Tartu Ülikool

Loodus- ja täppisteaduste valdkond

Tehnoloogiainstituut

Leonid Tšigrinski

Õpperoboti Robotont püsivara arhitektuuri uuendamine

Bakalaureusetöö (12 EAP)

Arvutitehnika eriala

Juhendaja:

PhD Veiko Vunder

Tartu 2024

Resümee/Abstract

Õpperoboti Robotont püsivara arhitektuuri uuendamine

Tehnoloogia edenedes muutuvad robootikaga seotud valdkondade kvalifitseeritud töötajad üha nõudlikumaks. Selleks, et valmistada ette tulevasi spetsialiste selles valdkonnas, on Tartu Ülikool loonud haridusroboti – Robotont. Varem oli Robotont kasutusel kõrgtasemel programmeerimise õpetamiseks ning roboti püsivara pakkus vaid baasfunktsionaalsust. Selle töö eesmärk on seda muuta. Töö käigus töötati välja uus püsivara arhitektuur, viidi olemasolev kood vastavusse uue arhitektuuriga ning lisati juhendeid, mis aitavad säilitada püsivara kvaliteeti. Töö tulemusena on tulevikus Robotondi püsivarasse uue funktsioonaalsuse lisamine lihtsam ning õppuritele on loodud näitematerjal madalatasemelise programmeerimise tööstuspraktikate omandamiseks.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T125 Automatiseerimine, robootika, control engineering;

Märksõnad: manussüsteem, püsivara, robootika

Education robot "Robotont" firmware architecture updating

As technology progresses, the demand for skilled workers in fields related to robotics is increasing. In order to prepare future professionals in this field, the University of Tartu has created an educational robot - Robotont. Previously, Robotont was used for teaching high-level programming, and the robot's firmware provided only basic functionality. The aim of this work is to change that. During the project, a new firmware architecture was developed, existing code was adapted to the new architecture, and guidelines were added to help maintain firmware quality. As a result of this work, adding new functionality to Robotont's firmware in the future will be easier, and teaching materials have been created for students to acquire low-level programming industrial practices.

CERCS: T120 Systems engineering, computer technology; T125 Automation, robotics, control engineering;

Keywords: emebdded system, firmware, robotics

Sisukord

Jooniste loetelu

Lühendid, konstandid, mõisted

ADC – *Analog to Digital Converter* ehk analoog-digitaalmuundur

API – *Application Programming Interface* ehk rakendustarkvara liides

ATtiny – 8-bitiste mikrokontrollerite perekond, mida toodab Microchip Technology

GPIO – *General-Purpose Input/Output* ehk mitmeotstarbeline sisend/väljund

HAL – *Hardware Abstraction Layer* ehk riistvara abstraktsiooni kiht

I2C – *Inter-Integrated Circuit* ehk mitme võimaliku ülemaga jadasiin

OSI – (*Open Systems Interconnection*) Rahvusvahelise Standardiorganisatsiooni loodud mudel andmesideprotokollide kirjeldamiseks

PID regulaator – *Proportional–Integral–Derivative controller* ehk proportsionaal-integraal-diferentsiaalregulaator

PWM – *Pulse-Width Modulation* ehk pulsilaiusmodulatsioon

ROS – *Robot Operating System* ehk roboti operatsioonisüsteem

RTOS – *Real Time Operating System* ehk reaalaja operatsioonisüsteem

STM32 – 32-bitiste mikrokontrollerite perekond, mida toodab STMicroelectronics

SWD – (*Serial Wire Debug*) Jadaühendusel põhinev silumisliides

USB – *Universal Serial Bus* ehk universaalne jadasiin

1 Sissejuhatus

Tehnoloogia kiire areng on viinud ettevõtete kasvava vajaduseni automatiseerida oma tegevusi robootika ja nutitehnoloogiate abil. See omakorda suurendab nõudlust haritud spetsialistide järele, kes suudaksid neid süsteeme arendada ja hallata. Tartu Ülikool vastab sellele väljakutsele mitmete haridusprogrammide ja projektidega, mille eesmärk on arendada vastava kvalifikatsiooniga spetsialiste. Üks sellistest algatustest on Robotont [1].

Robotont on avatud lähtekoodiga platvorm, mille arendust koordineerib Tartu Ülikooli tehnoloogiainstituut. See platvorm on mõeldud mitmekülgseks õppe- ja teadustööks ning selle keskmes on kolme omniliikuva rattaga [2] mobiilne robot, mida juhib kompaktne Intel NUC arvuti, toetudes ROS (*Robot Operating System*) [3, 4] tarkvarale. ROS on avatud lähtekoodiga raamistik robotite arendamiseks, pakkudes tööriistu andmete vahetamiseks, riistvara juhtimiseks ja andurite andmete töötlemiseks.

Madala taseme suhtluse eest vastutab Robotondil kolmandal versioonidel STM32F4 [5] perekonna mikrokontroller, mis on varustatud spetsiaalse püsivara. See mikrokontroller juhib alalisvoolu-mootoreid, valguslahendusi, OLED ekraani ning saadab asendiandmeid USB ühenduse kaudu ROS-i.

Mikrokontrollerite perekonna vahetuse tõttu kohandati eelnevalt kasutatud püsivara uue riistvara testimiseks. Selles etapis oli eesmärk võimalikult väikese ajakuluga käima suhtlus Intel NUC pardaarvutiga ja mootorid pöörlema. Kiiruga kohandatud püsivara täitis küll seatud eesmärki, kuid kuna puudus läbimõeldud jaotus abstraktsioonikihtideks või ühtne programmeerimisstiil, siis võib tulevikus riistvara või loogika muutmine nõuda ulatuslikke koodimuudatusi.

Seega on Robotondi püsivaraga seotud töö eesmärgiks luua püsivara arhitektuur ja implementatsioon, mis vastab kaasaegsetele püsivaraarenduse standarditele ning tagab platvormi pikaajalise kasutatavuse. Tartu Ülikooli õpilaste osalemine selles protsessis annab neile väärtuslikke kogemusi kaasaegsete püsivara arendamise meetodite jälgimisel, valmistades neid ette reaalseks tööks ja andes neile konkurentsieelise tööturul.

2 Ülevaade püsivara arendamisest

2.1 Sissejuhatus

Püsivara on sardtarkvara, mis on salvestatud mikrokontrolleri püsimällu. See pakub madala taseme juhtimisfunktsioone ühendades riistvara ja kõrgema taseme tarkvara. Mikrokontroller koosneb järgmistest osadest: püsimälust (*ROM*) programmi salvestamiseks, keskprotsessorist (*CPU*) programmi täitmiseks, muutmälust (*RAM*) ajutiseks programmi andmete salvestamiseks ja sisend/väljund (*I/O*) perifeeria seademetesse. Suhtlus perifeeria seadmetega toimub erinevate liideste kaudu, näiteks GPIO (*General-Purpose Input/Output*), USB (*Universal Serial Bus*) ja I2C (*Inter-Integrated Circuit*) [6, 7].

Püsivara saab kirjutada programmeerimiskeeltes, mis toetavad otsest juurdepääsu mälusse, näiteks Assembly, C, C++ ja Rust . Nende keelte lähtekood teisendatakse hex-failiks [8], mis sisaldab mikrokontrollerile arusaadavat masinkoodi. Seda protsessi nimetakse koostamiseks (*build*) või kompileerimiseks (*compile*). Inimloetava programmi masinkoodi teisendamiseks kasutatakse kompilaatoreid, näiteks gcc ja clang [9]. Konkreetse kompilaatori valik sõltub kasutuses olevast mikrokontrollerist või selle perekonnast [10].

