonomu sistēmu sadala neatkarīgās, testējamās funkcijās.

47.1. Nav2 functional tree

Nav2 funkcionālais koks

```
To navigate autonomously to a goal
├─ To know robot state
│   └─ receive odometry / state estimate
│       └─ maintain map→odom→base_link TF
├─ To represent environment
│   ├── maintain global costmap
│   ├── maintain local costmap
│   ├── insert obstacles
│   └─ inflate obstacle costs
├─ To plan motion
│   ├── compute global path
│   └─ replan when required
├─ To control motion
│   ├── follow path
│   ├── obey kinematic limits
│   └─ output velocity commands
├─ To evaluate progress and goal
└─ To handle failures
    ├── contextual recovery
    ├── clear costmap
    ├── spin / wait / back up
    └─ fail safely / request operator
```

47.2. Galvenie Nav2 serveri un to lomas

Komponents	Galvenā loma	Tipisks jautājums
Planner server	aprēķina globālu ceļu	Pa kuriem nokļūt līdz mērķim?
Controller server	ģenerē kustības komandas ceļa sekošanai	Kā braukt nākamajā laika solī?
Costmap2D	uztur šķēršļu un izmaksu modeli	Kur ir droši/nevēlami braukt?
BT Navigator	orķestrē navigācijas uzvedību	Ko darīt secīgi un ko darīt kļūmes gadījumā?
Behavior server	izpilda atsevišķas uzvedības/recovery	Vai jāatkāpjas, jāgaida, jāpagriežas?
Map/localization	nodrošina karti un pozas novērtējumu	Kur robots atrodas kartē?

47.3. Global costmap un local costmap

Global costmap parasti aptver lielāku navigācijas telpu un tiek izmantots globālajai plānošanai. Local costmap parasti ir mazāks rolling window ap robotu un tiek izmantots lokālajam kontrolēšanai, lai reaģētu uz tuvumā esošiem šķēršļiem un dinamisku vidi.

Svarīgi atšķirt karti no costmap. Karte apraksta vides occupancy informāciju, bet costmap ir navigācijas izmaksu modelis, kurā var apvienot statisko karti, sensoru šķēršļus, inflation zonu un citus slāņus.

47.4. Costmap slāņi

Slānis	Ko tas dara	Ko students novēro RViz
Static layer	ievieto iepriekšēju occupancy karti	pastāvīgās sienas/šķēršļi
Obstacle layer	ievieto/attīra šķēršļus no sensoriem	jauni objekti parādās/pazūd
Voxel layer	3D/voxel tipa šķēršļu modelis, ja vajadzīgs	augstuma informācijas ietekme uz 2D costmap
Inflation layer	palielina izmaksas ap šķēršļiem	drošības “halo” ap sienām/objektiem

47.5. Inflation un robota footprint

Robots nav punkts. Costmap konfigurācijā jāņem vērā reālais footprint vai radius. Inflation layer ne tikai aizliedz šūnas, bet veido izmaksu gradientu ap šķēršļiem, lai plānotājs/kontrolieris preferētu drošāku attālumu. Pārāk maza inflation zona var novest pie braukšanas bīstami tuvu sienai; pārāk liela var aizvērt šaurus koridorus, kas fiziski ir izbraucami.

Laboratorijā footprint jāsalīdzina ar reālā robota platumu, ieskaitot izvirzītus sensorus vai konstrukcijas daļas. Ja tiek izmantots tikai base šasijas izmērs, bet LiDAR vai kamera izvirzās sānis, collision model var būt nepilnīgs.

47.6. Global planner

Globālā plānotāja uzdevums ir atrast ceļu no starta līdz mērķim costmap telpā. Dažādi plānotāji atšķiras ar izmantoto grafu/režģi, kinemātikas ierobežojumu modelēšanu, optimizācijas kritēriju un ātrdarbību. Kursā algoritmu pamatos apgūtais A* ļauj saprast vienu no centrālajām idejām: meklēšanas telpa, izmaksas un heuristika.

Koledžas līmenī studentam jāspēj atšķirt “ceļš eksistē kartē” no “robots fiziski spēj tam sekot”. Universitātes līmenī jāanalizē arī kinemātiska izpildāmība, curvature un cost funkcijas.

47.7. Local controller

Kontrolieris saņem ceļu un pašreizējo stāvokli un ģenerē kustības komandas. Tas strādā atkārtoti ar daudz lielāku frekvenci nekā vienreizējs globālais plāns. Kontrolierim jāņem vērā robota ātruma, paātrinājuma, pagriezienu un collision ierobežojumi. Ja vide mainās, local costmap ietekmē kontroliera izvēles.

47.8. Behavior Trees kā augsta līmeņa autonomijas loģika

Behavior Tree (BT) ir hierarhiska uzvedību struktūra. Nav2 BT Navigator izmanto BT, lai koordinētu darbības, nosacījumus, atkārtotu plānošanu un recovery. Tas ir ļoti labs turpinājums kursa state machine tēmai: vienkāršai sistēmai FSM ir pārskatāma, bet sarežģītā navigācijā BT dod modulārāku un atkārtoti izmantojamu struktūru.

Vienkāršota mācību Behavior Tree XML ideja

```
<root BTCPP_format="4" main_tree_to_execute="MainTree">
  <BehaviorTree ID="MainTree">
    <PipelineSequence name="NavigateWithReplanning">
      <ComputePathToPose goal="{goal}" path="{path}"/>
      <FollowPath path="{path}"/>
    </PipelineSequence>
  </BehaviorTree>
</root>
```

Konkrēti BT node parametri un plugin nosaukumi jāņem no instalētās Nav2 versijas. Mācību mērķis ir saprast struktūru: plānošana → sekošana → kļūmes atpazīšana → kontekstuāls recovery → atkārtots mēģinājums vai droša atteice.

47.9. Recovery/behavior loģika

Nav2 dokumentācijā recovery koncepcija tiek izmantota, lai autonomi mēģinātu atrisināt īslaicīgas navigācijas problēmas. Piemēri ir costmap attīrīšana, pagriešanās, gaidīšana vai atkāpšanās. Recovery nav “burvju poga”: jābūt definētam, kāda kļūme konkrēto darbību varētu atrisināt un kad turpmākie mēģinājumi vairs nav droši vai jēgpilni.

```
Navigation action
|
|— SUCCESS → continue mission
|
|— FAILURE
|   |— contextual recovery
|   |   |— clear relevant costmap
|   |   |   |— retry
|   |— system recovery
|   |   |— clear maps
|   |   |— spin / wait / backup
|   |   |— retry or fail safely
```

47.10. Costmap konfigurācijas mācību skelets

Mācību costmap YAML skelets

```
local_costmap:
  local_costmap:
    ros__parameters:
      global_frame: odom
      robot_base_frame: base_link
      rolling_window: true
      width: 3.0
```

```
height: 3.0
resolution: 0.05
plugins: ["obstacle_layer", "inflation_layer"]
```

```
obstacle_layer:
  plugin: "nav2_costmap_2d::ObstacleLayer"
  observation_sources: scan
  scan:
    topic: /scan
    data_type: "LaserScan"
    marking: true
    clearing: true
```

```
inflation_layer:
  plugin: "nav2_costmap_2d::InflationLayer"
  inflation_radius: 0.35
```

Šīs vērtības nav universāls “pareizais” iestatījums. Width/height jāsaista ar robota ātrumu un kontroliera redzes horizontu; resolution – ar kartes un sensora vajadzīgo detalizāciju; inflation_radius – ar footprint un drošības rezervi.

47.11. R6 laboratorija — costmap izpratne un sensoru validācija

1. Palaid karti/lokalizāciju vai SLAM un pārlicinies par map→odom TF.
2. Palaid Nav2 costmaps bez navigācijas mērķa.
3. RViz parādi global un local costmap atsevišķi.
4. Novieto kastīti 0,5–1 m priekšā LiDAR un pārbaudi obstacle marking.
5. Noņem kastīti un pārbaudi clearing uzvedību.
6. Maini inflation_radius kontrolētā eksperimentā un saglabā ekrānattēlus.
7. Salīdzini footprint ar reālo robota izmēru.

47.12. R7 laboratorija — NavigateToPose

Mērķis: nosūtīt vienu mērķa pozu un pierādīt pilnu ķēdi localization → costmap → planner → controller → cmd_vel → motors.

1. Pārbaudi, ka robots stāv pareizajā pozā kartē.
2. Nosūti mērķi brīvā telpā ar RViz NavigateToPose.
3. Saglabā global path un local costmap novērojumus.
4. Reģistrē /cmd_vel, /odometry/filtered, /scan un navigācijas statusu rosbag2.
5. Atkārto ar mērķi šaurāku šķēršļu tuvumā un salīdzini trajektoriju.
6. Novērtē gala pozīcijas kļūdu metros un leņķa kļūdu grādos.

Mēģinājums	Goal x/y/yaw	Faktiskais rezultāts	Pozīcijas kļūda	Laiks	Statuss
1					
2					
3					

47.13. R8 laboratorija — Behavior Tree un recovery eksperiments

Mērķis: nevis speciāli “salauzt robotu”, bet kontrolētā vidē izraisīt droši atgriezenisku navigācijas problēmu un novērot, kā BT mēģina to risināt.

1. Izvēlies drošu testu ar nelielu pārvietojamu šķērslī, nevis fiziski iesprostotu robotu.
2. Palaid navigāciju uz mērķi un dokumentē normālo BT izpildi.
3. Ievieto šķērslī tā, lai sākotnējais ceļš kļūst nederīgs, bet ir iespējams alternatīvs ceļš.
4. Novēro replanning un local costmap reakciju.
5. Atsevišķā testā izveido situāciju, kur planner/controller īslaicīgi nevar turpināt, un dokumentē recovery darbības.
6. Definē stop nosacījumu: pasniedzējs jebkurā brīdī var apturēt robotu, ja recovery noved pie nedrošas kustības.

47.14. Nav2 diagnostikas domāšanas secība

```

Goal not reached?
|
├─ Is localization correct?
|   └─ NO → fix map/TF/state estimation first
|
├─ Does global costmap represent world correctly?
|   └─ NO → map/layers/TF/sensor issue
|
├─ Does planner generate a path?
|   └─ NO → goal/costmap/planner/footprint issue
|
├─ Does controller produce reasonable cmd_vel?
|   └─ NO → controller limits/path/local costmap issue
|
├─ Does base execute cmd_vel correctly?
|   └─ NO → motor/control/bridge issue
|
└─ Are recoveries helping or hiding root cause?

```

Avoti un versiju piezīme

Šajā nodaļā izmantota oficiālā projekta/ražotāja dokumentācija. ROS 2 pakotņu parametri, launch faili un noklusējuma konfigurācijas var atšķirties starp distribūcijām, tādēļ laboratorijā vienmēr jāpārbauda tieši instalētās versijas dokumentācija un pakotnes piemēra konfigurācija.

- Nav2 oficiālā dokumentācija: Navigation Concepts, Behavior Trees un Costmap2D konfigurācijas materiāli (docs.nav2.org).
- ros-navigation/navigation2 projekta README un nav2_bringup konfigurācijas piemēri.

48. rosbag2: datu registrēšana, atkārtošana un eksperimentu analīze

Robotikas sistēmā kļūdu bieži nevar atrisināt ar frāzi “tagad vairs neatkārtojas”. Lai problēmu analizētu, nepieciešams saglabāt sensoru, stāvokļa un vadības datus ar laiku. rosbag2 ļauj ierakstīt ROS 2 topic datu plūsmu un vēlāk to atkārtot. Tas pārvērš vienreizēju fizisku testu atkārtojamā datu kopā.

48.1. Kāpēc bag ir profesionāls instruments

- Atdalīt datu iegūšanu no algoritma analīzes.
- Atkārtot vienu un to pašu sensora scenāriju pēc koda izmaiņām.
- Salīdzināt divas algoritma/config versijas ar identisku ievadi.
- Veikt incidenta root-cause analīzi pēc testa.
- Saglabāt pierādījumus laboratorijas protokolam un projekta aizstāvēšanai.
- Radīt offline dataset vizualizācijai, ML vai algoritmu prototipēšanai.

48.2. Minimālā komandu secība

rosbag2 pamata CLI piemēri

```
# record selected topics
ros2 bag record /odom /odometry/filtered /imu/data /scan /tf /tf_static /cmd_vel -o run_001

# inspect metadata
ros2 bag info run_001

# replay
ros2 bag play run_001

# record all topics (use carefully: data volume can grow quickly)
ros2 bag record -a -o full_run_001
```

Pasniedzējam: pirms “record -a” jāizskaidro datu apjoms, kameru topic iespējamais liels datu ātrums, personas datu/privātuma jautājumi un diska vietas kontrole. Mācību laboratorijās labāk sākumā ierakstīt tikai konkrētajam testam vajadzīgos topic.

48.3. Testa datu komplekta struktūra

Ieteicamā eksperimenta mapes struktūra

```
experiment_2026_08_18/
├─ README.md
├─ config/
│ └─ ekf.yaml
│ └─ nav2_params.yaml
├─ bags/
│ └─ run_001/
│ └─ run_002/
```

```

| └─ run_003/
|   └─ measurements/
|       └─ reference.csv
|   └─ analysis/
|       └─ plots.py
|       └─ results.csv
|   └─ report/
|       └─ test_summary.md

```

48.4. Reproducibility jeb atkārtojamība

Bag pats par sevi nav pilns eksperiments. Jāsaglabā arī konfigurācijas, izmantotā koda versija, aparatūras stāvoklis, vides apraksts un references mērījumi. Git commit hash ļauj precīzi norādīt, ar kuru programmatūras versiju bag tika ierakstīts.

Lauks	Piemērs
Testa ID	NAV-R7-003
Datums/laiks	2026-08-18 11:20 EEST
Robot ID	2WD-LAB-01
Git commit	abc1234
Konfigurācija	config/nav2_params.yaml
Bag	bags/run_003
Reference	2.00 m, 90°
Piezīmes	grīda sausa, baterija 80%

48.5. QoS un bag ieraksta problēmas

ROS 2 topic komunikācija izmanto QoS politikas. Ja recorder un publisher QoS nav savietojami, dati var netikt ierakstīti tā, kā gaidīts. Tāpēc pirms svarīga testa jāpārbauda topic informācija un pēc ieraksta – bag metadata un faktiskā playback/echo uzvedība. Īpaši sensoru topic var izmantot sensor-data tipa QoS profilus.

Studentam jāiemācās princips: “komanda nostrādāja bez kļūdas” nav pierādījums, ka visi nepieciešamie dati tiešām ir saglabāti.

48.6. MCAP kā alternatīvs storage formāts

rosvbag2 atbalsta storage plugin pieeju. Dažās ROS 2 distribūcijās pieejams MCAP storage plugin, ko var izvēlēties ar storage identifikatoru, piemēram, `-s mcap`. Kursā tas tiek pasniegts kā piemērs, ka datu faila formāts un ROS topic semantika ir atsevišķi slāņi. Tieši pieejamie plugin jānoskaidro instalētajā sistēmā.

```
ros2 bag record -s mcap /scan /imu/data /odometry/filtered -o run_mcap
```

48.7. Offline analīzes domāšana

Lai salīdzinātu algoritmus, jādefinē metrika pirms grafika zīmēšanas. Piemēram, 2WD navigācijas testā metrika var būt gala pozīcijas kļūda, yaw kļūda, trajektorijas garums, laiks līdz mērķim, minimālais attālums līdz šķērslim un recovery skaits.

Metrika	Vienība	Ko tā pasaka
Final position error	m	precizitāte mērķa sasniegšanā
Final yaw error	deg/rad	orientācijas precizitāte
Path length	m	ceļa efektivitāte
Time to goal	s	izpildes ātrums
Minimum obstacle distance	m	drošības rezerve
Recovery count	skaits	navigācijas stabilitāte
Mean lateral path error	m	kontroliera sekošanas kvalitāte

48.8. R9 laboratorija — tas pats bag, divas konfigurācijas

Mērķis: pierādīt, ka konfigurācijas izmaiņas var salīdzināt uz identiska sensoru datu ieraksta. Izvēlas vienu algoritmu, kurš spēj strādāt offline/replay plūsmā, piemēram, stāvokļa novērtētāju vai datu analīzes mezglu.

1. Ieraksti vienu kontrolētu braucienu ar raw/avota topic.
2. Saglabā config_A un config_B atsevišķos failos.
3. Playback bag ar config_A un saglabā rezultātu topic vai metrikas.
4. Atkārti to pašu bag ar config_B.
5. Salīdzini abas versijas ar iepriekš definētu metriku, ne tikai vizuāli.
6. Pieraksti Git commit hash un konfigurācijas diff.
7. Secinājumā norādi, kura konfigurācija ir labāka konkrētajam kritērijam un kādi kompromisi paliek.

48.9. Datu kvalitātes kontrolsaraksts

- Bag satur visus plānotos topic.
- Topic frekvences ir ticamas.
- Timestamp ir loģiski un laika avots ir dokumentēts.
- TF un TF_STATIC ir ierakstīti, ja tie nepieciešami replay.
- Konfigurācija un Git commit ir saglabāti blakus bag.
- References mērījums nav tikai “aptuveni”.
- Datu failā nav nevajadzīgu sensitīvu kameru/audio datu.

Avoti un versiju piezīme

Šajā nodaļā izmantota oficiālā projekta/ražotāja dokumentācija. ROS 2 pakotņu parametri, launch faili un noklusējuma konfigurācijas var atšķirties starp distribūcijām, tādēļ laboratorijā vienmēr jāpārbauda tieši instalētās versijas dokumentācija un pakotnes piemēra konfigurācija.

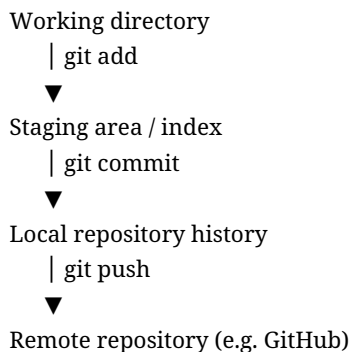
- ROS 2 rosbag2 oficiālā dokumentācija un rosbag2 record/play materiāli (docs.ros.org).
- rosbag2_storage_mcap pakotnes dokumentācija par MCAP storage plugin lietošanu.

49. Git un GitHub profesionālai programmatūras izstrādei

Programmēšanas kursā projekts bez versiju kontroles ir nepilnīgs profesionālās izaugsmes materiāls. Git saglabā projekta izmaiņu vēsturi lokāli, bet GitHub vai cita repozitoriju platforma nodrošina sadarbību, pull request, issue un code review darba plūsmu. Kursa mērķis nav iemācīt katru Git komandu, bet izveidot drošu ieradumu: mazas izmaiņas, saprotami commit, izolēti branch un pārskatāma integrācija.

49.1. Git mentālais modelis

Git pamata plūsma



Staging area ir apzinātas izvēles punkts: commit nav “viss, kas šobrīd mapē mainīts”. Students izvēlas, kuras izmaiņas veido loģisku vienību. Tas ir labs programmatūras inženierijas disciplīnas treniņš.

49.2. Minimālais komandu komplekts

Git lokālās darba plūsmas piemērs

```
git init
git status
git add README.md src/main.py
git commit -m "Add initial sensor parser"

git switch -c feature/imu-filter
# edit, test
git add src/imu_filter.py tests/test_imu.py
git commit -m "Add complementary IMU filter"

git switch main
git merge feature/imu-filter

git log --oneline --graph --decorate
```

49.3. Labs commit

Labs commit ir maza, loģiski pabeigta izmaiņa, kuru iespējams izskaidrot vienā teikumā un vajadzības gadījumā atcelt. Commit ziņai jāpaskaidro izmaiņas būtība, nevis tikai “update” vai “labojumi”.

Vājš commit	Labāks commit
update	Add encoder distance calibration
fix	Fix yaw sign in differential-drive odometry
final final	Add R7 navigation acceptance tests
changes	Validate MQTT temperature payload schema

49.4. Branch jeb izolēts darba zars

Branch ļauj izstrādāt funkciju vai labojumu izolēti no stabilā `main`. Git dokumentācija branch apraksta kā atsevišķu izstrādes līniju, un GitHub darba plūsmā feature branch ir tipisks veids, kā sagatavot pull request. Kursā katram lielākam noslēguma projekta uzdevumam var izmantot atsevišķu branch.

```
main
├─ feature/lidar-driver
├─ feature/ekf-config
├─ feature/nav2-bringup
└─ fix/encoder-sign
```

49.5. Merge un rebase konceptuālā atšķirība

Merge apvieno divas izstrādes vēstures un, ja nepieciešams, izveido merge commit. Rebase pārvieto/reizē izmaiņu commit uz citu bāzi, veidojot lineārāku vēsturi. Rebase pārraksta commit identitāti, tādēļ ar koplietotu vēsturi tas jālieto apzināti. Pamata kursā drošākais modelis ir saprast merge un konfliktu risināšanu; rebase ir paplašinājuma tēma.

49.6. Merge konflikts

Konflikts nav Git “kļūda”. Tas ir gadījums, kad sistēma nevar automātiski noteikt, kā apvienot divas savstarpēji konkurējošas izmaiņas. Studentam jāprot atvērt konflikta vietu, saprast abas versijas, izveidot pareizo gala saturu, palaist testus un tikai tad pabeigt merge.

```
<<<<<<< HEAD
STOP_DISTANCE = 0.40
=====
STOP_DISTANCE = 0.55
>>>>>> feature/safety-distance

# Nav pareizi mehāniski izdzēst markerus.
# Jānoskaidro prasība un jāizvēlas/testē pareizā vērtība.
```

49.7. GitHub Pull Request darba plūsma

GitHub mācību projekta sadarbības plūsma

```
Issue / task
↓
Create feature branch
↓
Implement + tests
↓
```

```

Commit small changes
↓
Push branch
↓
Open Pull Request
↓
Review / comments / CI
↓
Update branch
↓
Approve + merge
↓
Close issue / tag release

```

GitHub dokumentācijā pull request ir mehānisms, ar kuru izmaiņas atsevišķā branch tiek piedāvātas, apspriestas un pārskatītas pirms merge. Mācību kursā PR aizstāvēšana ir īpaši vērtīga: students paskaidro ne tikai gala kodu, bet arī izmaiņu motivāciju, testus un reviewer komentāru risinājumu.

49.8. Kursa repozitorija ieteicamā struktūra

```

robot-project/
├─ README.md
├─ LICENSE          # ja projekts tiek publiskots
├─ .gitignore
├─ docs/
│  └─ requirements.md
│  └─ functional_tree.md
│  └─ test_plan.md
├─ firmware/
│  └─ esp32/
├─ ros2_ws/
│  └─ src/
├─ python/
├─ config/
│  └─ ekf.yaml
│  └─ nav2_params.yaml
├─ tests/
├─ data/            # mazi piemēri; lielus bag publiski nelikt bez vajadzības
└─ CHANGELOG.md

```

49.9. .gitignore un sensitīvie dati

Repozitorijā nedrīkst nejauši publicēt Wi-Fi paroles, MQTT credentials, API keys, privātas SSH atslēgas, lielus build artefaktus vai sensitīvus sensoru ierakstus. `.gitignore` samazina nejaušas pievienošanas risku, bet jau commitots noslēpums nav “izdzēsts no vēstures” tikai tāpēc, ka failu pēc tam ieliek .gitignore.

.gitignore mācību piemērs

```

# Python
__pycache__/
.venv/

```

```
# ROS 2 / colcon
build/
install/
log/

# local secrets
.env
secrets.yaml

# large local datasets
bags/
*.db3
*.mcap
```

49.10. README kā tehniska dokumentācija

README minimums noslēguma projektam

- Problēma un projekta mērķis.
- Sistēmas arhitektūra / functional tree.
- Aparatūra un programmatūras prasības.
- Instalēšana/būvēšana.
- Palaišanas secība.
- Topic/interfeisu līgums.
- Drošības ierobežojumi.
- Testēšana un zināmie ierobežojumi.
- Autors(-i), licence un AI izmantošanas deklarācija, ja nepieciešams.

49.11. Code review kontrolesaraksts

- Vai izmaiņa atbilst prasībai/issue?
- Vai nosaukumi ir saprotami un nav liekas koda dublēšanas?
- Vai kļūdu/timeout gadījumi ir apstrādāti?
- Vai nav hard-coded parolu vai neatbilstošu absolūtu ceļu?
- Vai sensora vienības un koordinātu sistēmas ir dokumentētas?
- Vai ir tests vai eksperimenta pierādījums?
- Vai mainītas konfigurācijas ir pamatotas?
- Vai README vai tehniskā dokumentācija jāatjauno?

49.12. Git laboratorija G1 — no issue līdz pull request

1. Izveido issue “Add configurable obstacle stop distance”.
2. No main izveido branch feature/stop-distance-config.
3. Pārviesto hard-coded sliekšni uz konfigurācijas parametru.
4. Pievieno vismaz divus testus: normāla vērtība un nederīga vērtība.
5. Izveido 2–3 loģiskus commit ar saprotamiem ziņojumiem.

6. Atver pull request un aprakstā norādi prasību, risinājumu un testu rezultātus.
7. Cits students veic review un uzdod vismaz vienu saturisku jautājumu.
8. Autors izlabo/pamato, pēc tam PR tiek merge.

Avoti un versiju piezīme

Šajā nodaļā izmantota oficiālā projekta/ražotāja dokumentācija. ROS 2 pakotņu parametri, launch faili un noklusējuma konfigurācijas var atšķirties starp distribūcijām, tādēļ laboratorijā vienmēr jāpārbauda tieši instalētās versijas dokumentācija un pakotnes piemēra konfigurācija.

- Git oficiālā dokumentācija: git-branch, git-merge un Pro Git branching/rebasing materiāli (git-scm.com).
- GitHub Docs: branches un pull request darba plūsma (docs.github.com).

50. Eksāmenu, praktisko uzdevumu un aizstāvēšanas jautājumu banka

Šī nodaļa ir paredzēta kā ilgtermiņa jautājumu banka, no kuras pasniedzējs var veidot īsus testus, pārbaudes darbus, semestra eksāmenus, mutiskās aizstāvēšanas un universitātes līmeņa eksāmena biļetes. Jautājumi ir sadalīti pēc dziļuma, lai vienu un to pašu kursa materiālu varētu pielāgot profesionālās izglītības un augstskolas auditorijai.

50.1. Koledžas līmeņa teorijas jautājumi

Algoritmi un loģika

1. Kas ir algoritms un kādas īpašības tam jāpiemīt?
2. Izskaidro ievade → apstrāde → izvade modeli ar tehnisku piemēru.
3. Kāda ir atšķirība starp secību, zarošanos un atkārtošanos?
4. Kas ir pseidokods un kāpēc to lietot pirms programmas koda?
5. Kāda ir lēmuma (decision) simbola nozīme blokshēmā?
6. Kas ir robežgadījums un dod divus piemērus.
7. Kāda ir atšķirība starp AND un OR loģiku?
8. Kas ir interlock un kāpēc tas ir svarīgs tehniskā sistēmā?
9. Kas ir timeout un kādu kļūdu tas novērš?
10. Kas ir bezgalīgs cikls un kad nepārtraukts cikls ir apzināti vajadzīgs?
11. Izskaidro dekompozīciju un functional tree lomu.
12. Kā pārbaudīt algoritmu pirms programmēšanas?

Python

13. Kāda ir atšķirība starp int, float, bool un str?
14. Kāpēc input() rezultātu bieži nepieciešams konvertēt?
15. Izskaidro if/elif/else izpildes secību.

16. Kad izvēlēties for, kad while ciklu?
17. Kas ir funkcija un kāpēc tā uzlabo programmas struktūru?
18. Kas ir return un kā tas atšķiras no print?
19. Kas ir saraksts (list) un kā piekļūt tā elementam?
20. Kas ir dictionary un kā tas var reprezentēt sensora mērījumu?
21. Kā apstrādāt nederīgu lietotāja ievadi?
22. Kas ir exception un kāpēc nevajadzētu paslēpt visas kļūdas ar tukšu except?
23. Kā nolasīt CSV/JSON datus un kāpēc tie atšķiras?
24. Ko nozīmē debugging?

C++

25. Kāda ir C++ programmas kompilācijas ideja?
26. Kāpēc C++ mainīgajam deklarē datu tipu?
27. Kāda ir atšķirība starp = un ==?
28. Kāda ir const nozīme?
29. Kāda ir atšķirība starp if un switch?
30. Kad izmantot for un when while?
31. Kas ir funkcijas parametrs un return type?
32. Kas ir masīvs un kāds ir indeksa sākums?
33. Kas ir class un object?
34. Kas ir constructor?
35. Kāpēc mikrokontrolleru programmās jādomā par datu tipu izmēru?
36. Kāpēc blocking delay var būt problēma vadības sistēmā?

SQL, IoT un dati

37. Kas ir datubāzes tabula, rinda un kolonna?
38. Kas ir primary key?
39. Ko dara SELECT, INSERT, UPDATE un DELETE?
40. Ko dara WHERE?
41. Kas ir JOIN ideja?
42. Kāda ir atšķirība starp CSV un JSON?
43. Kas ir API?
44. Kāda ir atšķirība starp HTTP request/response un MQTT publish/subscribe modeli?
45. Kas ir MQTT broker?

- 46. Kas ir topic?
- 47. Kāpēc IoT datos vajadzīgs timestamp?
- 48. Kāpēc paroles un API keys nedrīkst publicēt repozitorijā?

Mehatronika

- 49. Kas ir mehatroniska sistēma?
- 50. Kāda ir sensora un aktuatora atšķirība?
- 51. Kas ir analogs un digitāls signāls?
- 52. Kas ir ADC?
- 53. Kas ir PWM un kāpēc tas nav tas pats, kas īsts analogs spriegums?
- 54. Kas ir open-loop un closed-loop vadība?
- 55. Kas ir feedback?
- 56. Kas ir setpoint, measured value un error?
- 57. Ko dara P, I un D daļa PID regulatorā konceptuāli?
- 58. Kas ir histerēze un kāpēc tā novērš biežu slēgāšanu pie robežas?
- 59. Kas ir fail-safe princips?
- 60. Kāpēc motora jaudas ķēdi nedrīkst barot tieši no mikrokontrollera GPIO?

Robotika un ROS 2

- 61. Kas ir DOF?
- 62. Kāda ir atšķirība starp forward un inverse kinematics?
- 63. Kas ir robota coordinate frame?
- 64. Kas ir LiDAR point/scan?
- 65. Ko mēra IMU accelerometer un gyroscope?
- 66. Kas ir sensor fusion?
- 67. Kas ir odometry un kāpēc tā driftē?
- 68. Kas ir SLAM?
- 69. Kas ir map, odom un base_link loma ROS 2 TF kokā?
- 70. Ko satur /scan?
- 71. Ko satur /imu/data?
- 72. Ko satur /odom?
- 73. Kāda ir global planner un local controller atšķirība?
- 74. Kas ir costmap?
- 75. Kas ir Behavior Tree?

76. Ko nozīmē recovery behavior?

77. Kāpēc rosbag2 ir noderīgs debugging?

78. Kāpēc Git commit hash ir noderīgs eksperimenta dokumentācijā?

50.2. Koledžas līmeņa īso atbilžu atslēgas principi

Atbildes vērtē pēc būtības, nevis konkrētas frāzes iegaumēšanas. Pilna atbilde satur definīciju, mehānisma skaidrojumu un, ja jautājums to prasa, tehnisku piemēru. 2 punktu jautājumam tipiska shēma: 1 punkts par pareizu jēdzienu, 1 punkts par pareizu piemēru/sekām.

Atbildes kvalitāte	Raksturojums
0	nav atbildes vai būtiski nepareiza
1	daļēja definīcija, bet trūkst mehānisma vai ir būtiska neprecizitāte
2	pareiza definīcija un saprotams tehnisks piemērs/sekas
3 (padziļināts)	papildus norāda ierobežojumus, kļūmes vai alternatīvas

50.3. Koledžas līmeņa praktiskie uzdevumi

P1. Izveido blokshēmu automātiskām durvīm ar drošības sensoru un timeout.

P2. Uzraksti pseidokodu tvertnes uzpildei ar MAX sensoru un sensora kļūmes stāvokli.

P3. Python: ievadi 10 temperatūras mērījumus, aprēķini vidējo, min, max un trauksmju skaitu.

P4. Python: nolasīt CSV ar timestamp, temperature un humidity un atrast periodus virs robežas.

P5. Python: izveidot state machine IDLE/RUN/FAULT/ESTOP.

P6. C++: izveidot funkciju, kas no ADC vērtības aprēķina procentus 0–100.

P7. C++: uzrakstīt drošu motora komandas ierobežošanu diapazonā -255...255.

P8. C++: izveidot Sensor klasi ar value, valid un timestamp.

P9. SQL: izveidot measurements tabulu un atlasīt pēdējās 20 temperatūras virs 30 °C.

P10. SQL: aprēķināt sensora vidējo vērtību noteiktā laika intervālā.

P11. IoT: izveidot MQTT topic struktūru trim telpām un diviem sensoru tipiem.

P12. IoT: definēt JSON payload ar value, unit, timestamp, sensor_id un quality.

P13. Mehatronika: izveidot ventilatora histerēzes algoritmu ON pie 30 °C un OFF pie 27 °C.

P14. Mehatronika: izskaidrot, kā noteikt motora drivera un barošanas prasības no motora datiem.

P15. Robotika: aprēķināt 2DOF manipulatora gala x/y dotajiem L1, L2, θ_1 , θ_2 .

P16. Robotika: no wheel encoder Δs_L un Δs_R aprēķināt Δs un $\Delta \theta$.

P17. LiDAR: no angle/range pārvērst vienu punktu x/y koordinātās.

P18. IMU: pārvērst gyro 90 deg/s uz rad/s.

P19. ROS 2: uzzīmēt korektu map→odom→base_link→laser_frame TF koku.

P20. ROS 2: sastādīt diagnostikas secību gadījumam, kad /scan ir redzams, bet RViz tas atrodas nepareizā vietā.

P21. ROS 2: noteikt, kuri topic jāieraksta rosbag, lai analizētu navigācijas driftu.

P22. Git: izveidot branch/commit/PR plūsmu vienai funkcionalitātei.

50.4. Debugging uzdevumu banka

D1. Python nosacījums

```
temperature = 35
if temperature = 30:
    print("Alarm")
```

Pasniedzēja atslēga: Sintakses kļūda: salīdzināšanai `==`; papildus prasībā jānosaka > vai >=.

D2. Python while

```
x = 0
while x < 10:
    print(x)
```

Pasniedzēja atslēga: x netiek mainīts → bezgalīgs cikls.

D3. C++ salīdzinājums

```
if (sensor = 1) { motorOn(); }
```

Pasniedzēja atslēga: Piešķiršana, nevis salīdzināšana; jālieto `==` vai bool semantika.

D4. C++ robeža

```
int values[10];
for (int i=0; i<=10; ++i) values[i]=0;
```

Pasniedzēja atslēga: i=10 ir ārpus masīva; nosacījumam jābūt i<10.

D5. MQTT payload

```
{"temp": 25.4, "unit": "C"}
```

Pasniedzēja atslēga: Trūkst timestamp/sensor identitātes, ja sistēmai vajag auditējamu telemetriju.

D6. Odometry

$$\Delta\theta = (\Delta s_L - \Delta s_R) / b$$

Pasniedzēja atslēga: Zīmes konvencija var būt pretēja kursā izmantotajai formulai; jāpārbauda definētais pozitīvais yaw un riteņu secība.

D7. IMU

```
angular_velocity.z = gyro_deg_s
```

Pasniedzēja atslēga: ROS Imu angular velocity jābūt rad/s, nevis deg/s.

D8. TF

map → base_link un odom → base_link publicē divi dažādi mezgli

Pasniedzēja atslēga: Pretrunīgs TF koks; jādefinē viens atbildīgais katrai transformācijai.

D9. Costmap

laser_frame nav statiska TF no base_link

Pasniedzēja atslēga: Sensoru dati nevar korekti transformēties costmap frame.

D10. Git

WiFi password ir commitots config.yaml un pēc tam fails pievienots .gitignore

Pasniedzēja atslēga: Noslēpums paliek Git vēsturē; jāmaina credential un jāattīra vēsture atbilstoši procedūrai.

50.5. Universitātes līmeņa jautājumu banka

Algoritmi un datu struktūras

U1. Analizē $O(1)$, $O(\log n)$, $O(n)$ un $O(n^2)$ atšķirību ar konkrētu datu apjoma piemēru.

U2. Kāpēc binary search prasa sakārtotu datu struktūru?

U3. Salīdzini array, linked list, stack, queue un hash map tipiskās operācijas.

U4. Izskaidro recursion bāzes gadījumu un stack izmantošanu.

U5. Kā izvēlēties algoritmu, ja laika un atmiņas prasības konfliktē?

C++ un sistēmu programmēšana

U6. Izskaidro stack un heap atšķirību.

U7. Kas ir pointer, reference un ownership?

U8. Kas ir RAII un kā tas palīdz resursu drošībā?

U9. Kas ir undefined behavior un kāpēc tas ir bīstams embedded sistēmās?

U10. Salīdzini `std::vector` un fiksētu array reāllaika/embedded kontekstā.

U11. Kas ir interrupt un kāpēc ISR jābūt īsai?

U12. Kas ir race condition un kā to var radīt shared volatile dati starp ISR un main loop?

Robotu matemātika un kinemātika

U13. Izskaidro homogēnās transformācijas matricas struktūru.

U14. Kā komponē transformācijas starp vairākiem frame?

U15. Izskaidro Denavit–Hartenberg parametru ideju.

U16. Kas ir forward kinematics un inverse kinematics risinājumu daudznozīmība?

U17. Kas ir Jacobian un kā tas saista joint ātrumus ar end-effector ātrumu?

U18. Kas ir singularitāte un kā tā ietekmē vadību?

U19. Derivē differential-drive v un ω no v_L un v_R .

U20. Kā integrācijas solis Δt ietekmē odometrijas kļūdu?

Vadība un stāvokļa novērtēšana

U21. Izskaidro closed-loop stabilitātes intuīciju.

U22. Kā P, I un D termini ietekmē transient response?

U23. Kas ir integral windup un kā to mazināt?

U24. Kas ir process noise Q un measurement noise R Kalman filtrā?

U25. Kas ir innovation?

U26. Kāpēc EKF izmanto Jacobian?

U27. Kā nepareiza covariance var padarīt sensor fusion sliktāku?

U28. Kāda ir atšķirība starp EKF un UKF konceptuāli?

SLAM un navigācija

U29. Izskaidro scan matching un tā iespējamās neveiksmes gadījumus.

U30. Kas ir pose graph?

U31. Kas ir loop closure un kāpēc tas var mainīt iepriekšējo trajektoriju?

U32. Salīdzini localization un mapping režīmu.

U33. Kāda ir global un local costmap atšķirība?

U34. Kā inflation radius un footprint ietekmē navigācijas izpildāmību?

U35. Kāda ir planner un controller atbildības robeža?

U36. Kāpēc Behavior Tree var būt piemērotāks par lielu FSM kompleksai navigācijai?

U37. Kā projektēt recovery, kas neizraisa bezgalīgu atkārtotānu?

U38. Kā rosbag var izmantot A/B algoritmu salīdzināšanai?

Programmatūras inženierija

U39. Kā prasība tiek sasaistīta ar testu traceability matricā?

U40. Kāda ir unit, integration un system test atšķirība?

U41. Kā definēt acceptance criterion robotikas funkcijai?

U42. Kāpēc konfigurācija jāversē kopā ar kodu?

U43. Salīdzini merge un rebase ietekmi uz Git vēsturi.

U44. Kas ir reproducible experiment un kādi artefakti tam vajadzīgi?

U45. Kā code review var atklāt drošības un integrācijas riskus, ko vienkāršs unit test neatklāj?

50.6. Universitātes līmeņa aprēķinu uzdevumi

UC1. Dots 2DOF manipulators $L_1=0,4$ m, $L_2=0,3$ m, $\theta_1=30^\circ$, $\theta_2=-45^\circ$. Aprēķini end-effector x,y un uzzīmē konfigurāciju.

UC2. Dotam 2R robotam atvasini 2×2 pozicionālo Jacobian un novērtē determinant singularitāti pie $\theta_2 = 0^\circ$.

UC3. 2WD robotam $r = 0,0325$ m, $b = 0,155$ m, wheel angular speeds 8 un 12 rad/s. Aprēķini v un ω .

UC4. Enkodera 600 ticks/rev ritenim ar $d = 65$ mm aprēķini distance_per_tick un teorētisko ticks skaitu 3 m.

UC5. IMU gyro bias ir 0,8 deg/s. Aprēķini sagaidāmo yaw driftu 60 s bez korekcijas.

UC6. Kalman 1D piemērā $P = 1$, $R = 4$. Aprēķini Kalman gain un interpretē uzticības sadalījumu.

UC7. PID regulatoram $e = 2$, integral = 4, $de/dt = -0,5$, $K_p = 1,2$, $K_i = 0,1$, $K_d = 0,3$. Aprēķini u.

UC8. A* režģī salīdzini Manhattan un Euclidean heuristic pielietojamību 4-neighbor un 8-neighbor kustībai.

UC9. Costmap resolution = 0,05 m, width = 4 m. Aprēķini šūnu skaitu vienā malā un kopējo šūnu skaitu.

UC10. Robots brauc 0,6 m/s un lokālā costmap no centra redz 1,5 m uz priekšu. Novērtē, cik sekundes telpiska horizonta tas dod ideālā gadījumā.

50.7. Mutiskās aizstāvēšanas jautājumi noslēguma projektam

O1. Kāda bija sākotnējā problēma, un kā to pārveidoji pārbaudāmās prasībās?

O2. Parādi functional tree un paskaidro, kura funkcija bija tehniski grūtākā.

O3. Kāpēc izvēlējies Python un/vai C++ konkrētām sistēmas daļām?

O4. Kā tiek validēti sensora dati pirms lēmumu pieņemšanas?

O5. Kas notiek, ja sensors pazūd vai atgriež nederīgu vērtību?

O6. Kā tu pierādi, ka programma strādā, nevis tikai demonstrē vienu veiksmīgu mēģinājumu?

O7. Kādas bija references vērtības testos?

O8. Kāda ir lielākā atlikusī kļūda vai nenoteiktība?

O9. Kur sistēmā var rasties race condition, timing vai blocking problēma?

O10. Kā ir nodalīta safety funkcija no normālās vadības?

O11. Kādas konfigurācijas ir versētas Git?

O12. Parādi vienu pull request vai commit, kurā kļūda tika labota, un izskaidro root cause.

O13. Ja tev būtu divas papildu nedēļas, ko mainītu arhitektūrā un kāpēc?

O14. Kā sistēma mērogotos no viena robota uz vairākiem?

O15. Kur projektā AI tika izmantots un kā pārbaudīji AI ģenerēto saturu/kodu?

50.8. Četri eksāmena varianti — koledžas līmenis

Variants A — 100 punkti

Daļa	Punkti	Uzdevums
1. Jēdzieni	20	10 īsie jautājumi no 50.1 bankas
2. Algoritms	20	pseudokods + blokshēma tehniskam procesam
3. Programmēšana	25	Python vai C++ uzdevums ar ievadi, nosacījumu, ciklu/funkciju
4. Sistēmu integrācija	20	SQL/IoT/mehatronikas/ROS 2 datu plūsmas analīze
5. Debugging	15	atrast un izskaidrot 3–5 kļūdas dotā fragmentā

Vērtēšanas prasība: programmēšanas daļā vismaz 40% punktu piešķir par algoritmu, testiem un kļūdu apstrādi, nevis tikai par sintaktiski strādājošu kodu.

Variants B — 100 punkti

Daļa	Punkti	Uzdevums
1. Jēdzieni	20	10 īsie jautājumi no 50.1 bankas
2. Algoritms	20	pseudokods + blokshēma tehniskam procesam
3. Programmēšana	25	Python vai C++ uzdevums ar ievadi, nosacījumu, ciklu/funkciju
4. Sistēmu integrācija	20	SQL/IoT/mehatronikas/ROS 2 datu plūsmas analīze
5. Debugging	15	atrast un izskaidrot 3–5 kļūdas dotā fragmentā

Vērtēšanas prasība: programmēšanas daļā vismaz 40% punktu piešķir par algoritmu, testiem un kļūdu apstrādi, nevis tikai par sintaktiski strādājošu kodu.

Variants C — 100 punkti

Daļa	Punkti	Uzdevums
1. Jēdzieni	20	10 īsie jautājumi no 50.1 bankas
2. Algoritms	20	pseudokods + blokshēma tehniskam procesam

Daļa	Punkti	Uzdevums
3. Programmēšana	25	Python vai C++ uzdevums ar ievadi, nosacījumu, ciklu/funkciju
4. Sistēmu integrācija	20	SQL/IoT/mehatronikas/ROS 2 datu plūsmas analīze
5. Debugging	15	atrast un izskaidrot 3–5 kļūdas dotā fragmentā

Vērtēšanas prasība: programmēšanas daļā vismaz 40% punktu piešķir par algoritmu, testiem un kļūdu apstrādi, nevis tikai par sintaktiski strādājošu kodu.

Variants D — 100 punkti

Daļa	Punkti	Uzdevums
1. Jēdzieni	20	10 īsie jautājumi no 50.1 bankas
2. Algoritms	20	pseudokods + blokshēma tehniskam procesam
3. Programmēšana	25	Python vai C++ uzdevums ar ievadi, nosacījumu, ciklu/funkciju
4. Sistēmu integrācija	20	SQL/IoT/mehatronikas/ROS 2 datu plūsmas analīze
5. Debugging	15	atrast un izskaidrot 3–5 kļūdas dotā fragmentā

Vērtēšanas prasība: programmēšanas daļā vismaz 40% punktu piešķir par algoritmu, testiem un kļūdu apstrādi, nevis tikai par sintaktiski strādājošu kodu.

50.9. Universitātes eksāmena biļetes paraugs

Jautājums	Punkti
1. Teorētiskais jautājums: EKF/SLAM/kinemātika/algoritmi	20
2. Matemātiskais aprēķins ar interpretāciju	25
3. Koda/konfigurācijas analīze un debugging	20
4. Sistēmas arhitektūras uzdevums	20
5. Mutisks follow-up un kompromisu analīze	15

50.10. Vērtēšanas robežas — ieteikums

Rezultāts	Raksturojums
90–100%	ļoti laba jēdzienu izpratne; risinājums pārbaudīts un pamatots; spēj analizēt kompromisus
75–89%	stabils risinājums un laba izpratne; nelielas nepilnības padziļinājumā
60–74%	pamati apgūti; risinājums strādā tipiskā gadījumā, bet vāja robežgadījumu/validācijas analīze
50–59%	minimāli sasniegti pamata rezultāti; nepieciešama būtiska palīdzība integrācijā/testēšanā
<50%	nav pierādīta pietiekama pamatu izpratne vai risinājums nav pārbaudāms

51. Integrētais ROS 2 autonomā robota ceļš: R4–R9

Šī nodaļa apvieno v0.8 R1–R3 laboratorijas un v0.9 R4–R9 laboratorijas vienā progresijā. Mērķis ir nepieļaut tipisku mācību kļūdu — uzreiz palaist Nav2 un pēc tam mēģināt minēt, kurā no desmit slāņiem atrodas problēma. Katrs līmenis tiek “atslēgts” tikai pēc tam, kad iepriekšējais ir izmērāmi pierādīts.

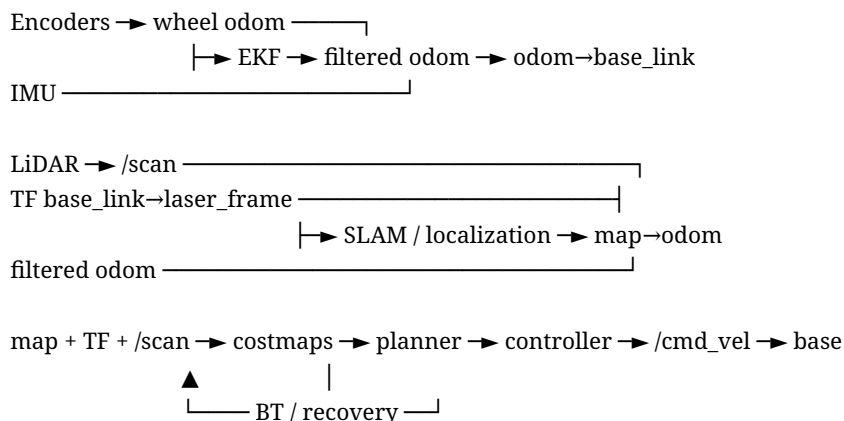
Posms	Galvenais rezultāts	Acceptance gate
R1	/odom + odom→base_link	2 m un 360° tests dokumentēts
R2	/imu/data + imu_link	vienības, asis, bias pārbaudīts
R3	/scan + laser_frame	references objekta attālums un TF pareizs
R4	/odometry/filtered	sensor fusion A/B tests ar metriku
R5	2D karte / SLAM	references dimensijas un loop closure novērtēts
R6	global/local costmaps	marking, clearing, inflation, footprint pārbaudīts
R7	NavigateToPose	≥3 mērķu testi ar gala kļūdu
R8	BT/recovery	kontrolēts failure/recovery scenārijs
R9	roslab A/B eksperiments	viena datu kopa, 2 konfigurācijas, salīdzināma metrika

51.1. “Stop and fix” princips

Ja R3 neiziet, students nepāriet uz R5. Ja map→odom lokalizācija ir nepareiza, netiek “tūnots controller”. Ja local costmap neredz šķērslī, netiek mainīti planner parametri. Šis princips māca root-cause analīzi un samazina nejaušu konfigurācijas mainīšanu.

51.2. Integrētā datu plūsma

Pilnā mācību autonomā robota arhitektūra



rosvag2 records evidence across all layers
Git versions code + configs + reports

51.3. Minimālā gala projekta demonstrācija

- Robots startē no zināmas pozas vai lokalizējas kartē.
- Students parāda TF koku un galveno topic statusu.
- Tiek dots NavigateToPose mērķis.
- Robots izplāno un izpilda ceļu, izvairoties no statiska šķēršļa.
- Tiek demonstrēts vismaz viens kontrolēts replanning/recovery scenārijs.
- Tiek saglabāts rosvag un pēc brauciena parādīta vismaz viena kvantitatīva metrika.
- Repozitorijā redzama programmatūras/config versija, ar kuru demonstrācija veikta.

51.4. Universitātes paplašinājums integrētajam projektam

- EKF covariance pamatota ar eksperimentāliem datiem.
- Salīdzināti vismaz divi controller/planner vai filtra parametru komplekti uz vienādiem scenārijiem.
- Aprēķināta trajektorijas kļūda pret references ceļu vai ārēju ground truth, ja pieejams.
- Analizēta laika aizture un topic frekvences ietekme.
- Izveidots pielāgots Behavior Tree vai custom BT node.
- Izveidota automatizēta testa/CI daļa vismaz programmatūras līmeņa komponentēm.
- Dokumentēti sistēmas drošības ierobežojumi un failure modes.

51.5. Gala projekta evidence package

Evidence package
└ requirements + acceptance criteria

- └ functional tree / architecture
- └ source code + config (Git)
- └ calibration records
- └ test plan
- └ R1–R9 protocols
- └ rosbag datasets
- └ quantitative analysis
- └ screenshots / plots
- └ issue + pull request history
- └ final report
- └ presentation / defense notes

52. [ARHĪVS] Versijas 0.9 izstrādes statuss un nākamais slānis

Versijā 0.9 vienotais dokuments papildināts ar praktisku ROS 2 autonomās navigācijas turpinājumu: EKF/robot_localization stāvokļa novērtēšanu, SLAM un slam_toolbox, Nav2 costmap/planner/controller/Behavior Tree arhitektūru, rosbag2 atkārtojamu eksperimentu metodiku, Git/GitHub profesionālo darba plūsmu, plašu koledžas un universitātes eksāmenu jautājumu banku un integrēto R1–R9 laboratoriju progresiju.

Nākamajam slānim paredzēts: pilns skolēna izdrukājamo R4–R9 darba lapu komplekts ar Expected/Actual/PASS-FAIL tabulām; robot_localization/slam_toolbox/Nav2 konfigurāciju etalona faili; automatizēti Python rosbag analīzes skripti un grafiki; ROS 2 launch/URDF/xacro pamati; simulatora (piem., Gazebo/Webots) darba plūsma; CI testu piemēri GitHub Actions vai analogā vidē; kā arī padziļināta Machine Vision/Visual SLAM un 3D point cloud nodaļa.

53. R4–R9 izdrukājamās ROS 2 laboratoriju darba lapas

Šajā nodaļā R4–R9 laboratoriju apraksti pārveidoti par izdrukājamu darba lapu komplektu. Katra darba lapa sākas ar priekšnosacījumiem, mērķi un obligāto evidence paketi, bet beidzas ar testa tabulu, analīzes jautājumiem un īsu pasniedzēja vērtēšanas atslēgu. Laboratoriju secība ir apzināti stingra: pirms nākamā līmeņa students pierāda iepriekšējā līmeņa datu kvalitāti.

R4 — EKF: wheel odometry + IMU → filtrēta odometrija

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	90 min
Repozitorijs/commit	

Mērķis

Izveidot 2D stāvokļa novērtētāju, kas apvieno riteņu odometriju un IMU, un eksperimentāli parādīt, kā covariance un sensora izvēle ietekmē /odometry/filtered rezultātu.

Priekšnosacījumi

- R1–R3 pabeigtas ar PASS; /wheel/odom un /imu/data publicē korektas SI mērvienības un timestamp.
- TF starp base_link un imu_link ir pārbaudīts.
- Riteņa rādiuss un track width kalibrēti; students spēj izskaidrot odometrijas drift cēloni.

Darba gaita

1. Reģistrē 30–60 s nekustīga robota /imu/data un /wheel/odom datus.
2. Aprēķini gyro Z vidējo vērtību, standartnovirzi un novērtē bias.
3. Izveido ekf.yaml un norādi two_d_mode, frame nosaukumus un sensoru konfigurācijas vektorus.
4. Palaid EKF un pārbaudi /odometry/filtered frekvenci.
5. Pārbaudi, ka tikai viens mezgls publicē odom→base_link TF.
6. Veic 3 m taisno testu un 360° CW/CCW pagriezienus.
7. Maini IMU yaw-rate covariance aptuveni 10× un atkārtoto testu.
8. Salīdzini rezultātus un pamato, kura konfigurācija labāk atbilst mērījumiem.

Mērījums	A konfigurācija	B konfigurācija	Piezīmes
Taisne: gala X kļūda, m			
Taisne: gala Y kļūda, m			

Mērījums		A konfigurācija		B konfigurācija		Piezīmes		
Pagrieziens CW: yaw kļūda, °								
Pagrieziens CCW: yaw kļūda, °								
/odometry/filtered vid. Hz								
Nr.		Testa gadījums		Expected		Actual		PASS/FAIL
1								
2								
3								
4								
5								
6								

Analīzes jautājumi

- Kāpēc gludāka trajektorija pati par sevi nepierāda labāku lokalizāciju?
- Kas notiktu, ja IMU ass zīme būtu pretēja enkoderu yaw zīmei?
- Ko nozīmē pārāk maza covariance?
- Kā atšķirt sensora bias no nejauša trokšņa?
- Kāpēc pirms EKF jākalibrē mehāniskā odometrija?

Pasniedzēja vērtēšanas atslēga

- PASS: TF, vienības un timestamp korekti; filtrs stabils; students parāda A/B eksperimentu un spēj pamatot covariance ietekmi.
- FAIL: filtrs tiek vērtēts tikai pēc RViz vizuālā iespaida; nav etalona mērījuma; pretrunīgi TF; nav saglabātu konfigurāciju.

R5 — SLAM: 2D kartes izveide un kvalitātes novērtēšana

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	90 min
Repozitorijs/commit	

Mērķis

Izveidot 2D occupancy grid karti ar LiDAR un filtrētu odometriju, pēc tam novērtēt kartes ģeometrisku konsekvenci, loop closure un atkārtotamību.

- R4 pabeigta ar PASS.

- /scan ir pareizā frame un diapazonā.
- TF ķēde odom→base_link→laser_frame ir stabila.

Darba gaita

1. Izvēlies kartējamo telpu un nomēri vismaz divus references izmērus ar mērlenti.
2. Palaid SLAM konfigurāciju un pārbaudi map→odom transformu.
3. Brauc lēni, izvairoties no straujas rotācijas un sensora aizsegšanas.
4. Izveido cilpu — atgriezies sākuma zonā no cita virziena.
5. Saglabā karti un ekrānuzņēmumu ar TF un /scan.
6. Izmēri kartē references sienu/koridora attālumu un aprēķini relatīvo kļūdu.
7. Atkārto kartēšanu ar vienu parametru izmaiņu vai ar sliktāku odometrijas konfigurāciju un salīdzini.

Metrika	Mērījums dabā	Karte A	Karte B	Kļūda / secinājums
References garums 1, m				
References garums 2, m				
Loop closure novirze sākumā, m	—			
Kartēšanas laiks, min	—			
Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1				
2				
3				
4				
5				
6				

Analīzes jautājumi

- Kāda ir odometrijas loma scan matching procesā?
- Kā izskatās neveiksmīgs loop closure?
- Kāpēc stikls, spoguļi un šauri objekti var radīt problēmas LiDAR kartē?
- Kā kustības ātrums un rotācijas ātrums ietekmē kartes kvalitāti?

R6 — Nav2 costmaps: šķēršļu un inflation slāņu analīze

Lauks	Aizpilda students

Mērķis

Saprast, kā no sensora datiem tiek veidota navigācijai izmantojama izmaksu telpa, un eksperimentāli parādīt footprint, obstacle un inflation parametru ietekmi uz drošu trajektoriju.

Darba gaita

1. Ielādē stabilu karti un lokalizē robotu.
2. Atver global_costmap un local_costmap RViz.
3. Novieto vienu statisku šķērsli un pārbaudi, vai tas parādās local costmap.
4. Pieraksti robota footprint/radius un inflation_radius.
5. Veic A/B testu ar mazāku un lielāku inflation radius.
6. Izmēri minimālo reālo attālumu līdz šķērslim abos testos.
7. Izskaidro, kāpēc pārāk liela inflation nav vienmēr drošāka — tā var bloķēt šaurus koridorus.

Parametrs / metrika		A	B	Secinājums	
inflation_radius, m					
cost_scaling_factor					
min. clearance, m					
vai koridors izbraucams?					
Nr.	Testa gadījums	Expected		Actual	PASS/FAIL
1					
2					
3					
4					
5					

R7 — NavigateToPose: plānotājs, kontrolieris un mērķa sasniegšana

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	90 min
Repozitorijs/commit	

Mērķis

Nosūtīt autonomam robotam mērķa pozu, izmērīt plāna izpildes kvalitāti un atdalīt globālā plānotāja, lokālā kontroliera un lokalizācijas problēmas.

Darba gaita

1. Definē starta un mērķa pozu kartē.
2. Pirms kustības saglabā global plan ekrānuņemumu vai topic datus.
3. Izpildi braucienu trīs reizes identiskā vidē.
4. Pieraksti izpildes laiku, gala pozīcijas kļūdu un replanning skaitu.
5. Vienā atkārtojumā pievieno statistiku šķērsli pēc plāna izveides.
6. Nosaki, vai robots pārplāno ceļu, apstājas vai aktivizē recovery.

Atkārtojums	Laiks, s	Gala pozīcijas kļūda, m	Gala yaw kļūda, °	Replanning/recovery
1				
2				
3				
Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1				
2				
3				
4				
5				
6				

R8 — Behavior Tree un recovery: kontrolēta atteices situācija

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	90 min
Repozitorijs/commit	

Mērķis

Radīt drošu, reproducējamu situāciju, kurā sākotnējais navigācijas mēģinājums nevar tikt pabeigts, un izsekot Behavior Tree lēmumu/recovery secībai.

Darba gaita

1. Izvēlies atteices scenāriju: īslaicīgi bloķēts koridors, mērķis pārāk tuvu šķērslim vai lokāla plāna neveiksme.
2. Pirms testa definē sagaidāmo reakciju.
3. Ieraksti rosbag ar /tf, /tf_static, /scan, /odom vai /odometry/filtered un Nav2 statusa/topic datiem.

- Aktivizē scenāriju un dokumentē recovery secību.
- Atjauno vidi un pārbaudi, vai misija turpinās vai korekti beidzas ar kļūdu.
- Paskaidro, kāpēc bezgalīga retry cilpa nav pieņemama drošības uzvedība.

Posms	Sagaidīts	Novērots	PASS/FAIL
Primārais plāns			
Atteices noteikšana			
Recovery 1			
Replanning			
Misijas rezultāts			

R9 — rosbag A/B eksperiments: viena datu kopa, divas konfigurācijas

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	90 min
Repozitorijs/commit	

Mērķis

Izmantot vienu un to pašu sensora ierakstu, lai godīgi salīdzinātu divas algoritma/config versijas un parādītu reproducējamu inženiertehnisku eksperimentu.

Darba gaita

- Reģistrē vienu standartizētu braucienu ar būtiskajiem sensoriem un TF.
- Izveido Git tag/commit konfigurācijai A un B.
- Atskaņo rosbag ar konfigurāciju A un saglabā rezultātus.
- Atskaņo tieši to pašu rosbag ar konfigurāciju B.
- Aprēķini vismaz trīs metrikas: trajektorijas garums, gala kļūda, apstrādes/frekvences stabilitāte vai cita projekta specifiska metrika.
- Uzraksti secinājumu, kas balstīts datos, nevis vizuālā iespaidā.

Metrika	A	B	Labāks	Pamatojums
Gala pozīcijas kļūda				
Trajektorijas garums				
Vid. frekvence / latency				

Metrika	A	B	Labāks	Pamatojums
Recovery skaits				
Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1				
2				
3				
4				
5				

R4-R9 kopējā evidence pakete

```

R4-R9 evidence/
├── configs/
│   ├── ekf_A.yaml
│   ├── ekf_B.yaml
│   ├── slam.yaml
│   └── nav2.yaml
├── bags/
├── screenshots/
├── metrics/
│   └── experiment.csv
├── source/
└── report.pdf / report.docx

```

54. Etalona konfigurācijas: EKF, SLAM un Nav2

Šīs konfigurācijas ir mācību etaloni, nevis universāls “copy-paste” risinājums. Parametri jāpielāgo robota mehānikai, sensoru frame nosaukumiem, ROS 2 distribūcijai un instalēto pakotņu versijām. Mācību metodikas mērķis ir, lai students spētu pamatot katru aktīvo parametru un spētu diagnosticēt, kas mainās pēc tā korekcijas.

54.1. robot_localization — 2D EKF etalona skelets

ekf.yaml — mācību sākuma konfigurācija

```

ekf_filter_node:
  ros__parameters:
    frequency: 30.0
    sensor_timeout: 0.2
    two_d_mode: true
    transform_time_offset: 0.0
    transform_timeout: 0.1
    print_diagnostics: true
    publish_tf: true

    map_frame: map
    odom_frame: odom
    base_link_frame: base_link
    world_frame: odom

```

```

odom0: /wheel/odom
odom0_config: [false, false, false,
               false, false, false,
               true,  false, false,
               false, false, true,
               false, false, false]
odom0_queue_size: 10

imu0: /imu/data
imu0_config: [false, false, false,
              false, false, false,
              false, false, false,
              false, false, true,
              false, false, false]
imu0_queue_size: 20
imu0_remove_gravitational_acceleration: true

```

Pasniedzējam uzsvērt: konfigurācijas vektors ir jālasa kā konkrētu stāvokļa komponentu izvēle, nevis kā nesaprotama TRUE/FALSE virkne. Studentam mutiski jāspēj pateikt, no kura sensora tiek izmantots v_x un no kura — yaw rate.

54.2. slam_toolbox — mācību kartēšanas skelets

slam.yaml — konceptuāls mācību skelets

```

slam_toolbox:
  ros__parameters:
    use_sim_time: false
    mode: mapping
    map_frame: map
    odom_frame: odom
    base_frame: base_link
    scan_topic: /scan

    resolution: 0.05
    max_laser_range: 8.0
    minimum_time_interval: 0.2
    transform_publish_period: 0.05
    map_update_interval: 2.0

    use_scan_matching: true
    use_scan_barycenter: true
    do_loop_closing: true

```

Svarīgi: konkrētās slam_toolbox versijas parametru nosaukumi un noklusējuma vērtības jāpārbauda instalētajā pakotnē. Mācību darbā students salīdzina vismaz vienu parametru izmaiņu uz vienādas trajektorijas vai pēc iespējas līdzīgas datu kopas.

54.3. Nav2 — minimālais parametru domāšanas modelis

Sistēma	Ko tā dara	Ko students novēro
map_server / localization	Nodrošina karti un robota pozu kartē	map→odom un pose stabilitāte
planner_server	Izveido globālo ceļu	global plan, ceļa garums, izbraucamība
controller_server	Ģenerē kustības komandas	/cmd_vel, svārstības, clearance
global_costmap	Ilgāka horizonta izmaksu telpa	karte + statiski/dinamiski šķēršļi
local_costmap	Lokāla, bieži ritoša izmaksu telpa	reālā laika šķēršļi
bt_navigator	Koordinē misijas loģiku	NavigateToPose statuss un recoveries

Nav2 local costmap parametru mācību fragments

Parametru fragments — piemērs, ne pilns nav2_params.yaml

local_costmap:

local_costmap:

ros__parameters:

global_frame: odom

robot_base_frame: base_link

rolling_window: true

width: 3.0

height: 3.0

resolution: 0.05

plugins: ["obstacle_layer", "inflation_layer"]

obstacle_layer:

plugin: "nav2_costmap_2d::ObstacleLayer"

observation_sources: scan

scan:

topic: /scan

data_type: "LaserScan"

marking: true

clearing: true

inflation_layer:

plugin: "nav2_costmap_2d::InflationLayer"

inflation_radius: 0.30

cost_scaling_factor: 3.0

54.4. Konfigurācijas pārvaldība kā programmatūras artefakts

- YAML faili tiek glabāti Git kopā ar kodu.
- Eksperimenta protokolā tiek norādīts commit hash vai tag.
- Parametru izmaiņas tiek veiktas pa vienam loģiskam blokam, nevis “mainīts viss reizē”.

- A/B testam maina skaidri definētu faktoru un saglabā pārējos apstākļus pēc iespējas nemainīgus.
- Config failam jābūt komentētam: kas ir izmērīts, kas pieņemts un kas vēl jāvalidē.

55. URDF/Xacro un TF — mobilā robota digitālais modelis

URDF apraksta robota kinemātisko struktūru, vizuālo ģeometriju, collision ģeometriju un inerciālos parametrus. Xacro ļauj šo XML aprakstu veidot parametrisku un atkārtoti lietojamu. ROS 2 TF savukārt nodrošina, lai sensora dati un robota komponentes tiktu interpretētas vienotā koordinātu sistēmu kokā.

55.1. Ieteicamais 2WD robota frame koks

2WD robota TF koka mācību modelis

```
map
├── odom
│   ├── base_footprint
│   │   └── base_link
│   │       ├── left_wheel_link
│   │       ├── right_wheel_link
│   │       ├── caster_link
│   │       ├── imu_link
│   │       ├── laser_frame
│   │       └── camera_link
```

map→odom parasti pieder lokalizācijas/SLAM slānim; odom→base_footprint vai base_link — odometrijas/stāvokļa novērtēšanas slānim; fiksētās transformācijas no base_link uz sensoru frame iegūst no robota ģeometrijas/URDF un robot_state_publisher.

55.2. URDF/Xacro pamatjēdzieni

Elements	Nozīme	Mācību piemērs
link	Stingrs ķermenis / frame	base_link, laser_frame
joint	Saikne starp linkiem	continuous wheel joint, fixed sensor joint
visual	Ko redz simulācijā/RViz	vienkāršota kaste/cilindrs
collision	Sadursmju ģeometrija	vienkāršota, stabila forma
inertial	masa un inerces tensors	vajadzīgs dinamiskai simulācijai
xacro property	parametrs	wheel_radius, track_width

55.3. Minimāls Xacro robota skelets

edu_bot.urdf.xacro — minimāls mācību skelets

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="edu_bot">
  <xacro:property name="wheel_radius" value="0.0325"/>
  <xacro:property name="track_width" value="0.155"/>

  <link name="base_footprint"/>

  <link name="base_link">
```

```

<visual>
  <geometry><box size="0.22 0.18 0.06"/></geometry>
</visual>
<collision>
  <geometry><box size="0.22 0.18 0.06"/></geometry>
</collision>
</link>

<joint name="base_joint" type="fixed">
  <parent link="base_footprint"/>
  <child link="base_link"/>
  <origin xyz="0 0 0.035" rpy="0 0 0"/>
</joint>

<link name="laser_frame"/>
<joint name="laser_joint" type="fixed">
  <parent link="base_link"/>
  <child link="laser_frame"/>
  <origin xyz="0.07 0 0.08" rpy="0 0 0"/>
</joint>

<link name="imu_link"/>
<joint name="imu_joint" type="fixed">
  <parent link="base_link"/>
  <child link="imu_link"/>
  <origin xyz="0 0 0.04" rpy="0 0 0"/>
</joint>
</robot>

```

55.4. Kāpēc sensora TF ir mērījuma daļa

Ja LiDAR atrodas 7 cm priekšā robota rotācijas centram, šo nobīdi nevar ignorēt. Rotācijas laikā sensora punkti pārvietojas citādi nekā robota centrs. Nepareizs sensor→base_link TF rada kartes “dubultās sienas”, lokālas deformācijas, nepareizu šķēršļu pozīciju un var sabojāt sensor fusion.

55.5. TF validācijas laboratorija U1

U1 — URDF/Xacro + TF validācija

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	60–90 min
Repozitorijs/commit	

- Izveidot Xacro ar base_link, imu_link, laser_frame un camera_link.
- Palaist robot_state_publisher.
- RViz parādīt RobotModel, TF un LaserScan/Camera frame, ja pieejams.
- Ar tf2_echo pārbaudīt vismaz divas transformācijas.

- Izmainīt laser_frame X nobīdi par +0.10 m un dokumentēt, kā RViz mainās sensora datu atrašanās vieta.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1				
2				
3				
4				
5				

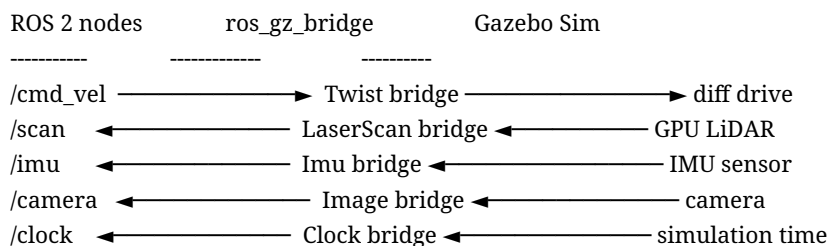
Tipiskās kļūdas

- Divi vecāki vienam frame.
- Cikls TF kokā.
- Sensora frame nav saistīts ar base_link.
- Nepareiza RPY ass/zīme.
- URDF vizuāli izskatās pareizi, bet collision/inertial dati simulācijā ir nereāli.

56. Robotu simulācija: Gazebo un Webots mācību plūsma

Simulators ļauj vienādos apstākļos atkārtot testu, radīt sensoru datus bez fiziska robota bojājuma riska un atdalīt programmatūras kļūdas no mehānikas problēmām. Simulācija tomēr nav realitāte: sensora troksnis, berze, riteņu slidēšana, tīkla aizture un mehāniskās tolerances jāievieš vai jānovērtē atsevišķi.

56.1. Gazebo + ROS 2 arhitektūra



Gazebo Harmonic dokumentācija un `ros_gz_sim_demos` demonstrē ROS 2 un Gazebo Transport savienošanu ar `ros_gz_bridge`, kā arī differential drive, camera, depth camera, GPU lidar un IMU datu piemērus. Mācību kursā galvenā vērtība ir vienots ROS 2 interfeiss: algoritms var klausīties `/scan` vai `/imu` neatkarīgi no tā, vai dati nāk no simulatora vai reālas ierīces.

56.2. Bridge YAML piemērs

ros_gz_bridge konfigurācijas konceptuāls piemērs

```
- ros_topic_name: /scan
  gz_topic_name: /scan
  ros_type_name: sensor_msgs/msg/LaserScan
  gz_type_name: gz.msgs.LaserScan
  direction: GZ_TO_ROS

- ros_topic_name: /imu/data
  gz_topic_name: /imu
  ros_type_name: sensor_msgs/msg/Imu
  gz_type_name: gz.msgs.IMU
  direction: GZ_TO_ROS

- ros_topic_name: /cmd_vel
  gz_topic_name: /cmd_vel
  ros_type_name: geometry_msgs/msg/Twist
  gz_type_name: gz.msgs.Twist
  direction: ROS_TO_GZ
```

56.3. Simulācijas modeļa validācijas kārtība

1. RobotModel/TF — vai ģeometrija un frame ir pareizi?
2. Joint/drive — vai pozitīvs `/cmd_vel` tiešām kustina robotu uz priekšu?
3. Odom — vai taisna kustība neizraisa mākslīgu yaw?
4. IMU — vai gravitācija/asu orientācija atbilst REP konvencijām?

5. LiDAR — vai /scan leņķi un range atbilst redzamajai videi?
6. Camera — vai CameraInfo/intrinsics ir pieejami un frame korekts?
7. Tikai pēc tam: EKF, SLAM, Nav2.

56.4. Webots kā alternatīvs mācību simulators

Webots nodrošina robotu un vides modelēšanu, sensorus/aktuatorus un kontrolieru programmēšanu vairākās valodās. To var izmantot kā alternatīvu laboratoriju vidi, īpaši, ja konkrētai platformai jau ir gatavs modelis vai students vēlas salīdzināt simulatoru uzvedību. Kursa metodikā svarīgākais ir saglabāt vienādu prasību un testa struktūru: input → algorithm → output → evidence.

56.5. S1 laboratorija — simulators pirms reāla robota

S1 — 2WD robots simulācijā: /cmd_vel → /odom + /scan + /imu

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	90 min
Repozitorijs/commit	

- Palaist 2WD simulatora modeli.
- Pārbaudīt /clock un use_sim_time.
- Nosūtīt taisnu 1 m kustību un 90° pagriezienu.
- Saglabāt /odom, /scan, /imu rosbag.
- Izveidot tabulu: komanda, simulācijas ground truth (ja pieejams), odom rezultāts.
- Pēc tam to pašu algoritmu palaist uz fiziska robota un salīdzināt atšķirības.

56.6. Sim-to-real diskusija

Simulācijā bieži idealizēts	Reālā robotā
riteņa rādiuss un track width precīzi	tolerances, riepas deformācija, nodilums
grīda vienmērīga	slīdēšana, nelīdzenumi, putekļi
timestamps/frekvences stabili	USB/Wi-Fi jitter, dropped frames
sensori ar kontrolētu troksni	temperatūra, atspīdumi, vibrācija, EMI
precīzs TF modelis	montāžas nobīdes un leņķa kļūdas

57. rosbag2 datu analīze ar Python un eksperimentu metriku

ROS sistēmā “strādāja vienu reizi” nav pietiekams inženiertehniskais pierādījums. rosbag2 ļauj saglabāt sensoru un sistēmas topic datus, bet Python ļauj no tiem iegūt kvantitatīvas metrikas. Šajā nodaļā students pāriet no ekrānuzņēmumiem uz datu analīzi: frekvence, latency aproksimācija, trajektorijas garums, gala kļūda, yaw drift, sensoru troksnis un A/B salīdzinājums.

57.1. Minimālā datu kopa autonomās navigācijas eksperimentam

- /tf un /tf_static
- /wheel/odom vai /odometry/filtered
- /imu/data
- /scan
- /cmd_vel
- Nav2 action/status un plāna topic, ja tie vajadzīgi konkrētajai metrikai
- /clock simulācijā

57.2. Python rosbag2 lasīšanas princips

ROS 2 Rolling satur rosbag2_py Python API. Praktiskajā kursā var izmantot SequentialReader vai augstāka līmeņa piemērus. Tā kā Python API un storage konfigurācija laika gaitā mainās, gatavo laboratorijas kodu jāpārbauda pret instalēto rosbag2_py versiju.

rosbag2_py SequentialReader — konceptuāls skelets

```
# Konceptuāls rosbag2_py lasīšanas skelets
import rosbag2_py

reader = rosbag2_py.SequentialReader()
storage_options = rosbag2_py.StorageOptions(uri='my_bag', storage_id='')
converter_options = rosbag2_py.ConverterOptions("", "")
reader.open(storage_options, converter_options)

while reader.has_next():
    topic, data, timestamp = reader.read_next()
    # data ir serializēta ROS ziņa; konkrētais tips jāzina no metadata
    # nākamais solis: deserializācija un metrikas aprēķins
    print(topic, timestamp)
```

57.3. Vienkāršas metrikas pēc eksportēšanas uz CSV

Python — trajektorijas garuma un gala kļūdas piemērs

```
import pandas as pd
import numpy as np

odom = pd.read_csv('odom.csv')

dx = np.diff(odom['x'])
dy = np.diff(odom['y'])
path_length = np.sum(np.sqrt(dx**2 + dy**2))
```

```

final_error = np.sqrt(
    (odom['x'].iloc[-1] - odom['x_ref'].iloc[-1])**2 +
    (odom['y'].iloc[-1] - odom['y_ref'].iloc[-1])**2
)

print(f"Path length: {path_length:.3f} m")
print(f"Final error: {final_error:.3f} m")

```

57.4. Topic frekvences aprēķins

```

timestamps = odom['t'].to_numpy()
dt = np.diff(timestamps)
frequency = 1.0 / dt

print('mean Hz =', np.mean(frequency))
print('std Hz =', np.std(frequency))
print('min Hz =', np.min(frequency))

```

Vidējā frekvence viena pati var slēpt periodiskus “caurumus”. Tāpēc jāskatās arī min/max, standartnovirze, percentiles vai atsevišķu dt histogramma. Reāllaika sistēmā dažas lielas pauzes var būt svarīgākas nekā labs vidējais rādītājs.

57.5. Metriku minimums R9 darbam

Metrika	Vienība	Ko tā palīdz atklāt
Trajektorijas garums	m	nevajadzīgi līkumi / replanning
Gala pozīcijas kļūda	m	lokālizācijas/navigācijas precizitāte
Gala yaw kļūda	° vai rad	orientācijas precizitāte
Vidējais + min topic Hz	Hz	datu plūsmas stabilitāte
Recovery skaits	reizes	navigācijas robustums
Misijas laiks	s	efektivitāte
Min. clearance	m	drošības rezerve

57.6. Datu analīzes grafiku noteikumi

- Grafīkam ir ass nosaukumi un mērvienības.
- A un B konfigurācijai izmanto vienu laika/koordinātu skalu.
- Grafīkā vai parakstā norāda datu failu un commit/config versiju.
- Ja dati filtrēti, dokumentē filtra tipu un parametrus.
- Nedrīkst izdzēst “sliktos punktus” bez tehniska pamatojuma.

58. Continuous Integration: GitHub Actions, testi un kvalitātes vārti

CI (Continuous Integration) automātiski pārbauda kodu pēc push vai pull request. Mācību kursā CI nav nepieciešams sarežģīts DevOps projekts — pietiek, ja students iemācās, ka kodam pirms merge automātiski jāiziet sintakses/lint, unit test un build pārbaudes. Tas nostiprina profesionālu disciplīnu un padara projekta kvalitāti reproducējamu.

58.1. CI pipeline mācību modelis

```
git push / pull request
↓
checkout source
↓
setup environment
↓
install dependencies
↓
lint / formatting checks
↓
unit tests
↓
build
↓
(optional) artifacts / test report
↓
PASS → review/merge
FAIL → fix before merge
```

58.2. Python GitHub Actions piemērs

.github/workflows/python-ci.yml — mācību piemērs

```
name: python-ci

on:
  push:
  pull_request:

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.12'
      - run: python -m pip install --upgrade pip
      - run: pip install -r requirements.txt
      - run: pytest -q
```

GitHub oficiālā dokumentācija uztur atsevišķus “building and testing Python” un citu valodu CI piemērus. Konkrēto actions major versiju laboratorijā jāpārbauda pirms semestra, jo Actions ekosistēma tiek regulāri atjaunināta.

58.3. C++/ROS 2 CI domāšanas modelis

```
# Konceptuāla darba plūsma ROS 2 paketēm
source ROS environment
rosdep install ...
colcon build --event-handlers console_direct+
colcon test
colcon test-result --verbose
```

Universitātes līmenī var izmantot ROS 2 konteineru vai sagatavotu runner vidi. Koledžas kursā pietiek ar vienu vienkāršu Python CI un lokālu C++/ROS 2 colcon build/test procedūru, ja pilna ROS 2 CI vides uzturēšana ir pārāk smaga.

58.4. Unit test piemērs Python kinemātikai

```
import math
from robot_math import forward_kinematics_2d

def test_zero_angles():
    x, y = forward_kinematics_2d(0.5, 0.3, 0.0, 0.0)
    assert math.isclose(x, 0.8, abs_tol=1e-9)
    assert math.isclose(y, 0.0, abs_tol=1e-9)

def test_vertical_first_link():
    x, y = forward_kinematics_2d(0.5, 0.3, math.pi/2, 0.0)
    assert math.isclose(x, 0.0, abs_tol=1e-9)
    assert math.isclose(y, 0.8, abs_tol=1e-9)
```

58.5. Kvalitātes vārti projektā

Vārti	Minimālā prasība
Q1 — Syntax/build	projekts kompilējas/palaižas tīrā vidē
Q2 — Unit tests	kritiskās funkcijas pārbaudītas ar robežgadījumiem
Q3 — Configuration	config faili versiju kontrolē; nav paroles/API keys
Q4 — Documentation	README satur instalēšanu, palaišanu un testu
Q5 — Evidence	commit hash ir sasaistīts ar testa protokolu
Q6 — Review	vismaz viens peer/pasniedzēja code review pirms gala merge

58.6. CI1 laboratorija — no kļūdas līdz zaļam pipeline

CI1 — Automātisks Python tests ar GitHub Actions

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	60–90 min
Repozitorijs/commit	

- Izveido mazu Python moduli ar 2–3 funkcijām.
- Izveido pytest testus, tostarp vienu robežgadījumu.
- Izveido GitHub Actions workflow.
- Apzināti ievieto vienu kļūdu un saglabā FAIL ekrānuzņēmumu/logu.
- Izlabo kļūdu un saglabā PASS evidence.
- Pull request aprakstā norādi, ko CI pārbaudīja un ko tas nepārbaudīja.

59. Visual SLAM un kameras ģeometrija

Visual SLAM izmanto attēlu secību, lai vienlaikus novērtētu kameras/robota kustību un veidotu vides reprezentāciju. Atšķirībā no 2D LiDAR SLAM, kamera tieši nemēra attālumu katram pikselim. Monokulārā sistēmā dziļums tiek secināts no kustības un ģeometrijas, stereo/RGB-D sistēmās ir papildu dziļuma informācija. Šī nodaļa paredzēta kā universitātes līmeņa turpinājums pēc Machine Vision un klasiskā SLAM ievada.

59.1. Pinhole kameras modelis

Kameras projekcijas pamatmodelis

$$\begin{aligned} s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \end{aligned}$$

K = camera intrinsic matrix

f_x, f_y = focal length pikseļu vienībās

c_x, c_y = principal point

Kameras kalibrācija nosaka intrinsics un lēcas deformācijas koeficientus. Bez kalibrācijas ģeometriskie algoritmi var radīt sistemātisku pozas un kartes kļūdu.

59.2. Visual odometry / SLAM apstrādes ķēde

```
Camera image
↓
preprocess / undistort
↓
feature detection + descriptors
↓
feature matching / tracking
↓
outlier rejection (RANSAC)
↓
relative pose estimation
↓
triangulation / map points
↓
local optimization
↓
loop detection + global correction
↓
robot/camera trajectory + map
```

59.3. ORB iezīmju demonstrācija Python

```
import cv2

img = cv2.imread('frame.png', cv2.IMREAD_GRAYSCALE)
orb = cv2.ORB_create(nfeatures=1000)
keypoints, descriptors = orb.detectAndCompute(img, None)
```

```

out = cv2.drawKeypoints(img, keypoints, None)
cv2.imwrite('orb_keypoints.png', out)
print('features:', len(keypoints))

```

Šis vēl nav SLAM. Tas ir tikai perception front-end pirmais solis: attēlā atrast atkārtojami identificējamus punktus un tiem aprēķināt descriptor. Students nedrīkst sajaukt “feature detection” ar lokalizāciju.

59.4. Divu kadru relatīvā poza

Ja ir atbilstoši punkti divos kalibrētas kameras kadros, var aprēķināt essential matrix un no tās atgūt relatīvo rotāciju un translācijas virzienu. Monokulārā gadījumā absolūtais translācijas mērogs no diviem kadriem vien nav zināms — tas ir fundamentāls ierobežojums, ko studentam jāspēj izskaidrot.

OpenCV — essential matrix un relatīvās pozas skelets

```

E, mask = cv2.findEssentialMat(
    pts1, pts2, K,
    method=cv2.RANSAC,
    prob=0.999,
    threshold=1.0
)

_, R, t, mask_pose = cv2.recoverPose(E, pts1, pts2, K)

```

59.5. PnP — pozas aprēķins no 3D–2D atbilstībām

Ja kartē jau ir 3D punkti un jaunā attēlā atrod to 2D projekcijas, Perspective-n-Point (PnP) ļauj novērtēt kameras pozu. OpenCV solvePnP ir klasisks instruments šīs ģeometrijas demonstrēšanai. Universitātes uzdevumā studenti salīdzina reprojection error pirms un pēc outlier filtrēšanas.

59.6. Visual SLAM atteices režīmi

Situācija	Kāpēc grūti	Iespējamā pieeja
vienkrāsaina siena	maz feature	LiDAR/IMU fusion, citas iezīmes
ātra kustība	motion blur	īsāka ekspozīcija, IMU, lielāks FPS
mainīgs apgaismojums	descriptor nestabilitāte	normalizācija, robustākas iezīmes
atkārtots raksts	nepareizi matches	RANSAC, ģeometriskā validācija
dinamiski cilvēki/auto	karte nav statiska	dynamic object filtering
monokulārs sākums	mēroga nenoteiktība	stereo/RGB-D/IMU vai zināms mērogs

59.7. V1 laboratorija — divu kadru visual odometry mini eksperiments

V1 — Feature matching → Essential matrix → Relative pose

Lauks	Aizpilda students
Vārds, uzvārds	
Datums	
Grupa	
Darba laiks	90–120 min
Repozitorijs/commit	

- Kalibrēta kamera vai dota K matrica.
- Uzņem divus kadrus ar nelielu pārvietojumu un statisku ainu.
- ORB + BFMatcher vai līdzvērtīga feature ķēde.
- Filtrē matches ar RANSAC, aprēķinot essential matrix.
- Atgūsti R un t virzienu.
- Parādi inlier/outlier skaitu un izskaidro translācijas mēroga problēmu.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1				
2				
3				
4				
5				

60. 3D LiDAR, PointCloud2 un punktu mākoņu apstrāde

2D LaserScan satur attālumu vienā skenēšanas plaknē. 3D LiDAR, depth camera vai RGB-D sensors var dot punktu mākonī — lielu 3D punktu kopu ar x, y, z un iespējami intensitātes, krāsas vai citiem laukiem. ROS 2 standarta ziņojums šim mērķim ir sensor_msgs/msg/PointCloud2.

60.1. PointCloud2 datu modelis

Jēdziens	Nozīme
header	timestamp + frame_id
height / width	organizēta vai neorganizēta punktu struktūra
fields	piem., x, y, z, intensity
point_step	viena punkta ieraksta izmērs baitos
row_step	vienas rindas izmērs
data	binārs punktu buferis
is_dense	vai nav nederīgu punktu

ROS 2 sensor_msgs dokumentācijā PointCloud2 ir paredzēts N-dimensionālu punktu kolekcijai ar papildu laukiem. Python pusē sensor_msgs_py.point_cloud2 nodrošina read_points un read_points_numpy palīgfunkcijas; C++ pusē ir iterator/modifier rīki un PCL konversijas.

60.2. Python PointCloud2 lasīšanas piemērs

```
from sensor_msgs_py import point_cloud2

def cloud_callback(msg):
    xyz = point_cloud2.read_points_numpy(
        msg,
        field_names=['x', 'y', 'z'],
        skip_nans=True
    )
    print('points:', xyz.shape[0])
    print('min xyz:', xyz.min(axis=0))
    print('max xyz:', xyz.max(axis=0))
```

60.3. Punktu mākoņa tipiskā apstrādes ķēde

```
raw PointCloud2
↓
frame transform (TF)
↓
range / crop filtering
↓
downsample (voxel grid)
↓
outlier removal
↓
plane / ground segmentation
↓
clustering / object extraction
```

↓
features / model / map
↓
planning or perception output

60.4. Voxel grid ideja

Voxel grid sadala telpu mazos 3D kubos un katru aizņemto kubu reprezentē ar vienu punktu vai centroid. Tas samazina punktu skaitu un aprēķinu slodzi. Leaf size ir kompromiss: pārāk mazs dod maz ietaupījuma; pārāk liels izdzēs ģeometriskas detaļas.

```
# Konceptuāls NumPy voxel downsample princips
voxel = 0.05 # 5 cm
keys = (xyz / voxel).astype(int)

# unikālie voxel indeksi; reālā algoritmā katram voxel aprēķina centroid
_, idx = np.unique(keys, axis=0, return_index=True)
xyz_down = xyz[idx]
```

60.5. PCL/ROS 2 ekosistēma

ROS 2 Rolling dokumentācijā `pcl_conversions` nodrošina konversijas starp PCL punktu struktūrām un `sensor_msgs/msg/PointCloud2`. `pcl_ros` satur filtrēšanas komponentes, piemēram, `PassThrough`, `CropBox`, `StatisticalOutlierRemoval` un `VoxelGrid`. Praktiskā laboratorijā šo bloku mērķis nav “izsaukt gatavu filtru”, bet izmērīt tā ietekmi uz punktu skaitu, latentumu un rezultāta ģeometriju.

60.6. TF un punktu mākoņi

Punktu mākonis vienmēr eksistē konkrētā frame. Lai apvienotu vairākus sensorus vai veidotu karti, punkti jāpārveido kopējā koordinātu sistēmā. `tf2_sensor_msgs` paredz transformācijas `sensor_msgs` tipiem, īpaši `PointCloud2`. Nepareizs timestamp vai frame rada “izsmērētu” karti kustībā pat tad, ja sensora mērījumi paši ir precīzi.

60.7. 3D LiDAR pret 2D LiDAR

Kritērijs	2D LiDAR	3D LiDAR
Datu forma	LaserScan plakne	PointCloud2 telpa
Datu apjoms	mazāks	daudz lielāks
Rampas/3D objekti	ierobežoti	labāk uztverami
Aprēķinu slodze	zemāka	augstāka
Kartēšana	2D occupancy	3D point/voxel/surface map
Mācību sākums	ļoti piemērots	pēc 2D SLAM pamatiem

60.8. P1 laboratorija — PointCloud2 filtrēšana un kvantitatīva analīze

P1 — 3D point cloud: crop → downsample → statistika

Lauks	Aizpilda students
Vārds, uzvārds	

Lauks	Aizpilda students
Datums	
Grupa	
Darba laiks	90–120 min
Repozitorijs/commit	

- Iegūt PointCloud2 no simulatora, depth camera vai 3D LiDAR datu faila.
- Pārveidot punktus izvēlētajā frame.
- Atņemt punktus ārpus noteiktas telpas (crop).
- Veikt downsample ar vismaz diviem voxel izmēriem.
- Pierakstīt punktu skaitu un apstrādes laiku.
- Vizuāli un kvantitatīvi salīdzināt detaļu zudumu.

Variants	Voxel, m	Punktu skaits	Apstrādes laiks, ms	Piezīmes
Raw	—			
A	0.02			
B	0.05			
C	0.10			
Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1				
2				
3				
4				
5				

61. Integrētais universitātes projekts — multisensoru autonomi robots

Šis projekts ir paredzēts kā materiāla augstākais integrācijas līmenis. Students neizstrādā visu no nulles; viņš projektē sistēmas arhitektūru, izvēlas datu avotus, integrē esošos ROS 2 komponentus ar savu Python/C++ kodu, veic kalibrēšanu, definē prasības un pierāda rezultātu ar atkārtotajiem eksperimentiem.

61.1. Projekta misija — piemērs

Mobilam robotam autonomi jāizbrauc zināmā iekštelpu kartē no starta zonas uz trim mērķpunktiem, jāizvairās no statiskiem un vismaz viena dinamiska šķēršļa, jāreģistrē sensora dati un pēc misijas jā sagatavo kvantitatīva atskaite. Paplašinātajā variantā kamera identificē mērķa objektu vai marķieri un sasaista to ar robota pozu.

61.2. Funkcionālais koks

To execute an autonomous inspection mission

- └─ To perceive the environment
 - | └─ To acquire 2D/3D LiDAR data
 - | └─ To acquire IMU data
 - | └─ To acquire camera data
- └─ To estimate robot state
 - | └─ To compute wheel odometry
 - | └─ To fuse odometry + IMU
- └─ To localize and map
 - | └─ To maintain map→odom transform
 - | └─ To detect localization degradation
- └─ To plan and navigate
 - | └─ To generate global path
 - | └─ To avoid local obstacles
 - | └─ To execute recoveries
- └─ To collect mission evidence
 - | └─ To record rosbag
 - | └─ To store config + commit hash
 - | └─ To calculate metrics
- └─ To fail safely
 - | └─ To stop on critical sensor loss
 - | └─ To report fault
 - | └─ To allow manual intervention

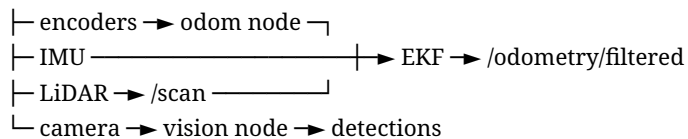
61.3. Minimālās prasības

ID	Prasība	Acceptance criterion
REQ-01	Robots izpilda 3 secīgus NavigateToPose mērķus	3/3 sasniegti bez manuālas kustības komandas
REQ-02	Šķēršļu izvairīšanās	nav kontakta; min. clearance $\geq$ definētā robeža
REQ-03	Stāvokļa novērtējums	/odometry/filtered bez kritiskas divergences testa laikā

ID	Prasība	Acceptance criterion
REQ-04	Datu reģistrācija	rosvbag satur definēto obligāto topic kopu
REQ-05	Reproducējamība	repo commit + config + bag ļauj atkārtot analīzi
REQ-06	Droša atteice	kritiska /scan vai odom zuduma gadījumā sistēma apstājas vai pāriet definētā drošā stāvoklī

61.4. Programmatūras arhitektūra

Sensors / hardware



Localization / mapping

/scan + filtered odom + TF → SLAM/localization → map→odom

Navigation

map + pose + /scan → Nav2 → /cmd_vel → motor controller

Evidence

selected topics → rosvbag2

configs + code → Git

bag + reference → Python metrics/report

61.5. Projekta izstrādes fāzes

Fāze	Rezultāts	Stop/Go kritērijs
P0 — Requirements	prasības + functional tree	prasības ir testējamas
P1 — Robot model	URDF/Xacro + TF	TF koks korekts
P2 — Sensors	/imu, /scan, odom	katrs sensors atsevišķi PASS
P3 — State estimation	EKF	R4 PASS
P4 — Mapping/localization	karte	R5 PASS
P5 — Navigation	costmap + Nav2	R6/R7 PASS
P6 — Robustness	BT/recovery	R8 PASS
P7 — Evidence	rosvbag A/B	R9 PASS
P8 — Advanced perception	vision/3D cloud	V1 vai P1 PASS
P9 — Final demo	pilna misija	acceptance criteria PASS

61.6. Universitātes līmeņa pētnieciskais papildinājums

- Salīdzināt EKF konfigurācijas uz identiska rosvbag un statistiski pamatot izvēli.

- Salīdzināt 2D LiDAR SLAM un Visual/Visual-Inertial odometry konkrētā scenārijā.
- Analizēt sensoru timestamp nobīdes ietekmi uz trajektorijas kļūdu.
- Veikt point-cloud downsampling parametru A/B/C eksperimentu un mērīt latency/geometry kompromisu.
- Izstrādāt custom ROS 2 node, kas publicē projekta specifisku diagnostikas metriku.
- Izveidot CI, kas automātiski palaiž unit testus kinemātikas/perception matemātikai.

61.7. Gala aizstāvēšanas struktūra

1. Problēma un prasības — 2 min.
2. Functional tree un sistēmas arhitektūra — 3 min.
3. Kalibrēšana un sensori — 3 min.
4. Programmatūra/ROS 2 un Git — 3 min.
5. Demonstrācija — 5–8 min.
6. Testu rezultāti un metrikas — 4 min.
7. Atteices režīmi un ierobežojumi — 2 min.
8. Jautājumi / koda aizstāvēšana — 5–10 min.

61.8. Projekta vērtēšanas rubrika — universitātes variants

Kritērijs	Svars	Ko vērtē
Prasības un arhitektūra	15%	traceability, functional tree, interfeisi
Matemātiskais pamatojums	15%	kinemātika, covariance, ģeometrija, metrikas
Programmatūras kvalitāte	15%	struktūra, Git, testi, CI
Sensoru/TF integrācija	15%	frames, timing, kalibrācija
Autonomā navigācija	15%	lokālizācija, planning, control, recovery
Eksperimentu kvalitāte	15%	roslab, A/B, reproducējamība
Dokumentācija/aizstāvēšana	10%	skaidrojums, ierobežojumi, profesionāls noformējums

62. [ARHĪVS] Versijas 1.0 statuss un turpmākā attīstības karte

Versija 1.0 noslēdz pirmo pilno vienotā mācību dokumenta ciklu: no algoritmiskās domāšanas un Python/C++ pamatiem līdz SQL/IoT, mehatronikai, sensoru programmēšanai, robotu kinemātikai, LiDAR/Machine Vision/IMU, ROS 2, EKF, SLAM, Nav2, rosbag, Git/CI, Visual SLAM un 3D punktu mākoņiem. Dokumentā vienlaikus ir kursa apstiprināšanas struktūra, pasniedzēja teorija, skolēna/studentu darba lapas, laboratorijas, pārbaudes darbi, projektu veidnes un universitātes līmeņa paplašinājumi.

62.1. Kas v1.0 ir pabeigts

- 36 × 90 min gada kursa karkass un pilni pasniedzēja scenāriji.
- Algoritmu, Python, C++, SQL/IoT, mehatronikas un robotikas teorijas slānis.
- H1–H12 aparatūras laboratorijas un R1–R9 ROS 2 progresija.
- Functional Tree metodika un sistēmu inženierijas dokumentēšana.
- URDF/Xacro/TF, simulācijas, EKF, SLAM, Nav2 un rosbag datu analīzes pāreja.
- Visual SLAM un 3D LiDAR/PointCloud2 universitātes ievads.
- Git/GitHub + CI profesionālās programmatūras izstrādes pamati.
- Eksāmenu un projektu vērtēšanas bankas.

62.2. Ieteicamā nākamā attīstība pēc v1.0

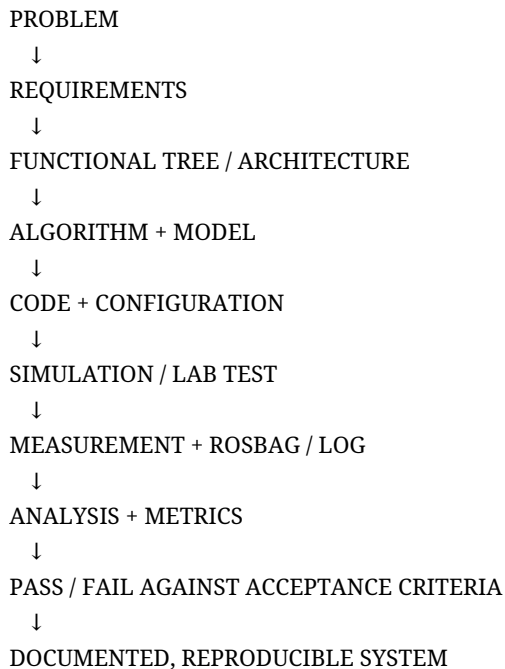
Virziens	Ko pievienot
Pasniedzēja komplekts	atsevišķi slaidi katrai lekcijai + demonstrāciju faili
Studentu grāmata	saīsināta versija bez pasniedzēja atbildēm
Repozitorijs	gatavs edu_bot ROS 2 workspace ar config/launch/test failiem
Simulācijas komplekts	Gazebo/Webots pasaules un standartizēti scenāriji
Datu komplekts	rosbag paraugi EKF/SLAM/Nav2/vision/point cloud laboratorijām
Universitātes 2. līmenis	optimizācija, factor graphs, nonlinear least squares, MPC
Embedded/RTOS	ESP32 FreeRTOS, interrupt timing, queues, CAN/RS-485
Industrial systems	PLC/OPC UA/Modbus, SCADA, industrial cybersecurity
AI robotics	object detection, semantic segmentation, learned perception ar stingru validāciju

62.3. Kursa galvenā profesionālā doma

Studentam kursa beigās jāspēj paskaidrot ne tikai “kā uzrakstīt kodu”, bet arī: kāpēc sistēma ir sadalīta funkcijās; no kurienes rodas dati; kā tie tiek mērīti un validēti; kā algoritms pieņem

lēmumu; kā programmatūra ietekmē fizisku procesu; kā tiek pierādīts, ka sistēma darbojas; un kā rezultātu iespējams atkārtot citam inženierim.

Kursa gala inženiertehniskā darba plūsma



Primārie tehniskie avoti un versiju piezīme

Avoti pārbaudīti 18.08.2026. ROS 2 Rolling, simulatoru un bibliotēku interfeisi var mainīties; laboratorijā vienmēr jāpārbauda tieši instalētās distribūcijas/pakotnes dokumentācija un piemēri. Zemāk norādītie avoti izmantoti kā tehnisko jēdzienu un interfeisu pārbaudes punkts, nevis kā vienīgā mācību literatūra.

- ROS 2 sensor_msgs Rolling: https://docs.ros.org/en/rolling/p/sensor_msgs/
- ROS 2 sensor_msgs_py / PointCloud2 Python tools: https://docs.ros.org/en/rolling/p/sensor_msgs_py/
- ROS 2 pcl_conversions / pcl_ros Rolling: https://docs.ros.org/en/rolling/p/pcl_conversions/ un https://docs.ros.org/en/rolling/p/pcl_ros/
- ROS 2 rosbag2 / rosbag2_py Rolling: <https://docs.ros.org/en/rolling/p/rosbag2/> un https://docs.ros.org/en/rolling/p/rosbag2_py/
- Gazebo Harmonic ROS 2 integration: https://gazebo-sim.org/docs/harmonic/ros2_integration/
- ROS + Gazebo Sim demos: https://docs.ros.org/en/rolling/p/ros_gz_sim_demos/
- Webots User Guide: <https://www.cyberbotics.com/doc/guide/>
- GitHub Actions — build and test code: <https://docs.github.com/en/actions/tutorials/build-and-test-code>
- OpenCV calib3d / pose geometry: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

63. Factor graphs, nelineārā optimizācija un Bundle Adjustment

Šī nodaļa paceļ lokalizācijas un SLAM tēmu no “filtru apvieno sensorus” uz vispārīgāku optimizācijas skatījumu. Factor graph apraksta, kā nezināmie sistēmas stāvokļi ir saistīti ar mērījumiem, kustības modeli un citiem ierobežojumiem. Tas ir īpaši noderīgi SLAM, Structure from Motion, Visual Odometry un multisensoru sapludināšanā.

63.1. Mainīgie, faktori un mērījumu ierobežojumi

Factor graph ir bipartīts grafiks: viena mezglu grupa attēlo nezināmos mainīgos, otra — faktoros jeb ierobežojumus. Robotikā mainīgie var būt robota pozas $x_0 \dots x_n$, orientācija, ātrums, IMU bias vai 3D landmark punkti. Faktori var būt odometrija, LiDAR scan matching, kamera, GPS, IMU vai loop-closure mērījums.

Factor graph vienkāršota shēma

```
VARIABLES:  x0 --- x1 --- x2 --- x3
              \   \       /
FACTORS:      O01  O12      LC30
              \   \       /
MEASUREMENTS: odom  odom  loop closure
```

Svarīgā doma auditorijai: sensors nedod “patiesību”; sensors dod mērījumu ar kļūdu. Optimizācija meklē tādu stāvokļu kopumu, kas vislabāk saskaņo visus mērījumus vienlaikus.

63.2. Residual, svars un izmaksu funkcija

Katrs faktors rada residual jeb atlikuma kļūdu: starpību starp to, ko paredz pašreizējais stāvokļa novērtējums, un to, ko sensors izmērīja. Vienkāršoti optimizācija minimizē svērtu residualu kvadrātu summu.

Factor graph izmaksu funkcijas ideja

$$r_i(x) = z_i - h_i(x)$$
$$J(x) = \sum r_i(x)^T W_i r_i(x)$$

kur:

z_i — mērījums,

h_i — mērījuma modelis,

W_i — uzticamības/svara matrica.

Ja mērījums ir ļoti trokšņains, tam piešķir mazāku svaru. Ja mērījums ir precīzs, tas vairāk ietekmē gala risinājumu. Šis pats princips jau bija sastopams Kalman filtrā, tikai tagad to skatām kā globālu optimizācijas problēmu.

63.3. Jacobian, Gauss–Newton un Levenberg–Marquardt ideja

Daudzi robotikas mērījumu modeļi ir nelineāri. Tāpēc funkciju lokāli linearizē ap pašreizējo novērtējumu, izmantojot Jacobian. Gauss–Newton atkārtoti aprēķina korekciju, līdz residuali vairs būtiski nesamazinās. Levenberg–Marquardt pievieno slāpēšanu, kas var palīdzēt situācijās ar sliktu sākotnējo novērtējumu.

- Sākuma novērtējums ir svarīgs — īpaši 3D rotācijām un lielām pozas kļūdām.

- Optimizācijas konverģence nenozīmē, ka atrasts globāli pareizais risinājums.
- Nepareiza datu asociācija var radīt matemātiski “labu”, bet fiziski nepareizu karti.

63.4. Robust kernels un outlier problēma

Reālā vidē daļa mērījumu ir kļūdaini: nepareizi feature match, kustīgs objekts, LiDAR atstarojums vai kļūdainas loop closure. Robust loss funkcijas samazina ļoti lielu residualu ietekmi, lai viens outlier neizjauktu visu optimizāciju.

Pieeja	Ideja	Kad noder
Least squares	Visi residuali tiek kvadrēti	Tīri dati bez būtiskiem outlier
Huber	Mazas kļūdas kvadrātiskas, lielas mazāk agresīvas	Vispārīgs robusts risinājums
Cauchy/Tukey	Spēcīgi ierobežo lielu residualu ietekmi	Daudz nepareizu datu asociāciju

63.5. Bundle Adjustment

Bundle Adjustment vienlaikus optimizē kameru pozas un 3D punktu koordinātas, minimizējot reprojection error. Tas ir viens no Visual SLAM un Structure from Motion centrālajiem optimizācijas posmiem.

Bundle Adjustment mērījuma ķēde