Saadud hex-fail saab laadida mikrokontrolleri püsimällu spetsiaalse silumise liidese kaudu, näiteks SWD (*Serial Wire Debug*). Sama liideste kaudu saab ka siluda püsimälus olevat masinkoodi - käivitada programmi ridahaaval ja uurida muutujate ning registrite väärtuseid. Selleks kasutatakse spetsiaalset programmaatorit (nt. ST-LINK [11]) vastava tarkvaraga (nt. ST-LINK utility [12] ja gdb [13]).

2.2 Tööristaahel

Püsivara kirjutamiseks, koostamiseks, üleslaadimiseks ja silumiseks on olemas spetsiaalsed programmid, mis aitavad neid tegevusi lihtsustada. Need programmid koos moodustavad tööriistaahela (*toolchain*) [14, 15].

Programmi koostamiseks on võimalik sisestada kompilaatori käske eraldi, kuid sel juhul peab kasutaja määrama kompilaatori jaoks iga sisendfaili eraldi. Näiteks *gcc -o programm fail1 fail2* [16]. Suuremates projektides on programmi lähtekood jagatud enamasti mitme faili vahel ja seega muutub eelneva käsu sisestamine üsna pea tülikaks. Selle protsessi lihtsustamiseks on olemas tarkvara nimega Make [17], mis kasutab sisendina *makefile*'i. Make võimaldab kasutajal grupeerida terminali käske nn reeglitesse ja luua reeglite vahel sõltuvusi. Ent *makefile*'i süntaks

võib muutuda keeruliseks, eriti kui lisatakse palju reegleid. Lisaks on Make platvormispetsiifiline, mis tähendab, et kui arendus toimub ühel operatsioonisüsteemil (näiteks Windows) ja tulemus saadetakse teisele operatsioonisüsteemile (näiteks Linux), võivad mõned *makefile*'is olevad käsud teises keskkonnas mitte töötada. Ühilduvusprobleemide lahendamiseks on olemas tarkvara nimega CMake [18], mis võtab sisendiks platvormist sõltumatu *CMakeLists* faili ning genereerib erinevad *makefile*'id sõltuvalt kasutaja operatsioonisüsteemist [19–21].

Arendaja võib koodi kirjutamiseks kasutada mistahes koodiredaktorit, nagu näiteks VSCode, Sublime Text või VIM. Need koodiredaktorid toetavad erinevaid funktsioone nagu keele süntaksi esiletõstmine, automaatne väljundi lõpetamine ja teksti vormistamine. Lisaks on mitmetes koodiredaktorites võimalik lisada pistikprogramme ja nende kaudu käivitada näiteks make käsku. Peamine erinevus koodiredaktorite vahel seisneb stiilisätetes ja kiirklahvides (*hotkeys*) [22, 23].

Integreeritud arenduskeskkond või IDE (*Integrated Development Environment*) [24] pakub veelgi ulatuslikumat funktsionaalsust, kuna ühes programmis on koodiredaktor ja kõik vajalikud tööriistad programmi koostamiseks, üleslaadimiseks ja silumiseks. IDE valik sõltub sageli mikrokontrolleri perekonnast. Näiteks STM32 perekonna arendamiseks võib kasutada STM32CubeIDE [25], mis sisaldab ka STM32CubeMX [26] tarkvara koodi genereerimiseks Arendajal on võimalus paigutada seda tarkvara eraldi. Arduino plaadi [27] programmeerimiseks on saadaval Arduino IDE [28], ESP32 plaadi [29] puhul aga Espressif IDE [30] ning PIC [31] perekonna mikrokontrollerite jaoks MPLAB X IDE [32].

Erilist tähelepanu väärib PlatformIO [33], mis on püsivara arendusvahendite kogum, mida saab kasutada nii koodiredaktori pistikprogrammina integreerituna [34] kui ka eraldi terminali kaudu. Oluline on märkida, et PlatformIO toetab kõiki eelnevalt mainitud koodiredaktoreid ja mikrokontrollerite perekondi [35].

Eelpool mainitud arenduskeskkondadel ja koodiredaktoritel on võime automaatselt teksti vormindada. Ent väljaspool selliseid platvorme on olemas ka eraldiseisev tarkvara clang-tidy [36] ja clang-format [37]. Clang-tidy on tarkvara, mis aitab tuvastada võimalikke stiilinõuete rikkumisi ning pakub nende parandamiseks soovitusi. Clang-format tagab koodi ühtlase vorminduse vastavalt eeldefineeritud reeglitele. Mõlemat tarkvara saab tasuta eraldiseisvana paigaldada, kuid on juba vaikimisi sisseehitatud näiteks VSCode koodiredaktoris [38–40].

2.3 Arendusraamistikud ja teegid

Mikrokontrollerite tootjad ja teised ettevõtted pakuvad arendajatele palju valmiskomponente ja teeke (*library*), mis on kokku kogutud raamistikesse (*framework*). Teekide all mõeldakse tavaliselt mingi funktsioonide kogumit, mis täidab konkreetse eesmärki [41]. Manussüsteemides kasutatakse teeke näiteks väliste seadmete (näiteks ekraani, servo mootori või infrapunaanduri) mugavaks juhtimiseks.

Raamistik on teekide ja teise komponentide kogum, mille peale ehitakse tarvararakendust [41]. Manussüsteemide maailmas on üheks raamistikuks on riistavara abstraktsiooni kiht ehk HAL (*Hardware Abstraction Layer*). HAL-i eesmärk on pakkuda arendajatele valmiskomponente mikrokontrolleri juhtimiseks. Näiteks LED-i vilkumise protseduur, mis vajab kontreetse biti muutmist GPIO registris, on HAL-i abil võimalik sooritada ühe *toggle* funktsiooniga [42]. Raamistiku (sealhulgas HAL-i) valik sõltub suuresti mikrokontrolleri perekonnast. Antud kontekstis keskendutakse näidetele, mis toetuvad STM32F4 perekonnale.

Paljudel platvormidel on võimalik on kasutada **Arduino** [43, 44] raamistikku, mis koosneb C++-põhisest keelest, funktsioonidest ja teekidest välisseadmetega suhtlemiseks ning riistvara juhtimiseks. Arduino eeliseks on lai kogukonna tugi, suur hulk õppematerjale ja teeke ning lihtne koodi kirjutamine. Puudustena on välja toodud piiratud paindlikkust mikrokontrolleri funktsioonide kasutamisel ja projekti suurenemisel kaasnevat haldamise keerukust [45].

STM32F4 perekonnale on olemas sarnane raamistik, mida nimetatakse **STM32F4 HAL**-iks [46]. STM32F4 HAL-i eeliseks Arduino ees on suurem paindlikkus mikrokontrolleri seadistamisel ja võimalus kasutada koodigeneraatorit. Puuduseks on aga suur õpikõver, kuna nõuab sügavamaid teadmisi mikrokontrolleri seadistamise ja välisseadmete kohta.

Lisaks on olemas püsivara arendusplatvorm **Mbed** [47], mis sarnaselt STM32F4 HAL-ile, funktsioonid mikrokontrolleri juhtimiseks. Mbed sisaldab mitmeid teeke, näiteks PID kontroller ja odomeetria arvutuse moodul. Suurem osa Mbed-i teekidest ja funktsioonidest on kirjutatud C++ keeles, mis võimaldab püsivara arhitektuuris kasutada objekt orienteerutud lähenemist. Samuti on Mbed-il lai kogukonna tugi ja palju dokumentatsiooni, mis muudab arendusprotsessi sujuvamaks. Kuna reeglina kasutavad kõrgtaseme keeled rohkem arvutusse, siis võib ka Mbed-i abstraktsioonikiht piirata arendajate kontrolli mikrokontrolleri ressursside üle.