```

3D point P_j
  ↓ project with camera pose X_i and intrinsics K
predicted pixel ũ_ij
  ↓ compare
measured pixel u_ij
  ↓
reprojection error e_ij = u_ij - ũ_ij

```

Ko uzsvērt: kamera mēra pikseļu koordinātas, nevis tieši 3D attālumu. 3D struktūra un kameras kustība tiek novērtēta no ģeometriskajām sakarībām starp daudziem kadriem.

63.6. GTSAM konceptuāls piemērs

Factor graph veidošanas pseidokods

```

# Pseudokods / GTSAM domāšanas modelis
graph.add(PriorFactorPose2(X0, prior_pose, prior_noise))
graph.add(BetweenFactorPose2(X0, X1, odom01, odom_noise))
graph.add(BetweenFactorPose2(X1, X2, odom12, odom_noise))
graph.add(BetweenFactorPose2(X2, X0, loop20, loop_noise))

initial.insert(X0, guess0)
initial.insert(X1, guess1)
initial.insert(X2, guess2)

result = optimizer.optimize()

```

63.7. Functional Tree — Factor Graph SLAM

Functional Tree — optimizācijas SLAM

To estimate a globally consistent robot trajectory and map

- └─ To acquire constraints
 - └─ odometry / IMU
 - └─ LiDAR or visual relative pose
 - └─ loop-closure constraints
- └─ To model uncertainty
 - └─ covariance / noise model
 - └─ robust loss for outliers
- └─ To optimize state variables
 - └─ linearize residuals
 - └─ solve state correction
 - └─ iterate until convergence
- └─ To validate result
 - └─ inspect residuals
 - └─ check trajectory consistency
 - └─ compare against reference/ground truth

FG1 — Sintētisks pose-graph eksperiments

Mērķis: Parādīt, kā loop closure var izlabot uzkrāto odometrijas drift un kā nepareizs loop closure var sabojāt trajektoriju.

Nepieciešamais aprīkojums / programmatūra:

- Python/NumPy vai GTSAM vide
- Iepriekš sagatavota sintētiska 2D trajektorija
- CSV tabula ar odometrijas un loop-closure mērījumiem

Darba secība:

1. Izveidot 8–12 pozu trajektoriju ar nelielu odometrijas drift.
2. Uzzīmēt tikai odometrijas trajektoriju.
3. Pievienot pareizu loop-closure ierobežojumu un optimizēt.
4. Salīdzināt gala pozīcijas kļūdu pirms/pēc optimizācijas.
5. Atkārtot ar vienu apzināti nepareizu loop closure.
6. Dokumentēt residualus un secināt, kā atpazīt kļūdainu constraint.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Odometrija bez loop closure	redzams drift		
2	Pareizs loop closure	gala kļūda samazinās		
3	Nepareizs loop closure	parādās deformācija/lieli residuali		

Analīzes jautājumi:

- Kāpēc loop closure ir spēcīgs, bet bīstams mērījums?
- Kā covariance maina faktora ietekmi?
- Kāpēc optimizatoram vajag sākuma novērtējumu?

Vērtēšanas / acceptance criteria:

- Students spēj izskaidrot mainīgo un faktoru atšķirību.
- Ir kvantitatīvs pirms/pēc salīdzinājums.
- Secinājumā apspriesti outlier un robust loss.

64. Visual-Inertial Odometry: kamera + IMU

Visual-Inertial Odometry (VIO) apvieno kameras ģeometrisku informāciju ar IMU augstas frekvences paātrinājuma un leņķiskā ātruma mērījumiem. Kamera palīdz ierobežot IMU drift, bet IMU palīdz uzturēt kustības novērtējumu starp attēlu kadriem un ātras kustības laikā.

64.1. VIO stāvokļa vektors

Tipisks VIO stāvoklis

$$\mathbf{x} = [\mathbf{R}, \mathbf{p}, \mathbf{v}, \mathbf{b}_g, \mathbf{b}_a]$$

$\mathbf{R}$ — orientācija

$\mathbf{p}$ — pozīcija

$\mathbf{v}$ — lineārais ātrums

$\mathbf{b}_g$ — žiroskopa bias

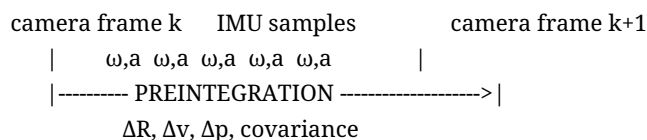
$\mathbf{b}_a$ — akselerometra bias

Augstskolas līmenī studentam jāspēj paskaidrot, kāpēc bias ir stāvokļa daļa. Ja bias netiek modelēts, integrācijas kļūda ar laiku uzkrājas un kļūst par ievērojamu pozas drift.

64.2. IMU preintegration

IMU parasti strādā daudz augstākā frekvencē nekā kamera. Tā vietā, lai optimizācijā ievietotu katru atsevišķo IMU mērījumu kā atsevišķu stāvokli, mērījumus starp diviem kameras kadriem var preintegrēt vienā kustības ierobežojumā. Preintegration akumulē relatīvo rotāciju, ātruma un pozīcijas izmaiņu kopā ar nenoteiktību.

IMU preintegration starp kameras kadriem



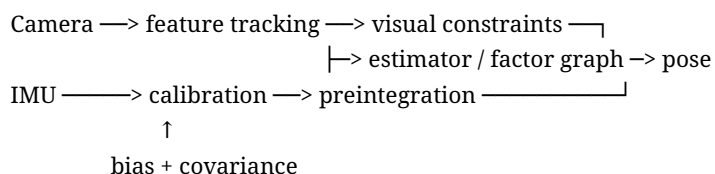
64.3. Laika sinhronizācija un extrinsic calibration

VIO sistēmā viena no kritiskākajām praktiskajām problēmām ir nevis formula, bet pareiza sensoru sinhronizācija. Kameras attēlam un IMU datiem jāattiecas uz korekti saskaņotu laiku. Tāpat jāzina transformācija starp kameras un IMU koordinātu sistēmām.

Kļūda	Simptoms	Diagnostikas ideja
Timestamp offset	poza “atpaliek” vai svārstās ātras kustības laikā	salīdzināt angular rate ar vizuāli novērotu rotāciju
Nepareiza IMU ass	roll/pitch/yaw mainās nepareizā virzienā	statiskas orientācijas tests + TF pārbaude
Nepareizs camera<->IMU extrinsic	trajektorija deformējas rotācijas laikā	kalibrācijas target + transformācijas validācija
Nezināms bias	drift pat stāvot mierā	ilgāks statisks IMU logs un vidējās vērtības analīze

64.4. VIO datu plūsma

Visual-Inertial pipeline



64.5. Kad VIO neizdodas?

- Vienveidīga siena vai ļoti maz tekstūras — vizuālo feature ir par maz.
- Motion blur vai pārāk ilgs exposure — feature tracking kļūst nestabils.
- Ļoti strauja rotācija — kamera var zaudēt atbilstības, IMU kļūst kritiski svarīgs.
- Ilgstoša konstanta kustība bez pietiekamas observability — daži stāvokļi kļūst vāji novērojami.
- Nepareiza sensoru kalibrācija vai laika nobīde — sistēma var būt nestabila pat ar labiem sensoriem.

VIO1 — Kameras un IMU datu sinhronizācijas eksperiments

Mērķis: Iemācīties kvantitatīvi novērtēt timestamp nobīdi un IMU bias, pirms mēģināt pilnu VIO algoritmu.

Nepieciešamais aprīkojums / programmatūra:

- IMU CSV ar t, gx, gy, gz, ax, ay, az
- Kameras/pose CSV vai video kadru timestamp saraksts
- Python, pandas, NumPy, matplotlib

Darba secība:

1. Atlasīt 10–20 s segmentu ar skaidru robota rotāciju.
2. Aprēķināt IMU gyroscope magnitūdu un attēlot laikā.
3. No kameras vai reference pose iegūt rotācijas notikumu laikā.
4. Nobīdīt vienu signālu ± 200 ms diapazonā un atrast korelācijas maksimumu.
5. Novērtēt statistiskā segmenta gyro bias.
6. Dokumentēt atrasto time offset un bias.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Statistsks IMU segments	vidējais gyro tuvu konstantam bias		
2	Rotācijas notikums	camera/IMU maksimumi sakrīt pēc korekcijas		

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
3	Nobīdes meklēšana	iegūta viena argumentēta offset vērtība		

Analīzes jautājumi:

- Kāpēc 20–50 ms nobīde ātras rotācijas laikā var būt būtiska?
- Kā atšķiras sensor timestamp un datora saņemšanas laiks?
- Kāpēc time synchronization ir sistēmu inženierijas, ne tikai algoritma problēma?

Vērtēšanas / acceptance criteria:

- Grafikos redzami abi signāli.
- Ir aprēķināts offset un bias.
- Secinājumā norādīti ierobežojumi un nenoteiktība.

65. 3D LiDAR registrācija: ICP un NDT

3D LiDAR registrācijas mērķis ir noteikt rigid transformāciju starp diviem punktu mākoņiem. To izmanto odometrijā, kartēšanā, loop closure kandidātu pārbaudē un 3D modeļu savietošanā.

65.1. Rigid transformācija

Punktu mākoņa pozas modelis

$$p_{\text{target}} \approx R \cdot p_{\text{source}} + t$$

R — 3×3 rotācijas matrica

t — 3×1 translācija

T — 4×4 homogēnā transformācija

65.2. ICP — Iterative Closest Point

ICP atkārti trīs pamatsoļus: atrod atbilstības starp source un target punktiem, aprēķina transformāciju, kas samazina attālumu starp atbilstībām, un atkārti līdz konverģencei. ICP ir lokāls algoritms — labs sākuma novērtējums būtiski ietekmē rezultātu.

ICP algoritma cikls

1. Initial guess T_0
 2. Transform source cloud
 3. Find correspondences
 4. Reject invalid/outlier correspondences
 5. Estimate ΔT
 6. $T \leftarrow \Delta T \cdot T$
 7. Check convergence; repeat
- Point-to-point ICP minimizē punktu savstarpējos attālumus.
 - Point-to-plane variants izmanto virsmas normāles un bieži labāk strādā strukturētās 3D vidēs.
 - Downsampling un outlier removal var samazināt aprēķinu izmaksas un trokšņu ietekmi.

65.3. NDT — Normal Distributions Transform

NDT sadala target point cloud telpas šūnās un punktus katrā šūnā apraksta ar statistisku sadalījumu. Reģistrācija meklē transformāciju, kas padara pārveidotos source punktus ticamus šajos sadalījumos. Tas var būt noderīgs lielākiem punktu mākoņiem un transportlīdzekļu lokalizācijas scenārijiem.

Aspekts	ICP	NDT
Datu modelis	punktu atbilstības	statistisks modelis voxel šūnās
Sākuma poza	parasti vajag labu initial guess	arī gūst labumu no initial guess
Tipisks tunings	max correspondence, iterations, epsilon	resolution, step size, epsilon
Rezultāta pārbaude	fitness + overlap + transformācija	fitness + convergence + transformācija

65.4. PCL ICP kodola piemērs

PCL ICP mācību piemērs

```
pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp;
icp.setInputSource(source);
icp.setInputTarget(target);
icp.setMaximumIterations(50);
icp.setMaxCorrespondenceDistance(0.5);

pcl::PointCloud<pcl::PointXYZ> aligned;
icp.align(aligned, initial_guess);

if (icp.hasConverged()) {
    std::cout << icp.getFitnessScore() << std::endl;
    std::cout << icp.getFinalTransformation() << std::endl;
}
```

65.5. Reģistrācijas kvalitāte

Nedrīkst vērtēt tikai “hasConverged()”. Jāpārbauda arī transformācijas fizikālā ticamība, overlap, residual/fitness, sākuma pozas jutība un rezultāts pret reference trajectory, ja tāda pieejama.

Metrika	Ko tā palīdz pamanīt
Translation error	nepareizu x/y/z nobīdi
Rotation error	yaw/pitch/roll kļūdu
Fitness / residual	sliktu ģeometrisku savietojumu
Inlier ratio	mazu faktiskā overlap daļu
Runtime	vai algoritms ir praktiski lietojams reālajā frekvencē

LR1 — ICP/NDT parametrizācijas un sākuma pozas jutība

Mērķis: Eksperimentāli salīdzināt reģistrācijas kvalitāti, ja mainās initial guess un point-cloud downsampling.

Nepieciešamais aprīkojums / programmatūra:

- Divi PCD punktu mākoņi ar daļēju pārklājumu
- PCL vai sagatavota laboratorijas programma
- Reference transformācija, ja pieejama

Darba secība:

1. Palaist reģistrāciju ar labu initial guess.
2. Saglabāt fitness, iterāciju skaitu, runtime un transformāciju.
3. Atkārtot ar palielinātu translācijas un yaw kļūdu sākuma pozā.
4. Atkārtot ar 2–3 voxel izmēriem.
5. Salīdzināt ICP un NDT vai divas ICP konfigurācijas.

6. Izveidot tabulu ar kļūdu un runtime kompromisu.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Labs initial guess	stabila konvergence		
2	Sliktāks initial guess	novērtēta konverģences robeža		
3	Downsampling	runtime samazinās; kvalitāte kvantificēta		

Analīzes jautājumi:

- Kāpēc “converged” var būt nepareizā lokālā minimumā?
- Kā voxel size ietekmē precizitāti un ātrumu?
- Kādus citus sensorus izmantotu initial guess iegūšanai?

Vērtēšanas / acceptance criteria:

- Ir vismaz 6 eksperimentu konfigurācijas.
- Rezultāti salīdzināti ar metriku, ne tikai ekrānattēlu.
- Students pamato izvēlēto parametru komplektu.

66. 3D kartēšana: voxel, occupancy un OctoMap

Punktu mākonis ir sensora novērojums konkrētā laikā; karte ir ilglaicīgāks telpas modelis. 3D occupancy mapping sadala telpu šūnās un glabā, vai telpa ir brīva, aizņemta vai vēl nezināma.

66.1. Voxel un izšķirtspēja

Voxel ir 3D analogs pikselim. Jo mazāks voxel izmērs, jo vairāk detaļu, bet lielāks atmiņas patēriņš un apstrādes laiks. Izšķirtspēja jāizvēlas atbilstoši robota izmēram, sensora precizitātei un uzdevumam.

Voxel izmērs	Ieguvums	Risks
liels	mazāka atmiņa, ātrāka plānošana	pazūd šauri šķēršļi un detaļas
vidējs	kompromiss	jāvalidē pret robota footprint
mazs	detalizēta ģeometrija	daudz mezglu, liels CPU/RAM patēriņš

66.2. Occupancy probability un log-odds

Probabilistiska karte nepasaka tikai 0 vai 1. Katrs sensora “hit” palielina ticamību, ka voxel ir aizņemts, bet rays pirms hit punkta var palielināt ticamību, ka šī telpa ir brīva. Log-odds forma ļauj atkārtotus Bayes update realizēt kā saskaitīšanu.

Occupancy update ideja

$$L_t(m) = L_{t-1}(m) + \text{inverse\_sensor\_model}(z_t) - L_0$$

occupied → probability virs occupancy threshold

free → probability zem free threshold

unknown → nav pietiekamas informācijas

66.3. Octree un OctoMap

Octree rekursīvi sadala 3D telpu astoņās apakššūnās. Homogēnas zonas var glabāt rupjākā līmenī, bet sarežģītas zonas — detalizētāk. Tas ļauj veidot kompaktu daudzizšķirtspējas 3D occupancy modeli.

Octree struktūras ideja

```
ROOT CUBE
├─ octant 0
├─ octant 1
├─ octant 2
├─ octant 3
├─ octant 4
├─ octant 5
├─ octant 6
├─ octant 7
└─ subdivide again only where detail is needed
```

66.4. 2D costmap pret 3D map

Jautājums	2D occupancy/costmap	3D OctoMap/voxel map
Ko modelē?	x,y plakni	x,y,z telpu
Navigācija pa grīdu	ļoti piemērota	jāprojicē/interpretē
Droni/manipulatori	ierobežoti	dabiski piemērota
Atmiņa	mazāka	lielāka
Sensori	2D LiDAR	3D LiDAR/depth camera/stereo

VXM1 — Vienkārša 3D voxel occupancy analīze Python

Mērķis: No PointCloud2/CSV punktiem izveidot voxel indeksus, analizēt punktu skaitu voxel šūnās un izšķirtspējas ietekmi.

Nepieciešamais aprīkojums / programmatūra:

- XYZ punktu CSV vai PCD→CSV eksports
- Python, NumPy, pandas

Darba secība:

1. Ielādēt x,y,z punktus.
2. Definēt voxel izmēru r.
3. Aprēķināt indeksus $\text{floor}(x/r)$, $\text{floor}(y/r)$, $\text{floor}(z/r)$.
4. Saskaitīt punktus katrā voxel.
5. Atkārtot ar vismaz 3 izšķirtspējām.
6. Salīdzināt voxel skaitu un atmiņas/izšķirtspējas kompromisu.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	r=0.20 m	rupja karte, maz voxel		
2	r=0.10 m	vidējs kompromiss		
3	r=0.05 m	vairāk detaļu un voxel		

Analīzes jautājumi:

- Kāpēc “occupied voxel” no viena point cloud vēl nav pilns occupancy sensor modelis?
- Kā modelēt free space gar LiDAR staru?
- Kāpēc unknown jāatšķir no free?

Vērtēšanas / acceptance criteria:

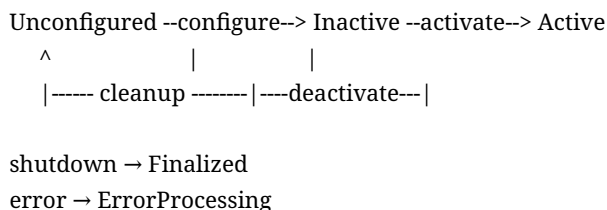
- Ir salīdzinātas vismaz 3 izšķirtspējas.
- Students izskaidro free/occupied/unknown.
- Secinājumā izvēlēta izšķirtspēja konkrētam robotam.

67. ROS 2 Lifecycle Nodes, diagnostics un sistēmas veselība

Autonomam robotam nav pietiekami, ka katrs mezgls “kaut kā darbojas”. Profesionālā sistēmā jāspēj kontrolēti konfigurēt, aktivizēt, apturēt un diagnosticēt programmatūras komponentes. ROS 2 lifecycle un diagnostic_updater dod standartizētus mehānismus šim mērķim.

67.1. Managed node dzīves cikls

ROS 2 managed node vienkāršots state model



Galvenie stāvokļi ir Unconfigured, Inactive, Active un Finalized. Konfigurācijas resurss var tikt sagatavots configure pārejā, bet reālo datu apstrādi un publikāciju sāk tikai Active stāvoklī. Tas ļauj supervisor mezglam nodrošināt pareizu sistēmas starta secību.

67.2. Kāpēc lifecycle ir svarīgs robotā?

- LiDAR mezgls nedrīkst tikt uzskatīts par gatavu, kamēr nav atvērts ports un pārbaudīta datu plūsma.
- SLAM vai localization mezglam nav jāsāk darbs pirms ir korekts TF un sensora topic.
- Nav2 komponentes var tikt aktivizētas tikai pēc kartes/lokālizācijas sagatavošanas.
- Kļūmes gadījumā supervisor var mēģināt deactivate → cleanup → configure → activate, nevis restartēt visu sistēmu.

67.3. Diagnostics

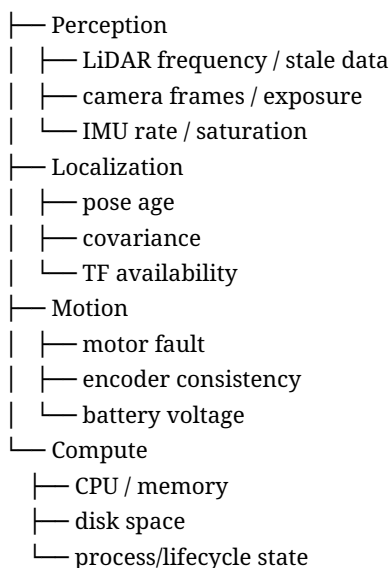
diagnostic_updater ļauj publicēt strukturētu veselības informāciju par sensoriem un mezgliem: statusa līmeni, tekstu un mērījumu vērtības. Labs diagnostikas ieraksts atbild uz trim jautājumiem: kas nav kārtībā, cik nopietni tas ir, un ar kādu mērāmu faktu to pamato.

Komponente	OK nosacījums	WARN piemērs	ERROR piemērs
LiDAR	scan 8–12 Hz	scan < 6 Hz	nav datu > 2 s
IMU	timestamp monotons, rate stabils	rate svārstās	driver disconnected
Motor controller	heartbeat un temperatūra normā	temperatūra tuvu robežai	overcurrent/fault
Localization	covariance zem robežas	covariance pieaug	pose nav atjaunota

67.4. Sistēmas health tree

Functional Tree — robota diagnostika

ROBOT HEALTH



LC1 — Lifecycle + diagnostics scenārijs

Mērķis: Izveidot mezgla/sistēmas starta un atkopšanas loģiku, kur komponentes netiek aktivizētas, kamēr nav izpildīti priekšnosacījumi.

Nepieciešamais aprikojums / programmatūra:

- ROS 2 vide
- Vienkāršs lifecycle demo mezgls vai pasniedzēja simulators
- /diagnostics apskates rīks

Darba secība:

1. Definēt configure, activate, deactivate un cleanup priekšnosacījumus.
2. Izveidot trīs diagnostikas noteikumus: frequency, stale data, value limit.
3. Palaist normālu starta secību.
4. Apzināti pārtraukt vienu sensoru/topic.
5. Reģistrēt diagnostikas pāreju OK→WARN/ERROR.
6. Izpildīt recovery secību un pierādīt atgriešanos Active stāvoklī.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Normāls starts	visi mezgli Active tikai pēc prerequisite		
2	Topic pazūd	diagnostics ERROR		
3	Recovery	mezgls kontrolēti atjaunots		

Analīzes jautājumi:

- Kāpēc process “running” vēl nenozīmē “healthy”?
- Kādus parametrus monitorētu real-time motoru kontrollerim?
- Kad automātisks restart var būt bīstams?

Vērtēšanas / acceptance criteria:

- Ir dokumentēta state-transition secība.
- Ir vismaz 3 diagnostikas metriķi ar robežām.
- Recovery nav tikai “restart computer”.

68. Docker un reproducējama robotikas programmatūras vide

Containerization ļauj definēt programmatūras vidi kā versējamu artefaktu: operētājsistēmas bāze, bibliotēkas, Python pakotnes, ROS 2 atkarības un palaišanas komandas. Tas samazina situācijas “uz mana datora strādā”.

68.1. Image, container, volume, network

Jēdziens	Nozīme robotikas projektā
Image	nemainīgs programmatūras pakotnes šablons ar atkarībām
Container	image palaista instance — izolēts process
Volume	noturīgi dati: bag faili, mapes, konfigurācijas
Network	komunikācijas telpa starp servisiem
Registry	image glabāšanas un izplatīšanas vieta

68.2. Dockerfile piemērs Python/ROS mācību servisam

Vienkāršots ROS 2 Dockerfile piemērs

```
FROM ros:rolling-ros-base
```

```
WORKDIR /ws
```

```
COPY src ./src
```

```
RUN . /opt/ros/rolling/setup.sh && \  
colcon build --symlink-install
```

```
CMD ["bash", "-lc", "source /opt/ros/rolling/setup.bash && source install/setup.bash && ros2 launch my_robot  
bringup.launch.py"]
```

Versiju piezīme: mācību vidē base image jāfiksē atbilstoši konkrētajai instalētajai ROS 2 distribūcijai, nevis akli jāizmanto “rolling”.

68.3. Docker Compose robotikas servisam

Compose mācību piemērs — robot app + datubāze

```
services:
```

```
  robot_app:
```

```
    build: .
```

```
    volumes:
```

```
      - ./config:/ws/config:ro
```

```
      - ./bags:/ws/bags
```

```
    environment:
```

```
      - ROS_DOMAIN_ID=23
```

```
    network_mode: host
```

```
  database:
```

```
    image: postgres:alpine
```

```
    volumes:
```

- dbdata:/var/lib/postgresql/data

volumes:

dbdata:

ROS 2/DDS tīkla discovery uzvedība container vidē ir jātestē konkrētajā OS, DDS implementation un network mode kombinācijā. Mācību mērķis ir saprast robežu starp reproducējamu programmatūras vidi un reālu hardware/network integrāciju.

68.4. Hardware access un drošība

- USB/serial ierīces jānodod containerim apzināti; “privileged” nav labs noklusējuma risinājums.
- Konfigurācijas un sertifikātus nedrīkst iebūvēt publiskā image kā paroles vai privātās atslēgas.
- Bag/map/DB dati jāglabā volume vai ārējā datu glabātnē, nevis container writable layer.
- Image tags un digest palīdz reproducēt konkrētu laboratorijas versiju.

DKR1 — Reproducējams ROS/Python serviss ar Docker Compose

Mērķis: Palaist vienu mācību servisu un datu glabātuvī reproducējamā container konfigurācijā un pierādīt datu noturību pēc container atjaunošanas.

Nepieciešamais aprikojums / programmatūra:

- Docker Engine/Desktop
- Vienkāršs Python vai ROS 2 publisher projekts
- compose.yaml un Dockerfile

Darba secība:

1. Izveidot Dockerfile.
2. Izveidot compose.yaml ar app un datu volume.
3. Build image un fiksēt tag.
4. Palaist servisu.
5. Izveidot testdatus volume.
6. Noņemt un atkārtoti izveidot container; pārbaudīt, ka volume dati saglabājas.
7. Dokumentēt image ID/tag un commit hash.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Build	image izveidots bez manuālas instalācijas containerī		
2	Restart/recreate	serviss atjaunojas no deklarētās konfigurācijas		
3	Volume	dati saglabājas		

Analīzes jautājumi:

- Kāda atšķirība starp image un container?
- Kāpēc datubāzi nevajag glabāt container layer?
- Ko nepieciešams fiksēt, lai eksperimentu var reproducēt pēc 6 mēnešiem?

Vērtēšanas / acceptance criteria:

- Dockerfile un compose faili atrodas Git.
- Nav manuālu “slepeno” soļu pēc build.
- Ir pierādīta datu noturība un versijas identifikācija.

69. Industriālie protokoli: OPC UA, Modbus un CAN/CANopen

Robotikas un mehatronikas risinājumi reti dzīvo izolēti. Industriālā vidē tiem jāapmainās ar datiem ar PLC, piedziņām, sensoriem, SCADA/MES sistēmām un edge datoriem. Šajā nodaļā protokoli tiek mācīti nevis kā “komandu saraksts”, bet kā dažādi datu modeļi un sistēmas integrācijas līmeņi.

69.1. OPC UA — semantisks informācijas modelis

OPC UA nav tikai “reģistru lasīšana”. Servera AddressSpace var modelēt objektus, mainīgos, metodes, eventus un to savstarpējās attiecības. Klients var atklāt datu struktūru, lasīt/rakstīt vērtības un veidot subscriptions. OPC UA specifikācijā būtiska loma ir arī autentifikācijai, sertifikātiem, integritātei un konfidencialitātei.

OPC UA informācijas modeļa mācību piemērs

```
Machine_01
├─ State: RUNNING
├─ SpeedSetpoint: 120.0 rpm
├─ ActualSpeed: 118.6 rpm
├─ Temperature: 42.1 °C
├─ AlarmActive: false
└─ Method: ResetFault()
```

69.2. Modbus — vienkāršs datu un funkciju modelis

Modbus Application Protocol definē funkciju kodus un klasisku datu modeli ar coils, discrete inputs, input registers un holding registers. Tas ir vienkāršs un plaši sastopams, bet semantika “ko nozīmē register 40017” bieži jādefinē atsevišķā integrācijas dokumentācijā.

Objekts	Tipisks tips	Piemērs
Coil	1 bit R/W	MotorStartCommand
Discrete Input	1 bit R	SafetyDoorClosed
Input Register	16 bit R	TemperatureRaw
Holding Register	16 bit R/W	SpeedSetpoint

Mācību uzdevumā vienmēr jānorāda arī scaling, signed/unsigned, endianness/word order, mērvienība un valid range — tieši šeit rodas daudz reālu integrācijas kļūdu.

69.3. CAN un CANopen

CAN ir kopnes datu pārraides pamats ar message identifiier un arbitrāciju. CANopen virs CAN definē standartizētāku ierīču komunikācijas modeli, tostarp Object Dictionary un tipiskus komunikācijas objektus. CANopen datu saites slānis balstās uz ISO 11898-1 un parasti izmanto ISO 11898-2 fizisko slāni.

Aspekts	CAN/CANopen piemērs
Identifier	ziņojuma prioritāte/identifikācija
Object Dictionary	ierīces parametru un datu strukturēts katalogs

Aspekts	CAN/CANopen piemērs
PDO	ātra procesa datu apmaiņa
SDO	parametru/objektu piekļuve
NMT	mezgla vadības stāvokļi

69.4. Protokolu salīdzinājums

Kritērijs	OPC UA	Modbus TCP/RTU	CAN/CANopen
Tipisks līmenis	PLC/SCADA/edge/enterprise	PLC/ierīces, legacy un vienkārša integrācija	device/drive/embedded bus
Datu semantika	bagāta informācijas modeļa pieeja	reģistri/bitī, semantika ārējā dokumentā	Object Dictionary CANopen gadījumā
Drošība	OPC UA definē security model	klasiskajam Modbus nav iebūvētas pilnas security; ir Modbus Security variants	parasti sistēmas/segmentācija s līmenī
Reālā laika raksturs	atkarīgs no profila/transporta	determinisms nav garantēts visos tīklos	CAN prioritātes/arbitrācija device līmenim

69.5. Functional Tree — industriāls robotikas gateway

Functional Tree — industrial gateway

To integrate robot data with industrial systems

- ├─ Acquire robot data
 - | └─ ROS 2 topics/services
 - | └─ controller/fieldbus signals
- ├─ Normalize semantics
 - | └─ engineering units
 - | └─ timestamps/quality
 - | └─ tag/object names
- ├─ Expose industrial interface
 - | └─ OPC UA information model
 - | └─ Modbus register map (legacy/simple)
- ├─ Enforce security boundary
 - | └─ authentication/certificates where applicable
 - | └─ network segmentation
 - | └─ write-command authorization
- └─ Log and diagnose
 - | └─ communication state
 - | └─ stale/bad quality
 - | └─ audit/change evidence

IP1 — Industriāla datu interfeisa specifikācija

Mērķis: No robota sensora/telemetrijas datiem izveidot korektu OPC UA/Modbus/CAN datu līguma piemēru ar mērvienībām, kvalitāti, diapazoniem un testiem.

Nepieciešamais aprikojums / programmatūra:

- Pasniedzēja dota robota datu tabula
- Spreadsheet vai dokumenta veidne
- Protokolu specifikāciju fragmenti

Darba secība:

1. Izvēlēties 10 signālus: stāvoklis, setpoint, actual value, alarm u.c.
2. Katram definēt canonical name, datu tipu, mērvienību, update rate un valid range.
3. Izveidot OPC UA objektu struktūru.
4. Izveidot 5 signālu Modbus register map alternatīvu.
5. Norādīt scaling un byte/word order.
6. Definēt 8 acceptance testus.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Normāla vērtība	pareiza semantika un scaling		
2	Out-of-range	quality/error noteikts		
3	Stale data	identificēts kā nederīgs/novecoji s		
4	Write command	autorizācija un validācija definēta		

Analīzes jautājumi:

- Kāpēc tag “R40017” nav pietiekama dokumentācija?
- Kā atšķiras command no telemetry no drošības viedokļa?
- Kurus datus nedrīkstētu automātiski pārsūtīt no IT/cloud uz safety control?

Vērtēšanas / acceptance criteria:

- Datu līgumā ir tips, mērvienība, diapazons, timestamp/quality.
- Ir definēti read/write ierobežojumi.
- Ir vismaz 8 pārbaudāmi acceptance criteria.

70. RTOS un FreeRTOS robotikā un mehatronikā

Mikrokontrolleru programmēšanā “while(true)” cikls ātri kļūst grūti pārvaldāms, ja vienlaikus jālasa sensori, jāvada motors, jāapstrādā komunikācija un jāuzrauga drošības stāvokļi. RTOS ļauj sadalīt sistēmu uzdevumos ar prioritātēm un skaidriem komunikācijas mehānismiem.

70.1. Task un scheduler

Task ir neatkarīga izpildes plūsma ar savu steku un stāvokli. Scheduler izvēlas, kura task konkrētajā brīdī izpildās. Reālā laika sistēmai būtiska nav tikai “ātra darbība”, bet paredzams atbildes laiks un termiņu ievērošana.

Task	Periods/prioritāte piemērs	Atbildība
MotorControl	1–5 ms, high	PID, PWM, overcurrent state
Encoder/IMU	5–10 ms, high/medium	sensoru paraugu ņemšana
Communication	10–100 ms, medium	CAN/MQTT/serial
Logger/UI	100–1000 ms, low	telemetrija, displejs, debug

70.2. Queue, semaphore un mutex

FreeRTOS queues ir primārs ziņojumu nodošanas mehānisms starp task un starp ISR un task. Semaphore parasti izmanto sinhronizācijai/notikumu signalizēšanai, bet mutex — kopīga resursa savstarpējai izslēgšanai. Mutex ietver priority inheritance mehānismu, kas samazina priority inversion ietekmi.

Mehānisms	Lietojums	Piemērs
Queue	datu nodošana	IMU sample → estimator task
Binary semaphore	notikums/sinhronizācija	ADC DMA pabeigts
Mutex	kopīga resursa aizsardzība	I2C bus vai shared config
Task notification	viegls tiešs task signāls	encoder event counter

70.3. Interrupt Service Routine noteikumi

- ISR jābūt īsai: nolasīt kritisko stāvokli, saglabāt/iesignalizēt notikumu un iziet.
- No ISR jāizmanto tieši ISR drošās API funkcijas, piemēram, FreeRTOS funkcijas ar FromISR sufiksu, ja to paredz API.
- ISR nedrīkst gaidīt mutex vai veikt ilgstošu failu/tīkla operāciju.

70.4. Priority inversion un determinisms

Priority inversion rodas, ja augstas prioritātes task gaida resursu, kuru tur zemas prioritātes task, bet vidējas prioritātes task neļauj zemajai task pabeigt darbu. Priority inheritance var mazināt šo efektu, taču sistēmas arhitektūrā joprojām jāizvairās no gariem kritiskajiem posmiem.

Priority inversion ilustrācija

HIGH task --- waits for mutex ----->
LOW task -- owns mutex --[preempted]-- releases -->

MED task ---- runs for long time ---->

Solution direction: short critical sections + correct mutex + priority design

70.5. Static un dynamic memory

Embedded/reālā laika sistēmās atmiņas stratēģija ir arhitektūras lēmums. FreeRTOS ļauj RTOS objektus izveidot gan dinamiski, gan ar lietotāja nodrošinātu statisku atmiņu. Dinamiska atmiņa ir elastīga, bet kritiskām sistēmām jāvērtē fragmentācija, determinisms un kļūdu apstrāde.

70.6. ESP32/FreeRTOS queue piemērs

FreeRTOS queue mācību kods

```
#include <Arduino.h>

struct Sample {
    uint32_t t_ms;
    float value;
};

QueueHandle_t q;

void sensorTask(void*) {
    Sample s;
    for (;;) {
        s.t_ms = millis();
        s.value = analogRead(34);
        xQueueSend(q, &s, pdMS_TO_TICKS(5));
        vTaskDelay(pdMS_TO_TICKS(10));
    }
}

void loggerTask(void*) {
    Sample s;
    for (;;) {
        if (xQueueReceive(q, &s, portMAX_DELAY) == pdTRUE) {
            Serial.printf("%lu,%.2f\n", s.t_ms, s.value);
        }
    }
}

void setup() {
    Serial.begin(115200);
    q = xQueueCreate(20, sizeof(Sample));
    xTaskCreate(sensorTask, "sensor", 2048, nullptr, 2, nullptr);
    xTaskCreate(loggerTask, "logger", 2048, nullptr, 1, nullptr);
}

void loop() { vTaskDelay(pdMS_TO_TICKS(1000)); }
```

RT1 — Sensoru-vadības-logger task arhitektūra

Mērķis: Sadalīt mikrokontrollera programmu vismaz trīs task un izmērīt, vai prioritātes un queue nodrošina prognozējamu datu plūsmu.

Nepieciešamais aprikojums / programmatūra:

- ESP32 vai FreeRTOS simulatora vide
- Sensors/potenciometrs
- Serial logger

Darba secība:

1. Izveidot sensorTask, controlTask un loggerTask.
2. Definēt periodus un prioritātes.
3. Datus starp sensorTask un controlTask nodot ar queue.
4. Izmērīt sensor sample perioda jitter vismaz 200 paraugiem.
5. Apzināti ievietot 100 ms “smagu” darbu low-priority logger task.
6. Atkārtot testu ar nepareizu prioritāšu konfigurāciju un salīdzināt.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Normāla slodze	sample periods tuvu mērķim		
2	Low-priority heavy load	high-priority control saglabā termiņus		
3	Nepareiza prioritāte	redzams jitter/term deadline risks		

Analīzes jautājumi:

- Kāda atšķirība starp priority un periodu?
- Kad queue ir labāka par global variable?
- Kāpēc Serial.print nevajadzētu likt ļoti kritiskā motor-control task?

Vērtēšanas / acceptance criteria:

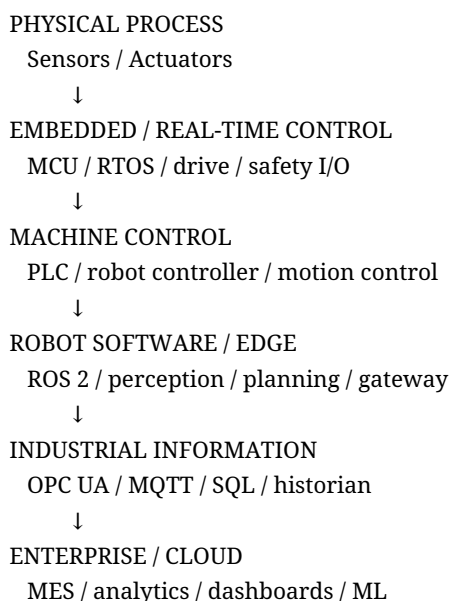
- Ir izmērīts jitter.
- Nav nekontrolētas shared global data piekļuves.
- Students pamato prioritāšu izvēli.

71. Industriālās sistēmas integrācija: no sensora līdz uzņēmuma datiem

Šī nodaļa saliek vienā arhitektūrā visu kursu: sensoru fizika, mikrokontroleris/RTOS, PLC vai motora kontroleris, ROS 2, industriāls datu interfeiss, SQL/time-series glabāšana un mākoņ/uzņēmuma analītika. Galvenais mērķis ir iemācīt robežas starp control, supervisory un information līmeņiem.

71.1. Slāņotā arhitektūra

No fiziska procesa līdz uzņēmuma datiem



Svarīgs profesionālais princips: ne visas komandas drīkst plūst atpakaļ pa šo ķēdi. Safety-critical funkcijas nedrīkst padarīt atkarīgas no cloud, Wi-Fi vai augsta līmeņa lietotnes pieejamības.

71.2. Real-time pret information-time

Līmenis	Tipisks laika mērogs	Kļūmes sekas
Motor current loop	μs–ms	nestabila piedziņa / aparatūras risks
Motion/control	ms	trajektorijas kļūda
Robot planning	10–100+ ms	lēnāka reakcija/ceļa maiņa
Telemetry/SCADA	100 ms–s	novecojis operatora skats
Cloud analytics	s–min–h	analitikas aizkave

71.3. Timestamp, quality un data lineage

Profesionālam datu ierakstam vajag vairāk par “value=42”. Jāzina sensor/source ID, event timestamp, acquisition timestamp, quality, engineering unit, program/config version un, ja dati izmantoti eksperimentā, arī dataset/rosbag ID.

Traceable telemetry ieraksta piemērs

```
{
  "source": "robot01/imu",
  "timestamp_ns": 1787040000123456789,
  "value": {"gz": 0.031},
  "unit": "rad/s",
  "quality": "GOOD",
  "software_commit": "a1b2c3d",
  "calibration_id": "imu-cal-2026-08"
}
```

71.4. Functional Tree — end-to-end datu ķēde

Functional Tree — uzticama datu ķēde

To provide trustworthy robot data for control and analytics

- └─ Measure physical quantity
- └─ Validate and timestamp at source
- └─ Transfer over deterministic/local interface
- └─ Normalize engineering units and semantics
- └─ Publish to robot/edge middleware
- └─ Expose industrial/IT interface
- └─ Persist with metadata and quality
- └─ Visualize/analyze
- └─ Trace every result back to software + calibration + raw data

INT1 — End-to-end sistēmas arhitektūras projekts

Mērķis: Izstrādāt vienas tehniskas sistēmas pilnu datu un vadības arhitektūru ar robežām starp real-time, supervisory un IT/cloud līmeņiem.

Nepieciešamais aprīkojums / programmatūra:

- A3/A2 lapa vai diagrams.net/Visio
- Pasniedzēja dotais scenārijs: mobilais robots, konveijers vai temperatūras stends
- Interfeisa prasību veidne

Darba secība:

1. Definēt sensorus un aktuatorus.
2. Definēt control loop un kur tas izpildās.
3. Definēt ROS/edge funkcijas.
4. Definēt OPC UA/MQTT/SQL datu plūsmu.
5. Atzīmēt safety boundary un write-command robežas.
6. Pievienot timestamp/quality/versioning principu.
7. Izveidot 10 acceptance criteria un 5 failure scenarios.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Cloud unavailable	lokālā drošā vadība turpinās		

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
2	Sensor stale	quality tiek degradēta un control reaģē definēti		
3	Wrong command	range/auth validation noraida		
4	Software update	datu versija ir izsekojama		

Analīzes jautājumi:

- Kur atrodas “source of truth” katram datu tipam?
- Kuras funkcijas nedrīkst pārvietot uz cloud?
- Kā pierādīt, ar kādu algoritma versiju rezultāts iegūts?

Vērtēšanas / acceptance criteria:

- Arhitektūrā ir skaidras robežas.
- Ir definēta datu semantika un kvalitāte.
- Safety un IT/cloud funkcijas nav sajauktas.

72. Uzticamība, observability, watchdog un fault injection

Sistēmas uzticamību nevar novērtēt tikai ar “demo nostrādāja”. Jāmāca apzināti radīt kļūmes, novērot to sekas un pārbaudīt, ka sistēma reaģē paredzami. Observability nozīmē spēju no logiem, metrikiem, diagnostikas un trace datiem saprast sistēmas iekšējo stāvokli.

72.1. Watchdog

Watchdog uzrauga, vai svarīga funkcija joprojām darbojas noteiktā laikā. Embedded watchdog var restartēt MCU, savukārt programmatūras watchdog var pārcelt sistēmu safe state, ja sensora heartbeat vai control update ir pārāk vecs.

Programmatūras watchdog ideja

```
if now - last_lidar_update > 0.5 s:  
    navigation_state = DEGRADED
```

```
if now - last_motor_heartbeat > 0.2 s:  
    command_velocity = 0  
    raise ERROR
```

72.2. Observability slāņi

Slānis	Artefakts	Piemērs
Logs	notikuma konteksts	"LiDAR reconnect attempt 3"
Metrics	skaitlis laikā	scan_hz, CPU, localization covariance
Diagnostics	strukturēts health	OK/WARN/ERROR + values
Traces	cēloņsakarību/latency analīze	sensor callback → planner → command
Dataset	atkārtojama izmeklēšana	rosvbag + config + commit

72.3. Fault injection

Fault injection ir kontrolēta kļūmes ievadišana, lai pārbaudītu recovery. Mācību laboratorijā kļūme var būt sensor topic dropout, mākslīgs 300 ms network delay, nepareizs timestamp, viena riteņa encoder scale kļūda vai IMU bias.

Kļūme	Ko sagaidīt no labas sistēmas
LiDAR dropout	navigation stop/degraded + diagnostics
IMU bias	estimator covariance/residual pasliktinās; nevis klusē
Wrong wheel radius	odom drift atklājas kalibrācijas testā
Database unavailable	control turpinās lokāli; telemetry bufferējas vai kļūda ir redzama
CPU overload	deadline/rate diagnostics parāda degradāciju

OBS1 — Fault injection un recovery pierādījums

Mērķis: Apzināti radīt vismaz četras kļūmes autonomā robota simulācijā vai datu pipeline un pierādīt sagaidīto drošo reakciju.

Nepieciešamais aprikojums / programmatūra:

- ROS 2/simulācijas vai Python pipeline
- diagnostics/log/rosvbag rīki
- Testa protokols

Darba secība:

1. Definēt 4 fault scenārijus un expected safe behavior.
2. Palaist baseline un saglabāt metriku.
3. Ievadīt katru fault pa vienam.
4. Reģistrēt detection latency un recovery laiku.
5. Pārbaudīt, ka kļūme nav “silent”.
6. Saglabāt logs/rosvbag un commit/config ID.

Nr.	Testa gadījums	Expected	Actual	PASS/FAIL
1	Sensor dropout	diagnostics + safe response		
2	Timestamp fault	estimator/data validation atklāj		
3	Network/data service loss	control neatkarīgs, kļūme redzama		
4	CPU load	rate/deadline degradation izmērīta		

Analīzes jautājumi:

- Kāda atšķirība starp fault detection un fault recovery?
- Kad automātiska recovery nav pieļaujama?
- Kāds pierādījums nepieciešams, lai testu varētu atkārtot?

Vērtēšanas / acceptance criteria:

- Visiem fault ir iepriekš definēts Expected.
- Ir mērīts detection/recovery laiks.
- Rezultāts ir reproducējams no saglabāta dataset/config.

73. Paplašinātie universitātes/industrijas projektu virzieni

Šie projekti paredzēti kā nākamais līmenis pēc pamatkursa. Tos var izmantot bakalaura līmeņa ievadprojektam, pētnieciskam darbam vai pasniedzēja pašmācībai. Katram projektam obligāti jābūt prasībām, Functional Tree, arhitektūrai, mērījumu metodikai un reproducējamam testam.

73.1. Projekts A — Visual-Inertial Odometry

- Ievade: kamera + IMU dataset.
- Mērķis: novērtēt trajectory un salīdzināt pret reference/ground truth vai alternatīvu odometriju.
- Pētījuma faktors: timestamp offset, IMU noise/bias, feature count vai camera frame rate.
- Metrikas: ATE/RPE vai vienkāršota pozas kļūda, tracking loss skaits, runtime.

73.2. Projekts B — 3D LiDAR localization un mapping

- Ievade: 3D LiDAR + wheel/IMU odometry.
- Salīdzināt ICP un NDT vai divus parametrizācijas režīmus.
- Izveidot 3D voxel/OctoMap reprezentāciju.
- Metrikas: registration fitness, trajectory drift, runtime, map size.

73.3. Projekts C — Industriāls robotikas gateway

- ROS 2 telemetrija → OPC UA informācijas modelis + SQL/historian.
- Atsevišķi definēt command un telemetry plūsmas.
- Iekļaut quality, timestamp, user authorization/security boundary un audit evidence.
- Testēt stale data, invalid command, restart, database unavailable.

73.4. Projekts D — RTOS mobilā robota kontrolieris

- ESP32/STM32 + FreeRTOS task arhitektūra.
- Encoder/IMU acquisition, motor PID, communication un logger atsevišķās task.
- Mērīt jitter, control update rate, queue usage un deadline misses.
- Pievienot watchdog/fail-safe un fault injection.

73.5. Vienotā projekta pierādījumu pakete

Artefakts	Obligātais saturs
Requirements	ID, requirement, acceptance criterion
Functional Tree	galvenā funkcija → apakšfunkcijas
Architecture	hardware, software, interfaces, data flow
Repository	commit history, README, config
Dataset	raw data/rosbag + metadata
Test protocol	Expected/Actual/PASS-FAIL
Metrics	grafiki un skaitliski KPI
Defense	5–10 min tehniskā demonstrācija + jautājumi

74. v1.1 eksāmenu un aizstāvēšanas jautājumu banka

Šī banka paredzēta diferenciācijai: koledžas līmenī var izmantot konceptuālos un praktiskos jautājumus, savukārt universitātes līmenī pievienot matemātisku pamatojumu, mērījumu nenoteiktību un sistēmas arhitektūras kompromisus.

74.1. Factor graph / VIO jautājumi

- Kas factor graph ir mainīgais un kas ir faktors?
- Kas ir residual un kāpēc tam vajag svaru/covariance?
- Kāpēc nelineāram optimizatoram vajag sākuma novērtējumu?
- Ko dara robust loss funkcija?
- Kas ir loop closure un kāpēc nepareizs loop closure ir bīstams?
- Kas ir Bundle Adjustment?
- Kāpēc kamera viena pati tieši nemēra metrisku 3D attālumu?
- Ko IMU preintegration apkopo starp diviem kameras kadriem?
- Kāpēc VIO jānovērtē gyro un accelerometer bias?
- Kā timestamp offset izpaužas VIO rezultātā?

74.2. 3D LiDAR / mapping jautājumi

- Kādu transformāciju meklē point-cloud registration?
- Apraksti ICP vienu iterāciju.
- Kāpēc ICP var konverģēt uz nepareizu lokālo minimumu?
- Ar ko NDT konceptuāli atšķiras no ICP?
- Kā voxel downsampling ietekmē runtime un detaļu līmeni?
- Ko nozīmē free, occupied un unknown 3D occupancy kartē?
- Kāpēc OctoMap izmanto octree?
- Kā sensor ray palīdz noteikt free space?
- Kādus KPI izmantotu 3D mapping eksperimentā?

74.3. ROS 2 lifecycle / diagnostics / Docker

- Nosauc managed node galvenos dzīves cikla stāvokļus.
- Kāpēc “process darbojas” nav tas pats, kas “sistēma ir healthy”?
- Kādus trīs diagnostikas nosacījumus definētu LiDAR?
- Kad lifecycle recovery ir labāka par pilnu sistēmas restartu?
- Kas ir Docker image un container?
- Kāpēc mapes/bag/DB datus glabā volume?
- Kāpēc reproducējamam eksperimentam jāfiksē image + Git commit + config?
- Kādi ir hardware access riski containerī?

74.4. Industriālie protokoli un RTOS

- Ar ko OPC UA informācijas modelis atšķiras no Modbus register map?
- Nosauc četrus klasiskos Modbus datu objektus.
- Kāpēc Modbus integrācijā jānorāda scaling un word order?

- Ko CANopen Object Dictionary dod vīrs CAN?
- Kāpēc safety command nevajag padarīt atkarīgu no cloud servisa?
- Kas ir RTOS task un ko nosaka priority?
- Atšķir queue, semaphore un mutex.
- Kas ir priority inversion?
- Kāpēc ISR nedrīkst bloķēties uz mutex?
- Kad statiska RTOS objektu atmiņa var būt priekšrocība?

74.5. Aprēķinu / analīzes uzdevumi

Uzdevums	Studentam jāizdara
A	Dots 10 Hz kamera un 200 Hz IMU. Aprēķināt vidējo IMU paraugu skaitu starp kadriem un apspriest jitter.
B	Dots 0.10 m voxel un 20×10×3 m telpas bounding box. Aprēķināt teorētisku pilna grid šūnu skaitu un paskaidrot, kā octree var ietaupīt atmiņu.
C	Dots LiDAR 10 Hz; watchdog timeout 0.35 s. Cik secīgi scan var iztrūkt, pirms watchdog noteikti nostrādā ideālā periodā?
D	Dots Modbus 16-bit register un scaling 0.1 °C/count. Ko nozīmē raw=253? Kā validēt raw ārpus paredzētā diapazona?
E	SensorTask periods 10 ms. 500 paraugu logā mean=10.02 ms, max=17 ms. Aprēķināt max jitter pret nominālu un argumentēt, vai tas pieņemams konkrētam control loop.
F	ICP rezultāts: translation error 0.08 m, yaw error 1.8°. Izvēlēties acceptance criteria un pamatot PASS/FAIL.

74.6. Aizstāvēšanas “dziļie” jautājumi

- Parādi vienu vietu savā sistēmā, kur covariance/uncertainty tieši ietekmē lēmumu.
- Kā pierādīsi, ka tavs algoritma uzlabojums nav tikai viena dataset nejaušība?
- Kāda ir bīstamākā silent failure tavai sistēmai?
- Kuru komponenti tu padarītu lifecycle-managed un kāpēc?
- Ja noņem vienu sensoru, kura sistēmas funkcija degradēsies pirmā?
- Kā tavā projektā atšķiras raw data, derived data un command?
- Kā tu reproducēsi šodienas rezultātu pēc gada?
- Kas tavā arhitektūrā ir real-time un kas nav?

75. Pasniedzēja padziļinātais pašmācības ceļš

Šī sadaļa paredzēta pasniedzējam, kurš vēlas no kursa pakāpeniski izaugt līdz autonomo robotu sistēmu un industriālās robotikas pasniegšanai augstskolā. Nav nepieciešams apgūt visu vienlaikus; katram posmam ir praktisks “pierādījums”, kas parāda, ka tēma tiešām saprasta.

Posms	Tēma	Praktiskais pierādījums
1	Factor graph matemātiskā ideja	sintētiska pose graph optimizācija + residualu analīze
2	Bundle Adjustment	mazs multi-view piemērs un reprojection error
3	IMU preintegration/VIO	timestamp + bias eksperimenti; pēc tam publiskais dataset
4	ICP/NDT	parametru sweep ar reference transformāciju
5	3D occupancy/OctoMap	point cloud → voxel/OctoMap + memory/resolution salīdzinājums
6	ROS lifecycle/diagnostics	lifecycle sensor node + health rules
7	Docker	pilna laboratorijas vide no fresh machine/VM
8	OPC UA/Modbus/CAN	datu līgums + simulēts gateway
9	FreeRTOS	3–4 task robot controller + jitter mērījums
10	Integrācija	ROS 2 → industrial interface → SQL/dashboard
11	Fault injection	rosvbag/log/metrics pierādīta recovery
12	Universitātes projekts	publiski reproducējams repo + dataset + technical report

75.1. Ieteicamais pašmācības princips

Pasniedzēja pašmācības cikls