RTOS (*Real Time Operating System*) ehk reaalaja operatsioonisüsteem on püsivara raamistik, mis võimaldab kontrollida erinevate loogikakomponentide täitmist reaalajas. See võimaldab

programmeerijatel määrata iga ülesande (*task*) täitmise ajaperioodi ning tagab ülesannete vahel optimaalse mikrokontrolleri ressursside jaotamise. RTOS pakub suurt jõudlust ja determinismi ning annab arendajatele täpset kontrolli iga koodisegmendi käitusaja üle [48]. Ent selle integreerimine rakendustesse nõuab arendajatelt põhjalikku arusaamist operatsioonisüsteemi üldistest toimimispõhimõtetest.

Mõningates olukordades (näiteks juhtudel, kui mälu programmi hoidmiseks on rangelt piiratud) on otstarbekas raamistikke mitte kasutada ning seadistada mikrokontroller ise. Näiteks on ATtiny85 [49] mikrokontrolleril on 8 KB programmi mälu (*flash memory*). GPIO-viikude oleku muutmiseks tuleb seadistada kaks registrit: suuna- ja andmeregister. Suunaregister määrab, kas viigud töötavad sisendina või väljundina. Andmeregistri kaudu on võimalik juhtida või lugeda viikude olekut, sõltuvalt nende suunast. [50]. Seda lähenemisviisi nimetatakse riistvaralähedaseks (*bare metal*). Riistvaralähedase viisi puhul seadistab arendaja kõik perifeeria seadmed registri tasemel ise. Riistvaralähedase püsivara arendamine hõlmab programmeerimist otse riistvara tasemel, mis annab programmeerijatele suure kontrolli manussüsteemi ressursside üle ja võimaldab suuremat tõhusust. Siiski nõuab riistvaralähedane arendamine põhjalikke teadmisi mikrokontrolleri arhitektuurist, sealhulgas protsessori, mäluhierarhia ja perifeeriaseadmete kohta [51].

2.4 Koodi kvaliteet

Koodi koostamisel on oluline järgida kvaliteedikriteeriume, mis lihtsustavad teistel arendajatel koodi toetamist kogu selle elutsükli vältel. On mitmeid kvaliteedikriteeriume, mille abil saab hinnata kirjutatud tarkvara kvaliteeti. Järgnevas vaatleme neist lähemalt kuute kriteeriumit: modulaarsus, madal sidusus (*low coupling*), eraldatus (*incapsulation*), paindlikus, ühilduvus ja pikkajaline tugi.

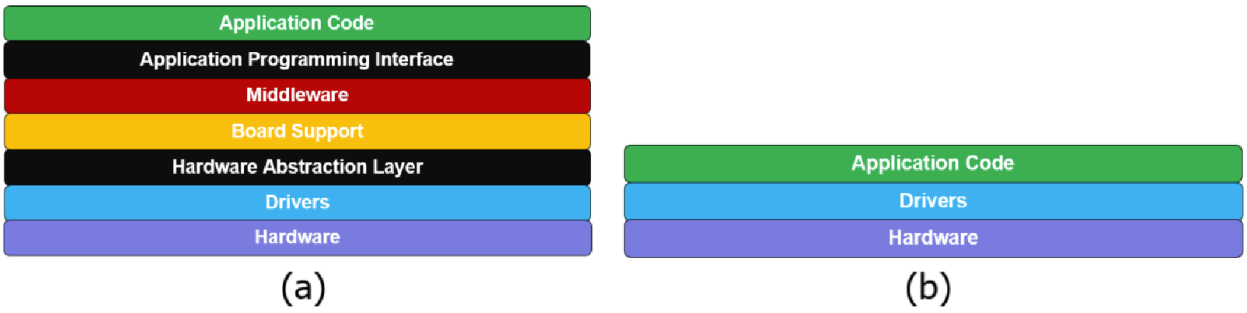
Modulaarsus eeldab, et kood jaotatakse mooduliteks, millest igaüks täidab konkreetset ülesannet. See võimaldab tulevikus kasutada valmiskoodi uute toodete loomisel. **Madal sidusus** hõlbustab teiste moodulite ümberkirjutamist juhul, kui ühte muudetakse. See tagab toote koodi muutmise paindlikkuse. **Eraldatus** tähendab, et andmetele juurdepääsu piiratakse mooduli piires. See aitab vältida tulevasi vigu, mis võivad tekkida soovimatu juurdepääsu korral ühelt moodulilt teisele. **Paindlikkus** tähendab, et tarkvara saab hõlpsasti laiendada uute moodulite lisamisega ilma vanu muutmata. See omadus on eriti oluline, kui toote funktsionaalsust pidevalt uuendatakse, nagu ka Robotondi puhul. **Ühilduvuse** kriteeriumi järgi loogikamoodulid ei tohiks sõltuda konkreetsest riistvarast. See tagab, et riistvara muutmisel (sh mikrokontrolleri vahetamisel) on koodi muutmise

vajadus minimaalne. **Pikaajaline tugi** vastab sellele, et kirjutatud kood järgib kindlaks määratud standardeid, sealhulgas omab vormindamisreegleid ja dokumentatsiooni. Selle kriteeriumi järgimine muudab koodi loogika mõistmise teistele arendajatele lihtsamaks [52, 53].

2.5 Arhitektuur

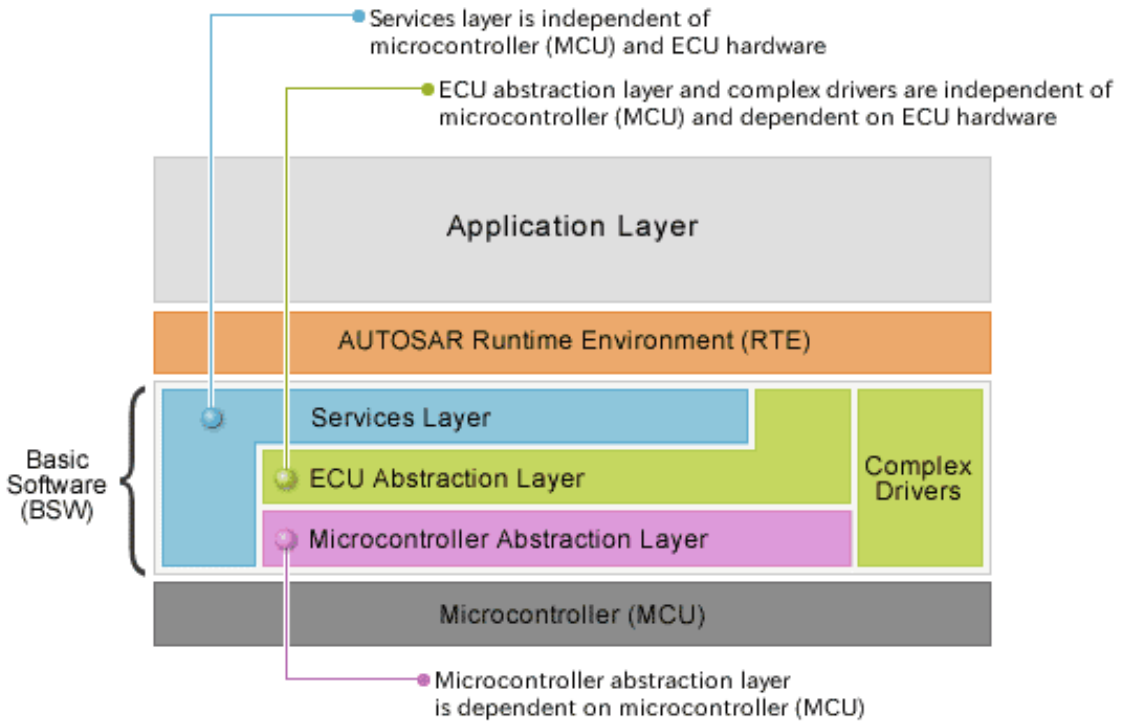
Kõigi eelnevalt nimetatud kvaliteedikriteeriumide täitmiseks võib arendaja jagada oma programmi abstraktsioonikihtideks. Jagamise eesmärk on kõigipeealt struktureerida koodi, nii et oleks selge iga komponendi eesmärk, mis tagab pikaajalise toe. Lisaks aitab abstraktsioonikihide kasutuselevõtt hoida koodi modulaarse ja eraldatuna, kuna jagamine sunnib arendajat kogu lahenduse enne implementeerimist läbi mõtlema. Lõpuks pakub selline lähenemine ühilduvust ja paindlikust, sest vajadusel saab muuta ainult ühe kihti komponente, nii et kood jääb töötavaks.

Alumine kiht hõlmab tavaliselt mooduleid, mis suhtlevad otse mikrokontrolleriga. Näiteks draiverid, mis saadavad sõnumeid USB kaudu või muudavad GPIO-viikude olekuid. Mida kõrgemal asub kiht, seda vähem sõltub see riistvarast. Järgmine kiht võib sisaldada koodi, mis pakub toe konkreetsele plaadile. Kõigi ülemine kiht on tavaliselt rakenduse kiht, kus paikneb vastava rakenduse loogika. Mida keerukam on toode, seda rohkem kihte võib olla alumise ja ülemise kihi vahel. Seda lähenemist nimetakse **kihiliseks arhitektuuriks**. Joonisel 2.1 (a) on toodud seitsme kihiga püsivara arhitektuur ning Joonisel 2.1(b) on kolme kihiga versioon [53, 54].



Joonis 2.1: Kihilise arhitektuuri lähenemine. (a) Seitse kihti; (b) Kolm kihti

Paljud elektroonikaseadmete tootjad kasutavad seda lähenemisviisi püsivara kirjutamisel. Näiteks on Toshiba rakendanud seda lähenemisviisi oma MCU Motor Studio 3.0 [55] programmiga mootorite juhtimiseks püsivaraga suhtlemisel. Lisaks on ettevõte AUTOSAR välja töötanud oma tööriistade ja standardite komplekti, mis kirjeldab abstraktsioonikihtidel põhineva arhitektuuri "AUTOSAR Layered Architecture" (Joonis 2.2) [56]. Nimetatud standardid on kasutuses mitmetel autotootjatel nagu BMW, Ford Motor Company ja Mercedes Benz [57].



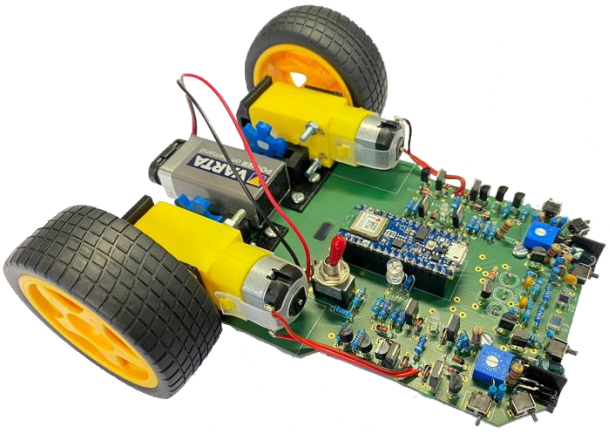
Joonis 2.2: Näide kihilist arhitektuurist: "AUTOSAR Layered Architecture"

Kihilise arhitektuuri näide võrgutehnoloogia valdkonnas on seadmetevahelist suhtlust kirjeldav OSI mudel. OSI mudel jaguneb samuti abstraktseteks kihtideks, millest igaüks sisaldab erinevaid sideprotokolle. Alumine kiht kirjeldab bittide ülekandmist füüsilises kihis, samas kui ülemine kiht kirjeldab protokolle, mida rakendused kasutavad omavahel suhtlemiseks [58].

2.6 Näited haridusrobotite püsivaradest

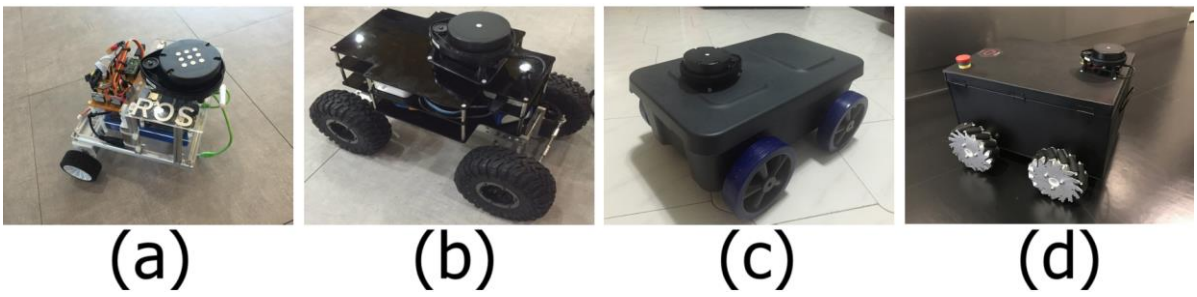
Käesolev alapeatükk annab ülevaate sellest, milliseid püsivara arendamise lähenemisviise tänapäeval robootika õppeplatvormidel kasutatakse.

Esimene näide on WHSR "Wizzer" (Joonis 2.3) [59], mis on Westfaleni Rakendusteaduste Ülikooli Gelsenkircheni, Saksamaa, omadus. Robot koosneb kahest mootorist, mis on ühendatud juhtplaadiga. Peamine juhtplaat on Arduino Nano [60]. Püsivara on kirjutatud C++ keeles, kasutades mõningaid Arduino keele funktsioone mikrokontrolleri juhtimiseks. Püsivara on jaotatud loogilisteks mooduliteks vastavalt riistvarale: ADC [61] juhtimine, nupu oleku lugemine, taimeri seadistamine ja roboti liikumine. Koodi pole jagatud selgeteks abstraktsioonikihtideks, näiteks taimer ja selle katkestused on konfigureeritud põhifailis (*WHSR.cpp*), kirjutades väärtusi otse mikrokontrolleri registrisse.