```
READ / UNDERSTAND
↓
REPRODUCE OFFICIAL EXAMPLE
↓
CHANGE ONE PARAMETER
↓
```

MEASURE EFFECT



BREAK IT ON PURPOSE



EXPLAIN WHY IT FAILED



TEACH IT TO A STUDENT

Svarīgākais posms ir “break it on purpose”. Tikai tad, kad vari paredzēt tipisku kļūmi, to izraisīt un diagnosticēt, tēma ir pietiekami saprasta profesionālai pasniegšanai.

75.2. Mini lekcijas, ko sagatavot pašmācībā

- 15 min: “Kāpēc covariance nav tikai matemātiska formalitāte?”
- 20 min: “No ICP līdz SLAM — kā viens scan kļūst par globālu karti?”
- 20 min: “RTOS un ROS 2 dara atšķirīgas lietas — kur ir robeža?”
- 20 min: “OPC UA pret MQTT pret ROS 2 — ko katrs risina?”
- 30 min: “Kā pierādīt autonoma robota algoritma kvalitāti ar reproducējamu eksperimentu?”

76. [ARHĪVS] Versijas 1.1 statuss un nākamā attīstības karte

Versija 1.1 paplašina vienoto mācību dokumentu no universitātes autonomās robotikas ievada līdz nākamajam līmenim: optimizācijas SLAM, visual-inertial odometry, 3D LiDAR reģistrācija un kartēšana, ROS 2 dzīves cikls/diagnostika, reproducējamās container vides, industriālie protokoli un RTOS.

76.1. Kas v1.1 ir pievienots

- Factor graphs, residuali, nelineārā optimizācija, robust loss un Bundle Adjustment.
- VIO stāvoklis, IMU preintegration, bias, time synchronization un calibration failure modes.
- ICP/NDT 3D LiDAR reģistrācija ar eksperimenta metriku.
- Voxel occupancy, log-odds, octree un OctoMap arhitektūra.
- ROS 2 Lifecycle Nodes, diagnostics un health/recovery dizains.
- Docker/Docker Compose reproducējamai robotikas videi.
- OPC UA, Modbus, CAN/CANopen un industrial gateway datu līgumi.
- RTOS/FreeRTOS task, queue, semaphore, mutex, ISR, priority inversion un timing.
- End-to-end industrial systems integration, traceability un data quality.
- Watchdog, observability un fault injection laboratorija.
- Jauni universitātes/industrijas projektu virzieni un v1.1 eksāmenu banka.

76.2. Ieteicamais nākamais slānis pēc v1.1

Virziens	Ko pievienot v1.2+
Control theory	state-space, LQR, model predictive control, system identification
Manipulators	trajectory generation, dynamics, torque control, MoveIt 2
Advanced perception	deep learning detection/segmentation, depth, 3D detection
Safety	risk assessment, safety PLC concepts, robot safety standards overview
Industrial data	time-series DB, historian, ISA-95 style context, digital twin
Research methods	ATE/RPE, confidence intervals, hypothesis testing, dataset protocol
Deployment	systemd, OTA/update strategy, secure boot concepts, fleet management

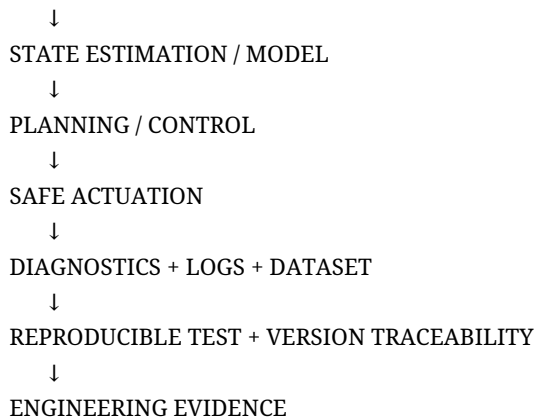
76.3. Galvenā profesionālā doma

v1.1 profesionālās sistēmu domāšanas ķēde

SENSOR / PROCESS

↓

TRUSTWORTHY DATA + TIME + QUALITY



Primārie tehniskie avoti un versiju piezīme — v1.1

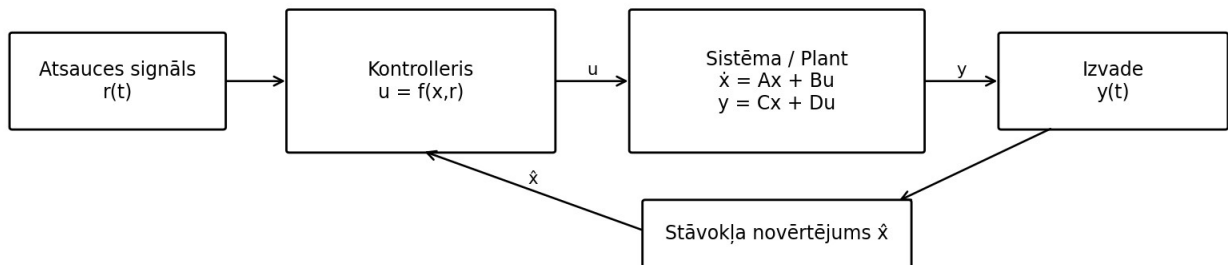
Avoti pārbaudīti 18.08.2026. Programmatūras API, ROS 2 distribūcijas un bibliotēku interfeisi var mainīties; pirms laboratorijas vienmēr jāpārbauda instalētās versijas oficiālā dokumentācija.

- GTSAM — Factor Graphs and GTSAM: <https://gtsam.org/tutorials/intro.html>
- GTSAM — IMU preintegration: <https://gtsam.org/notes/imu-factor/>
- PCL — Iterative Closest Point:
https://pointclouds.org/documentation/tutorials/iterative_closest_point.html
- PCL — Normal Distributions Transform:
https://pointclouds.org/documentation/tutorials/normal_distributions_transform.html
- OctoMap — probabilistic 3D mapping: <https://octomap.github.io/>
- ROS 2 design — Managed nodes/lifecycle: https://design.ros2.org/articles/node_lifecycle.html
- ROS 2 diagnostic_updater: https://docs.ros.org/en/ros2_packages/kilted/api/diagnostic_updater/
- Docker Docs — core concepts and Compose: <https://docs.docker.com/get-started/> un <https://docs.docker.com/compose/>
- OPC Foundation — OPC UA Overview and Security Model:
<https://reference.opcfoundation.org/specs/OPC-10000-1/> un <https://reference.opcfoundation.org/specs/OPC-10000-2/>
- Modbus Organization — specifications: <https://www.modbus.org/modbus-specifications>
- CAN in Automation — CANopen lower layers: <https://www.can-cia.org/can-knowledge/canopen-lower-layers>
- FreeRTOS Documentation: <https://www.freertos.org/Documentation/00-Overview>

77. State-space modeļi un modernās vadības teorijas pamati

Šī nodaļa pāriet no klasiskas “kļūda → PID → aktuators” domāšanas uz sistēmas iekšējā stāvokļa modelēšanu. State-space jeb stāvokļa telpas pieeja ir viens no pamatiem modernajai vadības teorijai, robotikai, lidojuma vadībai, mehatronikai un autonomām sistēmām.

State-space vadības sistēmas pamatstruktūra



77.1. attēls. State-space vadības sistēmas pamatstruktūra.

77.1. Stāvoklis, ievade un izvade

Sistēmas stāvoklis x ir minimāls mainīgo kopums, kas kopā ar zināmu ievadi ļauj prognozēt sistēmas nākotnes dinamiku. Piemēram, mobilam robotam stāvoklis var būt $[x, y, \theta, v, \omega]$, bet DC motoram — leņķiskais ātrums un tinuma strāva.

Nepārtrauktas lineāras sistēmas modelis

$$\begin{aligned}\dot{x} &= A x + B u \\ y &= C x + D u\end{aligned}$$

A matrica apraksta sistēmas iekšējo dinamiku, B — ievades ietekmi, C — to, kā no stāvokļa veidojas mērāmā izvade, bet D — tiešu ievades ietekmi uz izvadi. Daudzās mehāniskās sistēmās $D = 0$.

Uzsvērt, ka matrica nav “matemātika matemātikas dēļ”. Tā ir kompakta sistēmas arhitektūra: kuri stāvokļi ietekmē citus stāvokļus un kuri aktuatori spēj tos mainīt.

77.2. Diskrētais modelis un sampling time

Diskrēts state-space modelis

$$\begin{aligned}x[k+1] &= A_d x[k] + B_d u[k] \\ y[k] &= C_d x[k] + D_d u[k]\end{aligned}$$

Digitāls kontrolieris neveic aprēķinu nepārtraukti. Tas atkārtoti vadības ciklu ar noteiktu sampling time T_s . Tāpēc reālajā embedded/PLC/ROS vadībā modelis bieži tiek diskretizēts. Pārāk liels T_s var zaudēt svarīgu dinamiku; pārāk mazs T_s palielina procesora slodzi un trokšņa ietekmi.

77.3. Linearizācija ap darba punktu

Daudzas robotu un mehatronikas sistēmas ir nelineāras. Lokālā diapazonā tās var linearizēt ap darba punktu x_0, u_0 . Tad A un B iegūst no Jakobiāniem: $A = \partial f / \partial x$, $B = \partial f / \partial u$, novērtējot tos izvēlētajā darba punktā.

77.4. Controllability un observability

Controllability atbild uz jautājumu: vai ar pieejamām ievadēm sistēmu var aizvest uz vajadzīgo stāvokli? Observability atbild: vai no pieejamiem mērījumiem var rekonstruēt iekšējo stāvokli?

Kontrolējamības un novērojamības tests

$$C\_ctrb = [B \ AB \ A^2B \ \dots \ A^{n-1}B]$$
$$O\_obs = [C; CA; CA^2; \dots; CA^{n-1}]$$

Ja $\text{rank}(C\_ctrb)=n \rightarrow$ sistēma ir kontrolējama.

Ja $\text{rank}(O\_obs)=n \rightarrow$ sistēma ir novērojama.

Šie testi palīdz atklāt ļoti praktiskas problēmas. Piemēram, iespējams, ka modelis matemātiski satur sešus stāvokļus, bet esošie sensori sniedz nepietiekamu informāciju, lai visus stāvokļus novērtētu.

77.5. Python piemērs: masas-atsperes-slāpētāja modelis

Python demonstrācija — lineāras dinamiskas sistēmas simulācija

```
import numpy as np
from scipy.signal import StateSpace, lsim
import matplotlib.pyplot as plt
```

```
m = 1.0
```

```
c = 0.8
```

```
k = 4.0
```

```
A = np.array([[0.0, 1.0],
               [-k/m, -c/m]])
```

```
B = np.array([[0.0],
               [1.0/m]])
```

```
C = np.array([[1.0, 0.0]])
```

```
D = np.array([[0.0]])
```

```
sys = StateSpace(A, B, C, D)
```

```
t = np.linspace(0, 10, 1000)
```

```
u = np.ones_like(t)
```

```
tout, y, x = lsim(sys, U=u, T=t)
```

```
plt.plot(tout, y)
```

```
plt.xlabel('t, s')
```

```
plt.ylabel('position')
```

```
plt.grid(True)
```

```
plt.show()
```

SS1 — State-space modeļa izveide un validācija

Mērķis: No fizikāla apraksta iegūt state-space modeli, simulēt tā reakciju un salīdzināt ar sagaidāmo dinamiku.

Ko stāstīt auditorijai: Studentam jāsaprot, ka labs modelis nav “perfekta realitātes kopija”; tas ir mērķim pietiekams realitātes apraksts ar skaidriem pieņēmumiem.

Darba secība

1. Izvēlies masas–atsperes vai DC motora piemēru un nosaki stāvokļus.
2. Atvasini A, B, C un D matricas un pieraksti vienības.
3. Izveido Python simulāciju ar soli vai citu testa ievadi.
4. Maini vienu fizikālo parametru un novēro dinamiskās atbildes izmaiņu.
5. Pārbaudi controllability un observability rangū.
6. Dokumentē secinājumu: ko modelis apraksta labi un ko tas neņem vērā.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Nulles ievade	Stāvoklis attīstās atbilstoši sākuma stāvoklim			
Step input	Sistēma sasniedz prognozējamu pārejas procesu			
Parametra izmaiņa	Mainās atbilstošs dinamiskais raksturlielums			
Rank tests	Rangs interpretēts pret stāvokļu skaitu			

Paplašinājums

- A: izveidot modeli ar 2 stāvokļiem.
- B: pievienot traucējumu/disturbance modeli.
- C: salīdzināt lineāru modeli ar nelineāru simulāciju.

78. System Identification — modeļa iegūšana no mērījumiem

System Identification jeb sistēmas identifikācija ir process, kurā dinamiskās sistēmas modelis tiek iegūts no reāliem ievades un izvades mērījumiem. Tas ir īpaši vērtīgi, ja precīzs fiziskais modelis nav zināms, ir pārāk sarežģīts vai tā parametri mainās.

78.1. Kāpēc nepietiek ar “uzzīmēt step response”

Vienkāršs step tests var dot laika konstanti, pastiprinājumu un aizturi, tomēr kvalitatīvai identifikācijai nepieciešams pietiekami ierosināt sistēmas dinamiku, fiksēt sampling time, ievērot drošas amplitūdas robežas, atdalīt treniņa un validācijas datus un analizēt atlikumu/residual struktūru.

78.2. Eksperimenta projektēšana

- Ievadei jāierosina interesējošais frekvenču diapazons.
- Mērījumu signālam jābūt sinhronam un ar zināmu sampling time.
- Eksperimentam jāpaliek sistēmas drošajā darbības diapazonā.
- Jāfiksē sākuma stāvoklis, slodze, barošana, temperatūra un citi apstākļi.
- Identifikācijas un validācijas datiem jābūt atsevišķiem.

78.3. Vienkāršs ARX modelis

ARX piemēra forma

$$y[k] = -a_1 y[k-1] - a_2 y[k-2] + b_1 u[k-1] + b_2 u[k-2] + e[k]$$

Koeficientus var noteikt ar lineāru least-squares pieeju. Tas ļauj studentam sasaistīt lineāro algebru ar reālu dinamisku mērījumu datiem.

Least-squares parametru novērtēšanas kods

```
# theta = argmin ||Phi theta - y||^2
import numpy as np

theta, *_ = np.linalg.lstsq(Phi, y_target, rcond=None)
y_hat = Phi @ theta
rmse = np.sqrt(np.mean((y_target - y_hat)**2))
print(theta, rmse)
```

78.4. Modeļa kvalitāte

Rādītājs	Ko tas parāda	Brīdinājums
RMSE / MAE	Vidējo prognozes kļūdu	Mazs skaitlis pats par sevi neapliecina labu dinamisko modeli
Residuals	Kas paliek neizskaidrots	Strukturēti residuals var norādīt uz trūkstošu dinamiku
Validation fit	Generalizāciju uz citiem datiem	Vienmēr vērtēt ar neatkarīgu datu kopu

Rādītājs	Ko tas parāda	Brīdinājums
Parameter stability	Vai parametri saglabājas dažādos eksperimentos	Liela svārstības var norādīt uz nelinearitāti vai sliktu excitation

ID1 — DC motora / 2WD robota dinamiskā modeļa identifikācija

Mērķis: No reģistrētiem PWM un ātruma datiem iegūt vienkāršu dinamisku modeli un neatkarīgi to validēt.

Ko stāstīt auditorijai: Galvenais mērķis ir nevis “dabūt līkni”, bet iemācīties pierādīt, ka modelis ir validēts ārpus datiem, ar kuriem tas tika iegūts.

Darba secība

1. Reģistrē PWM komandu, laiku un enkodera ātrumu ar fiksētu sampling time.
2. Izmanto vismaz divus atšķirīgus eksperimentus: identifikācijai un validācijai.
3. Izveido pirmās vai otrās kārtas/ARX modeli.
4. Aprēķini prognozēto ātrumu un RMSE.
5. Uzzīmē measured vs predicted grafiku.
6. Pieraksti darba diapazonu, kurā modelis ir ticams.

Testēšanas protokols

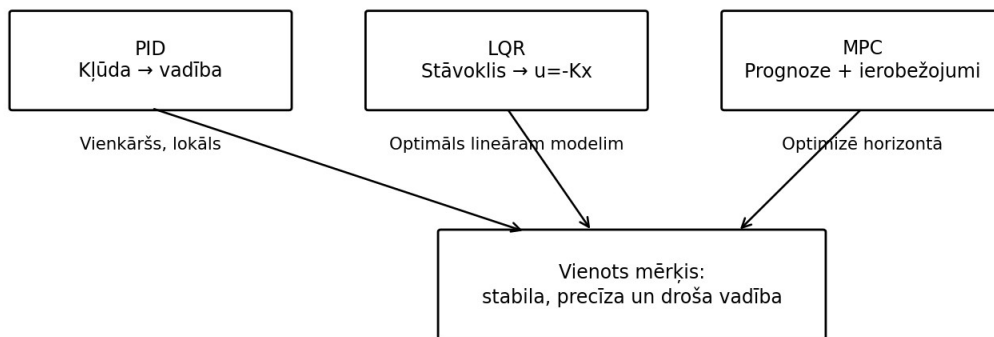
Test case	Expected	Actual	PASS/FAIL	Komentārs
Training data	Modelis pielāgots bez datu sajaukšanas			
Validation data	Aprēķināta neatkarīga kļūda			
Reverse direction	Novērtēta asimetrija vai paskaidrots, kāpēc nav testēts			
Different battery level	Novērtēta parametru jutība			

Paplašinājums

- A: step-response first-order modelis.
- B: ARX least squares.
- C: salīdzināt divus modeļu tipus un argumentēt izvēli.

79. LQR — optimālā stāvokļa atgriezeniskā saite

PID, LQR un MPC konceptuāls salīdzinājums



79.1. attēls. PID, LQR un MPC konceptuāls salīdzinājums.

Linear Quadratic Regulator (LQR) meklē stāvokļa atgriezeniskās saites matricu K , kas lineārai sistēmai līdzsvaro divus mērķus: mazu stāvokļa kļūdu un saprātīgu vadības piepūli.

79.1. Izmaksu funkcija

LQR optimizācijas ideja

$$J = \int (x^T Q x + u^T R u) dt$$

$$u = -Kx$$

Q nosaka, kuri stāvokļi ir dārgi jeb svarīgi, bet R — cik dārga ir liela vadības komanda. Tādēļ LQR tuning nav “magisku koeficientu meklēšana”; tas ir prasību prioritāšu formulējums matricās.

79.2. Riccati vienādojums

Continuous-time Algebraic Riccati Equation

$$A^T P + P A - P B R^{-1} B^T P + Q = 0$$

$$K = R^{-1} B^T P$$

Praksē Riccati vienādojumu risina uzticama numeriska bibliotēka. Studentam nav obligāti ar roku risināt lielu matricu vienādojumu, bet ir jāspēj izskaidrot, kā Q un R maina vadības uzvedību.

79.3. Reference tracking

Vienkāršs $u = -Kx$ stabilizē stāvokli ap nulles darba punktu. References sekošanai nepieciešams stāvokli definēt kā kļūdu pret mērķi, pievienot feedforward/prefilter vai paplašināt sistēmu ar integratoru, lai novērstu statisko kļūdu.

79.4. Python piemērs

LQR pastiprinājuma aprēķins Python

```
import numpy as np
from scipy.linalg import solve_continuous_are
```

```

A = np.array([[0., 1.],
              [0., 0.]])
B = np.array([[0.],
              [1.]])
Q = np.diag([20., 2.])
R = np.array([[0.5]])

P = solve_continuous_are(A, B, Q, R)
K = np.linalg.inv(R) @ B.T @ P
print('K =', K)

```

LQR1 — Pozīcijas/ātruma LQR regulators

Mērķis: Izveidot lineāru modeli, aprēķināt LQR regulatoru un eksperimentāli izpētīt Q/R ietekmi.

Ko stāstīt auditorijai: Svarīgi salīdzināt LQR ar PID nevis kā “labāks/sliktāks”, bet kā citu projektēšanas paradigmu ar modeli un vairākiem stāvokļiem.

Darba secība

1. Definē dubultintegratora vai cita vienkārša plant A/B modeli.
2. Pārbaudi controllability.
3. Izvēlies sākotnējo Q un R un aprēķini K.
4. Simulē closed-loop reakciju no vairākiem sākuma stāvokļiem.
5. Palielini pozīcijas svaru Q un salīdzini vadības piepūli.
6. Palielini R un analizē kompromisu starp agresivitāti un enerģiju.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Stability	Stāvokļi konverģē uz darba punktu			
Q position ↑	Pozīcijas kļūda tiek sodīta vairāk			
R ↑	Vadības komanda kļūst konservatīvāka			
Actuator limit	Atzīmēts, ka klasiskā LQR formulā limits nav tieši iekļauts			

Paplašinājums

- A: 2-state plant.
- B: reference tracking ar integratoru.
- C: LQR + Kalman estimator jeb LQG koncepta simulācija.

80. MPC — Model Predictive Control

Model Predictive Control (MPC) katrā vadības ciklā izmanto sistēmas modeli, lai prognozētu nākotni ierobežotā horizontā, atrastu optimālu vadības komandu secību un izpildītu tikai pirmo komandu. Nākamajā ciklā optimizācija tiek atkārtota ar jauniem mērījumiem — receding horizon princips.

80.1. Optimizācijas formulējums

Tipiska lineāra MPC problēma

$$\min_{\mathbf{u}} \sum_{k=0}^{N-1} [||\mathbf{x}_k - \mathbf{x}_{ref}||^2_Q + ||\mathbf{u}_k||^2_R] + ||\mathbf{x}_N - \mathbf{x}_{ref}||^2_P$$

subject to:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$

$$\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}$$

$$\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}$$

MPC lielākā didaktiskā vērtība ir tā, ka tas skaidri parāda prasību, modeļa un ierobežojumu saikni. Atšķirībā no vienkārša LQR var tieši iekļaut aktuatora, ātruma, temperatūras vai drošības ierobežojumus.

80.2. Prediction horizon un control horizon

Garāks prediction horizon ļauj redzēt tālāk nākotnē, taču palielina aprēķinu slodzi. Īss horizonts var būt pārāk tuvredzīgs. Reālā laika sistēmā optimizācijas sliktākais izpildes laiks ir tikpat svarīgs kā matemātiskais optimums.

80.3. Soft constraints

Dažos uzdevumos labāk ir atļaut īslaicīgu ierobežojuma pārkāpumu ar lielu sodu, nevis padarīt optimizācijas problēmu neizpildāmu. Tad ievieš slack mainīgos jeb soft constraints. Safety-critical robežām to drīkst izmantot tikai ar skaidru sistēmas drošības arhitektūru.

80.4. Python mācību piemērs — 1D robots

Receding horizon algorithms

```
# Konceptuāls MPC cikls
for k in range(steps):
    x0 = estimate_state()
    U = solve_optimization(x0, x_ref, constraints, horizon=N)
    apply_control(U[0])
    wait(Ts)
```

MPC1 — Ierobežota 1D/2D kustības vadība

Mērķis: Salīdzināt LQR un MPC, īpaši situācijā ar stingru aktuatora vai ātruma ierobežojumu.

Ko stāstīt auditorijai: MPC jāpasniedz kā tiešsaistes optimizācijas sistēma, kur algoritma izpildes laiks ir daļa no vadības prasībām.

Darba secība

1. Izmanto to pašu lineāro modeli kā LQR laboratorijā.
2. Definē $u_{\max}$ un, ja vajag, ātruma limitu.
3. Izveido finite-horizon optimizāciju ar izvēlētu N .
4. Simulē references maiņu un disturbance.
5. Salīdzini trajectory, $\max |u|$, settling time un aprēķina laiku ar LQR.
6. Pieraksti, vai optimizācija katrā solī pabeidzās sampling perioda laikā.

Testēšanas protokols

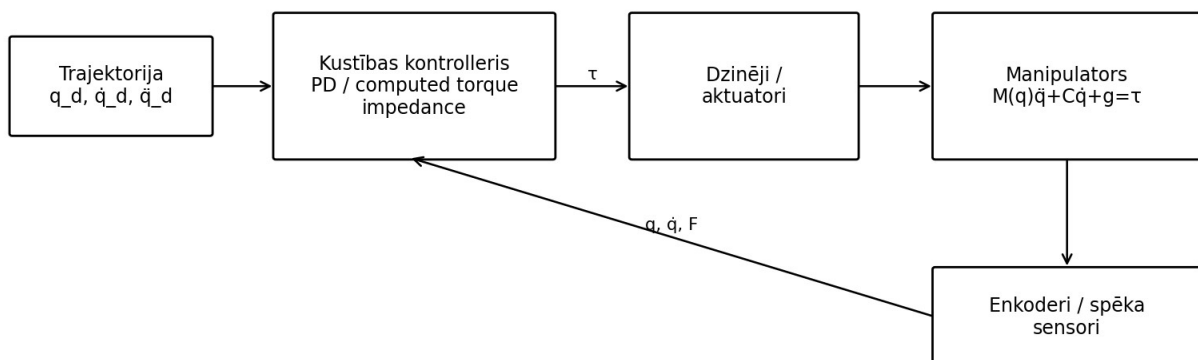
Test case	Expected	Actual	PASS/FAIL	Komentārs
Input constraint	$ u $ nepārsniedz noteikto limitu			
Reference step	Mērķis sasniegts bez constraint violation			
Disturbance	Kontrolleris pārplāno nākotnes komandas			
Runtime	Aprēķina laiks salīdzināts ar T_s			

Paplašinājums

- A: tikai input constraints.
- B: state + input constraints.
- C: soft constraint un runtime profiling.

81. Manipulatoru dinamika un torque control

Manipulatora slēgtās cilpas torque control arhitektūra



81.1. attēls. Manipulatora slēgtās cilpas torque control arhitektūra.

Kinematika atbild uz jautājumu “kur robots atrodas?”, bet dinamika — “kādi spēki un griezes momenti nepieciešami, lai robots kustētos?”. Manipulatoru dinamikā jāņem vērā inerces, Koriolisa/centrbēdzes efekti, gravitācija, berze un ārējie spēki.

81.1. Standarta manipulatora dinamiskais vienādojums

Manipulatora locītavu telpas dinamika

$$M(q) \ddot{q} + C(q, \dot{q}) \dot{q} + g(q) + f(\dot{q}) = \tau + J(q)^T F_{\text{ext}}$$

Termins	Nozīme
$M(q)$	Inerces matrica; sasaista locītavu paātrinājumu ar nepieciešamo torque
$C(q, \dot{q})\dot{q}$	Koriolisa un centrbēdzes efekti
$g(q)$	Gravitācijas kompensācijas torque
$f(\dot{q})$	Berze un citi zudumi
τ	Aktuatoru locītavu torque
$J^T F_{\text{ext}}$	Ārēja gala efektorā spēka pārnese uz locītavu torque

81.2. Lagrange pieeja

Viens sistemātisks veids dinamikas iegūšanai ir Lagrange metode: $L = T - V$, kur T ir kinētiskā enerģija un V potenciālā enerģija. Katram vispārinātajam koordinātu elementam q_i izmanto $d/dt(\partial L/\partial \dot{q}_i) - \partial L/\partial q_i = \tau_i$.

81.3. Gravity compensation

Ja robots kustas lēni vai statiski tur pozu, būtiska komandas daļa var būt gravitācijas kompensācija $\tau = g(q)$. Tas ir labs starpposms starp pozīcijas vadību un pilnu model-based torque control.

81.4. Computed torque control

Computed torque / feedback linearization ideja

$$\begin{aligned}e &= \mathbf{q}_d - \mathbf{q} \\ \dot{e} &= \dot{\mathbf{q}}_d - \dot{\mathbf{q}} \\ \mathbf{v} &= \ddot{\mathbf{q}}_d + \mathbf{K}_d \dot{e} + \mathbf{K}_p e \\ \boldsymbol{\tau} &= \mathbf{M}(\mathbf{q})\mathbf{v} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q})\end{aligned}$$

Ja modelis ir pietiekami precīzs, nelineārā dinamika tiek kompensēta un kļūdas dinamika kļūst līdzīga lineārai otrās kārtas sistēmai. Neprecīzs masas, inerces vai berzes modelis rada model mismatch, tāpēc reālā sistēmā saglabājas atgriezeniskās saites nozīme.

81.5. Jacobian transpose un spēks

Kartēšana no Cartesian spēka uz locītavu torque

$$\boldsymbol{\tau}_{\text{ext}} = \mathbf{J}(\mathbf{q})^T \mathbf{F}_{\text{ext}}$$

Šī sakarība savieno kinemātikas Jacobian ar dinamiku un force/impedance control. Tā ir centrāla, ja robots kontaktējas ar vidi, spiež, slīpē, montē vai sadarbojas ar cilvēku.

DYN1 — 2DOF manipulatora dinamikas simulācija

Mērķis: Izveidot vienkāršotu 2DOF dinamisko modeli un salīdzināt PD vadību ar gravity compensation / computed torque.

Ko stāstīt auditorijai: Mācību mērķis nav iegūt rūpnieciska robota absolūti precīzu modeli, bet saprast, no kurienes torque komandā rodas atsevišķie fizikālie komponenti.

Darba secība

1. Definē linku garumus, masas, masas centru un inerces.
2. Implementē $\mathbf{M}(\mathbf{q})$, $\mathbf{g}(\mathbf{q})$ un, ja iespējams, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$.
3. Simulē tikai PD vadību vairākās pozās.
4. Pievieno gravity compensation un salīdzini statisko kļūdu.
5. Pievieno desired trajectory un computed torque struktūru.
6. Pieraksti model mismatch testu, mainot vienu masas parametru.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Hold pose	Pozīcija noturēta, kļūda izmērīta			
Gravity compensation	Samazinās gravitācijas izraisīta kļūda			
Trajectory	Sekojava kļūda aprēķināta			
Wrong mass	Model mismatch ietekme dokumentēta			

Paplašinājums

- A: tikai gravity term.
- B: $M + g$.
- C: pilns $M/C/g$ un external-force perturbation.

82. Trajektoriju ģenerēšana, impedance un force control

82.1. Path un trajectory

Path ir ģeometrisks ceļš konfigurāciju vai Cartesian telpā. Trajectory pievieno laiku un līdz ar to arī ātrumu un paātrinājumu. Reālam robotam kustības izpildei nepieciešama trajectory, jo aktuatoriem ir ātruma, paātrinājuma, jerk un torque ierobežojumi.

82.2. Cubic un quintic polinomi

Locītavu trajektorijas bāzes formas

$$q(t) = a_0 + a_1t + a_2t^2 + a_3t^3 \quad \# \text{ cubic}$$
$$q(t) = a_0 + a_1t + a_2t^2 + a_3t^3 + a_4t^4 + a_5t^5 \quad \# \text{ quintic}$$

Cubic polinoms ļauj noteikt sākuma/beigu pozīciju un ātrumu. Quintic ļauj papildus noteikt arī paātrinājumu, tādēļ bieži dod gludāku kustības profilu.

82.3. Trapezoidal un S-curve profili

Trapezoidal velocity profile sadala kustību acceleration–constant velocity–deceleration posmos. S-curve papildus ierobežo jerk, kas mehāniski samazina triecienus un vibrāciju.

82.4. Impedance control

Vēlamā mehāniskā impedance

$$M_d \ddot{e} + D_d \dot{e} + K_d e = F_{\text{ext}}$$

Impedance control nekontrolē tikai pozīciju vai tikai spēku. Tas nosaka vēlamu attiecību starp kustības kļūdu un ārējo spēku — robots programmatiski iegūst “atsperes–slāpētāja” uzvedību. Tas ir būtiski drošā kontaktā, collaborative robotics un montāžas uzdevumos.

82.5. Force control un hibrīdā vadība

Dažos uzdevumos viena Cartesian ass jākontrolē pēc spēka, bet citas pēc pozīcijas. Hibrīdā position/force control sadala uzdevuma telpu atbilstošos virzienos. Studentam jāspēj paskaidrot, kāpēc “turēt Z spēku 20 N” un “turēt Z pozīciju 0.5 m” vienlaikus var būt savstarpēji konfliktējošas prasības.

TC1 — Trajektorija + computed torque / impedance simulācija

Mērķis: Ģenerēt gludu joint trajectory un analizēt tās sekošanu ar dažādiem vadības režīmiem.

Ko stāstīt auditorijai: Uzsvērt, ka trajektorijas projektēšana un controlleris ir divas atšķirīgas problēmas: pirmais nosaka vēlamu kustību laikā, otrais cenšas to fiziski realizēt.

Darba secība

1. Ģenerē cubic vai quintic trajectory no q_0 līdz q_f .
2. Aprēķini $\dot{q}_d$, $\ddot{q}_d$ un $\dddot{q}_d$ visā kustībā.
3. Simulē PD vai computed torque sekošanu.
4. Izmēri max error, RMS error, max torque un settling pēc kustības.
5. Pievieno ārēju islaicīgu disturbance.

6. Paplašinājumā implementē vienkāršu impedance response.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Boundary conditions	$q/\dot{q}/\ddot{q}$ atbilst sākuma un beigu nosacījumiem			
Tracking	Kļūda aprēķināta un attēlota			
Torque limits	Maksimālā komanda salīdzināta ar limitu			
Disturbance	Recovery izmērīts			

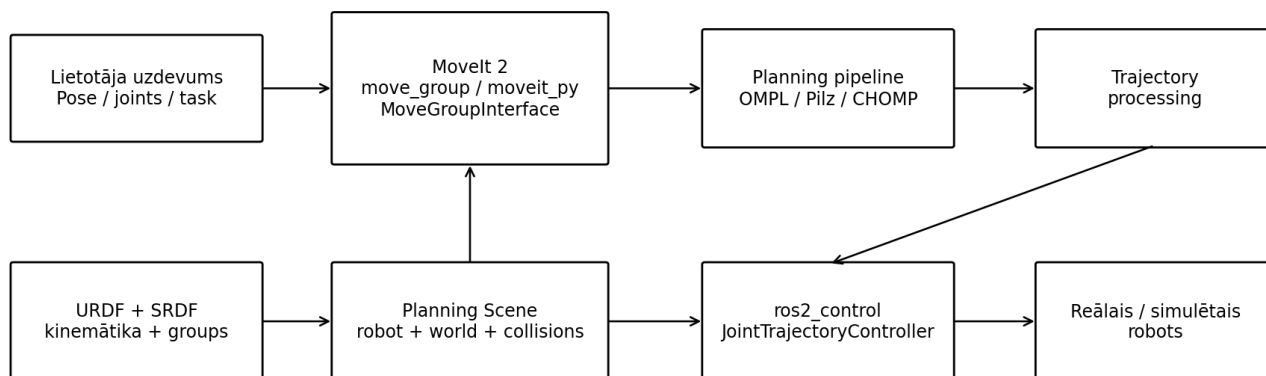
Paplašinājums

- A: cubic + PD.
- B: quintic + computed torque.
- C: Cartesian impedance / force perturbation.

83. MoveIt 2 — manipulatoru kustības plānošana un izpilde

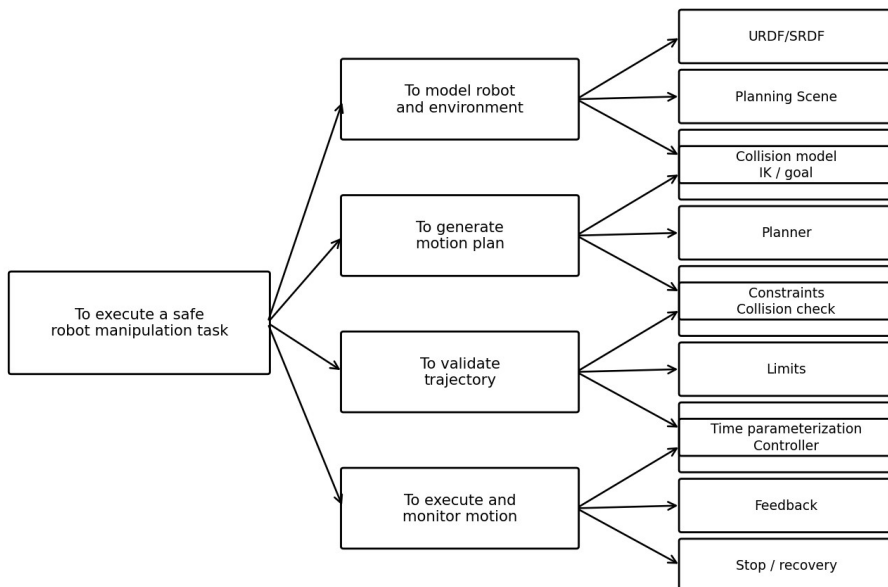
MoveIt 2 ir ROS 2 kustības plānošanas ietvars manipulatoriem. Mācību kursā to izmanto kā augstāka līmeņa slāni, kas sasaista robotu modeli, kinemātiku, collision checking, planning scene, kustības plānotājus, trajectory processing un kontrolierus.

MoveIt 2 kustības plānošanas un izpildes arhitektūra



83.1. attēls. MoveIt 2 kustības plānošanas un izpildes arhitektūra.

Functional Tree — drošs manipulatora kustības uzdevums



83.2. attēls. Functional Tree — drošs manipulatora kustības uzdevums.

83.1. URDF un SRDF loma

URDF apraksta robota linkus, joints, ģeometriju un kinemātisko struktūru. SRDF papildina semantisko informāciju, piemēram, planning groups, pasīvos joints, end effectors un noteiktas grupu konfigurācijas. MoveIt plāno konkrētai planning group, nevis abstrakti “visam robotam”.

83.2. move_group un Planning Scene

MoveIt arhitektūrā move_group integrē kustības plānošanas komponentes un izmanto robota modeli, joint state/TF informāciju, Planning Scene un kontrolieru interfeisus. Planning Scene satur robota pašreizējo stāvokli un pasaules ģeometriju, kas nepieciešama collision checking un constraint novērtēšanai.

83.3. Planning pipelines un planners

MoveIt atbalsta plānotāju plugin arhitektūru. Pašreizējā MoveIt dokumentācija kā tipiskus planning pipelines rāda OMPL, Pilz industrial motion planner un CHOMP. Plānotāja izvēle ir atkarīga no uzdevuma: brīvas telpas meklēšana, industriālas determinētākas kustības, optimizācija vai speciāli constraints.

83.4. Trajectory execution ar ros2_control

Plānotāja rezultāts nav tikai ģeometrisks ceļš; MoveIt ģenerē kustības trajektoriju. Izpilde tipiski tiek nodota kontrolierim ar FollowJointTrajectory action interfeisu. ros2_control JointTrajectoryController izpilda laika ziņā parametrizētu locītavu trajektoriju, izmantojot pieejamos hardware command/state interfaces.

83.5. MoveGroupInterface un moveit_py

C++ pusē MoveGroupInterface nodrošina augsta līmeņa API pozas/joint mērķiem, plānošanai, kustībai un collision objektu pārvaldībai. Python pusē MoveIt dokumentācijā pieejams moveit_py motion planning API, kas ļauj mācību projektos veikt plānošanu, planning scene operācijas un izmantot vairākas planning pipelines.

83.6. Collision objects un Planning Scene Interface

Objekta pievienošana planning scene — struktūras piemērs

```
// Konceptuāls C++ fragments
moveit::planning_interface::PlanningSceneInterface psi;
moveit_msgs::msg::CollisionObject object;
object.id = "workbench";
// ... geometry + pose ...
psi.applyCollisionObject(object);
```

Svarīgi, ka šķērslis nav tikai RViz vizualizācija. Tam jābūt Planning Scene daļai, lai collision checker varētu to izmantot kustības plānošanā.

83.7. Pick-and-place un MoveIt Task Constructor

Sarežģītu manipulatora uzdevumu ir lietderīgi sadalīt posmos. MoveIt Task Constructor (MTC) strukturē uzdevumu stages, piemēram: current state → open gripper → connect/approach → grasp → attach object → lift → move → place → detach → retreat. Šī pieeja ļoti labi sasaucas ar kursa Functional Tree un dekompozīcijas principiem.

83.8. Kustības uzdevuma validācija

Pārbaudes slānis	Jautājums
Robot model	Vai joint limits, frames un collision geometry ir korekti?
Planning Scene	Vai visi būtiskie šķēršļi ir modelēti?
IK/goal	Vai mērķis ir sasniedzams un nav singularitātes/constraint problēmas?
Plan	Vai plāns ir collision-free un kinemātiski derīgs?
Trajectory	Vai ātrums/paātrinājums atbilst limitam?
Controller	Vai FollowJointTrajectory interfeiss un joints sakrīt?
Execution	Vai faktiskā kļūda, stop un recovery ir uzraudzīti?

MV1 — MoveIt 2: pose goal + collision object + execution

Mērķis: Izveidot reproducējamu manipulatora plānošanas eksperimentu ar vienu collision objektu un mērījamiem acceptance criteria.

Ko stāstīt auditorijai: MoveIt nav "autopilots". Students joprojām ir atbildīgs par pareizu robot model, collision model, controllers, constraints un validācijas pierādījumu.

Darba secība

1. Startē robotu MoveIt demo/simulācijas konfigurācijā un pārbaudi joint_states + TF.
2. Izvēlies planning group un nosaki sasniedzamu pose goal.
3. Izplāno kustību bez šķēršļa un saglabā planning time/path rezultātu.
4. Pievieno collision objektu Planning Scene.
5. Atkārti plānošanu un pārbaudi, ka trajektorija apiet objektu vai korekti ziņo par neizpildāmu uzdevumu.
6. Izpildi plānu simulācijā un reģistrē faktiskos joint states.
7. Izveido testa protokolu: plan success, collision status, execution result, max tracking error.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Baseline plan	Plāns izveidots bez collision objekta			
Collision object	Objekts redzams Planning Scene un ietekmē plānošanu			
Trajectory limits	Nav acīmredzama joint limit violation			
Execution	Controller pabeidz action ar definētu rezultātu			

Paplašinājums

- A: tikai pose goal.
- B: pose goal + collision object.
- C: vairāki planners / constraints un rezultātu salīdzinājums.

MTC1 — Pick-and-place dekompozīcija ar MoveIt Task Constructor

Mērķis: Sadalīt manipulatora uzdevumu stages un sasaistīt katru stage ar ieejas nosacījumu, rezultātu un kļūmes gadījumu.

Ko stāstīt auditorijai: MTC ir ļoti labs piemērs, kā Functional Tree no sistēmu inženierijas pāriet izpildāmā robotikas uzdevuma arhitektūrā.

Darba secība

1. Izveido functional tree pick-and-place uzdevumam.
2. Definē current state, open hand, approach/grasp, attach, lift, transport, place, detach un retreat stage loģiku.
3. Katram stage pieraksti precondition un acceptance criterion.
4. Izmanto planning scene objektu kā paņemamo detaļu.
5. Plāno vismaz vienu veiksmīgu pilnu task solution.
6. Izraisīt vienu kontrolētu kļūmi (nesasniedzama grasp pose vai collision) un dokumentēt diagnostiku.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Task init	Robot model un stage properties ielādējas			
Pick	Objekts tiek sasniegts un attach logika ir konsekventa			
Place	Objekts tiek novietots un detached			
Fault case	Kļūme ir reproducējama un izskaidrota			

Paplašinājums

- A: task arhitektūra bez execution.
- B: pilna simulēta pick-and-place.
- C: vairāku grasp candidate/cost salīdzinājums.

84. v1.2 praktiskās laboratorijas un integrētais projekts

84.1. Laboratoriju progresija

Kods	Laboratorija	Galvenais pierādījums
SS1	State-space modelis	Modelis + simulācija + controllability/observability
ID1	System Identification	Training/validation dati + kļūdas metrika
LQR1	LQR	Q/R tuning + closed-loop reakcija
MPC1	MPC	Constraints + runtime + salīdzinājums ar LQR
DYN1	Manipulatora dinamika	M/C/g modelis + model mismatch
TC1	Trajectory/torque/impedance	Trajectory boundary + tracking error
MV1	MoveIt 2 planning	Planning Scene + collision-aware plan + execution
MTC1	Task Constructor	Stage dekompozīcija + pick/place + fault case

84.2. Integrētais projekts — 2DOF/6DOF manipulatora digitālais vadības dvīnis

Projekta mērķis: izveidot vienotu modelēšanas un robotikas demonstratoru, kurā students no sistēmas prasībām nonāk līdz state-space/dinamiskam modelim, trajektorijai, controllerim un MoveIt 2 kustības plānam.

Minimālās prasības

- URDF/Xacro vai cits dokumentēts manipulatora modelis ar frames un joint limits.
- Forward/inverse kinematics pārbaude.
- Vismaz vienas ass vai reducēta state-space dinamiskā modeļa validācija.
- Trajectory generation ar position/velocity/acceleration robežnosacījumiem.
- Vienas advanced control metodes demonstrācija: LQR, MPC vai computed torque.
- MoveIt 2 Planning Scene ar vismaz vienu collision objektu.
- Plan + execute simulācijā ar mērītu tracking/error rezultātu.
- Traceability: prasība → modelis/funkcija → kods/config → tests → pierādījums.

Acceptance criteria piemērs

ID	Prasība	Kritērijs	Pierādījums
AC-01	Kinematika	FK/IK gala pozas kļūda zem definētas robežas	Python tests / RViz

ID	Prasība	Kritērijs	Pierādījums
AC-02	Trajectory	Nav pārsniegtu definēto joint velocity limit	Trajectory log
AC-03	Control	RMS tracking error zem projekta noteiktās robežas	Grafiks + aprēķins
AC-04	Collision	Plāns nešķērso collision object	Planning Scene / RViz + result
AC-05	Repeatability	Vismaz 5 atkārtojumi ar dokumentētu success rate	Test protocol
AC-06	Fault	Vismaz viena kļūme tiek korekti diagnosticēta	Fault injection pierādījums

84.3. Funkcionālais koks projektam

Integrētā projekta Functional Tree

```

To execute a safe manipulation mission
├─ To model robot
│   ├── kinematics
│   ├── dynamics
│   └─ limits / frames
├─ To generate desired motion
│   ├── task / goal
│   ├── trajectory
│   └─ constraints
├─ To plan collision-free motion
│   ├── planning scene
│   ├── planner
│   └─ collision checking
├─ To control motion
│   ├── state feedback
│   ├── torque/position control
│   └─ actuator interface
└─ To verify execution
    ├── logging
    ├── acceptance tests
    └─ fault / recovery

```

85. v1.2 eksāmenu un aizstāvēšanas jautājumu banka

85.1. State-space un System Identification

1. Kas ir sistēmas stāvoklis, un ar ko tas atšķiras no mērāmas izvades?
2. Izskaidro A, B, C, D matricu fizisko nozīmi.
3. Kāpēc digitālai vadībai nepieciešams diskrets modelis?
4. Ko nozīmē controllability un kādas praktiskas sekas ir nekontrolējamam stāvoklim?
5. Ko nozīmē observability un kā tas saistās ar sensoru izvēli?
6. Kāpēc modelis jāvalidē ar datiem, kas netika izmantoti identifikācijai?
7. Kā nepietiekams excitation var sabojāt System Identification rezultātu?
8. Ko var secināt no strukturētiem residuals?

85.2. LQR un MPC

1. Ko fiziski nozīmē Q un R matricas LQR uzdevumā?
2. Kāpēc LQR bez papildu risinājuma var neatrisināt statistisku reference tracking kļūdu?
3. Kāda ir galvenā MPC priekšrocība, ja aktuatoram ir stingrs limits?
4. Kas ir receding horizon princips?
5. Kāda ir prediction horizon ietekme uz aprēķinu slodzi un uzvedību?
6. Kāda atšķirība starp hard un soft constraint?
7. Kāpēc optimizācijas izpildes laiks ir vadības sistēmas prasība?

85.3. Manipulatora dinamika un MoveIt 2

1. Izskaidro katru terminu $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$.
2. Kāpēc gravity compensation var samazināt pozīcijas kļūdu?
3. Kas ir computed torque control un kā model mismatch to ietekmē?
4. Kā $J^T F$ pārvērš Cartesian spēku locītavu torque?
5. Kāda atšķirība starp path un trajectory?
6. Kāpēc quintic trajectory var būt noderīgāka par cubic?
7. Kas ir Planning Scene un kāpēc collision object ir jāievieto tajā?
8. Kas ir planning group MoveIt 2?
9. Kāda loma ir move_group, planning pipeline un controller?
10. Kāpēc FollowJointTrajectory ir svarīgs savienojums starp plānu un kontrolieri?
11. Kāpēc sarežģītam pick-and-place uzdevumam MTC stage dekompozīcija ir labāka nekā viena monolīta funkcija?

85.4. Aprēķinu uzdevumi

- Dots 2×2 A un 2×1 B. Aprēķināt controllability matricu un tās rangu.
- Dots Q, R un jau aprēķināts P. Aprēķināt LQR $K = R^{-1}B^T P$.
- Dots sampling time un nepārtrauktas pirmās kārtas sistēmas parametrs. Izveidot vienkāršotu diskretu modeli.
- Dots q, $\dot{q}$, $\ddot{q}$ un vienkāršots M/g modelis. Aprēķināt nepieciešamo torque.
- Dots Jacobian un Cartesian force. Aprēķināt $\tau = J^T F$.

- Dotai cubic trajectory pārbaudīt sākuma/beigu pozīciju un ātrumu robežnosacījumus.
- Dots MPC horizonts N , input limits un candidate komandu secība. Aprēķināt cost un noteikt pieļaujamo kandidātu.

85.5. Aizstāvēšanas dziļie jautājumi

- Kuru sava projekta daļu tu neuzticētu modelim bez papildu safety layer un kāpēc?
- Kas tavā modelī ir pieņēmums, nevis izmērīts fakts?
- Kā tu pierādītu, ka labs simulācijas rezultāts pārnēsas uz reālu robotu?
- Kas notiks, ja collision geometry atšķirsies no reālās iekārtas?
- Kā tu izvēlētos starp PID, LQR un MPC konkrētam uzdevumam?
- Kādu mērījumu tu pievienotu, lai sistēma kļūtu labāk observable?
- Kādu fault injection testu tu uzskatītu par visvērtīgāko savam manipulatoram?

86. Pasniedzēja pašmācības ceļš — advanced control un manipulatori

Šī sadaļa paredzēta pasniedzēja paša profesionālajai izaugsmei. Ieteicams necensties vienlaikus apgūt visas formulas, bet katru tēmu sasaistīt ar vienu reproducējamu Python/simulācijas eksperimentu.

Posms	Ko apgūt	Praktisks rezultāts
1	Lineārā algebra + eigenvalues	2x2 state-space stabilitātes analīze
2	State-space + discretization	Mass-spring/DC motor simulācija
3	Controllability/observability	Rank testi un sensora/aktuatora interpretācija
4	System Identification	ARX/first-order modelis no reāliem datiem
5	LQR	Q/R tuning eksperiments
6	MPC	Input/state constraint eksperiments
7	Manipulator dynamics	2DOF M(q), C, g simulācija
8	Trajectory generation	Cubic/quintic + velocity/acceleration grafiki
9	Torque/impedance control	Disturbance un model mismatch eksperiments
10	MoveIt 2	Planning Scene + collision-aware plan
11	MoveIt Task Constructor	Pick-and-place stages
12	Integrācija	Modelis → planner → controller → test report

86.1. Pasniedzēja mini lekcijas pašpārbaude

- 5 minūtēs izskaidrot state-space bez matricas biedēšanas: “iekšējais stāvoklis + ievade → nākamais stāvoklis”.
- 5 minūtēs izskaidrot LQR kā kompromisu starp kļūdu un vadības piepūli.
- 5 minūtēs izskaidrot MPC kā atkārtotu nākotnes optimizāciju ar constraints.
- 5 minūtēs izskaidrot manipulatora dinamisko vienādojumu, sasaistot katru terminu ar fizisku efektu.
- 5 minūtēs izskaidrot, ko tieši MoveIt plāno un ko tas nevada tieši motoru līmenī.
- 5 minūtēs izskaidrot atšķirību starp Planning Scene, planner un JointTrajectoryController.

86.2. Pašmācības projekts

Ieteicamais personīgais projekts: 2DOF virtuāls manipulators Python → dinamikas simulācija → LQR/computed torque → URDF/Xacro → MoveIt 2 simulācija → collision object → MTC pick-and-place. Tas vienā ķēdē nostiprina matemātiku, programmēšanu, mehatroniku, robotiku un sistēmu integrāciju.

87. [ARHĪVS] Versijas 1.2 statuss un nākamais slānis

87.1. Kas pievienots v1.2

- State-space modeļi, discretization, linearizācija, controllability un observability.
- System Identification un mērījumu datu validācija.
- LQR, Riccati vienādojuma jēga, Q/R tuning un reference tracking.
- MPC, receding horizon, constraints, soft constraints un runtime prasības.
- Manipulatoru dinamika, gravity compensation, computed torque un J^TF.
- Trajectory generation, impedance un force control pamati.
- MoveIt 2 arhitektūra, Planning Scene, planning pipelines, ros2_control execution un MoveIt Task Constructor.
- 8 praktiskās laboratorijas, integrēts manipulatora projekts, eksāmenu banka un pasniedzēja pašmācības ceļš.

87.2. Nākamais plānotais slānis — v1.3

Nākamais tehniskais slānis paredzēts AI un 3D uztverei robotikā: Deep Learning robotu redzei, CNN pamati, transfer learning, object detection/segmentation, 3D object detection, point-cloud features un uztveres-plānošanas integrācija.

87.3. Galvenā profesionālā doma

Advanced control un MoveIt 2 jāsasaista ar vienu un to pašu inženiertehnisko principu: prasība → modelis → algoritms → realizācija → verifikācija → drošs izpildījums. Sarežģītāka matemātika nemaina nepieciešamību pēc izmērāmiem acceptance criteria un reproducējama testa pierādījuma.

Primārie tehniskie avoti un versiju piezīme — v1.2

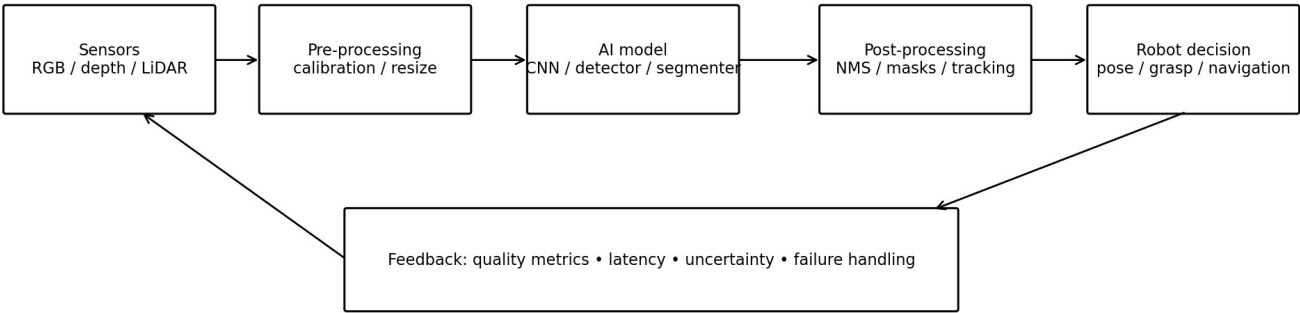
MoveIt 2 sadaļas tehniskā arhitektūra un termini pārbaudīti pret MoveIt Documentation (Main/Rolling) 2026-08-18: Concepts; The move_group node; Planning Scene Monitor; Motion Planning; Move Group C++ Interface; Motion Planning Python API; MoveIt Task Constructor. Trajectory execution termini pārbaudīti pret ROS 2 joint_trajectory_controller dokumentāciju. State-space, LQR, MPC un manipulatoru dinamikas matemātika dokumentā izmantota kā vispārpieņemta vadības teorijas/robotikas bāze; konkrētu bibliotēku API kods ir mācību piemērs un jāvalidē pret projekta izmantoto versiju.

- https://moveit.picknik.ai/main/doc/concepts/move_group.html
- https://moveit.picknik.ai/main/doc/concepts/planning_scene_monitor.html
- https://moveit.picknik.ai/main/doc/concepts/motion_planning.html
- https://moveit.picknik.ai/main/doc/examples/motion_planning_python_api/motion_planning_python_api_tutorial.html
- https://moveit.picknik.ai/main/doc/concepts/moveit_task_constructor/moveit_task_constructor.html
- https://docs.ros.org/en/ros2_packages/rolling/api/joint_trajectory_controller/doc/userdoc.html

88. Deep Learning robotu redzei — pamati un sistēmas domāšana

Deep Learning robotikā nav atsevišķs “AI triks”, bet viena no uztveres sistēmas komponentēm. Modelis saņem sensoru datus un rada klasifikāciju, bounding box, masku, keypoints, dziļuma novērtējumu vai citu semantisku rezultātu. Lai rezultāts būtu izmantojams robotam, tas jāsaista ar kalibrāciju, koordinātu sistēmām, laika zīmogiem, kvalitātes metriku, reāllaika prasībām un kļūmju apstrādi.

Deep Learning robotu redzes uztveres ķēde



88.1. attēls. Deep Learning uztveres ķēde robotā.

88.1. Machine Learning un Deep Learning atšķirība

Klasiskā Machine Learning pieejā cilvēks bieži iepriekš izstrādā pazīmes (features), piemēram, malu skaitu, formas aprakstus vai krāsu histogrammas. Deep Learning modelis daudzus pazīmju līmeņus iemācās no datiem pats, optimizējot tīkla parametrus pret definētu loss funkciju.

Pieeja	Raksturīgais process	Robotikas piemērs
Klasisks CV + noteikumi	Cilvēks definē pazīmes un sliekšņus	Krāsaina marķiera noteikšana ar HSV threshold
Machine Learning	Cilvēks sagatavo pazīmes, modelis mācās lēmumu	Defekta klasifikācija no tekstūras pazīmēm
Deep Learning	Tīkls mācās pazīmes un gala uzdevumu kopā	Objektu detektors vai segmentācijas modelis

88.2. Training un inference

Training laikā modelis salīdzina prognozi ar ground truth un ar backpropagation palīdzību maina svarus. Inference laikā svāri vairs netiek mācīti: modelis tikai aprēķina rezultātu jaunam sensoru ievades paraugam. Robotam inference latentums un determinētība bieži ir tikpat svarīgi kā precizitāte.

Mācīšanās cikla vienkāršota shēma

DATA → MODEL → PREDICTION → LOSS → BACKPROP → UPDATE WEIGHTS → atkārtot

88.3. Tensor, batch, epoch, parameter un gradient

- Tensor — daudzdimensiju skaitļu masīvs; attēls bieži tiek reprezentēts kā $C \times H \times W$ vai $N \times C \times H \times W$.
- Batch — vairāki paraugi, kurus apstrādā vienā optimizācijas solī.
- Epoch — viena pilna pāreja caur training dataset.
- Parameter — mācāms modeļa skaitlis, piemēram, konvolūcijas filtra svars.
- Gradient — norāda, kādā virzienā mainīt parametrus, lai samazinātu loss.

88.4. Loss nav tas pats, kas gala kvalitātes metrika

Loss ir optimizācijas funkcija, kuru izmanto treniņa laikā. Gala kvalitāti parasti vērtē ar atsevišķām metriku grupām: accuracy/F1 klasifikācijai, IoU/mAP detection un segmentation, kā arī latency, FPS, atmiņas patēriņu un kļūmju scenārijiem reālā robotā.

88.5. Overfitting un generalization

Overfitting rodas, ja modelis ļoti labi atceras training datus, bet slikti darbojas uz jauniem apstākļiem. Robotikā tas bieži atklājas, mainot apgaismojumu, kameru, fonu, objekta orientāciju, ātrumu vai vidi. Tāpēc dataset projektēšana ir daļa no sistēmas inženierijas, nevis tikai AI komandas darbs.

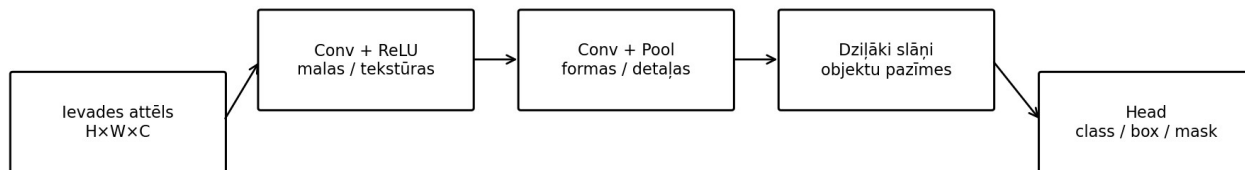
*Ko stāstīt auditorijai: Svarīgākā doma: “AI modelis nav gudrāks par datiem un prasībām, ar kurām tas validēts.”
Pirms demonstrēt modeļa arhitektūru, liec studentam definēt, ko nozīmē PASS reālā robotikas uzdevumā.*

88.6. Universitātes paplašinājums — CNN un Vision Transformers

CNN izmanto lokālas konvolūcijas un svaru koplietošanu. Vision Transformer tipa arhitektūras attēlu sadala tokenos/patches un izmanto attention mehānismus. Kursa pamatā CNN ir piemērotāks, lai izskaidrotu telpiskās pazīmes un receptive field; transformeru pieeju var izmantot kā paplašinājumu, uzsverot, ka konkrēta modeļa izvēle ir kompromiss starp datiem, precizitāti, latentumu un aparāturu.

89. CNN — konvolūciju neironu tīklu pamati

CNN pazīmju hierarhija — no pikseliem līdz semantiskam rezultātam



Receptive field pieaug → modelis apvieno arvien plašāku kontekstu

89.1. attēls. CNN pazīmju hierarhija.

89.1. Konvolūcija un kernel

Konvolūcijas slānis pārvieto nelielu filtru jeb kernel pāri attēlam un katrā pozīcijā aprēķina lokālu kombināciju. Treniņā kernel svāri tiek iemācīti. Agrīnie slāņi bieži reaģē uz vienkāršām malām un tekstūrām, bet dziļākie slāņi kombinē tās sarežģītākās pazīmēs.

Formula / matemātiskais modelis

$$Y[o,i,j] = b[o] + \sum_c \sum_m \sum_n W[o,c,m,n] \cdot X[c, i-s+m, j-s+n]$$

89.2. Stride, padding un feature map izmērs

Formula / matemātiskais modelis

$$H_{out} = \text{floor}((H_{in} + 2P - K) / S) + 1$$
$$W_{out} = \text{floor}((W_{in} + 2P - K) / S) + 1$$

Stride S nosaka filtra pārvietošanās soli, padding P pievieno robežvērtības, bet kernel K nosaka lokālā loga izmēru. Šie parametri maina feature map izmēru un aprēķinu slodzi.

89.3. Aktivācijas funkcija un nelinearitāte

Bez nelineāras aktivācijas vairāku lineāru slāņu virkne joprojām būtu ekvivalenta vienai lineārai transformācijai. ReLU ir klasisks piemērs: $\text{ReLU}(x) = \max(0, x)$. Mūsdienu arhitektūrās sastopamas arī citas aktivācijas, bet pamata mācību mērķis ir saprast, kāpēc nepieciešama nelinearitāte.

89.4. Pooling un downsampling

Downsampling samazina telpisko izšķirtspēju un palielina efektīvo receptive field. Tas var samazināt aprēķinu slodzi, bet vienlaikus zaudē precīzu telpisko informāciju; tāpēc segmentation un pose uzdevumos nepieciešami mehānismi, kas atjauno vai saglabā telpisko precizitāti.

89.5. Receptive field

Receptive field apraksta, cik liela ievades attēla zona var ietekmēt konkrētu dziļā slāņa aktivāciju. Robotikas objektu uztverē pārāk mazs konteksts var traucēt atšķirt lokālu tekstūru no pilna objekta; pārāk liela agresīva downsampling dēļ var pazust mazi objekti.

89.6. Vienkāršs PyTorch CNN demonstrators

CNN formas demonstrators — mācību piemērs

```
import torch
import torch.nn as nn

class SmallCNN(nn.Module):
    def __init__(self, classes=3):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((1, 1)),
        )
        self.head = nn.Linear(32, classes)

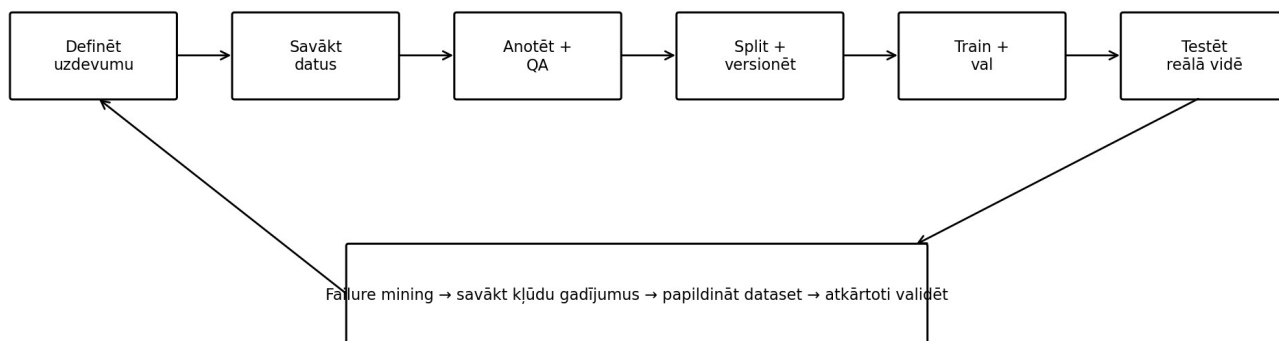
    def forward(self, x):
        x = self.features(x)
        x = x.flatten(1)
        return self.head(x)

model = SmallCNN(classes=3)
x = torch.randn(4, 3, 224, 224)
y = model(x)
print(y.shape) # torch.Size([4, 3])
```

Ko stāstīt auditorijai: Studentam nevajag sākumā memorēt visas PyTorch klases. Liec izsekot tikai tensoru formai: $4 \times 3 \times 224 \times 224 \rightarrow \text{feature maps} \rightarrow 4 \times 3 \text{ logits}$. Ja forma nav saprotama, arī sarežģītāks detektors paliks “melna kaste”.

90. Dataset inženierija, anotēšana un datu kvalitāte

Robotu redzes dataset dzīves cikls



90.1. attēls. Robotu redzes dataset dzīves cikls.

90.1. Sākt ar prasību, nevis ar attēlu mapīti

Pirms datu vākšanas jādefinē funkcija un acceptance criteria. “Atpazīt detaļu” ir pārāk vispārīgi. Labāka prasība: “Noteikt detaļas A klātbūtni 0.4–1.5 m attālumā, pie definēta apgaismojuma diapazona, ar recall $\geq$ noteikto robežu un inference laiku $\leq$ noteikto robežu uz mērķa aparatūras.”

90.2. Train / validation / test split

Training dati tiek izmantoti svaru optimizācijai, validation — hiperparametru un modeļa izvēlei, bet test dati jā saglabā gala neatkarīgai pārbaudei. Robotikā īpaši jā novērš data leakage: secīgi video kadri no viena brauciena nedrīkst nejauši sadalīties train un test tā, ka abās kopās nonāk gandrīz identiski kadri.

90.3. Anotāciju tipi

Uzdevums	Anotācija	Rezultāts
Classification	Attēla klases etiķete	Viena vai vairākas klases
Object detection	Class + bounding box	Objekta klase un 2D atrašanās vieta
Semantic segmentation	Klase katram pikselim	Pilna semantiskā karte
Instance segmentation	Maska katram objektam	Atsevišķu objektu formas
Keypoints / pose	Punktu koordinātas	Locītavas, satveršanas punkti, landmarks
3D detection	3D bbox + pose/class	Objekta atrašanās 3D telpā

90.4. Anotāciju QA

Modelis iemācās arī anotāciju kļūdas. Tāpēc jādefinē annotation guide: ko darīt ar daļēji aizsegtu objektu, minimālo redzamo laukumu, robežām, “unknown” gadījumiem un klasēm, kuras vizuāli pārklājas. Daļai dataset ieteicama neatkarīga otrā pārbaude.

90.5. Datu daudzveidība robotikas vidē

- apgaismojums un ēnas;
- objekta skala un distance;
- kamera, objektīvs un fokuss;
- motion blur un vibrācija;
- daļēja aizsegšana;
- netīrumi, atspīdumi un caurspīdīgas virsmas;
- retie drošības kritiskie gadījumi;
- negatīvie paraugi bez mērķa objekta.

90.6. Data augmentation

Augmentation mākslīgi maina training paraugus, piemēram, crop, flip, brightness, blur vai perspektīvu. Augmentation jābūt fizikāli pamatotai konkrētajam uzdevumam: piemēram, vertikāls flip var būt absurds, ja detaļa ražošanas līnijā nekad neparādās otrādi.

90.7. Dataset versēšana un traceability

AI izstrādes traceability ķēde

Requirement ID → dataset version → annotation guide → model/config commit → metrics → deployment artifact

Ko stāstīt auditorijai: AI projektā dataset ir konfigurācijas elements tāpat kā PLC parametrs vai kalibrācijas fails. Ja nav zināms dataset version, rezultātu nav iespējams korekti reproducēt.

91. Transfer learning un fine-tuning robotu redzes uzdevumiem

91.1. Kāpēc transfer learning

Praksē bieži nav nepieciešams trenēt lielu redzes modeli no nejaušiem svariem. Pretrained backbone jau satur vispārīgas vizuālās pazīmes, un jaunu uzdevumu var pielāgot ar mazāku dataset un aprēķinu apjomu. Divas pamata stratēģijas ir fixed feature extractor un fine-tuning.

Stratēģija	Kas tiek trenēts	Kad noder
Fixed feature extractor	Galvenokārt jaunais gala head	Mazs dataset, ātrs sākums, līdzīgs vizuālais domēns
Partial fine-tuning	Head + daļa dziļāko backbone slāņu	Vidējs dataset, domēna atšķirības
Full fine-tuning	Lielākā daļa vai viss modelis	Lielāks dataset un pietiekama aprēķinu jauda

91.2. PyTorch piemērs: pretrained ResNet kā klasifikators

Transfer learning mācību kodols

```
import torch
from torchvision.models import resnet18, ResNet18_Weights
import torch.nn as nn

weights = ResNet18_Weights.DEFAULT
model = resnet18(weights=weights)

# Freeze backbone
for p in model.parameters():
    p.requires_grad = False

in_features = model.fc.in_features
model.fc = nn.Linear(in_features, 3) # 3 robotikas detaļu klases

trainable = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.Adam(trainable, lr=1e-3)
```

Pēc sākotnējās head apmācības var atkausēt daļu dziļāko slāņu un izmantot mazāku learning rate. Jāsalīdzina validation rezultāts, nevis automātiski pieņemt, ka vairāk trenētu slāņu nozīmē labāku modeli.

91.3. Learning rate un catastrophic forgetting

Fine-tuning laikā pārāk liels learning rate var strauji izmainīt pretrained svarus un sabojāt iepriekš apgūtās pazīmes. Tāpēc bieži izmanto mazāku learning rate backbone slāņiem nekā jaunajam head.

91.4. Early stopping un model checkpoint

Training loss samazināšanās pati par sevi nenozīmē uzlabojumu. Jāseko validation metrikai, jā saglabā labākais checkpoint un jāanalizē brīdis, kad validation sāk pasliktināties.

91.5. Domēna maiņa

Ja pretrained modelis apmācīts uz ikdienas fotogrāfijām, bet jaunais uzdevums ir termokamera, rentgens vai ļoti specifiski industriāli attēli, pazīmju pārnesamība var būt vājāka. Tas jānoskaidro eksperimentā, nevis jāpieņem.

92. Object detection — objektu atrašana un klasificēšana

92.1. Detection rezultāts

Object detector katram atrastajam objektam parasti dod klasi, confidence score un bounding box. Robotam ar to vēl nepietiek: rezultāts jāsaista ar laika zīmogu, kameras kalibrāciju, nepieciešamības gadījumā dziļumu un koordinātu sistēmu.

Vienkāršots 2D detection rezultāts

```
Detection = {class_id, score, x_min, y_min, x_max, y_max}
```

92.2. Intersection over Union — IoU

Formula / matemātiskais modelis

$$\text{IoU} = \text{Area}(\text{Prediction} \cap \text{GroundTruth}) / \text{Area}(\text{Prediction} \cup \text{GroundTruth})$$

IoU mēra, cik labi prognozētais bounding box pārklājas ar ground truth. Tas tiek izmantots gan apmācības/novērtēšanas procesos, gan detection metrikās.

92.3. Confidence un threshold

Pazeminot confidence threshold, bieži palielinās recall, bet var pieaugt false positives. Palielinot threshold, sistēma kļūst konservatīvāka, bet var palaist garām objektus. Drošības kritiskā uzdevumā threshold jāizvēlas no prasības un riska, nevis tikai pēc vizuāli “skaista” demo.

92.4. Non-Maximum Suppression — NMS

Detektors var radīt vairākus pārklājošus kandidātus vienam objektam. NMS saglabā spēcīgākos kandidātus un nomāc pārāk līdzīgus pārklājošus box. NMS parametrs ietekmē blīvu objektu situācijas un jāvalidē uz reāla dataset.

92.5. Anchor-based un anchor-free ideja

Daļa detektoru prognozē objektus attiecībā pret iepriekš definētiem anchor, bet citi izmanto anchor-free formulējumu. Pamatkursā svarīgāk ir saprast gala rezultātu, loss komponentes un metriku, nevis iegaumēt konkrētas modeļu paaudzes arhitektūru.

92.6. Praktiska 2026 API piezīme

Praktiskās laboratorijās var izmantot TorchVision detection modeļus vai aktuālu YOLO saimes implementāciju. Bibliotēku API un modeļu nosaukumi mainās, tādēļ dokumentā tiek saglabāts algoritmiskais princips, bet konkrētais modelis pirms kursa sākuma jāpārbauda pret izvēlētajā bibliotēkas dokumentāciju.

Object detection integrācijas princips

```
# Pseudokods — neatkarīgs no konkrētas detector bibliotēkas
model = load_pretrained_detector()
results = model(image)

for det in results:
    if det.score >= CONF_THRESHOLD:
        publish(det.class_id, det.score, det.bbox)
```

92.7. Robotikas failure cases

- mazs objekts tālu no kameras;
- objekti pārklājas;
- spēcīgs atspīdums;
- motion blur;
- objekts ārpus training distribūcijas;
- detektors pārliecinoši kļūdās par nezināmu objektu;
- latency dēļ robots izmanto novecojušu detection.

93. Semantic un instance segmentation

93.1. Semantic segmentation

Semantic segmentation katram pikselim piešķir klasi. Divi atsevišķi viena tipa objekti rezultātā var būt vienas klases reģions. Tas ir noderīgi braucamai virsmai, sienām, grīdai, zonām vai defekta laukuma noteikšanai.

93.2. Instance segmentation

Instance segmentation papildus pikseļu maskai atšķir atsevišķus vienas klases objektus. Tas ir būtiski satveršanai vai detaļu skaitīšanai, ja vajadzīgs katras instances kontūrs un identitāte.

Uzdevums	Tipiska izvade	Robotikas pielietojums
Detection	Box + class + score	Ātra objekta atrašana
Semantic segmentation	Class map H×W	Braucama zona / virsmas
Instance segmentation	Maska katrai instancei	Satveršana, detaļu kontūras, skaitīšana

93.3. IoU un Dice maskām

Formula / matemātiskais modelis

$$\text{IoU} = |P \cap G| / |P \cup G|$$

$$\text{Dice} = 2|P \cap G| / (|P| + |G|)$$

93.4. No maskas līdz ģeometrijai

Maska var kalpot kā filtrs depth vai point cloud datiem: izvēlas tikai tos 3D punktus, kuru projekcija atrodas konkrētās objekta maskas iekšpusē. Tādējādi 2D semantiku var pārvērst 3D objekta punktu kopā.

93.5. Industriāls piemērs — defekta laukums

Ja uz detaļas jānovērtē skrāpējuma vai pārklājuma defekta laukums, semantic segmentation var būt informatīvāka par bounding box. Taču gala mm² aprēķinam nepieciešama kameras kalibrācija vai zināms mērogs.

94. Pose, keypoints, tracking un laika dimensija

94.1. Keypoint detection

Keypoint modelis prognozē semantiski nozīmīgus punktus: cilvēka locītavas, robota detaļas orientierus, satveršanas punktus vai objekta stūrus. No vairākiem 2D/3D atskaītes punktiem var iegūt objekta orientāciju un pose.

94.2. 2D pose nav 3D pose

Punkta koordinātas attēlā (u,v) nepietiek, lai zinātu tā telpisko koordinātu. Nepieciešams dziļums, stereo ģeometrija, zināms objekta modelis (PnP) vai cits 3D avots.

94.3. Tracking

Tracking sasaista detection starp kadriem. Tas ļauj saglabāt objekta identitāti, novērtēt kustību un samazināt lēmumu lēkāšanu. Tracking sistēmai jāapsver object ID dzīves cikls, occlusion, missed detections un re-identification.

Tracking cikls

Frame t: detection → association → track state → prediction

Frame t+1: detection → association → update / new track / lost track

94.4. Laika zīmogi un latency

Robotā redzes rezultāts ir derīgs tikai kopā ar laiku. Ja attēls uzņemts t_0 , bet detection publicēts t_0+150 ms, kustīgs robots pa šo laiku jau atrodas citur. Tāpēc jāglabā oriģinālais sensora timestamp un jāmēra end-to-end latency.

94.5. Optical flow kā kustības pazīme

Optical flow novērtē pikseļu kustību starp kadriem un var palīdzēt kustības, vizuālās odometrijas vai tracking uzdevumos. Deep Learning nav obligāts katrā perception komponentē — labs robotikas risinājums kombinē klasisku ģeometriju ar mācītiem modeļiem.

95. Modeļu novērtēšana, kļūdu analīze un robustums

95.1. Confusion matrix

Confusion matrix parāda, kuras klases modelis sajauc savā starpā. Robotikas uzdevumā tas ir vērtīgāk par vienu kopējo accuracy skaitli, jo dažādām kļūdām var būt ļoti atšķirīgas sekas.

Formula / matemātiskais modelis

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{F1} = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall})$$

95.2. Detection mAP

Detection kvalitāti bieži apkopo ar Average Precision un mAP pie definētiem IoU sliekšņiem. Studenta līmenī svarīgāk ir saprast, ka mAP apvieno precision-recall uzvedību un lokalizācijas prasību; profesionālā līmenī jāpieraksta tieši izmantotā mAP definīcija.

95.3. Reāllaika metrikas

Metrika	Ko tā atbild	Kāpēc robotā svarīgi
Inference latency	Cik ilgi modelis aprēķina rezultātu?	Novecojis rezultāts var radīt vadības kļūdu
End-to-end latency	No sensora timestamp līdz patērētāja rezultātam	Ietver transportu un post-processing
FPS / rate	Cik bieži saņem rezultātu?	Ietekmē reakcijas ātrumu
Memory/VRAM	Cik resursu izmanto?	Edge ierīcēm ir ierobežojumi
Worst-case behavior	Kas notiek sliktākajā scenārijā?	Drošības un recovery projektēšana

95.4. Error slicing

Kopējo test rezultātu sadala apakšgrupās: tumšs/gaišs, tuvs/tāls, mazs/liels objekts, kustīgs/statisks, clean/dirty lens u.c. Tas palīdz atrast konkrētu sistēmas vājumu, kuru vidējā metrika var paslēpt.

95.5. Unknown un out-of-distribution

Neironu tīkls var dot augstu confidence arī ievadei, kas būtiski atšķiras no training datiem. Tāpēc drošākā arhitektūrā rezultātam ir acceptance gate, diagnostika un fallback uz konservatīvu uzvedību, nevis pieņēmums, ka “augsts score = patiesība”.

95.6. Validation matrica

Testa dimensija	Piemēri	Acceptance evidence
Apgaismojums	min/nom/max, backlight	Recall/FP katrā zonā
Distance	0.4 / 1.0 / 1.5 m	mAP vai detection success
Occlusion	0 / 25 / 50 %	Failure threshold
Motion	0 / vidējs / max ātrums	Latency + missed detections

Testa dimensija	Piemēri	Acceptance evidence
Camera fault	frame drop / blur / covered lens	Diagnostic + safe fallback

Ko stāstīt auditorijai: Šeit sasaisti AI ar kvalifikācijas domāšanu: modelis nav “labs” abstrakti. Ir definēta intended use, acceptance criteria, test conditions un pierādījumi.

96. Edge inference, ONNX un reāllaika izvietošana robotā

96.1. Training dators un deployment dators var būt atšķirīgi

Modeli var trenēt uz GPU darbstacijas, bet robotā inference izpildīt uz industriāla PC, SBC vai specializēta accelerator. Tāpēc jāizvērtē model size, operatoru atbalsts, precision, atmiņa, siltums un latency.

96.2. ONNX kā starpformāts

ONNX bieži izmanto modeļa eksportam starp training framework un deployment runtime. Eksports pats par sevi negarantē identisku rezultātu: jāsalīdzina reference framework un deployment runtime izejas uz reprezentatīviem testiem.

Modeļa izvietošanas verifikācijas ķēde

```
# Konceptuāls eksporta workflow
trained_model
  ↓ export
model.onnx
  ↓ runtime load
inference(input)
  ↓ compare
reference_output vs deployed_output
```

96.3. OpenCV DNN

OpenCV DNN modulis var ielādēt vairākus neironu tīklu modeļu formātus un izmantot tos inference. Tas var būt praktisks ceļš, ja robotikas programma jau balstās OpenCV un nav nepieciešams pilns training framework mērķa ierīcē.

96.4. Quantization

Quantization samazina skaitļu precizitāti, piemēram, no FP32 uz FP16 vai INT8, lai samazinātu atmiņas un aprēķinu slodzi. Ieguvums jāvērtē kopā ar precizitātes izmaiņu un konkrētās aparatūras atbalstu.

96.5. Pipeline optimizācija

- neapstrādāt lielāku izšķirtspēju, nekā uzdevums prasa;
- izvairīties no nevajadzīgām kopijām CPU<->GPU;
- mērīt preprocessing, inference un post-processing atsevišķi;
- izmantot warm-up pirms latency mērījuma;
- mērīt ne tikai vidējo, bet arī p95/p99 vai worst-case laiku;
- jābūt noteiktai uzvedībai, ja inference kavējas vai modelis nav pieejams.

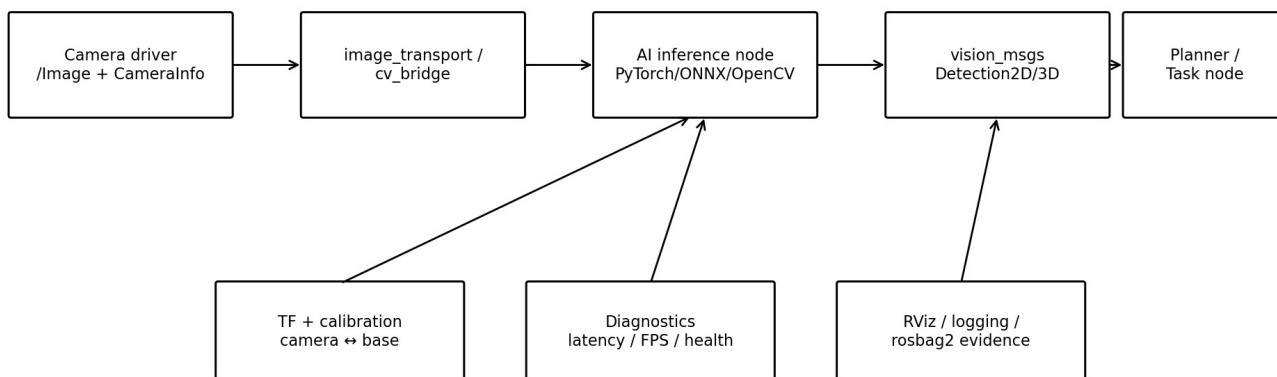
96.6. Deployment acceptance criteria piemērs

Prasība	Mērījums	Piemēra acceptance
Precizitāte	Validation dataset	Definēta mAP/F1 robeža

Prasība	Mērijums	Piemēra acceptance
Latency	End-to-end timestamp delta	$\leq$ projekta robeža
Rate	Publicēto rezultātu Hz	$\geq$ projekta robeža
Stabilitāte	30 min soak test	Nav crash / memory growth
Fault response	Atvienot kameru / model fail	Diagnostic + safe state

97. ROS 2 integrācija robotu redzei

ROS 2 robotu redzes AI arhitektūra



97.1. attēls. ROS 2 AI perception arhitektūra.

97.1. Attēlu interfeiss

ROS 2 kameru plūsmās centrālie interfeisi ir `sensor_msgs/Image` un `sensor_msgs/CameraInfo`. `CameraInfo` glabā kameras kalibrācijas parametrus, kas nepieciešami, lai pikseļu rezultātu korekti sasaistītu ar ģeometriju.

97.2. image_transport

`image_transport` ļauj izmantot dažādus attēla transporta veidus, nepārrakstot augstāka līmeņa algoritmu. Tas ir būtiski, ja kamera un AI node atrodas uz dažādiem datoriem vai neapstrādāta video plūsma ir pārāk smaga tīklam.

97.3. vision_msgs

`vision_msgs` definē standartizētus 2D un 3D redzes rezultātu message tipus, piemēram, `Detection2DArray` un `Detection3DArray`. Standarta interfeiss samazina savienojumu starp konkrētu modeli un pārējo robotikas sistēmu.

Ieteicams robotu redzes topic līguma piemērs