Joonis 2.3: Kokkupandud WHSR "Wizzer”

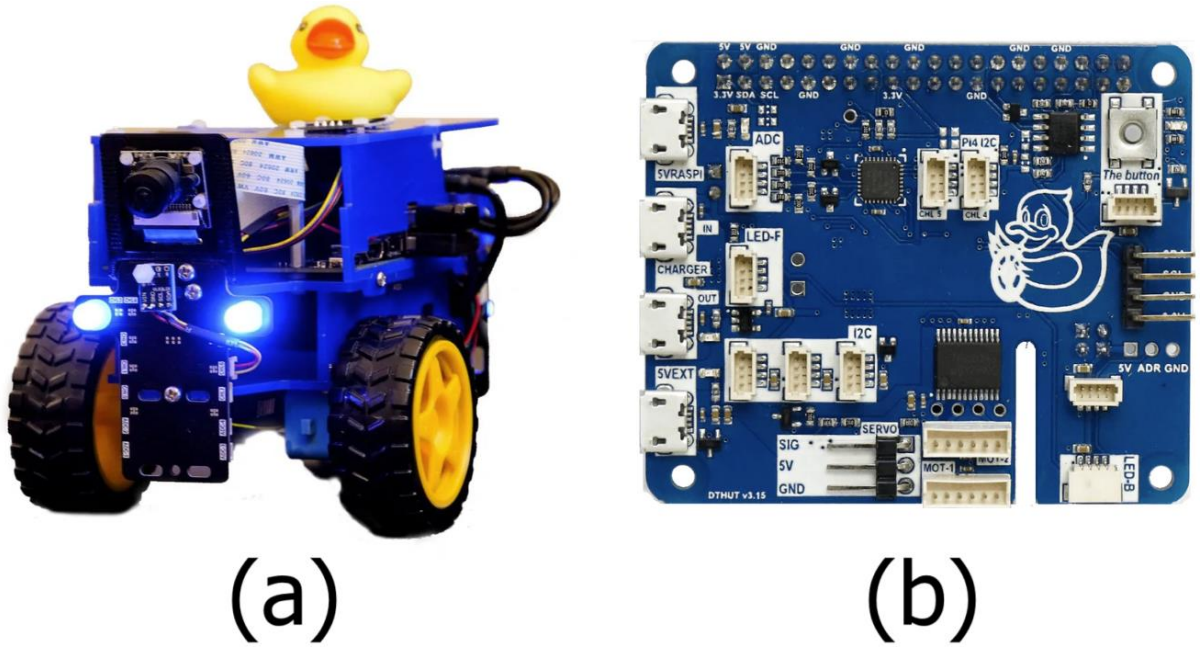
Teine näide on Linorobot (Joonis 2.4) [62, 63], mis on avatud lähtekoodiga õppeplatvorm, mille püsivara toetab kahte tüüpi roboteid: omniliikuvaid ja diferentsiaalliikuvaid. Kõrgemal tasemel suhtleb robot ROS-i kaudu. Selle projekti püsivara on modulaarne, kasutades valmis Arduino teeke mikrokontrolleri ja riistvara juhtimiseks, näiteks PID-regulaatorit, mootori juhtimise moodulit ja roboti kinemaatika arvutamise moodulit. Moodulid on eraldatud, mis võimaldab andmetele juurde pääseda liidestatud funktsioonide kaudu. Roboti konfiguratsiooni saab muuta ühes kohas vastavas failis, mis sisaldab rataste arvu ja tüüpi, PID-regulaatori koefitsiente, viigu määratlusi jne. Kuigi näidatud robotil puudub selge jaotus abstraktsioonikihtideks, on failid selgelt ja ühtlaselt struktureeritud ning vajadusel kommenteeritud.



Joonis 2.4: Kokkupandud Linorobotid. (a), (b) ja (c) on diferentsiaalliikuvad, (d) on omniliikuv

Kolmas näide käsitleb Duckieboti [64], mis on valmis komplekt roboti osadest, sealhulgas kahest rattast, kerest ja anduritest. Komplekt võimaldab roboti ise kokku panna ja seda kasutada näiteks isejuhtivate robotite programmeerimise harjutamiseks. Peamiseks komponendiks on Duckiehuti [65] plaat, mis juhib mootoreid ja LED-e. Plaadi püsivara [66] arhitektuur on väga sarnane eelnevale robotile: arhitektuur on modulaarne ja hästi kommenteeritud kuid puuduvad selged

abstraktsioonikihid. Erinevalt Linobot roboti püsivarale on kogu Duckieboti kood kirjutatud C keeles kasutades riistvaralähedast lähenemist. Duckiebot on toodud Joonisel 2.5(a) ning Duckiehut on toodud Joonisel 2.5(b).



(a) (b)

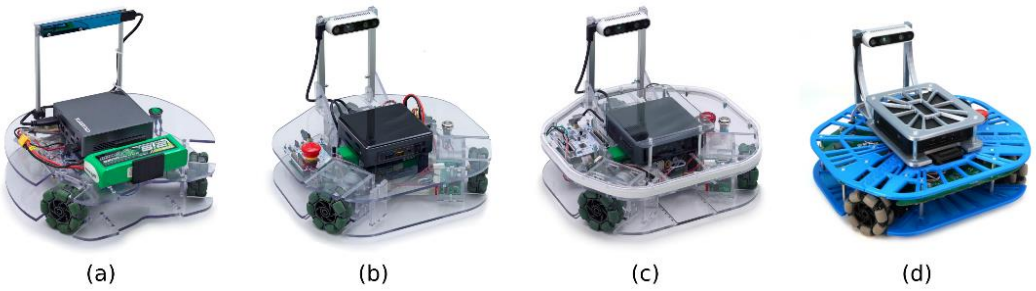
Joonis 2.5: (a) Kokkupandud Duckiebot robot. (b) Mootorite ja LED-ide juhtplaat Duckiehut

3 Robotont

3.1 Ülevaade

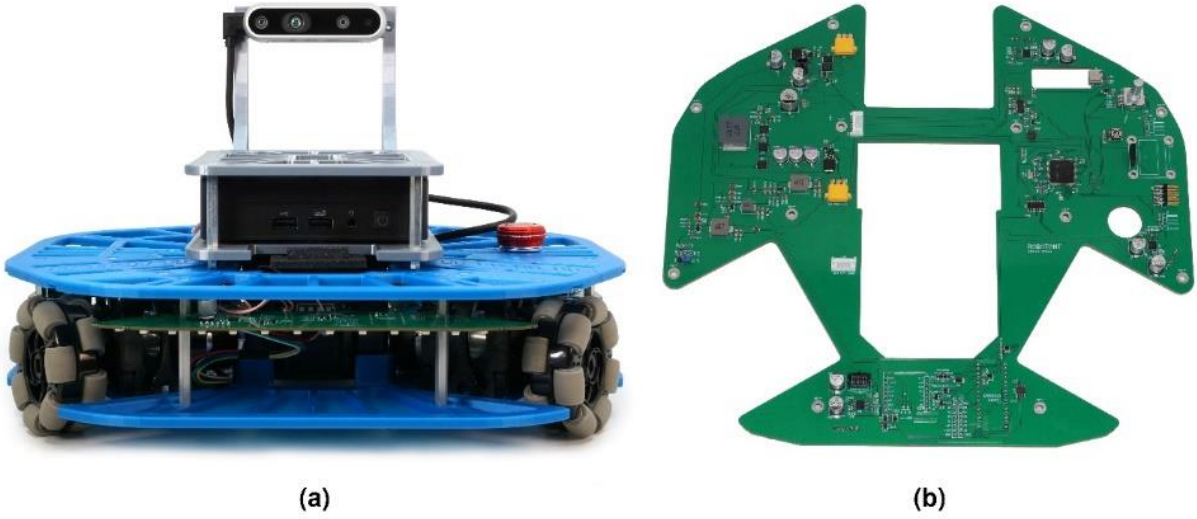
Käesolevas bakalaureusetöös kasutatavaks riistvaraplatvormiks on Robotont 3.0, mis on toodud Joonisel 3.1(d). Robotont on avatud lähtekoodiga haridusplatvorm, mis on loodud õpetama õpilastele ROS-i kasutamist roboti juhtimiseks [1, 67]. See on oluline osa ROS-i [68] algkursusest.

Robot on omniliikuv, mis tähendab, et robot saab vabalt valitud suunas liikuda ilma pööramata [2]. Mootorite kiiruse reguleerimiseks saadetakse USB-protokolli kaudu pardaarvutist roboti emaplaadile erinevaid käsklusi. Lisaks saadetakse roboti emaplaadilt tagasi pardaarvutisse andmeid asukoha kohta ehk odomeetriat [69], võimaldades kasutajal saada tagasisidet roboti kohta.



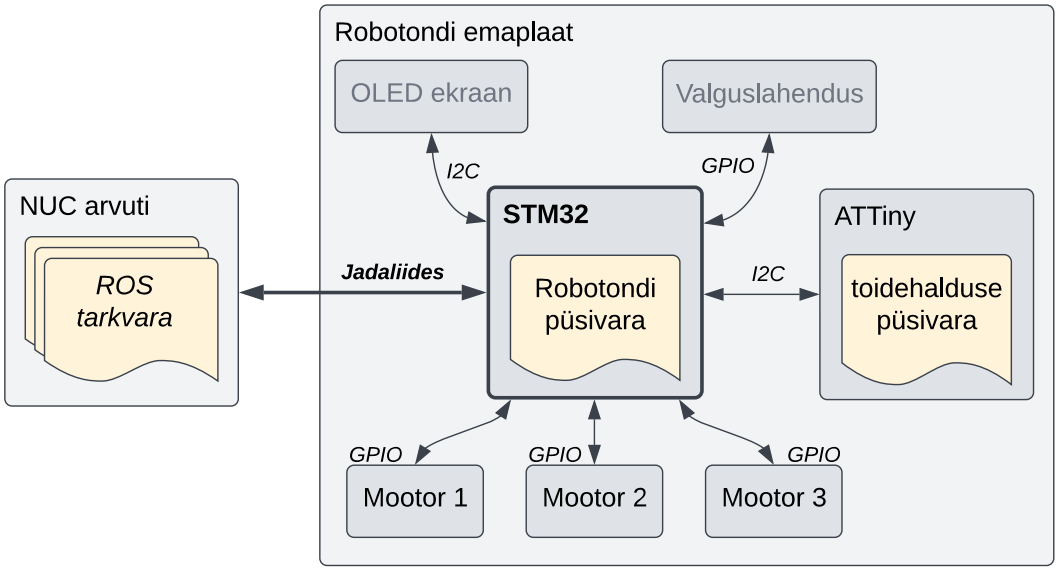
Joonis 3.1: (a) Robotont 1. (b) Robotont 2.0. (c) Robotont 2.1. (d) Robotont 3.0

Kolmanda põlvkonna Robotont koosneb mitmest komponendist: Intel NUC arvutist, kaamerast, emaplaadist ja 3D-printeriga prinditud šassiimoodulitest (Joonis 3.2) [67].



Joonis 3.2: (a) Kokkupandud Robotont 3.0 robot. (b) Robotondi 3.0 emaplaat

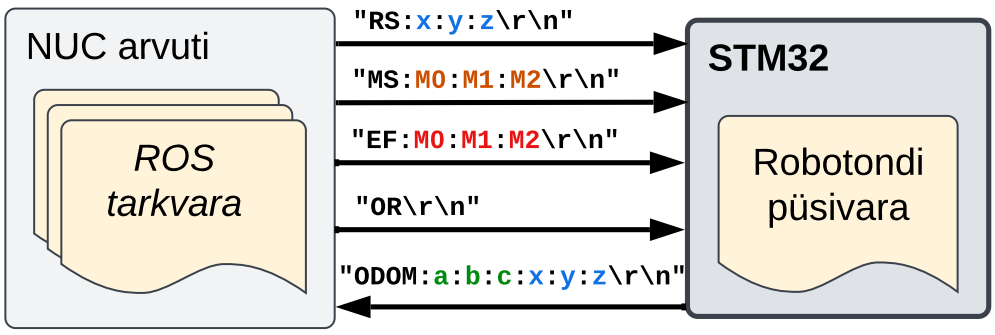
Kasutaja saab robotiga suhelda kasutades NUC-arvutisse paigaldatud ROS-i tarkvara või otse USB jadaliidese kaudu. Robotondi emaplaati haldab SMT32 mikrokontroller, mis toimib peamise kontrollerina roboti tegevuste ja interaktsioonide koordineerimiseks (Joonis 3.3).



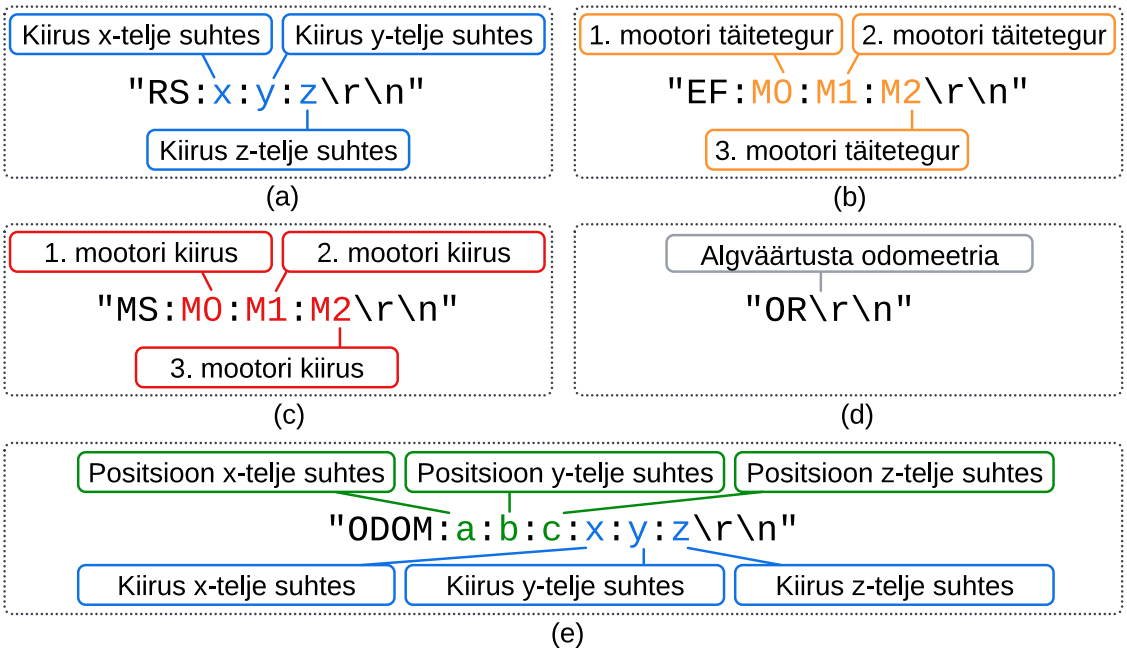
Joonis 3.3: Robotont 3.0 riistvara arhitektuuri ülevaade

Kuigi käesoleva töö keskmeks on STM32 mikrokontrolleri tarkvara, arendatakse samaaegselt teiste tudengite poolt robotondi OLED ekraanil põhinevat kasutajaliidest, adresseeritavatel LED valgustitel põhinevat valguslahendust ja juhtloogikat toitehalduskontrolleri jaoks.

Robotondi püsivara pakub kindlaksmääratud käsklusi roboti liikumise, andurite ja erinevate funktsioonide haldamiseks (Joonis 3.4). Esimese põlvkonna robotil (Robotont 1.0) oli defineeritud viis käsku, millest nelja kasutati info edastamiseks pardaarvutist mikrokontrollerisse ning ühte mikrokontrollerist pardaarvutisse. Käsustik (Joonis 3.5) jäi samaks kuni Robotont 3.0 versioonini kaasarvatud.