```
/camera/image_raw    sensor_msgs/Image
/camera/camera_info  sensor_msgs/CameraInfo
/perception/detections vision_msgs/Detection2DArray
/perception/detections3d vision_msgs/Detection3DArray
/perception/diagnostics diagnostic_msgs/DiagnosticArray
```

97.4. TF un frame_id

Detection koordinātas nav pilnvērtīgas, ja nav zināms frame. 2D rezultāts parasti ir saistīts ar kameras attēlu, bet 3D pose jānorāda definētā koordinātu sistēmā, piemēram, camera_optical_frame vai base_link. TF nodrošina transformāciju ķēdi starp sensoriem un robotu.

97.5. QoS un reāllaika plūsma

Kameras plūsmā bieži svarīgāks ir svaigākais kadrs nekā visu veco kadru saglabāšana. QoS jāizvēlas pēc sensora un patērētāja uzvedības. AI node nedrīkst nekontrolēti uzkrāt vecus kadrus, ja inference ir lēnāks par kameras frekvenci.

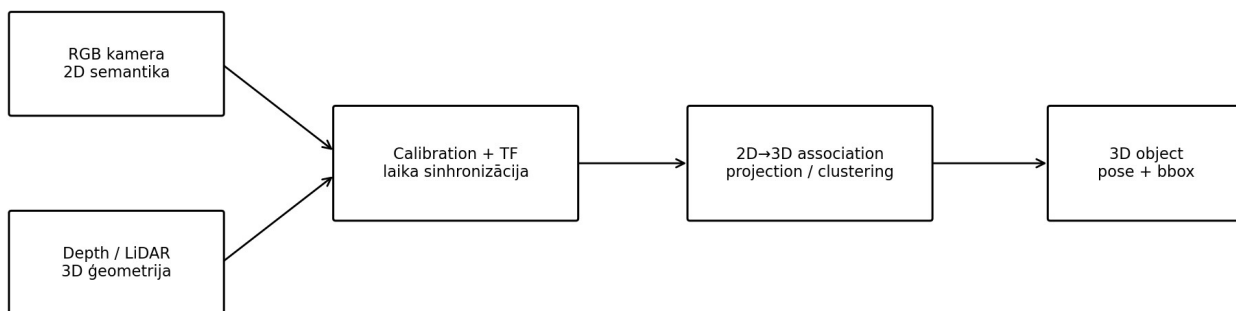
97.6. Diagnostika

Diagnostic key	Nozīme
input_rate_hz	Saņemto attēlu frekvence
inference_rate_hz	Pabeigto inference frekvence
latency_ms	End-to-end vai inference latentums
dropped_frames	Izlaisto kadru skaits
model_id/version	Izvietotā modeļa identitāte
last_error	Pēdējā kļūme / fallback iemesls

Ko stāstīt auditorijai: Ros topic nav tikai datu caurule. Tas ir sistēmas interfeisa līgums: type + units + frame + timestamp + QoS + expected rate + failure semantics.

98. 3D objektu uztvere — RGB-D, LiDAR un sensoru sapludināšana

RGB-D / LiDAR un Deep Learning uztveres sapludināšana



Pamatprincips: 2D modelis pasaka “kas?”, 3D sensors palīdz noteikt “kur?”

98.1. attēls. 2D semantikas un 3D ģeometrijas sapludināšana.

98.1. No pikseļa uz 3D punktu

Ja katram pikselim ir zināms dziļums Z un kameras intrinsics, 2D koordinātu var pārvērst 3D punktā kameras koordinātu sistēmā.

Formula / matemātiskais modelis

$$X = (u - c_x) \cdot Z / f_x$$

$$Y = (v - c_y) \cdot Z / f_y$$

$$Z = Z$$

98.2. Kameras kalibrācija

Intrinsics apraksta kameras iekšējo ģeometriju (focal length, principal point, distortion), bet extrinsics — kameras pozu attiecībā pret citu koordinātu sistēmu. Nepareiza kalibrācija rada sistemātisku 2D→3D kļūdu pat tad, ja AI detektors ir perfekts.

98.3. 2D detection + depth

Vienkārša pieeja: detektors atrod 2D box, tad box centrā vai maskas reģionā iegūst depth. Robustāk ir izmantot masku un statistiski filtrēt dziļuma vērtības, jo box var saturēt fonu.

98.4. 2D mask + point cloud

Instance masku var projicēt uz point cloud un atlasīt tikai objekta punktus. Pēc tam var aprēķināt centroid, PCA orientāciju, axis-aligned vai oriented bounding box un izmantot rezultātu satveršanai vai plānošanai.

98.5. LiDAR clustering

Ja nav 2D semantikas, point cloud var grupēt ar clustering, piemēram, DBSCAN. Tad klasterus var klasificēt vai sasaistīt ar kameras detection. Clustering parametri jāizvēlas atbilstoši point density un distance.

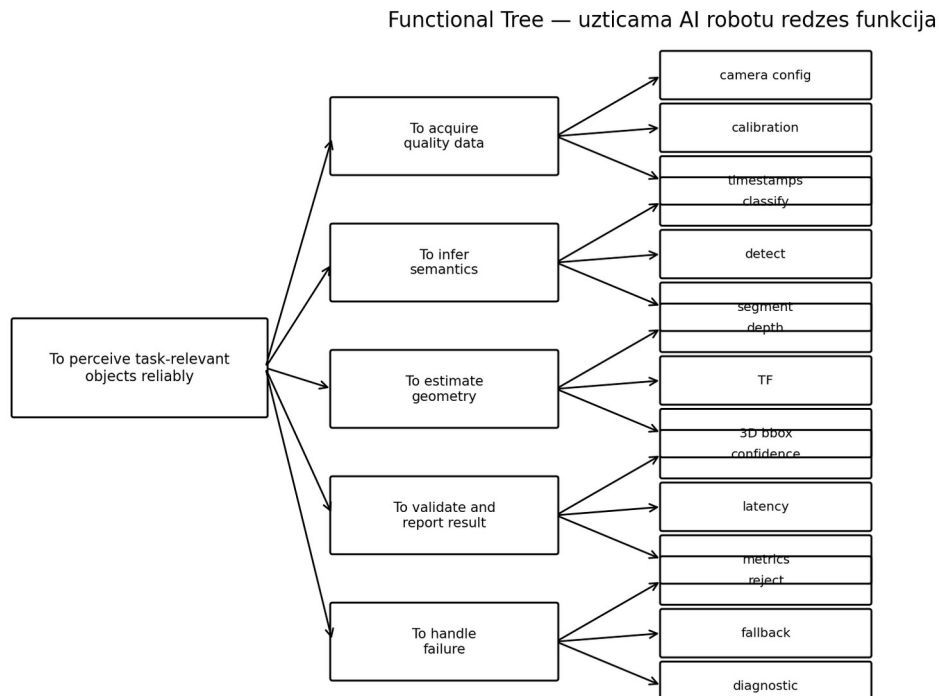
98.6. 3D detection rezultāts

3D detection tipiski satur class hypothesis, score, 3D bounding box un pose. ROS 2 vision_msgs satur Detection3D un Detection3DArray interfeisus šādu rezultātu apmaiņai.

98.7. Sensoru laika un telpas sinhronizācija

- camera $\leftrightarrow$ LiDAR extrinsic transform;
- timestamp atšķirība starp sensoriem;
- kustības kompensācija, ja robots vai objekts kustas;
- TF transform pie pareizā timestamp;
- depth invalid/no-return punkti;
- sensoru redzes lauku nepārklāšanās.

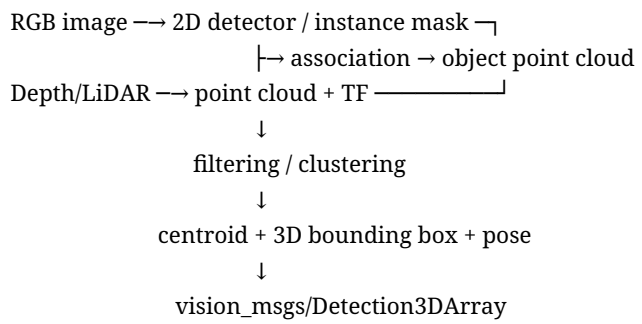
98.8. Functional Tree



98.2. attēls. Functional Tree — uzticama AI robotu redzes funkcija.

98.9. 3D object pipeline piemērs

2D/3D fusion datu plūsma



99. v1.3 praktiskās AI/robotu redzes laboratorijas

Laboratoriju secība paredzēta, lai students pakāpeniski pārietu no datu kvalitātes un klasifikācijas līdz pilnai ROS 2 3D perception funkcijai. Katras laboratorijas rezultātam jābūt reproducējamam: dataset/config/model versija, mērījumu tabula un īss secinājums.

DL1 — Dataset audits un leakage noteikšana

Mērķis: Izveidot reproducējamu robotu redzes dataset struktūru un atrast potenciālus data leakage riskus.

Ko stāstīt auditorijai: Slikts split var radīt “izcilu” test rezultātu bez reālas generalization. Video secīgi kadri ir klasisks leakage piemērs.

Iesniedzamie pierādījumi

- dataset_v1 struktūra
- split statistika
- dataset card
- īss leakage riska secinājums

Darba secība

1. Definē 3 klases un intended use.
2. Sakārto attēlus pa acquisition session, ne tikai failu nosaukumiem.
3. Izveido train/val/test split pa sesijām.
4. Saskaiti klases katrā split.
5. Atrodi vismaz 5 potenciālus duplicate/near-duplicate gadījumus.
6. Izveido dataset card ar versiju, datumu, klases definīcijām un ierobežojumiem.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Split neatkarība	Test sesijas nav train datos			
Klases	Visas klases pārstāvētas			
Dataset card	Versija + limitations dokumentēti			

A/B/C paplašinājums

- A: manuāls audits.
- B: Python skripts failu/klases statistikai.
- C: perceptual hash vai embedding balstīts near-duplicate audits.

DL2 — CNN klasifikators un overfitting novērošana

Mērķis: Apmācīt nelielu CNN un pierādīt atšķirību starp training un validation uzvedību.

Ko stāstīt auditorijai: Modelim jāseko abām līknēm. Ja training loss krīt, bet validation pasliktinās, tā ir sistēmiska norāde, ka modelis nevis “mācās labāk”, bet pārpielāgojas training datiem.

Darba secība

1. Sagatavo train/val datu loaderus.
2. Apmāci SmallCNN 10–30 epoch.
3. Saglabā train loss/accuracy un val loss/accuracy katrā epoch.
4. Uzzīmē līknes.
5. Atzīmē labāko validation checkpoint.
6. Atkārti ar augmentation vai regularization.
7. Salīdzini rezultātus.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Forward pass	Tensor shape korekta			
Validation log	Metrikas katrā epoch			
Checkpoint	Saglabāts labākais val modelis			

A/B/C paplašinājums

- A: tikai accuracy.
- B: loss + accuracy līknes.
- C: salīdzināt 2 regularization variantus un pamatot izvēli.

DL3 — Transfer learning ar pretrained modeli

Mērķis: Pielāgot pretrained classification backbone tehniskam 3 klašu uzdevumam un salīdzināt fixed-feature ar fine-tuning.

Ko stāstīt auditorijai: Transfer learning ir eksperiments, ne dogma. Studentam jāpierāda, vai atkausēti slāņi uzlabo validation un kā mainās treniņa laiks.

Darba secība

1. Ielādē pretrained modeli.
2. Nomaini final head uz savu klašu skaitu.
3. Apmāci tikai head.
4. Pieraksti labāko validation rezultātu.
5. Atkausē pēdējo backbone bloku.
6. Fine-tune ar mazāku learning rate.
7. Salīdzini kvalitāti, training time un stability.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Head-only	Training darbojas			
Fine-tune	Metrikas iegūtas			
Comparison	Secinājums balstīts datos			

A/B/C paplašinājums

- A: fixed feature.
- B: partial fine-tune.
- C: class-wise confusion matrix + error slicing.

DL4 — Object detection un threshold tuning

Mērķis: Izmantot pretrained detektoru un eksperimentāli noteikt confidence threshold ietekmi uz FP/FN.

Ko stāstīt auditorijai: Threshold nav kosmētisks parametrs. Tas maina sistēmas lēmumu risku un jāizvēlas no validation datiem.

Darba secība

1. Izvēlies pretrained detektoru.
2. Sagatavo 30–100 attēlu test kopu ar mērķa objektiem un negatīviem kadriem.
3. Izpildi inference ar vismaz 3 confidence thresholds.
4. Saskaiti TP/FP/FN.
5. Aprēķini precision/recall.
6. Izvēlies threshold un pamato to intended use kontekstā.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Low threshold	Augstāks recall / vairāk FP sagaidāms			
High threshold	Mazāk FP / iespējami FN			
Chosen threshold	Pamatots ar metriku un risku			

A/B/C paplašinājums

- A: manuāla TP/FP/FN tabula.
- B: automātisks metrics skripts.
- C: IoU matching un PR curve.

DL5 — Instance segmentation un 3D objekta punktu atlase

Mērķis: No 2D instance maskas un depth izveidot objekta 3D punktu kopu.

Ko stāstīt auditorijai: Šis ir ļoti svarīgs pārejas piemērs: neironu tīkls pasaka “kuri pikseļi pieder objektam”, bet ģeometrija pasaka “kur šie punkti atrodas telpā”.

Darba secība

1. Iegūsti RGB + depth pāri.
2. Izveido vai izmanto instance masku.
3. Atlasīt maskas pikseļus ar derīgu depth.
4. Pārreķini tos X,Y,Z ar intrinsics.
5. Noņem outliers.
6. Aprēķini centroid.
7. Vizualizē objekta point cloud.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Valid depth	Nederīgie depth punkti filtrēti			
Centroid	Aprēķināts 3D centroid			
Visualization	Objekts telpiski atdalīts			

A/B/C paplašinājums

- A: centra depth punkts.
- B: maskas median depth + point cloud.
- C: oriented bounding box + pose novērtējums.

DL6 — ROS 2 Detection2DArray inference node

Mērķis: Izveidot ROS 2 node, kas no Image saņem kadru un publicē standartizētu 2D detection rezultātu.

Ko stāstīt auditorijai: Svarīgais nav konkrētais modelis, bet interfeiss: timestamp, frame, detection type, diagnostics un reproducējams model version.

Darba secība

1. Subscribe uz /camera/image_raw.
2. Konvertē attēlu modeļa ievadei.
3. Izpildi inference.
4. Pārveido rezultātu uz vision_msgs/Detection2DArray.
5. Saglabā input header timestamp.
6. Publicē diagnostics ar latency un model version.
7. Pārbaudi topic ar ros2 topic echo/hz.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Message type	Detection2DArray			
Timestamp	Saglabāts sensora header			
Latency	Mērīta un publicēta			

A/B/C paplašinājums

- A: viena detection.
- B: vairāki objekti + labels.
- C: image_transport + QoS tests un rosbag evidence.

DL7 — 2D→3D Detection3DArray

Mērķis: Sasaistīt 2D detection ar depth/point cloud un publicēt 3D objekta rezultātu.

Ko stāstīt auditorijai: 3D rezultāts ir derīgs tikai tad, ja intrinsics, TF un timestamps ir korekti. AI score nevar kompensēt ģeometrijas kļūdu.

Darba secība

1. Saņem Detection2DArray + depth/point cloud.
2. Sinhronizē message pēc timestamp.
3. Atrodi 3D punktus detection/maskas reģionā.
4. Transformē uz base_link.
5. Aprēķini centroid un 3D bbox.
6. Publicē Detection3DArray.
7. RViz pārbaudi objekta pozīciju pret vidi.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Synchronization	Dati no viena laika loga			
TF	3D rezultāts base_link frame			
RViz	Bbox atbilst reālajam objektam			

A/B/C paplašinājums

- A: bbox centra depth.
- B: point cloud median/centroid.
- C: instance mask + oriented bbox + covariance/quality estimate.

DL8 — Edge deployment un latency budget

Mērķis: Salīdzināt reference modeli un deployment runtime pēc precizitātes un latency.

Ko stāstīt auditorijai: Deployment nav pabeigts, kamēr nav pierādīts, ka optimizētais modelis saglabā pieņemamu kvalitāti un izpildās noteiktajā laikā.

Darba secība

1. Izvēlies reference modeli.
2. Eksportē uz ONNX vai citu mērķa formātu.
3. Palaid deployment runtime.
4. Uz vieniem un tiem pašiem 50+ paraugiem salīdzini izejas.
5. Mēri preprocessing/inference/postprocessing laiku.
6. Aprēķini median un p95 latency.
7. Dokumentē atšķirības un acceptance.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Output equivalence	Atšķirības pieņemamā robežā			
Latency	P95 zem projekta robežas			
Soak	Ilgstošā testā nav memory growth/crash			

A/B/C paplašinājums

- A: tikai CPU ONNX.
- B: 2 runtimes salīdzinājums.
- C: FP16/INT8 kvantizācija + accuracy/latency trade-off.

DL9 — Failure injection robotu redzes sistēmai

Mērķis: Pierādīt drošu sistēmas uzvedību pie sensora un AI kļūmēm.

Ko stāstīt auditorijai: Uzticama robotika nav “modeļa accuracy”. Tā ir arī spēja konstatēt, ka perception nav derīgs, un pāriet kontrolētā stāvoklī.

Darba secība

1. Definē 5 kļūmes: camera disconnect, frozen frame, blur/covered lens, inference timeout, low confidence.
2. Katram gadījumam definē diagnostic condition.
3. Definē safe response.
4. Izpildi kontrolētu fault injection.
5. Reģistrē rosbag/log.
6. Pārbaudi, vai sistēma neizmanto stale perception.

7. Izveido FMEA tipa secinājumu.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Camera loss	Diagnostic + perception invalid			
Timeout	Stale result netiek izmantots			
Low confidence	Fallback/stop saskaņā ar prasību			

A/B/C paplašinājums

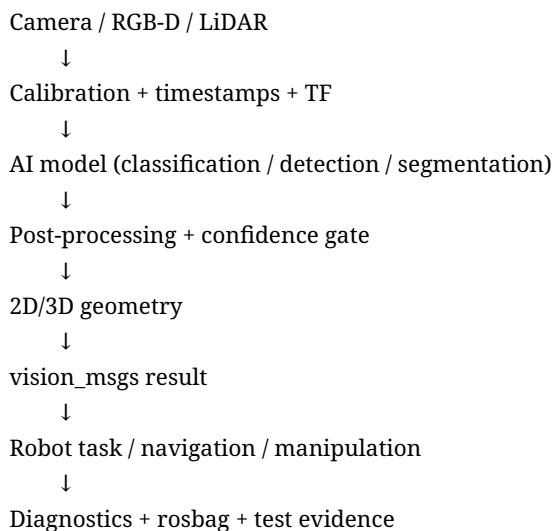
- A: manuāls fault test.
- B: automatizēti fault stimuli.
- C: recovery + health state machine + regression test.

100. Integrētais projekts — AI uztvere autonomam robotam

Projekta mērķis: izveidot dokumentētu perception apakšsistēmu, kas uztver uzdevumam svarīgus objektus, nosaka to 2D/3D stāvokli un droši nodod rezultātu robotam. Projekts var tikt realizēts simulācijā vai ar fizisku mobilo robotu/manipulatoru.

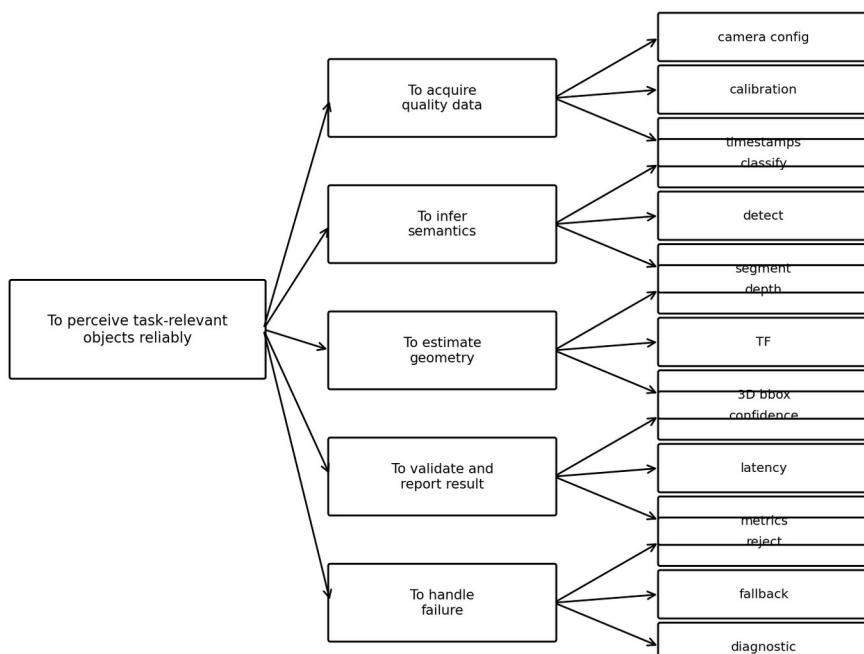
100.1. Minimālā arhitektūra

Integrētā projekta perception arhitektūra



100.2. Functional Tree projekta sākumam

Functional Tree — uzticama AI robotu redzes funkcija



100.1. attēls. AI perception Functional Tree.

100.3. Prasību piemērs

ID	Prasība	Verifikācijas metode
PER-001	Sistēma identificē definēto objekta klasi paredzētajā darbības zonā.	Marķēts validation dataset + reāls tests
PER-002	3D pozīcijas kļūda nepārsniedz projekta definēto robežu.	Mērījums pret references objektu
PER-003	End-to-end latency nepārsniedz projekta robežu.	Timestamp analīze
PER-004	Sensora vai inference kļūmes gadījumā perception tiek atzīmēts invalid.	Fault injection
PER-005	Katrs test rezultāts sasaistīts ar dataset/model/config versiju.	Traceability audits

100.4. Projekta fāzes

1. Definēt intended use un safety robežas.
2. Izveidot Functional Tree un interfeisu līgumus.
3. Sagatavot dataset un annotation guide.
4. Izvēlēties baseline modeli.
5. Izveidot training/validation pipeline.
6. Veikt error analysis un threshold tuning.
7. Integrēt ROS 2 node un diagnostics.
8. Pievienot 2D→3D ģeometriju.
9. Veikt edge deployment un latency testus.
10. Veikt fault injection un gala acceptance testu.
11. Sagatavot demonstrāciju un tehnisko aizstāvēšanu.

100.5. Vērtēšanas rubrika

Joma	Svars	Ko vērtē
Prasības un sistēmas arhitektūra	15%	Functional Tree, interfeisi, acceptance criteria
Dataset un reproducējamība	15%	Split, QA, versioning, leakage kontrole
Modelis un metrikas	20%	Baseline, validation, error analysis
ROS 2 / 3D integrācija	20%	Timestamp, TF, messages, geometry
Testēšana un drošums	20%	Latency, fault injection, evidence

Joma	Svars	Ko vērtē
Aizstāvēšana	10%	Spēja pamatot izvēles un ierobežojumus

101. v1.3 eksāmenu un aizstāvēšanas jautājumu banka

101.1. Deep Learning un CNN

1. Kas ir tensor un ar ko batch atšķiras no epoch?
2. Ko konvolūcijas kernel dara ar attēlu?
3. Kā stride un padding maina feature map izmēru?
4. Kāpēc CNN vajadzīga nelineāra aktivācija?
5. Kas ir receptive field?
6. Kāda ir atšķirība starp training un inference?
7. Kas ir overfitting un kā to atklāj?
8. Kāpēc training loss vien nepierāda labu modeli?

101.2. Dataset un transfer learning

1. Kas ir data leakage un kā tas var rasties no video kadriem?
2. Kāpēc test dataset nedrīkst izmantot hiperparametru izvēlei?
3. Kas ir annotation guide?
4. Kāda atšķirība starp fixed feature extractor un fine-tuning?
5. Kāpēc backbone fine-tuning bieži izmanto mazāku learning rate?
6. Kad pretrained modelis var slikti pārnēsties uz jaunu domēnu?

101.3. Detection un segmentation

1. Kas ir IoU?
2. Kā confidence threshold ietekmē precision un recall?
3. Kāpēc nepieciešams NMS?
4. Kāda atšķirība starp detection, semantic segmentation un instance segmentation?
5. Kā no instance maskas iegūt 3D objekta point cloud?
6. Kāpēc bounding box centra depth var būt neuzticams?

101.4. ROS 2 un 3D perception

1. Kāda loma ir CameraInfo?
2. Kāpēc detection jāsaģlabā sensora timestamp?
3. Kāda loma ir TF 3D perception sistēmā?
4. Ko dod vision_msgs?
5. Kāpēc AI node nedrīkst uzkrāt vecus kameras kadrus?
6. Kāda atšķirība starp Detection2DArray un Detection3DArray?
7. Kādi nosacījumi vajadzīgi korektai camera<->LiDAR fusion?

101.5. Aprēķinu uzdevumi

- Dots TP=80, FP=20, FN=10. Aprēķināt precision, recall un F1.
- Dots predicted box un ground truth box koordinātas. Aprēķināt IoU.
- Dots H_in=224, K=3, P=1, S=2. Aprēķināt H_out.
- Dots u,v,Z un f_x,f_y,c_x,c_y. Aprēķināt X,Y,Z kameras frame.

- Dots 30 Hz kamera un 100 ms inference. Izskaidrot backlog risku un piedāvāt frame-dropping stratēģiju.
- Dots inference p50=25 ms, p95=55 ms, p99=120 ms un vadības prasība 80 ms. Novērtēt sistēmu un pamatot acceptance.

101.6. Dzilje aizstāvēšanas jautājumi

- Kas tavā AI sistēmā ir izmērīts fakts un kas ir pieņēmums?
- Kā tu pierādītu, ka modelis darbojas ne tikai uz tava dataset, bet paredzētajā darbības vidē?
- Kura kļūda ir bīstamāka — false positive vai false negative — un kāpēc tieši tavā projektā?
- Ko robots dara, ja AI confidence ir zems?
- Kā tu noteiktu, vai kļūda rodas modelī, kalibrācijā vai TF?
- Kā tu pierādītu, ka 3D bbox nav tikai vizuāli “pareizā vietā”, bet metriskā precizitātē atbilst prasībai?
- Ko mainītu, ja deployment aparatūra spēj tikai pusi pašreizējā FPS?

102. Pasniedzēja pašmācības ceļš — Deep Learning un robotu redze

Pašmācības mērķis nav kļūt par pilna laika AI pētnieku. Mērķis ir spēt profesionāli izskaidrot perception pipeline, izvērtēt rezultātus un integrēt modeli robotikas sistēmā ar mērāmu kvalitāti.

102.1. 10 posmu ceļš

Posms	Praktiskais rezultāts
1. Tensor + NumPy/PyTorch	Spēj izsekot tensoru formām un normalizācijai
2. CNN	SmallCNN klasifikators + training curves
3. Dataset engineering	Dataset card + split + leakage audits
4. Transfer learning	Pretrained classifier fine-tuning
5. Detection	Threshold/IoU/precision-recall eksperimenti
6. Segmentation	Maska → objekta laukums/point cloud
7. Camera geometry	Calibration + 2D→3D projekcija
8. ROS 2 perception	Image → Detection2DArray + diagnostics
9. 3D fusion	Detection3DArray + TF
10. Deployment	ONNX/runtime + latency + fault injection

102.2. Pasniedzēja 5 minūšu pašpārbaude

- Izskaidrot CNN bez “AI maģijas”: lokāli filtri → feature maps → dziļāki konteksti → head.
- Izskaidrot, kāpēc data leakage var radīt maldinoši augstu test rezultātu.
- Izskaidrot precision/recall kompromisu ar reālu robotikas risku.
- Izskaidrot, kā 2D maska kļūst par 3D punktiem.
- Izskaidrot, kāpēc timestamp un TF ir daļa no AI sistēmas kvalitātes.
- Izskaidrot, ko darīt, ja modelis ir precīzs, bet pārāk lēns.

102.3. Personīgais pašmācības projekts

Ieteicamais projekts: RGB-D kamera vai ierakstīts dataset → pretrained object detector/segmenter → 2D metrics → depth fusion → 3D object centroid/bbox → ROS 2 Detection3DArray → RViz → rosbag evidence → ONNX deployment → controlled fault tests. Šāds projekts vienā ķēdē apvieno AI, computer vision, ģeometriju un ROS 2.

103. [ARHĪVS] Versijas 1.3 statuss un nākamais slānis

103.1. Kas pievienots v1.3

- Deep Learning sistēmu domāšana robotu redzei.
- CNN pamati, tensoru formas, convolution, receptive field un training/inference.
- Dataset engineering, anotāciju QA, leakage kontrole un traceability.
- Transfer learning un fine-tuning ar PyTorch principiem.
- Object detection, IoU, threshold, NMS un failure cases.
- Semantic/instance segmentation, pose/keypoints un tracking.
- AI metriku, latency, robustuma un fault scenāriju analīze.
- ONNX/OpenCV DNN un edge deployment principi.
- ROS 2 Image/CameraInfo, image_transport, vision_msgs un diagnostics integrācija.
- 2D→3D projekcija, RGB-D/LiDAR fusion un Detection3D.
- 9 praktiskās laboratorijas, integrēts AI perception projekts, eksāmenu banka un pasniedzēja pašmācības ceļš.

103.2. Nākamais plānotais slānis — v1.4

Nākamais slānis paredzēts robotu drošībai, riska analīzei un Digital Twin: hazard identification, risk reduction, safety functions, sensoru/aktuatoru atteices, verification/validation domāšana, droša cilvēka–robota sadarbība, kā arī digitālā dvīņa datu un modeļu sasaistīšana ar fizisku sistēmu.

103.3. Galvenā profesionālā doma

Deep Learning modelis robotā jāvērtē kā inženiertehniskas sistēmas komponents. Pareizs process ir: prasība → dataset → modelis → metrikas → kalibrācija/TF → deployment → integrācijas tests → fault test → pierādījums. Augsta accuracy bez šīs ķēdes nav pietiekams pamatojums autonomai darbībai.

Primārie tehniskie avoti un versiju piezīme — v1.3

Šīs v1.3 sadaļas tehniskie termini un API virziens 2026-08-18 pārbaudīts pret primāro PyTorch/TorchVision, OpenCV, ROS 2 vision_msgs/image_transport un Open3D dokumentāciju. Konkrēti modeļu nosaukumi un bibliotēku API laika gaitā mainās; mācību materiāls prioritizē stabilus algoritmiskos principus un paredz API pārbaudi pirms laboratorijas izpildes.

- https://docs.pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
- https://docs.pytorch.org/tutorials/intermediate/torchvision_tutorial.html
- https://docs.pytorch.org/vision/stable/auto_examples/index.html
- https://docs.opencv.org/5.0/tutorials/calib3d/camera_calibration/camera_calibration.html
- https://docs.opencv.org/doc/doxygen/html/d6/d0f/group__dnn.html
- https://docs.ros.org/en/ros2_packages/rolling/api/vision_msgs/
- https://docs.ros.org/en/rolling/p/vision_msgs/msg/Detection3D.html
- https://docs.ros.org/en/rolling/p/image_transport_plugins/
- https://www.open3d.org/docs/latest/tutorial/t_geometry/pointcloud.html

104. Robotu un mehatronisko sistēmu drošības inženierijas pamati

Drošības inženierija nav viena sensora, avārijas pogas vai programmas funkcija. Tā ir sistemātiska pieeja, kurā definē sistēmas robežas, identificē bīstamības, samazina risku ar prioritāru pasākumu hierarhiju un saglabā objektīvus pierādījumus, ka drošības prasības ir izpildītas.

104.1. Hazard, hazardous situation, hazardous event un risk

Jēdziens	Mācību skaidrojums	Piemērs robotikā
Hazard	Potenciāls kaitējuma avots	Robota kustīgā ass ar lielu enerģiju
Hazardous situation	Situācija, kur cilvēks/iekārta ir pakļauta hazardam	Operators ieiet robota darba zonā
Hazardous event	Notikums, kas var novest pie kaitējuma	Negaidīta robota kustības palaišana
Risk	Kaitējuma smaguma un notikuma iespējamības kombinācija	Sadursmes seku un iespējamības novērtējums

104.2. Safety, reliability un availability nav viens un tas pats

Iekārta var būt ļoti uzticama, bet nedroša; vai droša, bet bieži pāriet drošā apstāšanās stāvoklī un tāpēc tai ir zema pieejamība. Projektēšanā šie mērķi jānodala un jāmēra atsevišķi.

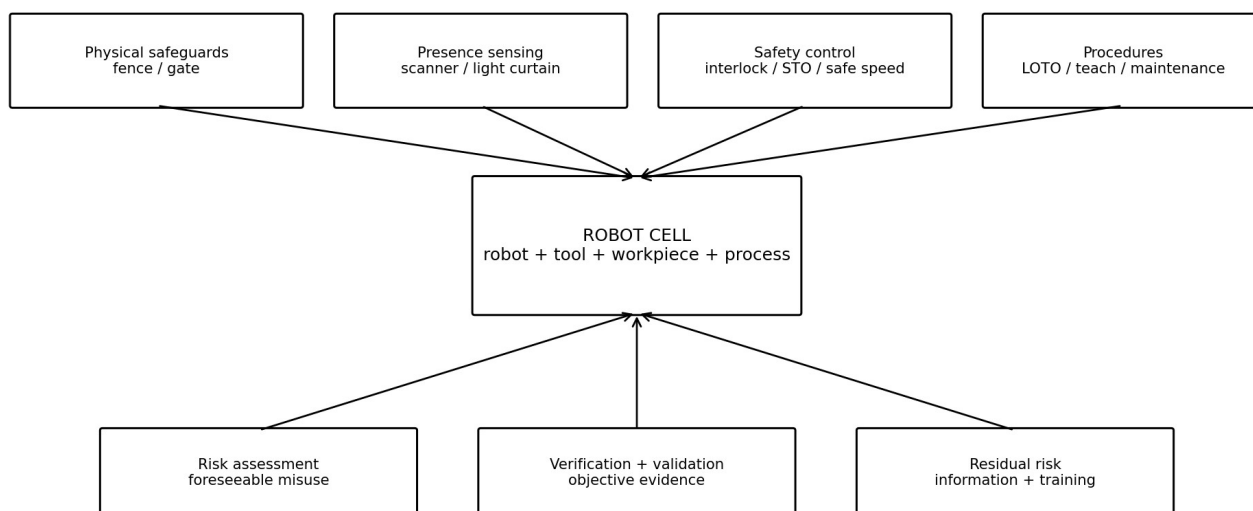
Īpašība	Jautājums
Safety	Vai kļūme var radīt nepieņemamu kaitējuma risku?
Reliability	Cik bieži sistēma izpilda funkciju bez atteices?
Availability	Cik lielu laiku daļu sistēma ir gatava darbam?
Security	Vai neatļauta/launprātīga darbība var ietekmēt sistēmu?

104.3. Safe state un fail-safe domāšana

Safe state nav universāli “power off”. Dažām sistēmām drošākais stāvoklis ir kontrolēta apstāšanās, enerģijas ierobežošana, bremzes aktivizēšana vai procesa uzturēšana noteiktā režīmā. Drošais stāvoklis jādefinē konkrētai hazardu analīzei.

Ko stāstīt auditorijai: Uzdod jautājumu: “Vai elektrības pazušana vienmēr ir droša?” Piemēri ar vertikālu asi, pneimatiku vai karstu procesu parāda, ka drošība ir sistēmas īpašība, nevis tikai OFF komanda.

Robotu šūnas drošība kā vairāku slāņu sistēma



104.1. attēls. Drošības aizsardzības slāņi robotu šūnā.

104.4. Safety by design un defense in depth

- Vispirms mēģina samazināt hazardu ar konstrukciju vai enerģiju.
- Tālāk izmanto aizsargus, detektēšanu un safety-related control functions.
- Procedūras, apmācība un PPE ir nepieciešamas, bet tās nav automātisks aizvietotājs tehniskai risku samazināšanai.
- Neviens viens slānis nedrīkst tikt uzskatīts par “magisku” risinājumu visiem hazardiem.

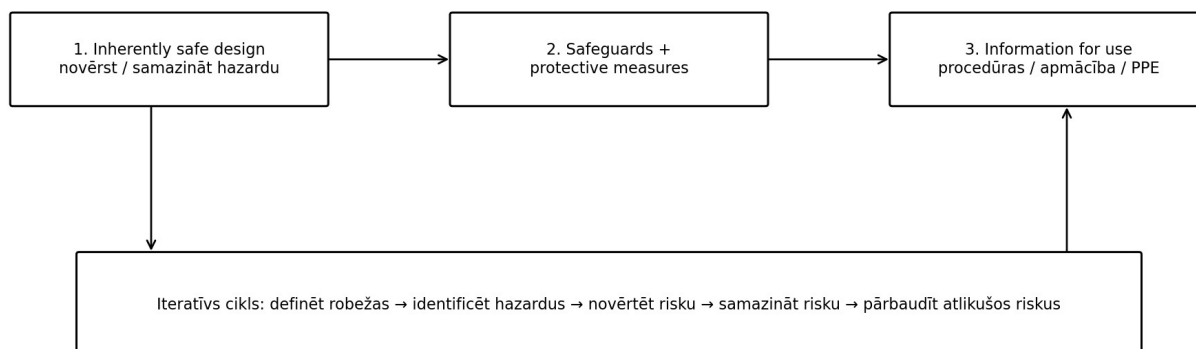
104.5. Standartu karte — mācību orientieris

Dokuments	Loma kursā	2026. gada statuss/piezīme
ISO 12100:2010	Vispārējā risku novērtēšanas un samazināšanas metodika	Joprojām publicēts; pārskatīšanas projekts ir izstrādē.
ISO 10218-1:2025	Industriālo robotu drošības prasības	2025. gada 3. izdevums.
ISO 10218-2:2025	Robotu lietojumu un šūnu integrācijas drošība	2025. gada 2. izdevums.
ISO 13849-1:2023	Safety-related control systems, PL pieeja	2023. gada 4. izdevums.
IEC 62061:2021 + amendments	Mašīnu funkcionālā drošība, SIL pieeja	2021 bāze; 2024 un 2026 grozījumi.
ISO/TS 15066:2016	Collaborative robotu papildu vadlīnijas	Joprojām publicēts; aizvietojošs darbs ir izstrādē.

Svarīgi: šī nodaļa ir izglītojošs pārskats, nevis standartu pilna satura aizvietotājs. Reālam projektam jāizmanto aktuālās oficiālās standartu versijas un piemērojamais normatīvais regulējums.

105. Risku novērtēšana un risku samazināšana

Risku samazināšanas hierarhija un iterācija (mācību shēma)



105.1. attēls. Risku samazināšanas prioritātes un iteratīvais cikls.

105.1. Sistēmas robežu definēšana

Pirms hazardu saraksta jādefinē, ko tieši vērtē: paredzētā lietošana, pamatoti paredzama nepareiza lietošana, darbības režīmi, cilvēki, enerģijas avoti, darba telpa, uzturēšana, tīrīšana, iestatīšana, transportēšana un ekspluatācijas beigas.

105.2. Hazardu kategorijas

- Mehāniskie: saspiešana, cirpšana, trieciens, iespiešana, krītoša masa.
- Elektriskie: tiešs/netiešs kontakts, īsslēgums, elektrostatika.
- Termiskie: karstas/aukstas virsmas un procesi.
- Vadības sistēmas: negaidīta palaišana, kļūdainais stāvoklis, sensora atteice, programmatūras kļūda.
- Ergonomiskie un cilvēkfaktori: neskaidras HMI komandas, slikta redzamība, nogurums.
- Procesa un materiālu riski: spiediens, ķīmiskās vielas, lāzers, metināšana u.c.

105.3. Vienkāršota riska matrica — tikai mācību vajadzībām

Lai trenētu domāšanu, kursā var izmantot vienkāršotu 4×4 matricu. Tā nav ISO 12100 noteikta universāla formula un nedrīkst aizvietot organizācijas apstiprinātu metodi vai piemērojama standarta procesu.

Iespējamība ↓ / Smagums →	1 — neliels	2 — vidējs	3 — nopietns	4 — katastrofisks
1 — reta	1	2	3	4
2 — iespējama	2	4	6	8
3 — ticama	3	6	9	12

Iespējamība ↓ / Smagums →	1 — neliels	2 — vidējs	3 — nopietns	4 — katastrofisks
4 — bieža	4	8	12	16

Formula / mācību modelis

Mācību score = Severity × Likelihood

Skaitlis palīdz salīdzināt scenārijus vienas mācību metodes ietvaros, bet pats par sevi nepierāda atbilstību vai drošību.

105.4. Risku samazināšanas trīs pakāpju princips

1. Inherently safe design measures — samazināt enerģiju, novērst saspiešanas vietu, mainīt ģeometriju vai procesu.
2. Safeguarding and complementary protective measures — guards, interlocks, presence sensing, safety-related control functions.
3. Information for use — brīdinājumi, instrukcijas, apmācība, darba organizācija un nepieciešamais PPE.

105.5. Residual risk

Pēc pasākumiem palikušais risks ir jānovērtē un jākomunicē. “Mums ir avārijas poga” nav pietiekams pierādījums, ka atlikušais risks ir pieņemams.

105.6. Risk register piemērs

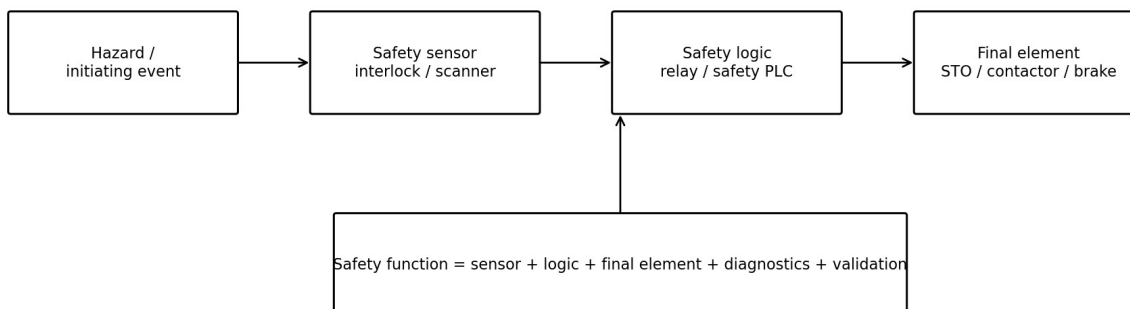
ID	Hazard/ scenārijs	Sākotnējais risks	Pasākums	Atlikušais risks	Pierādījums
R-01	Piekļuve kustīgam robotam automātiskā režīmā	12	Interlocked gate + safe stop	4	Gate test + safety function validation
R-02	Neparedzēta restartēšana pēc barošanas atjaunošanas	8	Restart interlock + state machine	2	Power-cycle test
R-03	Nepareizs Digital Twin stāvoklis tiek izmantots lēmumam	6	Freshness/ quality flag + reject stale data	2	Timestamp fault-injection test

Ko stāstīt auditorijai: Liec studentiem katram riskam piesaistīt pierādījumu. Ja kolonnā “Pierādījums” nav izmērāma testa vai dokumenta, tad pasākums vēl nav pilnībā verificēts.

106. Safety functions, PL/SIL un funkcionālā drošība

Safety function ķēde

Performance must be justified for the complete safety function, not one component alone.



106.1. attēls. Safety function pilnā ķēde.

106.1. Kas ir safety function?

Safety function ir funkcija, kuras atteice var palielināt risku. Piemēri: aizsargvārtu atvēršana izraisa drošu apstāšanos; scanner zonā konstatēts cilvēks samazina/aptur kustību; emergency stop komandai jāizraisa definēta reakcija.

106.2. Safety-related control system kā ķēde

Posms	Piemēri	Galvenais jautājums
Input	Safety switch, light curtain, scanner	Vai bīstamais stāvoklis tiek droši konstatēts?
Logic	Safety relay, safety PLC	Vai lēmumu loģika un diagnostika sasniedz prasīto integritāti?
Output	STO, safety contactors, brake control	Vai bīstamā enerģija tiek droši kontrolēta?

106.3. PL un SIL — konceptuāls salīdzinājums

ISO 13849-1 izmanto Performance Level (PL) pieeju, bet IEC 62061 izmanto Safety Integrity Level (SIL) pieeju mašīnu safety-related control systems. Šajā kursā mērķis nav mehāniski “pārvērst” PL par SIL, bet saprast kopīgo principu: safety function integritāte jāprojektē, jāaprēķina/argumentē un jāvalidē kā pilna ķēde.

Aspekts	PL pieeja	SIL pieeja
Tipiska atsauce	ISO 13849-1	IEC 62061

Aspekts	PL pieeja	SIL pieeja
Fokuss	SRP/CS arhitektūra, diagnostika, uzticamības parametri	Safety-related control system un SIL prasības
Rezultāts	PL a...e	SIL 1...3 mašīnu kontekstā
Kopīgais	Safety function prasība + arhitektūra + diagnostika + validācija	Safety function prasība + arhitektūra + diagnostika + validācija

106.4. PLr/SIL noteikšana

Nepieciešamo integritātes līmeni nosaka risku novērtējuma kontekstā. Kursā drīkst demonstrēt risk graph principu, bet konkrēta reāla projekta PLr vai SIL izvēle jāveic pēc aktuālā piemērojamā standarta, nevis pēc vienas no mācību tabulām.

106.5. Common cause, diagnostic coverage un proof/periodic testing

- Redundance nepalīdz, ja abus kanālus var vienlaikus sabojāt viens kopīgs cēlonis.
- Diagnostic coverage raksturo, cik efektīvi sistēma atklāj noteiktas bīstamas atteices.
- Safety functions jātestē ne tikai FAT/SAT brīdī; periodiskās pārbaudes var būt daļa no safety lifecycle.
- Konfigurācijas un programmatūras izmaiņas var invalidēt iepriekšējo safety validation.

106.6. Safety requirement piemērs

Piemērs — safety requirement (nevis universāla normatīva vērtība)

SF-01: When the interlocked access gate is opened during automatic operation, hazardous robot motion shall be brought to the defined safe state.

Verification: functional test + diagnostic fault injection.

Acceptance: response and final state meet the project safety requirement and applicable standard.

Ko stāstīt auditorijai: Atgādinī: “safety component” marķējums pats par sevi nepierāda visas safety function integritāti. Vienmēr vērtē sensoru, loģiku, output elementu, diagnostiku, kopējos cēloņus un validāciju kopā.

107. Industriālo robotu šūnu drošība — ISO 10218 konteksts

2025. gadā tika publicētas jaunas ISO 10218-1 un ISO 10218-2 versijas. Pirmā daļa koncentrējas uz industriālo robotu kā produktu, otrā — uz industriālo robotu application/cell integrāciju visā dzīves ciklā. Kursā šīs atsauces izmanto kā orientieri, nevis reproducē standartu tekstu.

107.1. Robots nav tas pats, kas robot application

Līmenis	Kas tajā ietilpst	Drošības jautājums
Industrial robot	Manipulatora mehānika, drives, robot controller	Vai robotam kā produktam ir vajadzīgās drošības iespējas?
Robot application	Robots + end-effector + workpiece + process	Kādi papildu hazardi rodas konkrētajā uzdevumā?
Robot cell/system	Application + guards + conveyors + operators + external machines	Vai visa integrētā sistēma ir droša visos režīmos?

107.2. Dzīves cikla režīmi

- Design / integration
- Commissioning
- Automatic production
- Teach / manual intervention
- Cleaning and maintenance
- Fault recovery
- Decommissioning

Hazardi un piekļuves noteikumi var būt atšķirīgi katrā režīmā. Īpaši bīstama ir “fault recovery” situācija, kad cilvēks mēģina ātri atjaunot ražošanu un var apiet standarta procesu.

107.3. Safeguarding un piekļuve

Aizsargžogs, interlocked gate, presence sensing device un safety-rated motion functions nav savstarpēji aizvietojami pēc noklusējuma. To kombinācija jāizvēlas pēc konkrētās risku analīzes, cilvēka piekļuves scenārijiem un stopping performance.

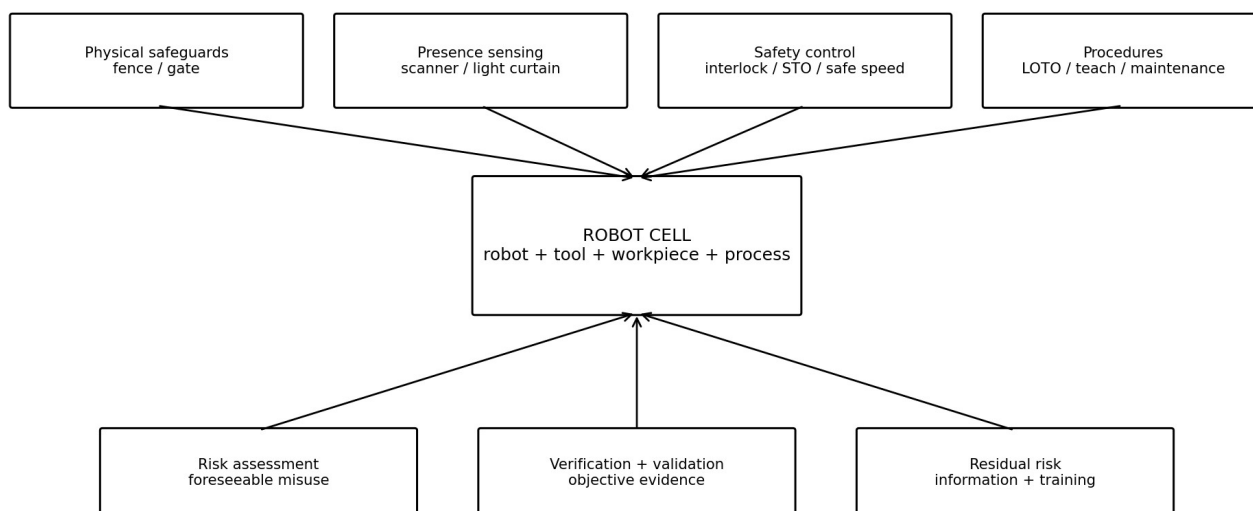
107.4. Stopping performance

Aizsardzības distances vai zonu definēšanai ir svarīga sistēmas faktiskā apstāšanās dinamika: sensora reakcija, loģikas reakcija, drive reakcija, bremzēšana un mehāniskā kustība. Mācību projektā šo var izmērīt un dokumentēt, bet rūpnieciskā validācijā jāievēro piemērojamie standarti un mērīšanas prasības.

107.5. End-effector un process hazard

Robotam drošības funkcijas var būt korekti realizētas, bet process joprojām var būt bīstams: vakuuma satvērējs var nomest detaļu; gripper var saspīest; welding torch rada karstumu un starojumu; spindle saglabā rotācijas enerģiju. Tāpēc risk assessment jāaptver pilna application.

Robotu šūnas drošība kā vairāku slāņu sistēma



107.1. attēls. Robot application drošības slāņi.

107.6. Robot cell acceptance checklist — mācību variants

Pārbaude	Jā/Nē	Pierādījums
Operating modes definēti		
Access points identificēti		
Safety functions sasaistītas ar hazards		
Restart behaviour pārbaudīts		
Loss of power/communication scenāriji testēti		
Tool/workpiece hazards novērtēti		
Safety function validation pabeigta		
Residual risks dokumentēti		

108. Collaborative robotika un cilvēka–robotu mijiedarbības drošība

Collaborative robotika nozīmē konkrētu robotu application, kur cilvēks un robotu sistēma noteiktos apstākļos var koplietot darba telpu. “Cobot” zīmols pats par sevi nenozīmē, ka jebkurš uzdevums automātiski ir drošs.

108.1. Collaborative application principi

Principi	Mācību skaidrojums	Piemērs
Safety-rated monitored stop	Robots atrodas drošā kontrolētā stāvoklī, kamēr cilvēks veic definētu darbību	Detaļas ievietošana shared workspace
Hand guiding	Cilvēks vada robota kustību ar paredzētu vadības interfeisu	Teach/ergonomic positioning
Speed and separation monitoring	Kustība tiek kontrolēta atbilstoši cilvēka attālumam un aizsardzības koncepcijai	Scanner-based shared workspace
Power and force limiting	Mehāniskā/elektriskā sistēma ierobežo kontakta enerģiju/spēkus konkrētā application	Zema enerģijas pick-and-place

108.2. Kāpēc nav universāla “droša cobot ātruma”?

Pieļaujamā kustība ir atkarīga no robota masas/inercijas, payload, tool ģeometrijas, kontakta zonas, saspiešanas iespējas, cilvēka ķermeņa daļas, procesa un safeguarding arhitektūras. Tāpēc kursā netiek dots viens universāls drošs ātrums vai spēks.

108.3. ISO/TS 15066 statusa piezīme

ISO/TS 15066:2016 joprojām ir publicēta un papildina collaborative industrial robot sistēmu drošības vadlīnijas. 2026. gadā ISO ir uzsācis darbu pie aizvietojošās ISO 15066-1 sērijas; tādēļ pasniedzējam pirms konkrētu normatīvu vērtību mācīšanas vienmēr jāpārbauda aktuālais statuss.

108.4. Collaborative workspace risk questions

- Vai cilvēku var iespiest starp robotu un fiksētu objektu?
- Vai tool ir asas malas vai satveršanas hazard?
- Kas notiek, ja payload tiek pazaudēts?
- Vai sensors redz visu nepieciešamo zonu?
- Kas notiek, ja tracking/sensor data kļūst stale?
- Kā tiek kontrolēta atgriešanās no safety stop uz automatic operation?

108.5. Praktisks scenārijs

Students analizē robotu, kas padod detaļu operatoram. Jāizveido hazardu saraksts, jāizvēlas collaborative koncepcija, jādefinē safety functions un jāuzraksta vismaz pieci acceptance tests. Uzdevuma mērķis nav “pierādīt sertifikāciju”, bet iemācīties sistemātisku argumentāciju.

109. Drošības funkciju arhitektūras — E-stop, interlock, STO, safe motion

109.1. Emergency stop

Emergency stop ir papildinoša aizsardzības funkcija ārkārtas situācijai. Tā nav ikdienas process stop poga un nav automātisks aizvietotājs guard/interlock risinājumam. ISO 13850 definē emergency stop funkcijas vispārējos projektēšanas principus.

109.2. Guard interlock

Interlock sasaista aizsarga stāvokli ar bīstamas funkcijas atļauju. Drošības projektā jādomā par bypass/tampering, restartēšanu, diagnostiku, fault masking un to, vai tikai stop funkcija ir pietiekama konkrētajam hazardam.

109.3. STO un citi safe motion jēdzieni

Funkcija/jēdziens	Intuitīvs skaidrojums	Ko nepārprast
STO — Safe Torque Off	Novērš drive spēju ģenerēt motora griezes momentu atbilstoši funkcijas dizainam	Tas ne vienmēr mehāniski aptur jau kustīgu/krietošu asi.
Safe stop	Definē drošu apstāšanās reakciju	Precīza realizācija ir drive/sistēmas specifiska.
Safe speed monitoring	Uzrauga, ka kustības ātrums nepārsniedz drošības prasību	Parasts encoder speed value nav automātiski safety-rated.
Safe position/zone	Uzrauga pozīciju vai darba zonu	Nepieciešama droša mērīšanas un loģikas ķēde.

109.4. Restart prevention

Pēc safety device atjaunošanas sistēma nedrīkst automātiski radīt negaidītu bīstamu kustību, ja risku novērtējums prasa apzinātu restartu. Restart loģika ir atsevišķa projektēšanas tēma, nevis tikai RESET pogas pievienošana.

109.5. Safety communication

Safety fieldbus/protocol ļauj pārraidīt safety-related informāciju pa kopīgu industriālo komunikācijas infrastruktūru, bet safety integrity rodas no sertificēta/projektēta end-to-end mehānisma, nevis no tā, ka parasts Ethernet rāmis ir “ātrs”.

109.6. State machine piemērs

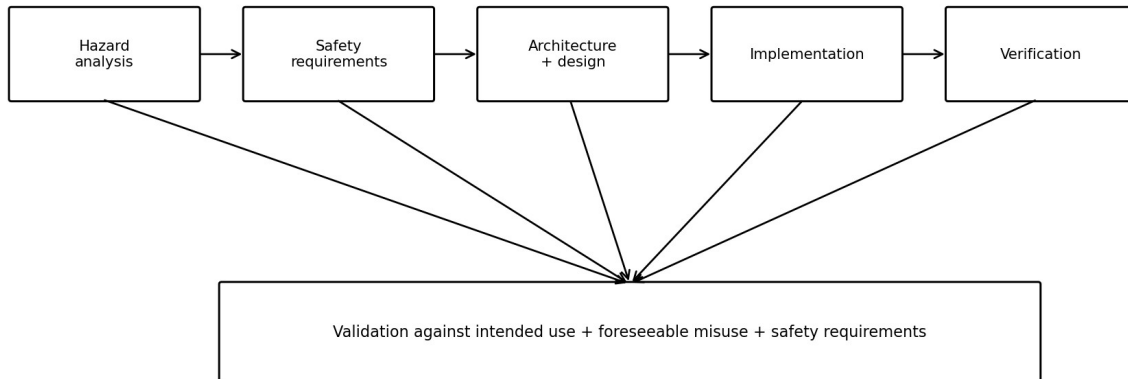
Mācību safety state machine

```
SAFE_STOPPED → RESET_PERMITTED → READY → AUTO_RUNNING
AUTO_RUNNING + gate_open → SAFE_STOPPING → SAFE_STOPPED
Any detected safety fault → FAULT_LOCKED
FAULT_LOCKED requires diagnosed fault clearance + defined reset procedure
```


Ko stāstīt auditorijai: Svarīgi atšķirt “software state machine, kas imitē safety” no īstas safety-rated funkcijas. Mācību kods māca loģiku, bet reālā mašīnā drošības funkcijām vajag atbilstošu aparatūru, arhitektūru un validāciju.

110. Verification, validation, traceability un safety case

Safety lifecycle un traceability



Traceability: hazard → safety function → requirement → implementation → test evidence → residual risk

110.1. attēls. Safety lifecycle un izsekojamība.

110.1. Verification vs validation

Termins	Jautājums	Piemērs
Verification	Vai mēs realizējam prasību pareizi?	Code review, wiring check, unit test
Validation	Vai safety function reālajā lietojumā samazina risku kā paredzēts?	Robot cell functional safety test ar reālu stopping behaviour

110.2. Traceability matrix

Hazard ID	Safety req.	Design element	Test ID	Result	Evidence
H-01	SF-01	Gate switch + safety logic + STO	T-SF-01	PASS	Test protocol + config hash
H-02	SF-02	Restart interlock	T-SF-02	PASS	Power-cycle video + log
H-03	DT-01	Twin freshness monitor	T-DT-01	PASS	Fault-injection CSV

110.3. Fault injection

Drošības un Digital Twin sistēmās “happy path” nav pietiekams. Jātestē sensora disconnect, stale timestamp, stuck-at value, komunikācijas zudums, power cycle, config mismatch un neatļauta restartēšana. Fault injection jāplāno tā, lai pats tests neradītu nepieņemamu risku.

110.4. Safety case

Safety case ir strukturēts arguments ar pierādījumiem, ka konkrētā sistēma noteiktā kontekstā sasniedz definētu drošības mērķi. Kursā var izmantot vienkāršotu formu: Claim → Argument → Evidence.

Vienkāršots safety case fragments

CLAIM: Access gate opening prevents hazardous automatic robot motion.

ARGUMENT: Hazard H-01 is controlled by safety function SF-01 with diagnosed input, safety logic and final element.

EVIDENCE: wiring review + configuration record + fault injection + stopping test + validation sign-off.

110.5. Configuration evidence

- Robot program version
- Safety PLC program checksum/version
- Drive safety parameter set
- Sensor firmware/configuration
- Risk assessment revision
- Test protocol revision
- Date, tester and calibrated instrument IDs where applicable

111. Kiberdrošības un funkcionālās drošības mijiedarbība

Safety un security ir atšķirīgas disciplīnas, bet mūsdienu savienotās sistēmās tās mijiedarbojas. Drošības funkcija var būt tehniski pareiza, bet tās konfigurācijas, tīkla vai autentifikācijas kompromitēšana var mainīt risku.

111.1. Security threat kļūst par safety concern

Security scenārijs	Iespējamā safety ietekme	Kontroles piemērs
Neatļauta safety parametru maiņa	Mainīta safety zone/speed limit	Access control + change management + independent validation
DoS uz perception tīklu	Autonoms robots zaudē uztveres datus	Timeout + degraded mode + safe stop policy
Spoofed Digital Twin telemetry	Operators/analytics pieņem nepareizu stāvokli	Authentication + provenance + freshness/quality checks
Firmware supply-chain compromise	Nedroša actuator behaviour	Signed updates + controlled deployment + rollback plan

111.2. Safety function nevajag balstīt tikai uz nekontrolētu cloud savienojumu

Latentums, pieejamība, autentifikācija un neatkarības prasības nozīmē, ka safety-critical reakcijām parasti vajag lokālu, determinētāku drošības arhitektūru. Cloud/Digital Twin var atbalstīt monitoringu, analīzi un maintenance, bet write-back ceļš jāprojektē ar stingrām robežām.

111.3. Defense in depth

- Network segmentation
- Least privilege
- Identity/authentication
- Secure configuration
- Signed software/firmware where supported
- Audit logs
- Backup/restore
- Safety-independent protective layer where risk requires

111.4. Change control

Programmatūras update, model update vai Digital Twin mapping izmaiņa var ietekmēt safety argumentu. Tāpēc jādefinē, kādas izmaiņas prasa atkārtotu risk assessment, verification vai validation.

112. ES mašīnu regulējuma un standartu versiju pārvaldība

112.1. Normatīvais konteksts 2026. gada augustā

ES Machinery Regulation (EU) 2023/1230 ir spēkā, bet lielākā daļa regulas prasību sāks piemēroties 2027. gada 20. janvārī saskaņā ar aktuālo EUR-Lex konsolidēto kopsavilkumu. Līdz pārejas datumam mašīnu atbilstības projektos jāpārbauda konkrētā produkta un laišanas tirgū laika konteksts, nevis automātiski jāpieņem viena režīma piemērojamība.

112.2. Standarts nav likums

Harmonizēti standarti var palīdzēt demonstrēt atbilstību noteiktām prasībām, taču standarta izmantošana, piemērojamība un juridiskā loma jāvērtē konkrētās jurisdikcijas un regulējuma kontekstā. Kursā nedrīkst jaukt “atbilst ISO standartam” ar “automātiski juridiski atbilstošs visām prasībām”.

112.3. Version check discipline

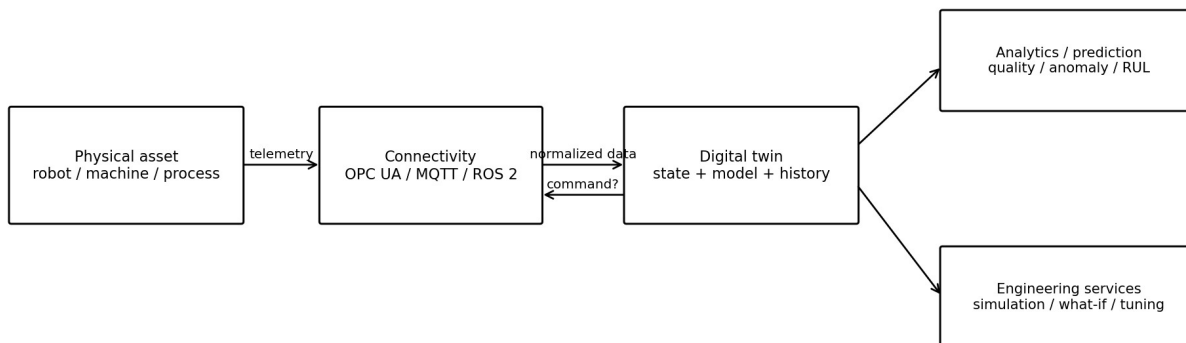
Ko pārbaudīt	Piemērs
Standarta numurs + izdevums	ISO 10218-1:2025, nevis automātiski 2011
Statuss	Published / withdrawn / under revision
Applicable product scope	Industrial robot vs service robot vs mobile robot
National/EU adoption/harmonization	Pārbaudīt aktuālo oficiālo sarakstu konkrētā projekta datumā
Project freeze date	Dokumentēt, ar kādu standartu versiju projekts validēts

112.4. Mācību dokumenta brīdinājums

Šī rokasgrāmata māca drošības inženierijas principus. Tā nav CE conformity assessment instrukcija un neaizvieto kompetentu risk assessment, sertifikācijas/atbilstības ekspertu vai aktuālo standartu pilno tekstu.

113. Digital Twin pamati un ISO 23247 ietvars

Digital Twin — fiziskais aktīvs, dati, modelis un pakalpojumi



Control write-back is optional and requires independent safety/security governance.

113.1. attēls. Digital Twin arhitektūras mācību modelis.

113.1. Kas ir Digital Twin?

Digital Twin kursā definējam kā pārvaldītu digitālu reprezentāciju, kas ir sasaistīta ar konkrētu fizisku vai loģisku aktīvu, satur tā stāvokli/modeli un tiek izmantota noteiktiem pakalpojumiem — monitoringam, analīzei, simulācijai, optimizācijai vai dzīves cikla izsekojamībai.

113.2. Digital model, digital shadow un digital twin — pedagoģisks nošķīrums

Līmenis	Datu saikne	Piemērs
Digital model	Nav obligātas automātiskas datu sinhronizācijas	CAD/kinemātikas modelis
Digital shadow	Fiziskais aktīvs automātiski atjaunina digitālo stāvokli	Telemetry dashboard
Digital twin	Pārvaldīta sasaite + modelis + pakalpojumi; write-back var būt, bet nav obligāts mācību definīcijā	Robot twin ar stāvokli, model-based residual un maintenance analytics

Šis trīs līmeņu iedalījums ir pedagoģisks palīgīdzeklis. ISO 23247 jāizmanto kā normatīvāks ražošanas Digital Twin ietvars, nevis jāpieņem, ka visi avoti terminus lieto identiski.

113.3. ISO 23247 sērija

Daļa	Fokuss
ISO 23247-1:2021	Overview un general principles
ISO 23247-2:2021	Reference architecture

Daļa	Fokuss
ISO 23247-3:2021	Digital representation of manufacturing elements
ISO 23247-4:2021	Information exchange
ISO 23247-5:2026	Digital thread for digital twin
ISO 23247-6:2026	Digital twin composition

113.4. Observable Manufacturing Element (OME) domāšana

Ražošanas Digital Twin sākas ar skaidru atbildi, ko tieši reprezentējam: robotu, instrumentu, conveyor, telpu, sensoru, produktu, procesu vai to kombināciju. Katram objektam vajag stabilu identitāti, atribūtus, stāvokli, saites un datu kvalitātes/provenance informāciju.

113.5. Digital Twin nav tikai 3D modelis

- 3D ģeometrija var būt tikai viena daļa.
- Twin var būt bez 3D vizualizācijas, ja tā funkcija ir datu analīze vai stāvokļa novērtēšana.
- Svarīga ir sasaite ar konkrētu aktīvu, datu nozīme, modeļa validācija un lietošanas mērķis.

114. Asset Administration Shell un standartizēta aktīva digitālā reprezentācija

IEC 63278-1:2023 definē Asset Administration Shell (AAS) struktūru kā standartizētu digitālu aktīva reprezentāciju industriālām lietojumprogrammām. AAS mērķis ir dot vienotu piekļuvi informācijai un pakalpojumiem visā aktīva dzīves ciklā.

114.1. Asset + AAS

Vienkāršots AAS mācību piemērs

Physical / digital / intangible ASSET
 📌 stable identification
ASSET ADMINISTRATION SHELL
 ├─ submodel: Nameplate
 ├─ submodel: TechnicalData
 ├─ submodel: ConditionMonitoring
 ├─ submodel: Documentation
 └─ submodel: Maintenance

114.2. Kāpēc AAS ir interesants robotikā?

- Vienoti metadati dažādu ražotāju iekārtām.
- Stabila identifikācija un semantika.
- Submodel pieeja dažādiem dzīves cikla aspektiem.
- Iespēja sasaistīt tehniskos datus, dokumentāciju, condition monitoring un maintenance informāciju.

114.3. AAS, OPC UA, MQTT un ROS 2 nav viens un tas pats

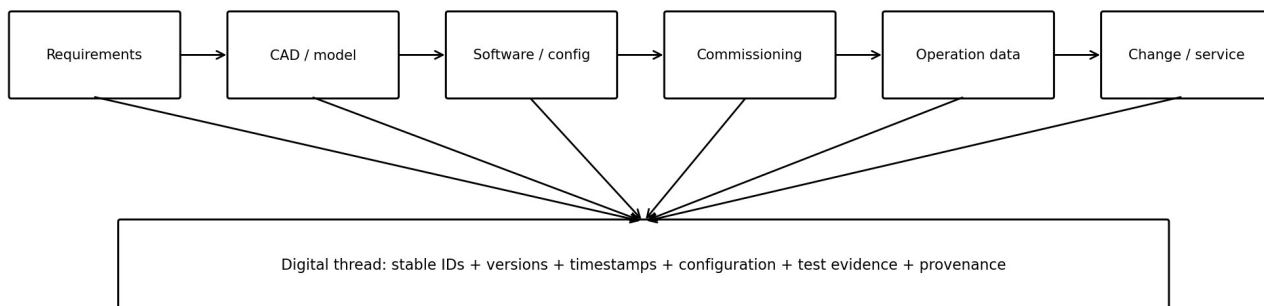
Tehnoloģija	Galvenā loma
AAS	Aktīva standartizēta digitālā reprezentācija un semantika
OPC UA	Industriāla informācijas modelēšana un komunikācija
MQTT	Viegls publish/subscribe ziņapmaiņas protokols
ROS 2	Robotikas programmatūras middleware, message/service/action ekosistēma

114.4. Mapping piemērs

Robot data	ROS 2	Digital Twin/AAS slānis
Joint positions	/joint_states	Operational state / live data reference
Serial number	parameter/database	Nameplate submodel
Maintenance hours	diagnostic topic	Condition/Maintenance submodel
URDF/Xacro version	repository artifact	Technical/documentation reference

115. Digital Thread, konfigurācijas pārvaldība un datu izsekojamība

Digital Thread — izsekojama informācijas ķēde visā dzīves ciklā



115.1. attēls. Digital Thread dzīves cikla informācijas ķēde.

115.1. Digital thread mērķis

Digital thread saista prasības, modeļus, konfigurācijas, programmatūru, testus, ražošanas/ekspluatācijas datus un izmaiņas tā, lai varētu atbildēt: “Kura konfigurācija radīja šo rezultātu?” un “Kāds pierādījums apstiprina šo prasību?”

115.2. Minimālie identifikatori

- asset_id
- configuration_id
- software commit/hash
- model version
- calibration version
- timestamp + time source
- test protocol ID
- operator/tester identity where required
- data quality / validity flag

115.3. Provenance

Provenance ir informācija par datu izcelsmi: kurš sensors, kura programmatūras versija, kāda kalibrācija, kāds laiks, kādas transformācijas. Bez provenance Digital Twin var būt vizuāli iespaidīgs, bet inženiertehniski vājš.

115.4. Data contract piemērs

Digital Twin telemetry ieraksta piemērs

```
{  
  "asset_id": "robot_02",  
  "timestamp_utc": "2026-08-18T11:00:00Z",  
  "config_id": "cfg_17",  
  "source": "joint_state_bridge",  
  "quality": "VALID",  
  "joint_1_rad": 0.512,  
  "model_version": "dt_model_3.2"  
}
```

115.5. Stale data un freshness

Formula / mācību modelis

$\text{age} = \text{current\_time} - \text{source\_timestamp}$

Ja age pārsniedz konkrētajam signālam pieļaujamo robežu, datiem jāmaina quality/validity statuss. Digital Twin nedrīkst klusējot rādīt pēdējo vērtību kā “aktuālu”, ja savienojums ir pārtrūcis.

116. Digital Twin modeļa ticamība, kalibrācija un validācija

116.1. Model fidelity

Modeļa detalizācijas līmenim jāatbilst lietošanas mērķim. Vizualizācijai pietiek ar ģeometriju, kinemātikas analīzei vajag joint/link parametrus, enerģijas vai dinamiskas vadības analīzei — inerces, friction un drive modeli.

116.2. Calibration, verification, validation

Darbība	Jautājums	Piemērs
Calibration	Kādi parametri jāpielāgo, lai modelis atbilstu mērījumiem?	Wheel radius, friction, sensor bias
Verification	Vai modeļa/programmas vienādojumi ir realizēti pareizi?	Unit test pret analītisku 2DOF kinemātiku
Validation	Vai modelis ir pietiekami labs paredzētajam lietojumam?	Trajectory error ir zem projekta pieļaujamās robežas

116.3. Residuals

Formula / mērījumu modelis

$$\text{residual}(t) = \text{measured}(t) - \text{predicted}(t)$$

Residuals ļauj salīdzināt fizisko sistēmu ar modeļa prognozi. Sistemātiska nobīde var nozīmēt sliktu kalibrāciju; pēkšņa residual izmaiņa — sensoru, mehānikas vai procesa stāvokļa maiņu.

116.4. RMSE

Formula / mērījumu modelis

$$\text{RMSE} = \sqrt{(1/N) \cdot \sum (\text{measured}_i - \text{predicted}_i)^2}$$

RMSE ir ērta kopsavilkuma metrika, bet viena pati var paslēpt transient kļūdas. Tāpēc validācijā jāskatās arī max error, laika grafiks un konkrēti failure scenāriji.

116.5. Uncertainty

Digital Twin nevajadzētu attēlot prognozi kā absolūtu patiesību. Model uncertainty var rasties no sensor noise, parametru neprecizitātes, nezināmas slodzes, temperatūras, nolietojuma vai vienkāršota fizikas modeļa.

116.6. Python validācijas piemērs

Modeļa residual un RMSE aprēķins

```
import numpy as np

measured = np.array([1.00, 1.08, 1.19, 1.28])
predicted = np.array([1.01, 1.10, 1.17, 1.31])
residual = measured - predicted
rmse = np.sqrt(np.mean(residual**2))
print("RMSE:", rmse)
print("max abs error:", np.max(np.abs(residual)))
```

117. Digital Twin mehatronikā, robotikā un autonomās sistēmās

117.1. Robot arm Digital Twin

- URDF/Xacro/kinematic model
- joint states
- controller state
- tool/payload metadata
- collision geometry
- calibration
- maintenance counters
- trajectory/test evidence

117.2. Mobile robot Digital Twin

- pose / odometry / localization covariance
- battery
- wheel encoder health
- IMU/LiDAR status
- map version
- navigation configuration
- mission state
- fault/recovery history

117.3. Predictive maintenance

Twin var kombinēt operation hours, motor current, vibration, temperature un maintenance history, lai izveidotu condition indicators. Predictive maintenance modelis ir jāvalidē atsevišķi; “AI anomaly” nav automātiski pierādīta komponentes atteice.

117.4. What-if simulation

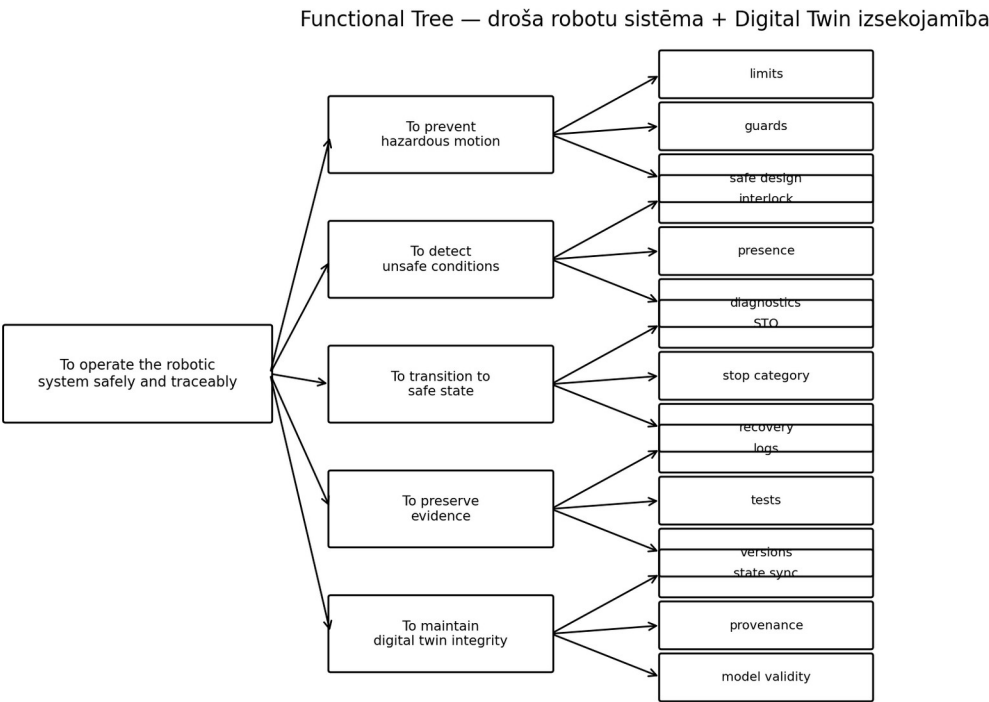
Digital Twin ļauj droši pārbaudīt konfigurācijas idejas simulācijā: citu payload, trajectory, cycle time, controller gain vai sensor placement. Taču pirms write-back uz fizisku iekārtu jāievēro change control un safety validation robežas.

117.5. Twin-in-the-loop un Hardware-in-the-loop

Pieeja	Ko savieno	Mērķis
Software-in-the-loop	Simulēts modelis + reāla programmatūras loģika	Agrīna algoritma verifikācija
Hardware-in-the-loop	Reāls controller/IO + simulēta plant	Vadības un IO integrācijas testi
Twin-in-the-loop	Dzīvā aktīva dati + modelis + analytics	Monitoring, estimation, anomaly detection, optimization support

117.6. Safety boundary

Digital Twin var būt spēcīgs diagnostikas un decision-support slānis, bet safety function atkarība no twin jāprojektē apzināti. Stale data, modeļa kļūda vai tīkla zudums nedrīkst klusējot pārvērsties bīstamā vadības komandā.



117.1. attēls. Functional Tree — drošība un Digital Twin izsekojamība.

118. v1.4 praktiskās Safety/Digital Twin laboratorijas

SAFE1 — Hazard identification un risk register

Mērķis: izveidot sistemātisku risku reģistru vienkāršai robotu šūnai

Ko stāstīt auditorijai: Mērķis nav iegūt “pareizo ciparu”, bet pierādāmu saiti starp hazardu, pasākumu un testu.

Iesniedzamie pierādījumi

- Risk register
- System boundary diagram
- 3 prioritāro risku argumentācija

Darba secība

1. Definē system boundary un operating modes.
2. Identificē vismaz 10 hazard/scenarios.
3. Piešķir mācību severity/likelihood un prioritāti.
4. Izvēlies risk reduction pasākumus trīs pakāpju hierarhijā.
5. Katram pasākumam pievieno verification/validation evidence.
6. Pārskati residual risk un noformē risk register revision.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
10 hazards identified	≥10 scenāriji ar mode/reasonable misuse			
Mitigation traceability	Katram high/medium risk ir pasākums + evidence			
Residual risk	Nav tukšu “solved” ierakstu bez pamatojuma			

A/B/C paplašinājums

- A: 6 hazard scenarios.
- B: 10+ hazards + residual risk.
- C: pievienot failure modes, common-cause un change-control analīzi.

SAFE2 — Safety function traceability matrix

Mērķis: izveidot safety function prasības un testu izsekojamību

Ko stāstīt auditorijai: Drošības prasībai jābūt testējamai; izvairies no vārdiem “droši” bez mērījama gala stāvokļa.

Iesniedzamie pierādījumi

- Safety requirements

- Traceability matrix
- Test protocol

Darba secība

1. Izvēlies trīs hazardus no SAFE1.
2. Katram definē safety function ID un trigger.
3. Definē safe state un reset/restart nosacījumu.
4. Sasaisti input–logic–output elementus.
5. Izveido vismaz divus fault injection testus katrai funkcijai.
6. Aizpildi traceability matrix.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Requirement testable	Visām SF ir trigger + response + safe state			
Fault injection	≥ 2 fault testi katrai safety function			
Traceability	Hazard→SF→design→test saite pilna			

A/B/C paplašinājums

- A: viena safety function.
- B: trīs funkcijas ar fault injection.
- C: pievienot independence/common-cause argumentu un configuration baseline.

SAFE3 — Restart, stale data un communication loss simulācija

Mērķis: demonstrēt kļūmju apstrādi bez fiziski bīstama testa

Ko stāstīt auditorijai: Safety loģikas mācību simulācija nav safety-rated controller, taču tā ir laba vieta, kur saprast restart un degraded mode principu.

Iesniedzamie pierādījumi

- State diagram
- Source code
- Fault event log

Darba secība

1. Izveido Python vai C++ state machine ar READY/RUNNING/SAFE_STOPPED/FAULT.
2. Simulē gate_open vai obstacle_detected.
3. Simulē communication timeout.
4. Simulē power-cycle/restart.

5. Pārbaudi, ka pēc fault clearance sistēma neatsāk RUNNING bez definēta reset nosacījuma.
6. Saglabā event log ar timestamp un reason.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Gate event	RUNNING→SAFE_STOPPED			
Comm timeout	Sistēma pāriet definētā degraded/safe state			
Restart	Nav automātiskas bīstamas kustības			

A/B/C paplašinājums

- A: viena fault pāreja.
- B: visas pārejas + logs.
- C: property-based tests vai model checking ievads.

DT1 — Digital Twin telemetry contract

Mērķis: izveidot izsekojamu un kvalitātes karodziņiem bagātu telemetry shēmu

Ko stāstīt auditorijai: Digital Twin sākas ar datu nozīmi, nevis dashboard. Katram laukam jābūt identificējamam, laicīgam un interpretējamam.

Iesniedzamie pierādījumi

- Data contract
- Sample dataset
- Validation script

Darba secība

1. Izvēlies vienu aktīvu (robots, motors vai sensors).
2. Definē asset_id, timestamp, config_id, source un quality.
3. Pievieno vismaz 5 procesa/robota mainīgos ar unit.
4. Izveido JSON schema vai tabulētu data contract.
5. Simulē VALID, STALE un INVALID statusus.
6. Saglabā 100 ierakstu dataset ar dažādiem quality statusiem.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Identity	Katram record ir asset/config/source			

Test case	Expected	Actual	PASS/FAIL	Komentārs
Freshness	STALE tiek pareizi noteikts			
Units	Mainīgajiem definētas mērvienības			

A/B/C paplašinājums

- A: JSON contract.
- B: schema validation + freshness.
- C: pievienot provenance un digital signature/hash demonstrāciju.

DT2 — Physical vs Twin residuals

Mērķis: kalibrēt vienkāršu modeli un novērtēt modeļa kļūdu

Ko stāstīt auditorijai: Modelis ir “labs” tikai pret konkrētu use case un acceptance criterion. Vienmēr salīdzini paredzējumu ar mērījumu.

Iesniedzamie pierādījumi

- CSV
- Python notebook/script
- Validation conclusion

Darba secība

1. Izvēlies pirmās kārtas, kinemātikas vai kustības modeli.
2. Iegūsti measured un predicted datu sērijas.
3. Aprēķini residual, RMSE un max absolute error.
4. Uzzīmē measured vs predicted un residual grafiku.
5. Maini vienu modeļa parametru un atkārti.
6. Definē, kura konfigurācija labāk atbilst use case.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Residual computed	Residual sērija ir korekta			
RMSE	Metrika reproducējama no datiem			
Decision	Izvēle pamatota ar metriku + grafiku			

A/B/C paplašinājums

- A: RMSE.

- B: RMSE + max error + plot.
- C: calibration optimization un validation dataset atdalīšana.

DT3 — Digital Thread un configuration provenance

Mērķis: pierādīt, kura programmatūras/modela konfigurācija radīja konkrētu testu

Ko stāstīt auditorijai: Šī laboratorija sasaista Git, testēšanu un Digital Twin: rezultātam jābūt atkārtojamam pēc versijas un konfigurācijas.

Iesniedzamie pierādījumi

- Version table
- Two test artifacts
- Comparison conclusion

Darba secība

1. Izveido projekta commit/tag.
2. Saglabā config_id un model_version.
3. Palaid testu un pieraksti software hash, config un timestamp.
4. Maini vienu konfigurācijas parametru un izveido jaunu revision.
5. Atkārto testu.
6. Izveido comparison table ar abiem run un secinājumu.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Run traceability	Katram run ir commit/config/model IDs			
Revision	Izmaiņa identificējama un aprakstīta			
Reproducibility	Var atjaunot testēto konfigurāciju			

A/B/C paplašinājums

- A: manual version table.
- B: automātisks metadata export.
- C: CI pipeline, kas pievieno test artifact hash un report.

DT4 — Twin freshness un fault injection dashboard

Mērķis: demonstrēt, ka dashboard neuzrāda stale data kā aktuālu

Ko stāstīt auditorijai: Vizualizācija ir riskanta, ja “pēdējā vērtība” izskatās dzīva pēc connection loss. Quality state jābūt redzamam un mašīnlasāmam.

Iesniedzamie pierādījumi

- Code/config
- Screenshots/logs
- Fault-injection protocol

Darba secība

1. Izveido simulatoru, kas sūta telemetry 5–10 Hz.
2. Pievieno timestamp freshness monitoru.
3. Pārtrauc datu plūsmu uz definētu periodu.
4. Dashboard/twin state jāmaina uz STALE/UNKNOWN.
5. Atjauno datu plūsmu un pārbaudi controlled recovery.
6. Saglabā screenshot/log pierādījumus.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Normal data	VALID un age zem threshold			
Connection loss	Statuss kļūst STALE/UNKNOWN			
Recovery	Atjaunošana notiek ar pareizu timestamp continuity			

A/B/C paplašinājums

- A: console status.
- B: dashboard + log.
- C: timeout policy + alert + automatic incident record.

DT5 — AAS/asset metadata prototips

Mērķis: strukturēt robota aktīva digitālo reprezentāciju submodel domāšanā

Ko stāstīt auditorijai: Nav vajadzīgs pilns AAS serveris, lai iemācītos principu: stabila identitāte, submodel un semantiski skaidri atribūti.

Iesniedzamie pierādījumi

- Asset model
- Submodel map
- Peer review checklist

Darba secība

1. Definē asset identity.
2. Izveido Nameplate, TechnicalData, ConditionMonitoring un Maintenance submodel tabulas.
3. Sasaisti live ROS/IoT telemetry ar ConditionMonitoring laukiem.

4. Pievieno dokumentācijas/URDF/software version references.
5. Definē quality/provenance laukus.
6. Veic peer review pēc completeness checklist.

Testēšanas protokols

Test case	Expected	Actual	PASS/FAIL	Komentārs
Identity	Asset ID ir unikāls un konsekvents			
Submodels	≥4 jēgpilni submodel bloki			
Traceability	Live data un documentation references sasaistītas			

A/B/C paplašinājums

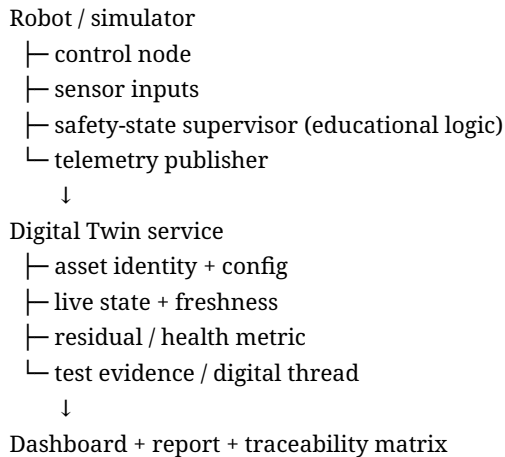
- A: tabulēta struktūra.
- B: JSON/AAS-like representation.
- C: izmēģināt AAS tooling/serveri un salīdzināt ar savu modeli.

119. Integrētais projekts — drošs robots ar Digital Twin un pierādījumiem

119.1. Projekta uzdevums

Izstrādāt simulētu vai zema riska mācību robotu sistēmu, kurai ir dokumentēts risk assessment, vismaz divas safety-related mācību funkcijas, Digital Twin telemetry, freshness/provenance mehānisms un reproducējams testu komplekts. Projekts nav sertificēta safety system izveide; mērķis ir sistēmu inženierijas un pierādījumu disciplīna.

119.2. Minimālā arhitektūra



119.3. Obligātie deliverables

- System boundary + Functional Tree
- Risk register
- Safety requirements + traceability matrix
- Robot/simulator code
- Digital Twin data contract
- Model validation or residual analysis
- Fault injection tests
- Configuration/version evidence
- Final demonstration
- Technical report + lessons learned

119.4. Acceptance criteria

ID	Criterion	Evidence
AC-01	Vismaz 10 hazard scenarios un residual risk	Risk register
AC-02	Vismaz 2 safety function requirements ar fault tests	Traceability + test protocol
AC-03	Twin telemetry satur identity/timestamp/config/quality	Dataset/data contract

ID	Criterion	Evidence
AC-04	Stale data tiek detektēta un skaidri marķēta	Fault injection log
AC-05	Modeļa/twin validācija satur vismaz 2 metriku veidus	Plots + metrics
AC-06	Katram test run var noteikt software/config version	Digital thread evidence

119.5. Vērtēšanas rubrika

Kritērijs	Svars
Risk assessment un safety argument	20%
Software/simulation kvalitāte	15%
Digital Twin datu/modela kvalitāte	20%
Verification/validation un fault injection	20%
Traceability/configuration management	15%
Aizstāvēšana un tehniskā argumentācija	10%

120. v1.4 eksāmenu un aizstāvēšanas jautājumu banka

120.1. Drošības pamati

1. Definē hazard, hazardous situation, hazardous event un risk.
2. Kāpēc reliability nav tas pats, kas safety?
3. Kas ir safe state un kāpēc tas nav vienmēr power off?
4. Izskaidro risk reduction pasākumu prioritāšu principu.
5. Kas ir residual risk?

120.2. Funkcionālā drošība

1. Kas ir safety function?
2. Uzzīmē input–logic–output safety function ķēdi.
3. Konceptuāli salīdzini PL un SIL pieeju.
4. Kāpēc redundancy bez common-cause analīzes var būt nepietiekama?
5. Kāpēc safety component sertifikāts nepierāda visas funkcijas integritāti?

120.3. Robotu šūnas un collaborative robotika

1. Atšķir industrial robot, robot application un robot cell.
2. Kāpēc cobot nav automātiski “drošs”?
3. Nosauc četrus collaborative application principus.
4. Kāpēc nav universāla droša cobot ātruma?
5. Kādi hazard rodas no end-effector un workpiece?

120.4. Verification/validation

1. Atšķir verification un validation.
2. Kas ir traceability matrix?
3. Kas ir safety case?
4. Dod trīs fault injection piemērus.
5. Kāpēc restart behaviour ir atsevišķi jātestē?

120.5. Digital Twin

1. Definē Digital Twin mācību kontekstā.
2. Kāpēc Digital Twin nav sinonīms 3D modelim?
3. Kas ir digital thread?
4. Kas ir provenance?
5. Kā stale data var radīt bīstamu decision-support rezultātu?
6. Kas ir model fidelity?
7. Atšķir calibration, verification un validation Digital Twin modelim.

120.6. Aprēķinu uzdevumi

1. Doti severity=3 un likelihood=2 mācību matricā. Aprēķini score un paskaidro, kāpēc score nav normatīvs pierādījums.

2. Dota measured/predicted sērija. Aprēķini residuals, RMSE un max absolute error.
3. Telemetry timestamp ir 2,4 s vecs un freshness requirement ir 0,5 s. Nosaki quality state un argumentē reakciju.
4. Diviem config run ir RMSE 0,12 un 0,07, bet otrajam max transient error ir lielāks. Kādu papildu analīzi veiksi pirms izvēles?

120.7. Aizstāvēšanas dziļie jautājumi

1. Vai Digital Twin drīkst tieši komandēt safety actuator? Atbildi nevis ar “jā/nē”, bet ar arhitektūras argumentu.
2. Ko darīsi, ja twin modelis labi atbilst laboratorijā, bet slikti ražošanā?
3. Kā change request safety PLC konfigurācijā ietekmē iepriekšējo validation?
4. Kādu evidence komplektu prasīsi pirms robot cell nodošanas ekspluatācijā?

121. Pasniedzēja pašmācības ceļš — Safety Engineering + Digital Twin

121.1. 12 posmu ceļš

Posms	Ko apgūt	Praktiskais rezultāts
1	ISO 12100 domāšana	System boundary + hazard list
2	Risk reduction hierarchy	Risk register ar measures
3	Safety functions	Input–logic–output decomposition
4	ISO 13849/IEC 62061 koncepti	PL/SIL terminology map
5	Robot cell safety	Mode/access/restart analysis
6	Collaborative applications	Shared workspace risk case
7	Verification/validation	Traceability + fault injection
8	Safety-security interaction	Threat-to-safety table
9	ISO 23247 Digital Twin	Reference architecture map
10	AAS/asset representation	Asset/submodel prototype
11	Model validation	Residual/RMSE analysis
12	Integrated project	Safe robot + Twin + evidence package

121.2. 5 minūšu pašpārbaude

1. Vai varu izskaidrot, kāpēc E-stop nav risk assessment aizvietotājs?
2. Vai varu izskaidrot pilnu safety function ķēdi?
3. Vai spēju formulēt testējamu safety requirement?
4. Vai spēju atšķirt Digital Twin no dashboard?
5. Vai spēju pierādīt, kura config version radīja testu?
6. Vai spēju paskaidrot, kā stale data ietekmē lēmumu?

121.3. Personīgais pašmācības projekts

Izveido savu 2WD vai manipulatora simulatoru kā “mini industrial system”: risk register, safety-state simulation, ROS 2/IoT telemetry, Digital Twin state, residual analysis, Git versioning un fault injection. Katru nedēļu pievieno vienu evidence artifact. Mērķis ir nevis tikai strādājošs demo, bet reproducējams inženiertehniskais stāsts.

122. [ARHĪVS] Versijas 1.4 statuss un nākamais slānis

122.1. Kas pievienots v1.4

- Formalizēti robotu/mehatronikas drošības inženierijas pamati.
- Risk assessment un risk reduction bloks.
- Safety functions, PL/SIL koncepti un validation.
- Aktuāls ISO 10218:2025 robotu šūnu drošības konteksts.
- Collaborative robotics drošības bloks.
- Safety-security mijiedarbība un ES regulējuma version check.
- ISO 23247 Digital Twin ietvars, AAS, digital thread un provenance.
- Digital Twin model fidelity, calibration/validation un residual analīze.
- 8 jaunas Safety/Digital Twin laboratorijas un integrēts projekts.

122.2. Nākamais plānotais slānis — v1.5

Nākamajā versijā jaunu tehnisko tēmu pievienošana tiek būtiski samazināta. Prioritāte būs skolēna/studenta gala rokasgrāmatas sakārtošana: konspektēti teorijas kopsavilkumi, formulas, terminoloģija, Python/C++/SQL/ROS 2/robotikas cheat sheets, darba lapu navigācija, projekta ceļvedis un progresijas kartes.

122.3. Standartu versiju piezīme — 2026-08

Šīs versijas safety sadaļa sagatavota, pārbaudot oficiālos ISO/IEC/EUR-Lex avotus 2026. gada augustā. ISO 12100:2010 joprojām ir publicēts, bet revision projekts ir izstrādē; ISO 10218-1/-2:2025 ir aktuālās robotu drošības versijas; ISO/TS 15066:2016 ir publicēts, bet aizvietojošs darbs notiek; IEC 62061 bāzes 2021 izdevumam ir 2024 un 2026 grozījumi. Pirms reāla projekta vienmēr jāpārbauda jaunākais oficiālais statuss.

Primārie tehniskie avoti un versiju piezīme — v1.4

- ISO 12100:2010 — Safety of machinery — General principles for design — Risk assessment and risk reduction.
- ISO 10218-1:2025 — Robotics — Safety requirements — Part 1: Industrial robots.
- ISO 10218-2:2025 — Robotics — Safety requirements — Part 2: Industrial robot applications and robot cells.
- ISO 13849-1:2023 — Safety-related parts of control systems — General principles for design.
- IEC 62061:2021 with amendments through 2026 — Functional safety of safety-related control systems.
- ISO 13850:2015 — Emergency stop function — Principles for design.
- ISO/TS 15066:2016 — Collaborative robots (published; revision work ongoing in 2026).
- ISO 23247-1/-2/-3/-4:2021; ISO 23247-5:2026; ISO 23247-6:2026 — Digital twin framework for manufacturing.
- IEC 63278-1:2023 — Asset Administration Shell for industrial applications — structure.
- Regulation (EU) 2023/1230 on machinery — check consolidated EUR-Lex applicability dates for the specific project.

123. Skolēna/studenta rokasgrāmata — ātrais starts un navigācija

Šī sadaļa ir izveidota kā ātra ieeja vairāk nekā 500 lappušu mācību materiālā. Tās mērķis nav aizstāt detalizētās nodaļas, bet palīdzēt skolēnam/studentam ātri saprast, ko mācīties, kur meklēt piemēru un kā pierādīt, ka prasme patiešām apgūta.

Galvenais princips — Neiemācies tikai sintaksi. Katru tēmu apgūsti ķēdē: PROBLĒMA → FUNKCIONĀLAIS SADALĪJUMS → ALGORITMS → KODS → TESTS → PIERĀDĪJUMS.

123.1. Kur sākt?

Ja tev vajag...	Atver vispirms	Pēc tam
Saprast algoritmu	6. nodaļu	21. nodaļas DL1–DL4 darba lapas un 22.1. pārbaudes darbu
Sākt Python	6.9. nodaļu	23. nodaļas Python quick reference un laboratorijas
Sākt C++	7. nodaļu	23. nodaļas C++ quick reference un C++ laboratorijas
Strādāt ar datiem	8. nodaļu	SQL/CSV/JSON darba lapas un integrācijas laboratoriju
Saprast mehatroniku	9. nodaļu	feedback, PID, sensori/aktuatori un fiziskās laboratorijas H1–H12
Saprast robotu	10.–14. nodaļu	kinemātika → sensori → SLAM → ROS 2 → navigācija
Taisīt projektu	15. nodaļu	28. nodaļas inženiertehniskās veidnes un 130. nodaļas ceļvedi
Mācīties universitātes līmenī	16., 31. un turpmākās padziļinājuma nodaļas	advanced control, VIO, 3D LiDAR, AI vision, safety un Digital Twin

123.2. Trīs darba režīmi

Režīms	Ko dari	Kad izmantot
Ātra atkārtošana	Lasi konspektu + formulu/cheat sheet + 3 pašpārbaudes jautājumus	Pirms stundas, pārbaudes darba vai aizstāvēšanas
Praktiskā apguve	Izpildi vienu darba lapu/laboratoriju, testē un aizpildi Expected/Actual/PASS-FAIL	Kad jāpierāda praktiska prasme
Padziļināta apguve	Lasi universitātes paplašinājumu, atkārto aprēķinu un realizē savu variantu	Pašmācībai, augstskolai, projektam

123.3. Ko nozīmē “es protu tēmu”?

- Es varu izskaidrot ideju saviem vārdiem bez koda kopēšanas.
- Es varu uzzīmēt datu vai signālu plūsmu un nosaukt ievades/izejas.
- Es varu uzrakstīt minimālu strādājošu piemēru.
- Es varu nosaukt vismaz divus failure cases un izveidot tiem testu.
- Es varu parādīt pierādījumu: kodu, mērījumu, grafiku, logu, testu protokolu vai demonstrāciju.

124. Zināšanu progresijas karte — PAMATI / PROFESIONĀLI / UNIVERSITĀTE

Visas tēmas dokumentā var lasīt trīs dziļumos. Skolēnam nav jāapgūst viss universitātes slānis uzreiz; svarīgi redzēt nākamo soli un saprast, kā pamati pārvēršas profesionālā kompetencē.

Joma	PAMATI — obligāti	PROFESIONĀLI — lieto sistēmā	UNIVERSITĀTE — modelē/analizē
Algoritmi	secība, IF, cikls, pseidokods, blokhēma	state machine, dekompozīcija, testēšana, complexity intuīcija	datu struktūras, Big-O, graph/search algoritmi
Python	mainīgie, input/output, IF, for/while, funkcijas	CSV/JSON/SQLite, NumPy, plotting, API/ROS node	robotikas matemātika, optimizācija, datu analīze, ML/DL
C++	tipi, nosacījumi, cikli, funkcijas	OOP, STL, embedded/ROS 2, state machine	memory model, RAI, real-time, control software
Dati/SQL	tabula, INSERT, SELECT, WHERE	JOIN, transakcijas, time-series, data contracts	indeksi, datu modelēšana, reproducējami eksperimenti
IoT	sensor → network → broker/server	MQTT QoS, HTTP/API, edge/cloud, offline buffering	drošība, arhitektūra, industrial gateway, digital thread
Mehatronika	sensors, aktuatori, feedback	PWM, enkodēri, PID, interlock, calibration	state-space, LQR/MPC, system identification
Robotika	DOF, frames, FK/IK ideja	odometrija, LiDAR/IMU/vision, ROS 2, SLAM, Nav2	Jacobian, dynamics, VIO, factor graphs, MoveIt 2
AI redze	attēls, klase, detection ideja	transfer learning, mAP, segmentation, ROS 2 inference	3D perception, sensor fusion, edge deployment, failure analysis
Drošība	hazard, risk, safe state	risk register, safety function, traceability, validation	PL/SIL konteksts, safety case, standards discipline

Joma	PAMATI — obligāti	PROFESIONĀLI — lieto sistēmā	UNIVERSITĀTE — modelē/analizē
Digital Twin	physical <-> digital model ideja	telemetry, freshness, residuals, provenance	fidelity, calibration/validation, AAS/digital thread

124.1. Ieteicamā secība pirmajam gadam

1. Algoritmi un problēmu sadalīšana.
2. Python kā ātra algoritma realizācijas valoda.
3. C++ kā stingrāka tipizācija un tuvāka saite ar aparatūru.
4. SQL/IoT, lai programma sāk strādāt ar sistēmas datiem.
5. Mehatronikas un robotikas pamati, lai kods ietekmē fizisku procesu.
6. Noslēguma projekts ar testēšanu un dokumentāciju.
7. Padziļinājuma tēmas tikai pēc stabila pamata.

125. Kā mācīties ar šo rokasgrāmatu — 90 minūšu un pašmācības cikls

125.1. 90 minūšu mācību stundas personīgais cikls

Laiks	Ko dara students	Rezultāts
0–10 min	Atkārto 3 iepriekšējās tēmas jautājumus; formulē šodienas mērķi	Aktivizētas priekšzināšanas
10–30 min	Teorija + viens pasniedzēja demonstrējums	Saprasts princips
30–50 min	Vadīts piemērs — risina kopā ar pasniedzēju	Pirmais pareizais modelis
50–75 min	Patstāvīgs A/B/C līmeņa uzdevums	Praktiska prasme
75–85 min	Tests/debugging/Expected vs Actual	Pierādījums, ka risinājums darbojas
85–90 min	Exit ticket: 1 lieta ko sapratu, 1 kļūda, 1 jautājums	Refleksija un nākamais solis

125.2. 45 minūšu pašmācības cikls

1. 5 min — uzraksti, ko tieši vēlies spēt izdarīt pēc sesijas.
2. 10 min — izlasi tikai konkrētās tēmas konspektu un vienu piemēru.
3. 20 min — aizver paraugu un uzraksti/izveido savu risinājumu.
4. 5 min — izveido vismaz 3 test cases, ieskaitot robežgadījumu.
5. 5 min — pieraksti vienu kļūdu un vienu secinājumu savā portfolio.

Svarīgi — Ja 20 minūšu praktiskajā daļā tikai pārraksti gatavu kodu, pašmācības cikls nav izpildīts. Jāspēj mainīt ieejas datus, prasību vai arhitektūras daļu un prognozēt rezultātu.

126. Kodola konspekti — algoritmi, Python, C++, SQL, IoT, mehatronika un robotika

126.1. Algoritmi — viena lapa

- Algoritms ir galīga un precīzi definēta darbību secība, kas no ievades noved līdz rezultātam.
- Trīs pamatstruktūras: secība (sequence), zarošanās (selection) un atkārtošanās (iteration).
- Pirms koda formulē ievadi, apstrādi, izvadi, nosacījumus un beigu nosacījumu.
- Dekompozīcija sadala lielu problēmu mazās funkcijās/moduļos ar skaidru atbildību.
- Labs algoritms ietver arī kļūdu/robežgadījumu apstrādi, timeout vai safe state, ja tas vajadzīgs.

126.2. Python — viena lapa

- Python ir interpretēta, augsta līmeņa valoda, kas īpaši ērta algoritmu, datu apstrādes un robotikas prototipēšanai.
- Pamatķēde: input → tipu pārveidošana/validācija → aprēķins/logika → output.
- if/elif/else realizē zarošanos; for — zināmu iterāciju; while — stāvokļa vadītu atkārtošanos.
- Funkcija samazina dublēšanos un nodala atbildību; saraksts/dict glabā datu kopas.
- Tehniskos projektos Python bieži savieno sensoru/ROS/DB/API datus ar analīzi un vizualizāciju.

126.3. C++ — viena lapa

- C++ dod stingrāku kontroli pār tipiem, atmiņu un izpildes struktūru nekā Python.
- Programma tiek kompilēta; tipa izvēle ietekmē diapazonu, precizitāti un aparatūras resursus.
- const dokumentē nemainīgu vērtību; funkcijas un klases palīdz veidot moduļus.
- OOP tehniskā piemērā: Sensor klase glabā stāvokli un metodes, bet konkrēts sensors ir objekts.
- Embedded/ROS 2 vidē svarīga ir deterministiska logika, atmiņas disciplīna un failure handling.

126.4. SQL un dati — viena lapa

- Datubāze ļauj saglabāt mērījumus strukturēti un atkārtojami.
- Tabula = lauki/kolonnas + ieraksti/rindas; primārā atslēga identificē ierakstu.
- INSERT pievieno datus; SELECT atlasa; WHERE filtrē; ORDER BY kārtō; JOIN savieno saistītas tabulas.
- Tehniskam mērījumam parasti vajag vismaz value, unit, timestamp un source/device identitāti.
- Datu kvalitāte nozīmē arī missing/invalid/stale vērtību apstrādi un izsekojamību.

126.5. IoT un mākoņdati — viena lapa

- IoT ķēde: sensors → edge device → network → broker/server → database → application/dashboard.
- MQTT ir publish/subscribe modelis ar brokeri un topic; HTTP/REST vairāk balstās request/response.
- JSON ir biežs datu apmaiņas formāts, bet vajadzīgs datu līgums: lauki, tipi, units, timestamp, version.
- Reālai sistēmai jāparedz connection loss, buffering, retry, authentication un data freshness.

- Cloud nav automātiski drošības kontroles slānis; safety funkcijām vajag piemērotu lokālu arhitektūru.

126.6. Mehatronika — viena lapa

- Mehatronika savieno mehāniku, elektroniku, vadību un programmatūru vienā sistēmā.
- Closed-loop ķēde: setpoint → controller → actuator → process → sensor → feedback.
- Sensors pārveido fizikālu lielumu datus; aktuators pārveido komandu fizikālā darbībā.
- PWM regulē vidējo piedziņas jaudu/ātrumu; enkoders dod kustības atgriezenisko saiti.
- PID izmanto kļūdu starp setpoint un measured value; pirms regulēšanas svarīga ir kalibrācija un sampling.

126.7. Robotika un kinemātika — viena lapa

- Robota pose apraksta pozīciju un orientāciju noteiktā koordinātu sistēmā.
- Forward kinematics: joint values → end-effector pose; inverse kinematics: vēlamā pose → joint values.
- Mobilam differential-drive robotam kustību nosaka kreisā/labā riteņa ātrumu attiecība.
- Frames/TF ir obligāti, lai sensoru un robotu dati būtu vienā ģeometriskā kontekstā.
- Kinemātika neapraksta spēkus; dinamika pievieno masu, inerci, gravitāciju un torque.

126.8. Autonomi robots — viena lapa

- Autonomijas cikls: SENSE → PERCEIVE → LOCALIZE/MAP → DECIDE → PLAN → CONTROL → ACT.
- LiDAR, kamera, IMU un enkoderi sniedz dažāda tipa informāciju un dažādas kļūdas.
- SLAM vienlaikus veido karti un novērtē robota atrašanās vietu; localization izmanto karti, ja tā jau ir.
- Path planning atrod ceļu; trajectory pievieno laiku, ātrumu un paātrinājumu; controller izpilda kustību.
- Recovery/degraded mode ir daļa no autonomijas — robots nedrīkst pieņemt, ka sensori un plāns vienmēr darbojas.

126.9. ROS 2 — viena lapa

- ROS 2 ir sadalītas robotikas sistēmas middleware un rīku ekosistēma.
- Node veic konkrētu funkciju; topic straumē datus; service ir īss request/response; action — ilgāks mērķis ar feedback.
- TF uztur koordinātu transformāciju koku; sensor_msgs/nav_msgs standartizē biežus robotikas datus.
- Nav2 savieno localization/map, costmaps, planner, controller un behavior/recovery loģiku.
- ROS 2 debugging sākas ar graph, topic, frame, timestamp un QoS pārbaudi, nevis ar nejaušu parametru mainīšanu.

126.10. Drošība un Digital Twin — viena lapa

- Hazard ir potenciāls kaitējuma avots; risk ņem vērā kaitējuma smagumu un iespējamību/eksponāciju atbilstoši izvēlētai metodei.

- Risk reduction prioritāte: inherently safe design → safeguards/protective measures → information for use.
- Safety function jāformulē testējami: trigger/input, logic, output/safe response un acceptance criteria.
- Digital Twin nav tikai 3D modelis — svarīga ir datu saite, konfigurācijas identitāte, freshness, model fidelity un validācija.
- Digital Twin rezultāts ir decision-support pierādījums tikai tad, ja zināma datu/modela kvalitāte un izsekojamība.

127. Formulu un aprēķinu rokasgrāmata

Šī sadaļa apkopo biežākās formulas no kursa. Tā nav matemātikas teorijas aizvietotājs — pie katras formulas jāzina lielumu nozīme, mērvienības un pieņēmumi.

Formula/jēdziens	Pieraksts	Kur izmanto
Vidējā vērtība	$\bar{x} = (x_1 + \dots + x_n) / n$	Sensoru mērījumu kopsavilkums
Procentuālā daļa	$p = \text{daļa} / \text{kopā} \times 100 \%$	Brāķis, izpildes procents, kļūdas īpatsvars
Rotāciju $\rightarrow$ distance	$s = N \cdot \pi \cdot D$	Enkodera/riteņa distances vienkāršots modelis
2DOF FK	$x = L_1 \cos \theta_1 + L_2 \cos(\theta_1 + \theta_2);$ $y = L_1 \sin \theta_1 + L_2 \sin(\theta_1 + \theta_2)$	Plaknes manipulatora gala punkts
Differential drive	$v = (v_R + v_L)/2; \omega = (v_R - v_L)/b$	Mobilā robota lineārais/leņķiskais ātrums
Pose integrācija	$x_{k+1} \approx x_k + v \cos \theta \cdot \Delta t; y_{k+1} \approx y_k + v \sin \theta \cdot \Delta t; \theta_{k+1} \approx \theta_k + \omega \Delta t$	Vienkāršota odometrija
Kontroles kļūda	$e(t) = r(t) - y(t)$	Setpoint pret measured output
PID	$u = K_p \cdot e + K_i \int e \, dt + K_d \cdot de/dt$	Closed-loop regulēšana
RMSE	$RMSE = \sqrt{(\sum (y_i - \hat{y}_i)^2 / n)}$	Model/twin kļūdas novērtējums
Precision	$TP / (TP + FP)$	Cik no pozitīvi prognozētajiem bija pareizi
Recall	$TP / (TP + FN)$	Cik no īstajiem pozitīvajiem tika atrasti
F1	$2 \cdot \text{precision} \cdot \text{recall} / (\text{precision} + \text{recall})$	Precision/recall līdzsvars
IoU	intersection / union	Detection/segmentation pārklājums
LiDAR polar $\rightarrow$ Cartesian	$x = r \cos \theta; y = r \sin \theta$	2D scan punktu koordinātas
Sampling period	$f = 1/T$	Loop/sensor frekvence no perioda

Formula/jēdziens	Pieraksts	Kur izmanto
Lineāra state-space forma	$\dot{x} = Ax + Bu; y = Cx + Du$	Modernās vadības sistēmas modelis
LQR izmaksu funkcija	$J = \int (x^T Q x + u^T R u) dt$	Stāvokļa kļūdas un vadības piepūles kompromiss

127.1. Aprēķina disciplīna

1. Pieraksti zināmos lielumus un to mērvienības.
2. Izvēlies formulu un nosauc tās pieņēmumus.
3. Pārveido vienības pirms ievietošanas formulā.
4. Aprēķini un saglabā pietiekamu precizitāti starprezultātos.
5. Pārbaudi rezultāta fizisko jēgu un robežvērtības.
6. Ja rezultāts izmantots kodā, uzraksti vismaz vienu unit test vai numerisku salīdzinājumu.

128. Ātrās atsaucēs kartes — kods, komandas un protokoli

128.1. Python minimālā karte

```
# ievade un tipi
value = float(input("T = "))

# nosacījums
if value > 30:
    state = "HOT"
elif value < 10:
    state = "COLD"
else:
    state = "OK"

# cikls un saraksts
values = [20.1, 20.4, 20.2]
for v in values:
    print(v)

# funkcija
def mean(xs):
    return sum(xs) / len(xs)
```

128.2. C++ minimālā karte

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    double value{};
    cin >> value;
    const double LIMIT = 30.0;

    if (value > LIMIT) {
        cout << "HOT\n";
    } else {
        cout << "OK\n";
    }
    return 0;
}
```

128.3. SQL minimālā karte

```
CREATE TABLE measurements (
    id INTEGER PRIMARY KEY,
    ts TEXT NOT NULL,
    value REAL NOT NULL,
    unit TEXT,
    source TEXT
```

```
);

INSERT INTO measurements(ts,value,unit,source)
VALUES('2026-08-18T12:00:00',24.6,'degC','T01');

SELECT ts,value
FROM measurements
WHERE value > 25
ORDER BY ts;
```

128.4. MQTT / JSON minimālā karte

```
topic: lab/robot1/temperature

{
  "timestamp": "2026-08-18T12:00:00+03:00",
  "value": 24.6,
  "unit": "degC",
  "device_id": "robot1",
  "schema_version": 1
}
```

128.5. Git minimālā karte

```
git status
git switch -c feature/my-change
git add .
git commit -m "Add validated sensor parser"
git log --oneline --decorate -10
git diff main...HEAD
```

128.6. ROS 2 diagnostikas minimums

```
ros2 node list
ros2 topic list
ros2 topic info /scan
ros2 topic echo /odom --once
ros2 topic hz /imu/data
ros2 interface show sensor_msgs/msg/LaserScan
ros2 run tf2_ros tf2_echo odom base_link
ros2 bag info <bag_folder>
```

129. Debugging un problēmu risināšanas ceļvedis

Debugging noteikums — Maini vienu lietu vienā reizē un pirms izmaiņas pieraksti, ko sagaidi. Pretējā gadījumā nav zināms, kura izmaiņa palīdzēja vai sabojāja sistēmu.

Simptoms	Pārbaudi vispirms	Tad
Programma nepalaižas	sintakse, imports/includes, compiler/interpreter error	minimāls reproducējams piemērs
Rezultāts nepareizs	ievaddati, tipi, vienības, robežnosacījumi	starp rezultātu log/print un unit test
Cikls nebeidzas	WHILE beigu nosacījums, state update	timeout/watchdog un kļūdas stāvoklis
SQL neatgriež datus	schema, table/column names, WHERE filtrs	SELECT * un soli pa solim sašaurini query
MQTT nav datu	broker adrese/ports, topic, connection status	QoS/auth/network un publisher/subscriber logi
Sensors rāda muļķības	barošana, GND, pin/bus, units, calibration	raw data un datasheet diapazons
Robots brauc šķībi	riteņi/berze, encoder counts, radius/track-width calibration	closed-loop wheel speed un odometrijas logs
ROS topic ir, bet dati neder	message type, frame_id, timestamp, units	TF, QoS, rate un covariance
SLAM karte deformējas	odom drift, scan quality, TF/timestamps	motion speed, calibration, loop closure/config
AI modelis “labs”, bet robotā slikts	dataset shift, preprocessing, latency	failure-set, threshold, real-world test data

129.1. Piecu jautājumu algoritms

1. Kas tieši nedarbojas un kas darbojas?
2. Kāds bija Expected un kāds ir Actual?
3. Kurš ir mazākais sistēmas bloks, kurā atšķirība pirmo reizi parādās?
4. Kādu vienu hipotēzi varu pārbaudīt ar mērījumu/logu/testu?
5. Vai pēc labojuma atkārtotu sākotnējo un vismaz vienu regresijas testu?

130. Projektu ceļveži — no idejas līdz aizstāvēšanai

130.1. Universālais projekta cikls

Posms	Galvenais jautājums	Obligāts pierādījums
1. Problēma	Kam un kāpēc sistēma vajadzīga?	1–3 teikumu problem statement
2. Prasības	Ko sistēmai jādara un kā zināsim, ka izdarīts?	testējamas requirements + acceptance criteria
3. Functional Tree	Kā galvenā funkcija sadalās apakšfunkcijās?	functional tree ar ievadēm/izejām
4. Arhitektūra	Kuras funkcijas būs Python/C++/DB/IoT/ROS/aparatūrā?	bloku shēma + interfeisi
5. Realizācija	Kāds ir minimālais strādājošais kodols?	Git commit/tag vai versijas identitāte
6. Testēšana	Vai katrai prasībai ir tests?	Expected/Actual/PASS-FAIL + logs/mērījumi
7. Validācija	Vai risinājums reāli risina sākotnējo problēmu?	demonstrācija un secinājumi
8. Aizstāvēšana	Vai vari pamatot izvēles un kļūdas?	5–10 min tehniska prezentācija + Q&A

130.2. Mini projekts Python

- Prasība: vismaz viena ievade, validācija, nosacījums, cikls vai datu kopa un funkcija.
- Pierādījums: 5 testa gadījumi, tostarp kļūdaina ievade/robežgadījums.
- Paplašinājums: CSV/SQLite vai grafiks.

130.3. Mini projekts C++

- Prasība: skaidri tipi, const, funkcijas un vismaz viena state-machine vai class struktūra.
- Pierādījums: compile bez kļūdām/brīdinājumu analīze, testu tabula, debugging piemērs.
- Paplašinājums: embedded/sensoru simulācija vai C++ komponente ROS 2 projektā.

130.4. IoT integrācijas projekts

- Datu ceļš: source → edge → MQTT/HTTP → storage → visualization.
- Definē JSON/data contract un topic/API nosaukumu.
- Obligāti testē connection loss, invalid value un timestamp/freshness.

130.5. Robotikas noslēguma projekts

- Obligāti: functional tree, frames/TF vai koordinātu sistēmu skaidrojums, sensoru/aktuatoru interfeisi, testu plāns.
- Ja autonomi robots: perception/localization/planning/control ķēde un vismaz viens recovery scenārijs.
- Ja manipulators: FK/IK vai trajectory/planning daļa un skaidri kustības ierobežojumi.
- Ja izmantots AI: dataset/model version, metriku tabula un failure-set testēšana.

131. Portfolio un pierādījumu komplekts profesionālai izaugsmei

Labs portfolio parāda ne tikai gala rezultātu, bet inženiertehnisko domāšanu. Viena labi dokumentēta sistēma ir vērtīgāka par desmit neizskaidrotiem koda failiem.

Artefakts	Ko tas pierāda	Ieteicamais formāts
Problem statement + requirements	spēju saprast uzdevumu un formulēt testējamu mērķi	README/PDF/Markdown
Functional Tree / arhitektūra	sistēmu domāšanu un dekompozīciju	diagramma + īss skaidrojums
Kods + versiju vēsture	programmēšanas un darba plūsmas disciplīnu	Git repository
Testu protokols	spēju pierādīt rezultātu	Expected/Actual/PASS-FAIL tabula
Datu logs/grafiks	mērījumu un datu analīzi	CSV/SQLite + plot
Failure/debugging pieraksts	problēmu risināšanas spēju	issue/engineering log
Demo video/foto	reālu sistēmas darbību	īss demonstrācijas ieraksts
Secinājumi	kritisko domāšanu un ierobežojumu izpratni	1 lappuses technical summary

131.1. Vienas projekta mapes ieteicamā struktūra

```
project_name/  
├─ README.md  
├─ requirements/  
├─ architecture/  
├─ src/  
├─ config/  
├─ tests/  
├─ data/  
├─ results/  
├─ docs/  
└─ CHANGELOG.md
```

132. Terminoloģijas vārdnīca A–Z

Termins	Īss skaidrojums
ADC	Analog-to-Digital Converter — analogā signāla pārveide digitālā vērtībā.
Actuator	Izpildmehānisms, kas vadības komandu pārveido fizikālā darbībā.
Algorithm	Precīzi definēta darbību secība uzdevuma risināšanai.
API	Application Programming Interface — noteikts veids, kā programmatūras komponentes savstarpēji sazinās.
AAS	Asset Administration Shell — standartizēta aktīva digitālās reprezentācijas struktūra industriālā kontekstā.
Behavior Tree	Hierarhiska lēmumu/uzvedības organizēšanas struktūra, ko bieži izmanto robotikā.
Broker	MQTT starpniekserveris, kas saņem un izplata ziņojumus starp publisher un subscriber.
Calibration	Parametru pielāgošana, izmantojot references mērījumus.
CAN	Controller Area Network — robusta kopnes komunikācija iekārtām/transportam/embedded sistēmām.
Closed-loop	Vadība ar atgriezenisko saiti no procesa mērījuma.
CNN	Convolutional Neural Network — neironu tīkls, kas īpaši piemērots attēlu pazīmju apguvei.
Compiler	Programma, kas avota kodu pārveido izpildāmā/mašīnkoda formā.
Costmap	Navigācijas karte ar šķēršļu un izmaksu informāciju plānošanai.
Covariance	Nenoteiktības un mainīgo savstarpējās saistības matemātisks raksturojums.

Termins	Īss skaidrojums
Digital Thread	Izsekojama saite starp prasībām, konfigurācijām, datiem, testiem un sistēmas dzīves cikla artefaktiem.
Digital Twin	Digitāla sistēmas reprezentācija ar datu saiti, konfigurācijas identitāti un modelēšanas/analīzes funkcijām.
DOF	Degrees of Freedom — neatkarīgu kustības koordinātu skaits.
EKF	Extended Kalman Filter — nelineāru modeļu stāvokļa novērtēšanas filtrs.
Encoder	Sensors, kas mēra rotāciju/pozīciju un bieži tiek izmantots odometrijai vai closed-loop vadībai.
End-effector	Manipulatora gala instruments vai darba orgāns.
Fail-safe	Projektēšanas princips, kur kļūmes gadījumā sistēma pāriet paredzētā drošā stāvoklī/celī.
F1 score	Precision un recall harmoniskais vidējais.
Feedback	Mērījums no procesa, kas tiek izmantots vadības lēmuma koriģēšanai.
Forward Kinematics	Locītavu stāvokļu pārveide gala efektorā pozā.
Frame	Koordinātu sistēma, attiecībā pret kuru definē pozīciju vai mērījumu.
Functional Tree	Sistēmas galvenās funkcijas hierarhisks sadalījums pārbaudāmās apakšfunkcijās.
GPIO	General Purpose Input/Output — programmējami mikrokontrollera digitālie pini.
IMU	Inertial Measurement Unit — parasti akselerometrs + žiroskops, dažreiz magnetometrs.
Interlock	Loģika/mehānisms, kas atļauj vai bloķē darbību atkarībā no noteiktiem nosacījumiem.

Termins	Īss skaidrojums
IoT	Internet of Things — tīklā savienotas fiziskas ierīces, kas vāc/apmaina datus.
IoU	Intersection over Union — pārklājuma metrika detection/segmentation uzdevumos.
Inverse Kinematics	Vēlamās gala efektora pozas pārveide locītavu konfigurācijā.
Jacobian	Matrica, kas saista locītavu ātrumus ar gala efektora lineāro/leņķisko ātrumu.
JSON	Teksta datu apmaiņas formāts ar key-value struktūru.
LiDAR	Light Detection and Ranging — attāluma/vides mērīšana ar gaismas signālu.
Localization	Robota pozas novērtēšana zināmā vai veidotā kartē.
LQR	Linear Quadratic Regulator — state-space vadības metode ar kvadrātisku izmaksu funkciju.
MQTT	Viegls publish/subscribe ziņojumapmaiņas protokols.
MPC	Model Predictive Control — vadība, kas atkārtoti optimizē nākotnes vadības darbības, izmantojot modeli un ierobežojumus.
NMS	Non-Maximum Suppression — pārklājošos detection rezultātu filtrēšanas metode.
Node	ROS 2 procesa/komponentes loģiska vienība ar konkrētu funkciju.
Odometry	Robota kustības/pozas novērtējums no kustības sensoriem/modeli.
Open-loop	Vadība bez procesa mērījuma atgriezeniskās saites.
PID	Proporcionāli-integrāli-diferenciāls regulators.

Termins	Īss skaidrojums
Point Cloud	3D punktu kopa ar koordinātām un iespējami papildu atribūtiem.
Pose	Pozīcija + orientācija.
Precision	TP/(TP+FP) — pozitīvo prognožu precizitāte klasifikācijas/detection kontekstā.
Publisher	ROS 2/MQTT komponente, kas publicē datus.
PWM	Pulse Width Modulation — impulsa darba cikla modulācija vadības signālam.
QoS	Quality of Service — ziņojumu piegādes/uzglabāšanas/uzticamības politika.
Recall	TP/(TP+FN) — spēja atrast īstos pozitīvos gadījumus.
Residual	Atšķirība starp measured un predicted/expected vērtību.
RMSE	Root Mean Square Error — vidējās kvadrātiskās kļūdas sakne.
ROS 2	Robot Operating System 2 — middleware, message interfeisi un rīki robotikas programmatūras arhitektūrai.
Safe state	Iepriekš definēts stāvoklis, kas attiecīgā riska scenārijā samazina bīstamību.
Sensor Fusion	Vairāku sensoru informācijas kombinēšana labākam stāvokļa/vides novērtējumam.
SLAM	Simultaneous Localization and Mapping — vienlaicīga kartes veidošana un lokalizācija.
SQL	Structured Query Language — relāciju datubāžu definēšanas un vaicājumu valoda.
State Machine	Sistēmas uzvedības modelis kā stāvokļi un pārejas starp tiem.
Subscriber	Komponente, kas saņem/publicēta topic datus.

Termins	Īss skaidrojums
TF	ROS 2 koordinātu transformāciju koks/laika vēsture.
Timeout	Maksimālais gaidīšanas laiks, pēc kura notiek kļūdas vai fallback reakcija.
Traceability	Spēja izsekot saiti starp prasību, realizāciju, konfigurāciju, testu un rezultātu.
Trajectory	Kustības ceļš ar laika/ātruma/paātrinājuma informāciju.
URDF	Unified Robot Description Format — ROS robotu ģeometrijas/locītavu/saišu apraksts.
Validation	Pierādījums, ka sistēma atbilst paredzētajam lietojumam un vajadzībai.
Verification	Pierādījums, ka realizācija atbilst definētām prasībām/specifikācijai.
Watchdog	Uzraudzības mehānisms, kas konstatē iestrēgušu vai neatbildošu procesu un izraisa definētu reakciju.
Xacro	Makro valoda URDF robotu modeļu parametrizētai ģenerēšanai.

133. Pašvērtējuma un kompetenču kontrolsaraksti

Atzīmē tikai to, ko vari izdarīt bez gatava risinājuma kopēšanas. “Daļēji” nozīmē, ka vari ar dokumentāciju vai nelielu pasniedzēja norādi.