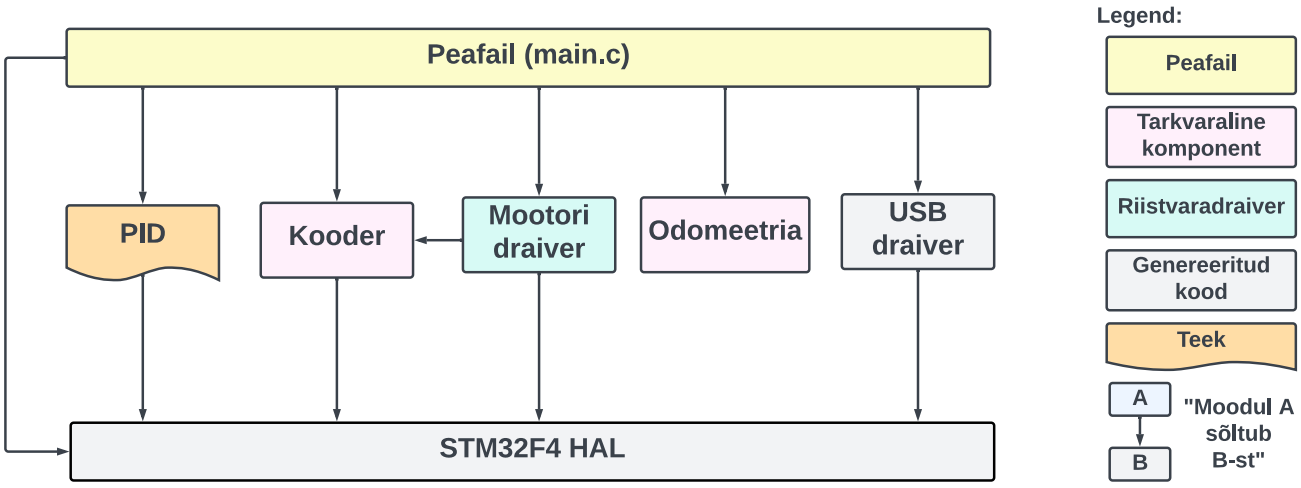
Joonis 3.4: Robotont 3.0 suhtlusprotokolli ülevaade



Joonis 3.5: Robotont 3.0 suhtlusprotokolli argumentide kirjeldus

3.2 STM32 kontrolleri püsivara

Algselt loodi Robotont 1.0 ja Robotont 2.0 robotite püsivara kasutades C++ keelt koos Mbed-raamistikuga. See lahendus töötas hästi kuni Robotont 3.0 robotite saabumiseni. Eelmise mikrokontrollerite puudus turul ei võimaldanud Robotont 3.0 emaplaadil jätkata Mbed toega kontrolleriga vaid leiti asenduseks STM32F407VGT6 [70]. Selle tulemusena viidi kogu loogika üle C programmeerimiskeelde, kasutades STM32F4 HAL raamistikku ja STM32CubeMX pakutavaid koodi genereerimise võimalusi. Robotont 3.0 püsivara arhitektuuri ülevaade on toodud Joonisel 3.6.

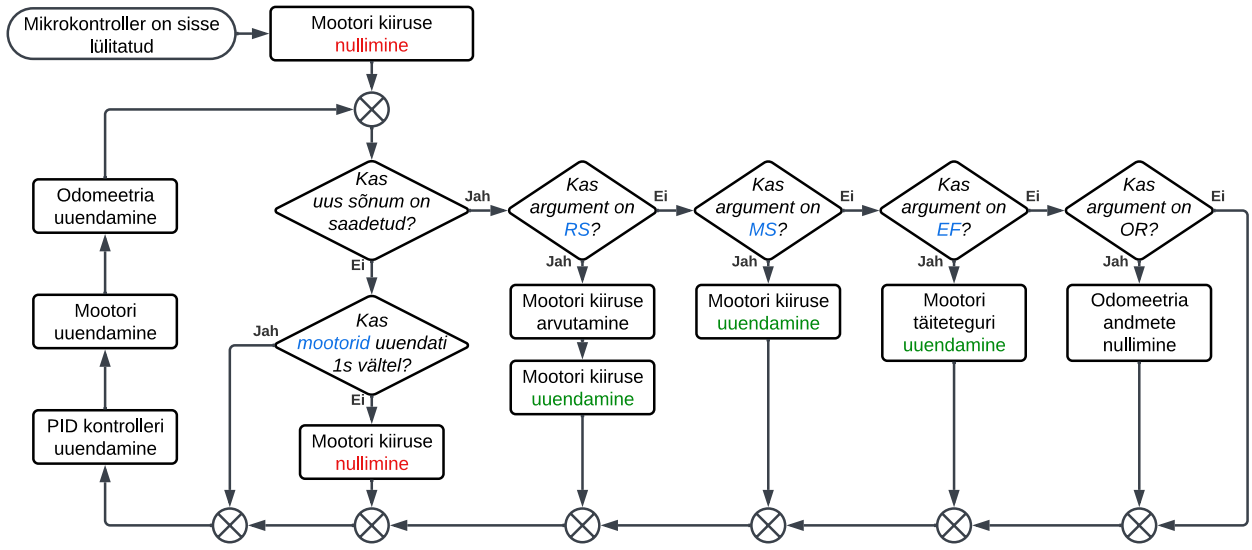


Joonis 3.6: Robotont 3.0 püsivara arhitektuur

Roboti püsivara tugineb mitmele olulisele komponendile:

- Mootori draiver, mis haldab suhtlust alalisvoolumootoriga. Draiveri eesmärk on arvutada mootori juhtsignaali täitetegur ja saata igale mootorile õige PWM signaal.
- PID kontrolleri teek, mis reguleerib mootori täitetegurit. Sisendina kasutatakse mootori kiirust, mida mõõdetakse koodri abil. Väljundiks on PWM juhtsignaali täitetegur.
- Odomeetria moodul, mis arvutab roboti asendi x, y ja z-telje suhtes mootorite kiiruste ja positsiooni põhjal. Odomeetria annab kasutajale infot roboti asendi ja roboti liikumise kiiruse kohta ruumis.
- Jadaliidese (USB) draiver, mis haldab suhtlust kasutaja arvutiga. Seda komponenti kasutatakse jadaliidese kaudu saabuvate käskude lugemiseks ja näiteks odomeetria info saatmiseks mikrokontrollerist pardaarvutisse.
- SMT32F4 HAL raamistik, mis pakub liideseid mikrokontrolleri juhtimiseks. Selle abil seadistatakse mikrokontroller ja kasutatakse perifeeriaseadmeid. Näiteks seadistatakse raamistiku abil taimer PWM signaali genereerimiseks.
- Peafail, mis seob kõik komponendid kokku üheks rakenduseks.

Püsivara kõige olulisem ülesanne on töödelda pardaarvutist saabunud ja sõnumite põhjal roboti liikumist juhtida. Kui kasutaja saadab liikumisega seotud käsu, hakkab robot liikuma. Kui 1 sekundi vältel uut käsku ei saadeta, jääb robot seisma. Juhtkäskudest sõltumata arvutatakse pidevalt mikrokontrolleris roboti odomeetria andmeid ja saadetakse tagasi pardaarvutisse. Detailsem juhtloogika on kirjeldatud Joonisel 3.7.