133.1. Algoritmi

Prasme	Jā	Daļēji	Vēl nē
Sadalīt problēmu ievadē, apstrādē un izvadē	[]	[]	[]
Uzrakstīt pseidokodu ar IF un ciklu	[]	[]	[]
Uzzīmēt blokshēmu	[]	[]	[]
Izveidot 5 test cases	[]	[]	[]
Paskaidrot timeout/watchdog nepieciešamību	[]	[]	[]

133.2. Python

Prasme	Jā	Daļēji	Vēl nē
Nolasīt un validēt ievadi	[]	[]	[]
Lietot if/for/while	[]	[]	[]
Izveidot funkciju	[]	[]	[]
Apstrādāt sarakstu/CSV	[]	[]	[]
Atrisināt kļūdu ar sistemātisku debugging	[]	[]	[]

133.3. C++

Prasme	Jā	Daļēji	Vēl nē
Izvēlēties tipu un const	[]	[]	[]

Prasme	Jā	Daļēji	Vēl nē
Uzrakstīt funkciju	[]	[]	[]
Izveidot class/object piemēru	[]	[]	[]
Izmantot vector/array	[]	[]	[]
Izskaidrot compile/runtime kļūdas atšķirību	[]	[]	[]

133.4. Dati/IoT

Prasme	Jā	Daļēji	Vēl nē
Izveidot SQL tabulu un vaicājumu	[]	[]	[]
Izskaidrot JSON datu līgumu	[]	[]	[]
Publicēt/saņemt MQTT topic	[]	[]	[]
Paredzēt connection loss	[]	[]	[]
Pārbaudīt timestamp/freshness	[]	[]	[]

133.5. Mehatronika

Prasme	Jā	Daļēji	Vēl nē
Atšķirt sensoru un aktuatoru	[]	[]	[]
Uzzīmēt feedback loop	[]	[]	[]
Izskaidrot PWM	[]	[]	[]
Kalibrēt enkodera/riteņa parametru	[]	[]	[]

Prasme	Jā	Daļēji	Vēl nē
Paskaidrot PID P/I/D komponentes	[]	[]	[]

133.6. Robotika

Prasme	Jā	Daļēji	Vēl nē
Definēt DOF/frame/pose	[]	[]	[]
Aprēķināt vienkāršu FK	[]	[]	[]
Izskaidrot odometriju	[]	[]	[]
Paskaidrot LiDAR/IMU/vision lomu	[]	[]	[]
Uzzīmēt SENSE→PLAN→CONTR OL ķēdi	[]	[]	[]

133.7. ROS 2

Prasme	Jā	Daļēji	Vēl nē
Atšķirt topic/service/action	[]	[]	[]
Pārbaudīt topic type/rate	[]	[]	[]
Izskaidrot TF koku	[]	[]	[]
Paskaidrot SLAM vs localization	[]	[]	[]
Diagnosticēt /odom, /imu, /scan datu plūsmu	[]	[]	[]

133.8. Inženierija

Prasme	Jā	Daļēji	Vēl nē
Uzrakstīt testējamu requirement	☐	☐	☐
Izveidot functional tree	☐	☐	☐
Veidot traceability matrix	☐	☐	☐
Dokumentēt Expected/Actual/PASS-FAIL	☐	☐	☐
Izskaidrot vienu projekta failure case un labojumu	☐	☐	☐

134. Profesionālās izaugsmes ceļi pēc kursa

Virziens	Nākamās prasmes	Ieteicamais portfolio projekts
Automatizācija / PLC	IEC 61131-3, Structured Text/Ladder, HMI/SCADA, industrial networks	Konveijera vai procesa simulatora state machine + interlock + datu logging
Embedded / IoT	ESP32/STM32, C/C++, RTOS, buses, low-level debugging	Sensoru mezgls ar MQTT, offline buffer un testētu datu līgumu
Mehatronika	motori, drives, encoders, PID, system identification	2WD vai servo sistēma ar kalibrāciju un closed-loop vadību
Robotika	ROS 2, kinematics, TF, SLAM, Nav2, MoveIt 2	Autonoms mobilais robots vai manipulatora simulācija
Computer Vision / AI	OpenCV, PyTorch, dataset engineering, deployment	Detection/segmentation pipeline ar failure-set un ROS 2 output
Datu inženierija	SQL, APIs, time-series DB, dashboards, ETL	IoT telemetry pipeline + DB + analīzes dashboard
Robotu drošība / validation	risk assessment, requirements, traceability, test/validation	Robot cell risk register + safety function test plan
Universitātes robotika	linear algebra, probability, state-space, optimization, estimation	VIO/SLAM/control simulācijas eksperiments ar reproducējamajiem datiem

135. Laboratoriju, pārbaudes darbu un projektu navigācijas indekss

Šis indekss palīdz izvēlēties praktisko darbu pēc mērķa, nevis tikai pēc nodaļas numura. Precīzās instrukcijas un etalona risinājumi atrodas iepriekšējās dokumenta sadaļās.

Mērķis	Ieteicamais materiāls	Pirms tam jāprot
Algoritmu domāšana	DL1–DL4 + algoritmu pārbaudes darbs	ievade/izvade, IF, cikls
Python pamati	DL5–DL7 + Python mini projekts	algoritms un datu tipi
C++ pamati	DL8–DL9 + C++ pārbaudes darbs	algoritmi + Python idejas
SQL/IoT	DL10–DL11 + L/IoT laboratorijas	Python, datu tipi
Mehatronikas aparatūra	H1–H8	C++/Arduino pamati + drošs pieslēgums
IoT fiziska sistēma	H9	ESP32, MQTT/JSON
LiDAR/Vision	H10–H11	Python/C++ + sensoru datu pamati
2WD autonomš robots	H12 + R1–R9	enkoderi, IMU, LiDAR, ROS 2
Kinemātika	2DOF laboratorijas + SS/advanced moduļi	trigonometrija + Python
ROS 2 navigācija	R4–R9	/odom, /imu, /scan, TF
AI perception	DL1–DL9 (v1.3 bloks)	Python, OpenCV, ML pamati
Safety/Digital Twin	SAFE1–SAFE3 + DT1–DT5	requirements, testēšana, sistēmu arhitektūra
Noslēguma projekts	Nodarbības 25–36 + 28. nodaļas veidnes	vismaz Python/C++ un viena integrācijas joma

135.1. Pārbaudes darba atkārtotāšanas secība

1. Izlasi attiecīgās 126. nodaļas vienas lapas konspektu.
2. Izpildi 5–10 A līmeņa jautājumus bez materiāla.
3. Izpildi vismaz vienu B/C praktisko uzdevumu.
4. Izskaidro vienu tipisku kļūdu un tās diagnostiku.
5. Tikai pēc tam pildi pilno pārbaudes darba variantu ar laika ierobežojumu.

136. [ARHĪVS] Versijas 1.5 statuss un pāreja uz pasniedzēja gala komplektu

136.1. Kas pievienots v1.5

- Skolēna/studenta ātrā navigācija pa lielo dokumentu.
- Trīs līmeņu progresijas karte: PAMATI / PROFESIONĀLI / UNIVERSITĀTE.
- 90 minūšu un pašmācības darba cikli.
- 10 kodola vienas lapas konspekti.
- Formulu un aprēķinu rokasgrāmata.
- Python/C++/SQL/MQTT/Git/ROS 2 quick-reference kartes.
- Debugging lēmumu ceļvedis.
- Projektu ceļveži un portfolio struktūra.
- Paplašināta A–Z terminoloģijas vārdnīca.
- Kompetenču pašvērtējuma lapas un profesionālās izaugsmes ceļi.
- Laboratoriju/pārbaudes darbu/projektu navigācijas indekss.

136.2. Ko šī versija apzināti nedara

- Nepievieno vēl vienu lielu jaunu robotikas vai AI teorijas nozari.
- Neaizvieto detalizētās laboratoriju instrukcijas ar īsiem kopsavilkumiem.
- Neizmaina iepriekšējās drošības/normatīvo standartu piezīmes bez atsevišķas versiju pārbaudes.
- Neuzskata cheat sheet par zināšanu pierādījumu — prasme joprojām jāapstiprina ar uzdevumu/testu/projektu.

136.3. Nākamais plānotais slānis — v1.6

v1.6 prioritāte: pasniedzēja gala komplekts un vērtēšanas sistēmas normalizācija. Plānots sakārtot visu 36 nodarbību teacher checklist, vienotu lesson-delivery karti, pārbaudes darbu variantu matricu, laboratoriju atbilžu indeksu, projekta vērtēšanas rubriku vienoto versiju, semestra/gada atzīmes aprēķina ieteikumu un kursa apstiprināšanas kopsavilkumu.

136.4. Galvenā doma studentam

Kursa rezultāts — Mērķis nav zināt 500+ lapas no galvas. Mērķis ir spēt strukturēt problēmu, izvēlēties pareizo rīku, uzbūvēt risinājumu, izmērīt rezultātu un tehniski pamatot, kāpēc risinājumam var uzticēties.

137. Pasniedzēja gala komplekts — lietošanas princips un dokumentu kontrole

Šī nodaļu grupa apvieno iepriekš izstrādāto kursa saturu vienotā pasniedzēja darba sistēmā. Mērķis nav atkārtot teoriju vai laboratoriju instrukcijas, bet noteikt, ko pasniedzējs sagatavo pirms nodarbības, kādu minimālo pierādījumu savāc stundā, kā vērtē sasniedzamos rezultātus un kā uztur kursu reproducējamu vairākus mācību gadus.

Pasniedzējam: v1.6 ir “operacionālais slānis”. Detalizētā teorija, kodi un laboratoriju risinājumi paliek savās iepriekšējās nodaļās; šeit tiek normalizēta lietošana, vērtēšana un dokumentu aprīte.

137.1. Pasniedzēja darba princips

1. Pirms nodarbības definē vienu galveno sasniedzamo rezultātu un vienu obligāto pierādījumu.
2. Nodarbībā vispirms aktivizē iepriekšējās zināšanas, pēc tam demonstrē jauno principu un tikai tad dod patstāvīgu uzdevumu.
3. Skolēna/studentu darbs tiek vērtēts pēc novērojama rezultāta: algoritma, koda, mērījuma, testa, dokumentācijas vai aizstāvēšanas.
4. Kļūdas tiek izmantotas kā mācību materiāls. Debugging process ir vērtējams pierādījums, nevis tikai “neveiksme”.
5. Katram projektam saglabā prasības, versiju, testus un rezultātus, lai vērtējums būtu izsekojams.
6. Paplašinātās universitātes tēmas izmanto diferenciācijai un pasniedzēja pašmācībai; tās nav automātiski obligātas visiem audzēkņiem.

137.2. Vienas nodarbības minimālais dokumentu komplekts

Elements	Kas pasniedzējam jābūt gatavs	Kas paliek kā pierādījums
Mērķis	1 skaidrs sasniedzamais rezultāts	Stundas ieraksts / plāns
Teorija	3–5 galvenie jēdzieni un viens tehniskais piemērs	Konspekts vai slaidu/lapas atsauce
Prakse	Obligātais A/B uzdevums; C paplašinājums pēc vajadzības	Fails, darba lapa, kods vai mērījumu tabula
Testēšana	Vismaz 2 normāli + 1 robežgadījums, ja tēma to atļauj	Expected/Actual/PASS-FAIL vai līdzvērtīgs pieraksts
Noslēgums	Exit ticket vai 1–3 refleksijas jautājumi	Īsa atbilde / pašvērtējums
Versija	Izmantoto failu, koda vai konfigurācijas versija	Datums, commit/tag vai faila versija

137.3. Dokumentu kontroles minimums

- Kursa pamatdokuments: viena kontrolēta master versija; skolēniem izsniedz PDF/izrakstus vai konkrētas darba lapas.

- Koda piemēri: glabā mapēs pēc nodarbības/laboratorijas identifikatora; neizmanto failus ar nosaukumiem “final_final2”.
- Konfigurācijas: YAML/JSON/URDF un cita konfigurācija tiek versēta kopā ar kodu.
- Vērtēšanas dokumenti: glabā uzdevuma versiju, atslēgu/rubriku un skolēna pierādījumu vienā sasaistāmā struktūrā.
- Aparatūras laboratorijās pieraksta izmantoto plates/sensora modeli un būtiskās bibliotēku versijas, ja tās ietekmē rezultātu.

138. 36 nodarbību teacher checklist un minimālais pierādījumu komplekts

Šī karte ir operatīvs kontroles saraksts visam 36 nodarbību pamatkursam. Pilnie runas scenāriji atrodas 37. nodaļā; šeit katrai nodarbībai ir norādīts, kas obligāti jāsagatavo un kādam novērojamam rezultātam jāpaliek pēc 90 minūtēm.

138.1. Teacher checklist — Algoritmi

Nr.	Tēma	Pirms stundas obligāti sagatavot	Minimālais pierādījums pēc stundas
1	Kas ir algoritms un kā domā dators?	Automātisko durvju piemērs; ievade/apstrāde/izvade shēma	8–12 soļu algoritms + identificētas I/O
2	Secība, zarošanās un atkārtotās	3 īsi pseidokoda piemēri; IF/loop kartītes	Pareizi atpazītas 3 pamatstruktūras + pseidokods
3	Blokshēmas	Simbolu lapa; viens kļūdainais piemērs	Blokshēma ar sākumu/beigām, lēmumu un plūsmas bultām
4	Mainīgie un datu tipi	Datu tipu piemēri no sensoriem	Mainīgo tabula: nosaukums, tips, vienība, piemēra vērtība
5	Loģika un nosacījumi	AND/OR/NOT piemēri; interlock scenārijs	Patiesības nosacījums + 3 testa gadījumi
6	Cikli un atkārtota sistēmas darbība	FOR/WHILE piemēri; timeout/watchdog scenārijs	Cikla pseidokods + bezgalīga cikla riska skaidrojums
7	Dekompozīcija, moduļi un sistēmas domāšana	Functional tree piemērs	Functional tree ar vismaz 3 galvenajiem zariem
8	Pārbaudes darbs: algoritmi	Izdrukāts/elektronisks variants; atslēga; laika plāns	Pārbaudes darbs + vērtēšanas pieraksts

138.2. Teacher checklist — Python

Nr.	Tēma	Pirms stundas obligāti sagatavot	Minimālais pierādījums pēc stundas
9	Python vide, programmas izpilde un datu plūsma	Darbojoša Python vide; starter fails	Programma ar input/print, mainīgo un aprēķinu
10	Python nosacījumi, Boolean loģika un validācija	3 validācijas piemēri	Programma ar if/elif/else + nederīgas ievades apstrādi
11	Python cikli, datu kopas, debugging un testēšana	Kļūdainais kods; neliela datu kopa	Cikla risinājums + atrasta/labota kļūda + testi
12	Python mini projekts	Prasību lapa; projekta rubrika	Darbojošs mini projekts + īsa demonstrācija

138.3. Teacher checklist — C++

Nr.	Tēma	Pirms stundas obligāti sagatavot	Minimālais pierādījums pēc stundas
13	C++ struktūra, kompilācija un pirmā programma	Kompilators/IDE; hello/sensor piemērs	Kods tiek kompilēts; students izskaidro compile/run

Nr.	Tēma	Pirms stundas obligāti sagatavot	Minimālais pierādījums pēc stundas
14	C++ datu tipi, operatori, const un precizitāte	Tipu/robežu piemēri	Aprēķins ar pamatotu tipu izvēli un const
15	C++ zarošanās, switch un sistēmas stāvokļi	State piemērs	if/switch risinājums + stāvokļu tabula
16	C++ cikli, izvēlnes un atkārtota procesa vadība	Izvēlnes starter kods	Atkārtota izvēlne ar korektu exit nosacījumu
17	Funkcijas, parametri un programmas dekompozīcija	Monolīts kods refaktorēšanai	Vismaz 2 funkcijas ar skaidriem parametriem/return
18	Masīvi, mērījumu datu apstrāde un algoritmi	Mērījumu masīvs	min/max/average vai meklēšana + testa dati
19	OOP ievads: class, object un tehniska komponente	Sensor klases skelets	Klase ar datiem, metodi un vienu validācijas noteikumu
20	C++ pārbaudes darbs un debugging	Variants; atslēga; kļūdaini fragments	Pārbaudes darbs + debugging pierādījums

138.4. Teacher checklist — SQL/IoT/integrācija

Nr.	Tēma	Pirms stundas obligāti sagatavot	Minimālais pierādījums pēc stundas
21	SQL pamati tehnisko mērījumu datiem	SQLite/DB vide; measurements shēma	CREATE/INSERT/SELECT vai sagatavota DB + 3 vaicājumi
22	CSV, JSON un Python + SQLite	CSV/JSON paraugi	Dati nolasīti/pārveidoti/saglabāti + validācija
23	IoT, MQTT, HTTP un mākoņdatu arhitektūra	Topic/API piemērs; JSON payload	Datu plūsmas shēma + topic/endpoint + payload
24	Integrācijas mini darbs	Starter repo/mapes; integrācijas prasības	Sensors/simulācija -> kods -> DB/dashboard pierādījums

138.5. Teacher checklist — Noslēguma projekts

Nr.	Tēma	Pirms stundas obligāti sagatavot	Minimālais pierādījums pēc stundas
25	Problēma, lietotājs un prasības	Projekta brief veidne	Problēmas formulējums + 5–10 pārbaudāmas prasības
26	Functional tree, arhitektūra un testu stratēģija	Functional tree un traceability veidnes	Functional tree + arhitektūras shēma + testu saraksts
27	Python realizācija I — kodols	Repo struktūra; coding checklist	Pirmais darbojošais Python kodols + commit/versija
28	Python realizācija II — dati, testi un vizualizācija	Testu tabula; grafika paraugs	Testi + datu rezultāts/grafiks + kļūdu pieraksts
29	C++ realizācija I — pārnese/low-level komponente	C++ skeleton; interfeisa līgums	Kompilējama C++ komponente ar definētu I/O
30	C++ realizācija II — funkcijas/OOP un testi	Code review checklist	Refaktorēts kods + testi + review piezīme

Nr.	Tēma	Pirms stundas obligāti sagatavot	Minimālais pierādījums pēc stundas
31	IoT/robotikas/sensora integrācija	Interfeisu un drošības checklist	Integrēta datu/komandu plūsma + vismaz 1 atteices tests
32	Testēšana, validācija un atteices scenāriji	Test plan veidne	Expected/Actual/PASS-FAIL + robežgadījumi/atteices
33	Dokumentācija un reproducējamība	README/requirements/template	README + versijas + palaišanas instrukcija
34	Pilnveidošana un tehniskais review	Rubrika; peer-review forma	Defektu saraksts -> labojumi -> atkārtots tests
35	Aizstāvēšana I — demonstrācija	Demo secība; laika taimeris	Dzīva demonstrācija + pierādījumu pakete
36	Aizstāvēšana II, refleksija un profesionālās attīstības ceļš	Refleksijas un portfolio forma	Aizstāvēšana + pašvērtējums + nākamais mācību solis

138.6. Teacher checklist — pirms katras nodarbības

- Vai sasniedzamais rezultāts ir formulēts kā darbība, ko students var demonstrēt?
- Vai demonstrācijas kods/fails ir pārbaudīts uz konkrētā datora/vides?
- Vai ir A minimums, B standarta uzdevums un C paplašinājums?
- Vai ir sagatavots neveiksmīgs/klūdainš piemērs debugging diskusijai?
- Vai ir skaidrs, ko students iesniedz vai parāda kā pierādījumu?
- Vai fiziskā laboratorijā ir pārbaudīta barošana, GND, signālu līmeņi un kustīgo daļu drošība?
- Vai ir rezerves variants, ja internets, aparatūra vai ārējais serviss nav pieejams?

138.7. Teacher checklist — pēc katras nodarbības

- Atzīmē, kuri sasniedzamie rezultāti bija sasniegti un kuri jāpārceļ uz nākamo stundu.
- Saglabā 1–3 anonimizētus tipiskus kļūdu piemērus nākamā gada mācību materiālam.
- Atjauno starter/etalona failu tikai tad, ja izmaiņas tiešām uzlabo mācību procesu; pieraksti versiju.
- Ja vairākums kļūdās vienā punktā, labo skaidrojumu vai uzdevuma formulējumu, nevis tikai “palielini skaidrošanas laiku”.
- Pieraksti laboratorijas aparatūras/bibliotēku problēmas, lai tās var novērst pirms nākamās grupas.

139. Vienota 90 minūšu lesson-delivery karte un nodarbību veidnes

Vienota struktūra samazina pasniedzēja slodzi un dod studentiem paredzamu darba ritmu. Laiki ir orientējoši; tos var mainīt atkarībā no grupas, taču katrai nodarbībai jāatstāj laiks gan aktīvai praksei, gan rezultāta pārbaudei.

139.1. Veidne A — teorija + vadīta prakse

Laiks	Posms	Pasniedzēja darbība	Studenta darbība
0–10 min	Aktivizācija	2–3 jautājumi par iepriekšējo tēmu; tehnisks konteksts	Atbild, prognozē, formulē iepriekšējās zināšanas
10–30 min	Jaunā teorija	Skaidro jēdzienu ar vienu konsekventu sistēmas piemēru	Pieraksta, uzdod jautājumus, skaidro ar saviem vārdiem
30–45 min	Demonstrācija	Rāda algoritmu/kodu/shēmu un komentē lēmumus	Seko, paredz nākamo soli, atrod riskus
45–75 min	Patstāvīga prakse	Konsultē; nedod gatavu risinājumu pirmajā pieprasījumā	Izpilda A/B uzdevumu; C paplašinājums pēc vajadzības
75–85 min	Testēšana	Vada 2–3 testa gadījumus un kļūdu analīzi	Pārbauda rezultātu; labo kļūdas
85–90 min	Exit ticket	Uzdod vienu “kāpēc?” un vienu “kas notiks, ja?”	Īsi formulē izpratni

139.2. Veidne B — fiziska/datorlaboratorija

Laiks	Posms	Minimums
0–10	Drošība + mērķis	I/O, barošana, kustība, avārijas apturēšana, datu faili
10–20	Sistēmas pārbaude	Savienojumi, versijas, ping/topic/sensoru testa minimums
20–60	Eksperiments	Viena mainīgā princips; mērījumi; Expected/Actual
60–75	Analīze	Grafiks/tabula, kļūda, residuals vai salīdzinājums
75–85	Atteices tests	Invalid input, sensor disconnect, timeout vai cits drošs scenārijs
85–90	Pierādījumu saglabāšana	Fails + konfigurācija + rezultāts + īss secinājums

139.3. Veidne C — pārbaudes darbs

Laiks	Darbība
0–5	Noteikumi, varianta piešķiršana, atļautie materiāli.

Laiks	Darbība
5–65	Individuālais darbs. Pasniedzējs skaidro tikai uzdevuma tehnisko formulējumu, ne risinājumu.
65–80	Praktiskās daļas palaišana/testēšana vai īsa koda/blokslēmas pārbaude.
80–87	Darba iesniegšana; failu nosaukumi/versijas; pierādījumu pārbaude.
87–90	Īsa refleksija: kura daļa bija sarežģītākā un kā students pārbaudīja savu rezultātu?

139.4. Veidne D — projekta sprinta nodarbība

Laiks	Darbība	Obligātais rezultāts
0–10	Stand-up: ko izdarīju / ko darīšu / kas bloķē	1 konkrēts sprinta mērķis
10–20	Pasniedzēja mini review vai tehnisks skaidrojums	Lēmums / prioritāte
20–65	Izstrāde, testēšana, mērījumi	Kods/modelis/konfigurācija/testa rezultāts
65–80	Integrācija vai code review	Defektu un nākamo darbību saraksts
80–90	Commit/evidence + īsa demonstrācija	Versēts pierādījums un nākamais solis

139.5. Diferenciācijas noteikums A/B/C

Līmenis	Kam paredzēts	Prasības raksturs	Pasniedzēja atbalsts
A — minimums	Studentam, kam nepieciešama strukturēta palīdzība	Viens jēdziens, skaidri dati, paredzams rezultāts	Starter fails, soļu saraksts, 1–2 norādes
B — standarts	Kursa obligātais līmenis	Jāapvieno 2–3 jēdzieni un jātestē	Dokumentācija un jautājumi, ne gatavs kods
C — paplašinājums	Ātrākajiem / universitātes līmenim	Atvērts uzdevums, alternatīvi risinājumi, optimizācija vai fiziska sistēma	Pasniedzējs darbojas kā reviewer/mentors

140. Vērtēšanas sistēmas normalizācija un semestra/gada modelis

Šī sadaļa piedāvā konsekventu vērtēšanas modeli, ko iespējams pielāgot konkrētās izglītības iestādes noteikumiem. Modelis apzināti nevērtē “aktivitāti” vai klātbūtni kā zināšanu aizvietotāju — punktus piešķir par pierādāmu kompetenci.

Svarīgi: procentu sadalījums un atzīmju pārvēršana ir metodisks paraugs, nevis normatīva prasība. Pirms kursa apstiprināšanas tas jāsaskaņo ar iestādes vērtēšanas kārtību.

140.1. Gada vērtējuma ieteicamā struktūra

Komponente	Svars	Ko tā pierāda
Zināšanu/pārbaudes darbi	20%	Algoritmi, C++ un nepieciešamā teorētiskā izpratne
Praktiskie/laboratorijas darbi	25%	Spēja realizēt, mērīt, testēt un atklādot
Mini projekti un integrācijas darbi	20%	Spēja apvienot vairākas tēmas vienā sistēmā
Noslēguma patstāvīgais projekts	30%	Pilns cikls: prasība -> arhitektūra -> realizācija -> tests -> aizstāvēšana
Dokumentācija, reproducējamība un refleksija	5%	Versijas, README, testu protokols, secinājumi

140.2. 1. semestra ieteicamā struktūra

Darbs	Svars semestrī
Algoritmu pārbaudes darbs (N8)	30%
Python mini projekts (N12)	30%
Darba lapas/laboratorijas/praktiskie uzdevumi	30%
Testēšanas un dokumentācijas pierādījumi	10%

140.3. 2. semestra ieteicamā struktūra

Darbs	Svars semestrī
C++ pārbaudes darbs (N20)	15%
SQL/IoT integrācijas darbs (N24)	15%
Praktiskie/laboratorijas darbi	20%
Noslēguma projekts (N25–N36)	45%
Dokumentācija un profesionālā refleksija	5%

140.4. Universāla 4 līmeņu rubrika

Līmenis	Raksturojums	Tipisks pierādījums
4 — pārliecinoši	Patstāvīgi realizē, testē, pamato izvēli un atrod ierobežojumus	Darbojas + testi + skaidrojums + robežgadījumi
3 — sasniegts	Pareizi realizē obligāto funkcionalitāti un veic pamatpārbaudi	Darbojas + vismaz pamata testi
2 — daļēji	Ir pareiza pieeja, bet risinājums ir nepilnīgs vai nestabils	Daļēji darbojas; students spēj norādīt kļūdu
1 — vēl nav sasniegts	Nav pierādīta prasītā kompetence vai students nespēj izskaidrot risinājumu	Nav reproducējama rezultāta / nav izpratnes pierādījuma

140.5. Vērtēšanas pierādījumu hierarhija

1. Spēcīgākais pierādījums: students pats demonstrē risinājumu, testu un izskaidro “kāpēc”.
2. Labs pierādījums: kods/shēma + Expected/Actual rezultāti + versija.
3. Papildu pierādījums: darba lapa, grafiks, SQL rezultāts, rosbag, mērījumu protokols, commit vēsture.
4. Vājāks pierādījums: tikai ekrānattēls bez atkārtojamas procedūras.
5. Nepietiekams pierādījums: tikai gatavs kods bez spējas to izskaidrot vai reproducēt.

140.6. Kavēta/neizpildīta darba atjaunošanas princips

- Studentam jāatjauno tas pats sasniedzamais rezultāts, ne obligāti identisks uzdevums.
- Praktiskam darbam var izmantot citu datu kopu, sensoru vai kontekstu, saglabājot vienādu grūtības līmeni.
- Atkārtotā darbā pasniedzējs saglabā atsevišķu varianta identifikatoru.
- Ja darbs ir projekts, jāiesniedz versija + izmaiņu saraksts + atkārtoti testi, ne tikai “jauns gala fails”.

141. Pārbaudes darbu variantu matrica un godīgas vērtēšanas principi

Variantiem jāmaina konteksts un skaitliskie dati, bet ne sasniedzamais rezultāts. Tas samazina tiešu kopēšanu un saglabā salīdzināmu grūtības līmeni.

141.1. Algoritmu pārbaudes darba variantu matrica

Prasmes bloks	A variants	B variants	C variants	D variants
I/O un secība	Automātiskās durvis	Autostāvvietas barjera	Konveijers	Tvertnes uzpilde
IF/ELSE	Temperatūra	Līmenis	Attālums	Spiediens
Cikls	10 mērījumi	5 detaļas	Atkārtot līdz sensoram	Skaitīt līdz distancei
Blokslēma	Ventilators	Sūknis	Signāllampa	Motora aizsardzība
Robežgadījums	Sensors nav derīgs	Nav brīvu vietu	Timeout	E-stop aktīvs

141.2. Python pārbaudes/mini projekta variantu matrica

Variants	Konteksts	Obligāti jāparāda
P-A	Temperatūras monitorings	input/validācija + if + cikls + min/max/avg
P-B	Noliktavas detaļu kvalitāte	saraksts + klasifikācija + brāķa % + tests
P-C	Enerģijas patēriņš	datu kopa + robeža + kopsavilkums + CSV paplašinājums
P-D	Motora stāvoklis	state/threshold + while/for + kļūdaina sensora apstrāde

141.3. C++ pārbaudes darba variantu matrica

Variants	Galvenais uzdevums	Papildu debugging fragments
C-A	Sensoru state machine ar enum/switch	Nepareiza salīdzināšana vai trūkstošs break
C-B	Mērījumu masīvs + funkcijas	Integer division / nepareizs tips
C-C	Izvēlne ar while + funkcijām	Cikls nebeidzas / nepareizs exit
C-D	Sensor klase ar validāciju	Neinicializēts lauks / nepareizs const

141.4. SQL/IoT varianti

Variants	Datu avots	SQL prasība	IoT prasība
I-A	Temperatūra/mitrums	WHERE + ORDER BY + AVG	MQTT topic + JSON
I-B	Motora vibrācija	WHERE + MAX + laika filtrs	HTTP POST + timestamp
I-C	Enerģijas patēriņš	GROUP BY + SUM/AVG	MQTT + quality/status
I-D	Robota distance	WHERE + COUNT + robeža	Telemetry + stale-data scenārijs

141.5. Pārbaudes darba kvalitātes kontrolesaraksts

- Visiem variantiem ir vienāds punktu skaits un vienādas sasniedzamo rezultātu kategorijas.
- Nevienā variantā nav nepieciešamas zināšanas, kas nav mācītas obligātajā kursa līmenī.
- Praktiskajai daļai ir vismaz viens redzams testa rezultāts, ne tikai koda esamība.
- Atslēga/rubrika ir sagatavota pirms darbu labošanas.
- Ja izmanto AI rīkus vai dokumentāciju, noteikumi ir vienādi visiem un ir skaidri norādīti pirms darba.

142. Laboratoriju atbilžu, etalonu un verifikācijas indekss

Šis indekss pasniedzējam ļauj ātri atrast, kur lielajā dokumentā atrodas skolēna darba lapa, etalons vai pārbaudes fokuss. Dažās laboratoriju grupās viens un tas pats identifikators tiek lietots citā tematiskā kontekstā; tādēļ norādīta arī nodaļa.

Laboratoriju grupa	Temats	Kur meklēt atbildi/etalonu	Pasniedzēja verifikācijas fokuss
DL1–DL12 (21. nodaļa)	Algoritmi/Python/C++/SQL/IoT/robotika	Darba lapas + diferencētie uzdevumi	Pārbaudīt domāšanas procesu, ne tikai atbildi
L1–L14	Programmēšana, dati, kinemātika, LiDAR, CV, IMU, A*	27. nodaļā studentu lapas + pasniedzēja versijas	Kods + Expected/Actual + skaidrojums
H1–H12	Arduino/ESP32/sensori/motori/robots	41. nodaļa darba lapas; 42. nodaļa etalona kodi	Barošana, I/O, mērījumi, drošība, kalibrācija
R1–R3	ROS 2 /odom, /imu, /scan	44. nodaļa	Frame, timestamp, topic un datu jēga
R4–R9	EKF, SLAM, Nav2, BT, rosbag	45.–48., 53.–54. nodaļa	Konfigurācija + TF + metrikas + reproducējamība
U1 / S1 / CI1 / V1 / P1	URDF, simulācija, CI, visual odometry, point cloud	55.–60. nodaļa	Interfeisi un verifikācijas pierādījums
FG1–OBS1	Factor graph līdz observability/fault injection	63.–72. nodaļa	Ekspérimenta metrika un kļūdu analīze
SS1–MTC1	State-space, ID, LQR, MPC, dinamika, MoveIt 2	77.–84. nodaļa	Modelis + simulācija + kontroles/plānošanas pierādījums
AI-DL1–AI-DL9	Dataset, CNN, detection, segmentation, ROS AI, edge	88.–100. nodaļa	Dataset/model version + metrikas + failure set
SAFE1–SAFE3 / DT1–DT5	Riski, safety funkcijas, Digital Twin	108.–119. nodaļa	Traceability, residuals, freshness, fault scenario

142.1. Etalona risinājuma lietošanas noteikums

- Etalons nav vienīgais pareizais risinājums, ja students sasniedz prasības ar citu korektu arhitektūru.
- Salīdzinot kodu, vērtē I/O, datu tipus, kļūdu apstrādi, testus un izskaidrojumu, ne tikai identisku sintaksi.
- Aparatūras laboratorijā etalona pin mapping nedrīkst automātiski pārnest uz citu plates/moduļa revīziju bez pārbaudes.
- ROS 2/AI/industrial konfigurācijās bibliotēku vai pakotņu versijas var mainīt detaļas; jāvērtē princips un dokumentēta konfigurācija.
- Ja students izmanto AI ģenerētu kodu, viņam jāspēj izskaidrot kritiskos blokus un demonstrēt paša veiktus testus.

142.2. Ātrais “PASS” jautājumu komplekts laboratorijai

1. Vai rezultāts ir reproducējams pēc restartēšanas?
2. Vai ievades un izvades vienības ir skaidras?

3. Vai ir vismaz viens testa gadījums ar paredzamo rezultātu?
4. Vai students var parādīt, kas notiek nederīgas/robežvērtības gadījumā?
5. Vai konfigurācija/kods ir saglabāts ar identificējamu versiju?
6. Vai students spēj izskaidrot vienu būtisku tehnisko izvēli?

143. Noslēguma projekta gala rubrika un aizstāvēšanas protokols

Noslēguma projekts ir kompetenču integrācijas darbs. Visiem projektiem nav obligāti jāizmanto identiskas tehnoloģijas, tādēļ rubrika nodala universālo inženiertehnisko kodolu no izvēlētās tehniskās specializācijas.

143.1. 100 punktu gala rubrika

Kritērijs	Punkti	Ko vērtē
Problēma un prasības	10	Skaidrs lietotājs/problēma; pārbaudāmas funkcionālās un nefunkcionālās prasības
Functional Tree un arhitektūra	10	Funkciju dekompozīcija, I/O, komponentes, datu/komandu plūsma
Realizācijas kvalitāte	20	Koda struktūra, nosaukumi, tipi, funkcijas/OOP, interfeisi, lasāmība
Izvēlētās tehnoloģijas dziļums	15	Python/C++/SQL/IoT/robotika/AI/control u.c. atbilstoši projekta scope
Testēšana un validācija	15	Test plan, robežgadījumi, Expected/Actual, atkārtota pārbaude pēc labojuma
Robustums, atteices un drošība	10	Invalid/stale/disconnect/timeout/recovery; drošības riski atbilstoši projektam
Dokumentācija un reproducējamība	10	README, versijas, konfigurācijas, palaišanas instrukcija, pierādījumu mape
Demonstrācija un aizstāvēšana	10	Stabila demo secība; spēja pamatot lēmumus un atbildēt uz jautājumiem

143.2. Tehnoloģiju dziļuma 15 punktu izvēles bloks

Virziens	Pilna līmeņa piemērs
Python/dati	Modulārs risinājums + datu validācija + testi + vizualizācija/DB
C++/embedded	Tipi/const + funkcijas/OOP + hardware/state-machine + timing/robustums
SQL/IoT	Datu modelis + query + MQTT/HTTP + timestamp/quality + offline/error scenārijs
Robotika	Frames/kinematics + sensors + planning/control + recovery/testi
AI/Vision	Dataset/model version + metrics + inference + failure set + robot integration
Vadības teorija	Modelis + regulatora izvēle + metrikas + stabilitātes/ierobežojumu diskusija

143.3. Aizstāvēšanas 10–12 minūšu protokols

Laiks	Saturs
0–1 min	Problēma, lietotājs, mērķis.
1–3 min	Functional Tree / arhitektūra / galvenie interfeisi.
3–6 min	Dzīva demonstrācija vai reproducējams simulācijas eksperiments.
6–8 min	Testi, atteices scenārijs, metrikas un būtiskākā atrastā kļūda.
8–10 min	Tehniskie lēmumi, ierobežojumi, ko mainītu nākamajā versijā.
10–12 min	Pasniedzēja jautājumi un viena neparedzēta robežgadījuma diskusija.

143.4. Aizstāvēšanas jautājumu minimums

- Kura ir sistēmas kritiskākā ievade un kas notiek, ja tā kļūst nederīga?
- Kāpēc izvēlējies tieši šo datu tipu/algoritmu/protokolu/komponenti?
- Parādi vienu testu, kas sākumā neizdevās. Ko tu mainīji?
- Kā var reproducēt tavu rezultātu uz cita datora vai nākamajā nedēļā?
- Kur ir lielākais tehniskais risks vai ierobežojums?
- Kuru funkciju tu pievienotu nākamajā versijā un kā to testētu?

143.5. Autentiskuma un AI izmantošanas pārbaude

- Students drīkst izmantot dokumentāciju un AI rīkus saskaņā ar kursa noteikumiem, taču iesniedzamais rezultāts ir jāizprot.
- Pasniedzējs var lūgt izskaidrot 5–15 rindas no paša iesniegtā koda vai mainīt vienu prasību demonstrācijas laikā.
- Ja students nespēj izskaidrot kritisko risinājuma daļu, vērtējums par šo kompetenci tiek atlikts līdz atkārtotai demonstrācijai.
- AI izmantošana jānorāda projekta dokumentācijā: kādam uzdevumam tika lietots rīks, ko students pārbaudīja un ko mainīja.

144. Kursa apstiprināšanas kopsavilkums un ieviešanas pakete

Šo sadaļu var izmantot kā īso pielikumu kursa programmas apspriešanai vai apstiprināšanai. Tā apkopo kursa obligāto kodolu, nevis visu paplašināto 500+ lappušu rokasgrāmatas saturu.

144.1. Kursa pases projekts

Lauks	Saturs
Kursa nosaukums	Programmēšanas un datorizētu sistēmu pamati
Mērķauditorija	Tehniskās/profesionālās izglītības audzēkņi pēc 9. klases; materiāls paplašināms universitātes ievadlīmenim
Apjoms	36 nodarbības x 90 min = 54 astronomiskās stundas = 72 akadēmiskās stundas pa 45 min
Norise	1 nodarbība nedēļā; teorija, vadīta prakse, laboratorijas, pārbaudes darbi un projekti
Pamatvalodas/tehnoloģijas	Python, C++, SQL, IoT datu plūsmas; paplašinājumā mehatronika, robotika, ROS 2, AI, Digital Twin
Priekšzināšanas	Datora pamatlietošana; pamata matemātiskās darbības; nav obligāta iepriekšēja programmēšanas pieredze
Galvenā metodika	Problēma -> algoritms -> realizācija -> tests -> dokumentācija -> aizstāvēšana
Noslēguma rezultāts	Patstāvīgs tehnisks projekts ar Python/C++ vai atbilstošu sistēmas komponenti, testiem un dokumentāciju

144.2. Obligātā 72 akadēmisko stundu struktūra

Bloks	90 min nodarbības	45 min akadēmiskās stundas	Galvenais rezultāts
Algoritmi	8	16	Algoritmiskā domāšana, blokslēma, loģika, cikli, dekompozīcija
Python	4	8	Programmas pamati, validācija, cikli, mini projekts
C++	8	16	Kompilācija, tipi, nosacījumi, cikli, funkcijas, masīvi, OOP
SQL/IoT/integrācija	4	8	Datu glabāšana, formāti, komunikācija, sistēmas datu plūsma
Noslēguma projekts	12	24	Prasības, Functional Tree, Python/C++, integrācija, testi, dokumentācija, aizstāvēšana

144.3. Sasniedzamo rezultātu īsa komplekts

- Izskaidro algoritma jēdzienu un izstrādā algoritmu ar secību, zarošanos un ciklu.
- Izveido un testē vienkāršu Python programmu ar validētu ievadi un strukturētu datu apstrādi.
- Izveido un atklādo C++ programmu, korekti izmantojot datu tipus, funkcijas un pamata OOP.

- Izmanto SQL/CSV/JSON un izskaidro IoT sistēmas datu plūsmu no sensora līdz glabāšanai/vizualizācijai.
- Dekomponē tehnisku sistēmu Functional Tree un sasaista funkcijas ar arhitektūru un testiem.
- Plāno, realizē, testē, dokumentē un aizstāv patstāvīgu programmatūras/datorizētas sistēmas projektu.
- Atpazīst datu kvalitātes, drošības, atteices un reproducējamības riskus atbilstoši apgūtajam līmenim.

144.4. Mācību metodes

Metode	Pielietojums
Īsa lekcija/demonstrācija	Jauna jēdziena ieviešana ar tehnisku piemēru
Vadīta programmēšana	Pirmā sintakses/algoritma realizācija kopā ar pasniedzēju
Problem-based learning	No reālas sistēmas problēmas līdz algoritmam un testam
Laboratorija/eksperiments	Sensori, dati, vadība, IoT, simulācija vai robotika
Debugging uzdevumi	Kļūdu diagnostika, nevis tikai koda rakstīšana no nulles
Projektu darbs	Prasības, arhitektūra, izstrāde, testēšana, dokumentācija, aizstāvēšana
Peer/code review	Tehniska argumentācija un kvalitātes kritēriji

144.5. Nepieciešamie resursi — obligātais minimums

- Dators katram studentam vai pārim ar Python un C++ izstrādes vidi.
- SQLite vai līdzvērtīga lokāla SQL vide; teksta/CSV/JSON redaktors.
- Git ir ļoti vēlams, taču pirmajās nedēļās var izmantot vienkāršu failu versēšanu.
- IoT/robotikas fiziskā aparatūra nav obligāta pamatkursa apstiprināšanai — attiecīgos uzdevumus var simulēt.
- Paplašinātajām laboratorijām: Arduino/ESP32, sensori, motora draiveris, enkoderi, IMU, 2D LiDAR, kamera un 2WD platforma pēc pieejamības.

144.6. Starppriekšmetu saites

Joma	Saikne ar kursu
Matemātika	Loģika, funkcijas, koordinātas, vektori, matricas, statistika, kļūda
Fizika/elektronika	Spriegums, strāva, sensori, motori, signāli, mērījumu kļūda
Mehatronika	Sensors -> controller -> actuator -> process -> feedback
Robotika	Kinematika, uztvere, lokalizācija, plānošana, vadība
Datu inženierija	Datu formāti, SQL, timestamps, kvalitāte, mākoņdatu plūsmas
Programmatūras inženierija	Prasības, Git, testi, CI, dokumentācija, reproducējamība

Joma	Saikne ar kursu
Darba/drošības kultūra	Riska domāšana, fail-safe principi, verificējama sistēmas uzvedība

145. Pasniedzēja sagatavošanās, kvalitātes kontrole un kursa uzturēšana

Kursa kvalitāti nosaka ne tikai saturs, bet arī tas, vai nākamajā gadā ir iespējams atkārtot to pašu mācību procesu ar skaidrām versijām, sagatavotu vidi un uz pierādījumiem balstītiem uzlabojumiem.

145.1. Pirms mācību gada

1. Pārbaudi 36 nodarbību kalendāru un skolas brīvlaikus; iezīmē rezerves nedēļas.
2. Pārbaudi Python/C++/SQLite vidi uz studentu datoriem vai sagatavo instalācijas instrukciju.
3. Izveido kursa mapju/repository struktūru un studentu iesniegšanas noteikumus.
4. Pārskati pārbaudes darbu variantus un aizstāj iepriekš izmantotus publiski pieejamus uzdevumus, ja nepieciešams.
5. Fiziskajām laboratorijām inventarizē plātes, sensorus, vadus, barošanas avotus un rezerves detaļas.
6. Veic 1 pilnu dry-run vismaz kritiskajām laboratorijām: motors/enkodērs, MQTT, LiDAR/kamera, ROS 2, ja tās paredzētas konkrētajai grupai.
7. Saskaņo vērtēšanas procentus un atkārtotas vērtēšanas principu ar iestādes noteikumiem.

145.2. Pirms katra semestra

- Nosaki, kuri paplašinātie moduļi konkrētajai grupai būs obligāti, izvēles vai tikai demonstrācija.
- Pārbaudi studentu priekšzināšanu līmeni ar īsu diagnosticējošu uzdevumu bez atzīmes.
- Pielāgo A/B/C uzdevumu proporciju grupas līmenim.
- Izvēlies 1–2 konsekvēntus tehniskos kontekstus, ko atkārtoti izmantot (piem., temperatūras vadība, konveijers, 2WD robots), lai nesadrumstalotu uzmanību.

145.3. Kursa kvalitātes metrikas

Metrika	Ko novērot	Iespējamā reakcija
N8 algoritmu rezultāti	Kurš jēdziens rada visvairāk kļūdu	Pievienot/uzlabot piemēru pirms Python
N12 Python projekts	Cik studentu nespēj validēt ievadi/testēt	Pastiprināt debugging/testēšanas mini uzdevumus
N20 C++ rezultāti	Tipu, funkciju, ciklu vai OOP kļūdu sadalījums	Pārplānot nākamā gada C++ tempu
N24 integrācija	Vai studenti saprot datu plūsmu, ne tikai atsevišķus rīkus	Pievienot vairāk architecture/data-contract darba
Noslēguma projekts	Cik projekti ir reproducējami un testēti	Stiprāk ieviest README/test plan jau N25–N28

Metrika	Ko novērot	Iespējamā reakcija
Laboratoriju atteices	Aparatūras/bibliotēku problēmu biežums	Standartizēt komplektu, versijas vai simulācijas backup

145.4. Izmaiņu kontroles ieraksta paraugs

Datums	Versija	Nodaļa/ laboratorija	Izmaiņa	Iemesls	Pārbaudīja
—	—	—	—	—	—
—	—	—	—	—	—
—	—	—	—	—	—

145.5. Kad jāpārskata tehniskā informācija

- Mainās izmantotās programmatūras/ROS 2/pakotņu API vai atbalstītās versijas.
- Mainās aparatūras modelis, plates revīzija vai sensora bibliotēka.
- Mainās oficiāls drošības standarts vai iestādes drošības prasība, uz kuru materiāls atsaucas.
- Laboratorija vairs nav reproducējama uz tipiskās studentu vides.
- Studentu rezultāti sistemātiski rāda, ka formulējums rada nevajadzīgu neskaidrību.
- Tiek pievienota jauna tehnoloģija — tai jābūt skaidram mācību rezultātam, ne tikai “modernai tēmai”.

145.6. Pasniedzēja kursa mapes ieteicamā struktūra

```

course_master/
  00_programma/
  01_teacher_guides/
  02_student_handouts/
  03_assessments/
  04_answer_keys/
  05_code_starters/
  06_reference_solutions/
  07_lab_configs/
  08_project_templates/
  09_examples_anonymized/
  10_change_log/
  archive/

```

146. [ARHĪVS] Versijas 1.6 statuss un pāreja uz v2.0 FINAL

146.1. Kas pievienots v1.6

- Vienots pasniedzēja darba princips un dokumentu kontroles minimums.
- 36 nodarbību teacher checklist ar obligāto sagatavošanu un pierādījumu.
- Četras 90 min lesson-delivery veidnes un A/B/C diferenciācijas noteikums.
- Normalizēts gada un semestru vērtēšanas modelis ar 4 līmeņu rubriku.
- Algoritmu, Python, C++, SQL/IoT pārbaudes darbu variantu matricas.
- Laboratoriju etalonu/verifikācijas indekss.
- 100 punktu noslēguma projekta rubrika un aizstāvēšanas protokols.
- Kursa apstiprināšanas īsā pase: 36 x 90 min = 72 akadēmiskās stundas.
- Pasniedzēja sagatavošanās, kvalitātes metriku un izmaiņu kontroles sistēma.

146.2. Kas paliek v2.0 FINAL kā redakcionāls darbs

1. Pārbaudīt un vienādot visu dokumenta satura rādītāju, nodaļu numerāciju un iekšējās atsauces.
2. Normalizēt tabulu, attēlu, kodu bloku un pasniedzēja piezīmju vizuālo stilu visā dokumentā.
3. Izveidot vienotu avotu/literatūras sarakstu un atdalīt normatīvos/versijjutīgos avotus no mācību literatūras.
4. Veikt gala terminoloģijas un valodas redakciju (Latviešu/English technical terms).
5. Atdalīt studentam izsniedzamo saturu no pasniedzēja atbildēm ar skaidriem marķējumiem, nezaudējot vienotā master dokumenta principu.
6. Veikt gala accessibility/layout pārbaudi un sagatavot publicējamo v2.0 FINAL kopiju.

146.3. Satura iesaldēšanas princips

No v1.6 līdz v2.0 FINAL jaunas lielas tehniskās nozares netiek pievienotas, ja vien tās neaizstāj novecojušu saturu vai nav nepieciešamas kursa obligāto sasniedzamo rezultātu nodrošināšanai. Galvenais uzdevums ir padarīt esošo materiālu konsekventu, pārskatāmu, pārbaudāmu un gatavu ilgtermiņa lietošanai.

Pasniedzējam: šajā brīdī master dokuments jau ir daudz plašāks par viena gada 72 akadēmisko stundu pamatkursu. Kursa apstiprināšanai izmanto 144. nodaļas kodolu; pārējās nodaļas ir metodiskais, profesionālais un universitātes paplašinājuma resurss.

147. v2.0 FINAL — avoti, standartu statuss un versiju vēsture

Šī sadaļa fiksē gala redakcijas pārbaudes datumu, galvenos primāros avotus un principu, kā uzturēt materiālu aktuālu. Programmatūras bibliotēkas, ROS 2 distribūcijas, drošības standarti un normatīvais regulējums laika gaitā mainās; pirms profesionāla projekta jāizmanto konkrētajā datumā spēkā esošā oficiālā dokumentācija.

147.1. Galvenie standarti un normatīvie avoti — pārbaudīts 18.08.2026

Avots	Tēma	Statuss/piezīme	Oficiālā saite
ISO 10218-1:2025	Robotics — Safety requirements — Part 1: Industrial robots	Publicēts; aizstāj 2011. gada izdevumu	Oficiālais avots
ISO 10218-2:2025	Robotics — Safety requirements — Part 2: Industrial robot applications and robot cells	Publicēts	Oficiālais avots
ISO 13849-1:2023	Safety-related parts of control systems — General principles for design	Publicēts	Oficiālais avots
IEC 62061:2021 + AMD1:2024 + AMD2:2026	Functional safety of safety-related control systems	Konsolidētā versija ar 2026. gada grozījumu	Oficiālais avots
ISO/TS 15066:2016	Collaborative robots	Publicēts, bet atrodas pārskatīšanā/aizvietošanas procesā	Oficiālais avots
ISO 23247-1...4:2021	Digital twin framework for manufacturing	Pamatdaļas: principi, arhitektūra, digitālā reprezentācija, informācijas apmaiņa	Oficiālais avots
ISO 23247-5:2026	Digital thread for digital twin	Publicēts 2026-06	Oficiālais avots
ISO 23247-6:2026	Digital twin composition	Publicēts 2026-07	Oficiālais avots
IEC 63278-1:2023	Asset Administration Shell for industrial applications — Part 1	Publicēts	Oficiālais avots
Regula (ES) 2023/1230	Machinery Regulation	Vispārēji piemērojama no 20.01.2027; pārejas normas jāpārbauda konkrētam projektam	Oficiālais avots

147.2. Programmatūras un platformu primārie avoti

- **Python:** Python 3.14 oficiālā dokumentācija — [oficiālā dokumentācija](#)
- **Git:** Git Reference — [oficiālā dokumentācija](#)
- **ROS 2:** ROS 2 dokumentācija; 2026. gada LTS distribūcija — Lyrical Luth — [oficiālā dokumentācija](#)
- **Nav2:** Navigation2 dokumentācija — [oficiālā dokumentācija](#)
- **MoveIt 2:** MoveIt 2 dokumentācija — [oficiālā dokumentācija](#)
- **OpenCV:** OpenCV dokumentācija — [oficiālā dokumentācija](#)
- **PyTorch:** PyTorch dokumentācija — [oficiālā dokumentācija](#)

147.3. Avotu lietošanas princips

- Standartu nosaukumi un izdevumi šajā dokumentā ir orientieris mācību procesam, nevis standarta pilna teksta aizvietotājs.
- Drošības funkciju, PL/SIL, aizsardzības distanču un robotu šūnu validācijas prasības profesionālā projektā jānosaka ar aktuālo standartu pilno tekstu un kompetentu risku novērtējumu.
- ROS 2 un citu programmatūras ietvaru piemēros pirms laboratorijas pārbauda izmantotās distribūcijas/API versiju; mācību kods var prasīt pielāgošanu jaunākām versijām.
- Kursa materiāla atjaunināšanas laikā katrai būtiskai izmaiņai jāreģistrē datums, iemesls, skartā nodaļa un pārbaudes pierādījums.

147.4. Versiju vēstures kopsavilkums

Versija	Galvenais papildinājums
v0.1–v0.5	Kursa karkass, algoritmi, Python/C++, SQL/IoT, darba lapas, pārbaudes darbi un laboratorijas.
v0.6–v1.0	Padziļināta robotika, kinemātika, LiDAR/IMU/Machine Vision, ROS 2, SLAM, Nav2, fiziskās laboratorijas, URDF/simulācija un 3D uztvere.
v1.1–v1.4	Industrijas slānis: RTOS, OPC UA/Modbus/CAN, modernā vadība, manipulatori/MoveIt 2, Deep Learning, funkcionālā drošība un Digital Twin.
v1.5	Skolēna/studenta rokasgrāmatas navigācija, quick-reference, formulas, vārdnīca un pašvērtējums.
v1.6	Pasniedzēja gala komplekts, 36 nodarbību checklist, vērtēšanas sistēma un kursa apstiprināšanas pakete.
v2.0 FINAL	Gala redakcija: pareizs saturs, klikšķināma navigācija, pieejamība, avotu statuss, publicēšanas metadati un gala QA.

147.5. Gala statusa deklarācija

v2.0 FINAL ir šī kursa pamatmateriāla gala redakcija. Turpmākas izmaiņas ieteicams veidot kā v2.1, v2.2 utt., saglabājot izmaiņu žurnālu un nemainot apstiprināto 36 × 90 min pamatkursa struktūru bez atsevišķa metodiska lēmuma.