Joonis 3.7: Robotont püsivara loogika. Kehtib kuni Robotont 3.0 (kaasarvatud)

4 Töö eesmärk ja nõuded

Selle bakalureusetöö eesmärgiks on Robotondi platvormi püsivara arhitektuuri väljatöötamine ja selle rakendamine. Tulemuseks on Robotont 3.1, mis erineb Robotont 3.0-st vaid püsivara osas. Uus arhitektuur koos dokumentatsiooni ja näitelahendustega on oluliseks osaks, et Robotondi püsivara täiendamine oleks lihtne nii õpilastele, teadlastele kui ka edasijõudnud arendajatele. Arhitektuur peab pakkuma piisavalt paindlikkust, et kohandada püsivara Robotondi pidevalt uueneva riistvaraga ja võimaldama rakenduse koodi laiendada või muuta ilma madalama taseme komponente mõjutamata. Antud töö põhirõhk on püsivara täiendamisel ja parandamisel.

Lähtuvalt kaasaegse tarkvaraarenduse kvaliteedistandarditest seati püsivarale ning programmeerimisstandardile allpool toodud nõuded.

Nõuded arhitektuurile:

- Arhitektuur on riistvaraülene ja toetab riistvarakomponentide (näiteks protsessori) väljavahetamist.
- Arhitektuur on modulaarne ja toetab roboti funktsionaalsuse lisamist ja eemaldamist.
- Arhitektuur on testitud sõitmisel ja odomeetria näitel.

Nõuded arenduskeskkonale:

- Tööriistad peavad ühilduma nii Windows kui ka Linux operatsioonisüsteemiga.
- Tööriistad peavad olema tasuta ligipääsetavad (võimalusel avatud lähtekoodiga).
- Dokumentatsioon võimaldab arenduskeskkonna ülesseadmist ka ilma eelneva püsivara arendamise kogemuseta.

Nõuded vormistusele ja dokumentatsioonile:

- Koodi vormistus järgib manussüsteemides üldtuntud tavasid.
- Koodi vormistusel kasutatud reeglid on dokumenteeritud püsivara repositoriumis.

5 Lahendus

5.1 Ülevaade

Varasemalt on Robotondi püsivara tuginenud STM32F4 HAL-ile ja kogu roboti käitumisloogikat kirjeldati põhifailis. Seetõttu puudusid arhitektuuris selged abstraktsioonikihid. Samuti ei olnud defineeritud selgeid vormistuse reegleid ega määratud konkreetset standardit, millele peaks arendamisel tuginema. Seetõttu ei vastanud kood varasemalt toodud kvaliteedi kriteeriumidele. Kui lühiajaliselt võib seesugune reegliteta ja raamideta koodi kirjutamine oli lihtsam ja kiirem, võib see pikemas perspektiivis kaasa tuua mitmed probleeme:

1. Selge standardi puudumine koodi kirjutamisel (nii süntaktiliselt kui ka arhitektuuriliselt) tekitab probleeme koodi mõistmisega, sest igal arendajal on oma eelistatud programmeerimisstiil.
2. Riistvara muudatuste korral (seal hulgas mikrokontroller ning trükkplaadi disain) tuleb muuta suurt osa koodist.
3. Uue funktsionaalsuse lisamisel võib samuti olla vaja muuta suurt osa koodist.

Käesolev peatükk tutvustab kihilist püsivara arhitektuuri disaini, mis väldib eelnevalt toodud probleemide tekkimist ja sätestab selged süntaktiliste -ja stiilireeglite kogumid koos näidetega Robotondi püsivara arendamiseks.

5.2 Tööriistaahela valik

Püsivara arendamise esimene samm seisneb tööriistadeahela valimises. On oluline, et need programmid oleksid tasuta ja toetaksid nii Linuxi kui ka Windowsi platvorme.

Alguses uuriti kompileerimistööriistu, et välja selgitada, kas leidub paremini soobiv viis kui see, mida seni kasutati. Kaaluti võimalust kirjutada ise *makefile*. Sellest otsustati loobuda, kuna see muudaks arenduskeskkonna seadistamise oluliselt keerulisemaks – kasutaja peaks ise alla laadima vajaliku kompilaatori, siluri ja silumisrakendused. Samuti võib *makefile*'i keerukas seadistamine olla algajale kasutajale hirmutav. Sama põhjusega loobuti ka CMake'i kasutamisest.

Otsustati kasutada PlatformIO'd. Esiteks on seda juba kasutatud alates Robotont 1.0 versioonist, seega on see juba Robotondi arendajatele tuttav. Teiseks on PlatformIO kasutamine lihtne ka kogenematutele kasutajatele. Kasutaja peab konfiguratsioonifailis määrama platvormi ja vajaliku raamistiku ning programm laeb ise alla vajalikud kompilaatori versioonid, siluri ja silumisrakendused (nagu Jlink ja STlink) ning kõik kasutatud raamistikud ja teegid, sealhulgas

STM32F4 HAL-i. PlatformIO jaoks on välja töötatud ka VSCode'i pistikprogramm, mis lihtsustab kasutamist veelgi, visualiseerides kõiki PlatformIO funktsioone.

Tarkvararaamistike osas otsustati vältida riistvaralähedast lähenemist, kuna see suurendaks oluliselt töömahtu ja vajaks täiendavate abstraktsioonikihtide disainimist ning implementeerimist. Kaaluti mitmeid tasuta võimalusi, mis on tänapäeval manussüsteemide maailmas populaarsed ning toetavad kasutatud mikrokontrollerit: Arduino ja STM32F4 HAL.

Kuigi Arduino pakub lihtsalt programmeerimisvõimalust, millega algajad robotihuvilised suure tõenäosusega juba tuttavad on, otsustati Arduino platvormist loobuda. Arduino piirab madalataseme juhtimist perifeeriaseadmete funktsioone, mis muudaks mõne mikrokontrolleri funktsiooni kasutamise keerulisemaks (näiteks taimeri režiimid või mõned katkestused).

Valik langes STM32F4 HAL-ile mitmel põhjusel. Esiteks kirjutati Robotont 3.0 koodi algselt selle raamistiku abil. Teiseks pakuvad raamistiku arendajad (STMicroelectronics) ka koodi genereerimise programmi nimega STM32CubeMX, mis lihtsustab oluliselt perifeeriaseadmete seadistamist ja kiirendab arendust.

Koodiredaktori valikust, soovitab autor kasutada VSCode'i. Selle peamine eelis teiste koodiredaktorite ees on eelnevalt mainitud Platform IO ühilduvus. Piisab vaid pistikprogrammi paigaldusest VSCode'i poest, misjärel projekti kompileerimine on ühe nupuvajutuse kaugusel. Kuna PlatformIO'd saab kasutada nii käsurealt kui ka liidestades teiste IDE-de (*Integrated Development Environment*) või koodiredaktoritega, jätab PlatformIO piisavalt paindlikust keskkonnavalikul vastavalt arendaja enda eelistustele.

5.3 Vormistuse reeglid

Ühtlane koodivormistus ja stiilireeglite järgimine lihtsustavad koodi arendamist ja mõistmist. Selle töö raames võeti aluseks olemasolev reeglite kogum, et luua oma lihtsustatud versioon Robotondi arendamiseks. Kasutatud on "Barr Group Embedded C Coding Standard" [71] ja "ROS C++ Style Guideline" reeglite kogumeid [72].

Barr-i reeglite valimise põhjus on nende C-keelepõhine olemus, levik manussüsteemide maailmas ning see, et nad on tasuta. ROS-i C++ reeglite kasutamine võimaldab täpsustada neid reegleid, mis pole eelnevas standardis täpsustatud. ROS-i reeglite kasuks otsustati, sest ajalooliselt on Robotondi arenduses kasutatud enamasti C++ keelt. Seega on mõned ROS-i reeglid arendajatele juba tuttavad.

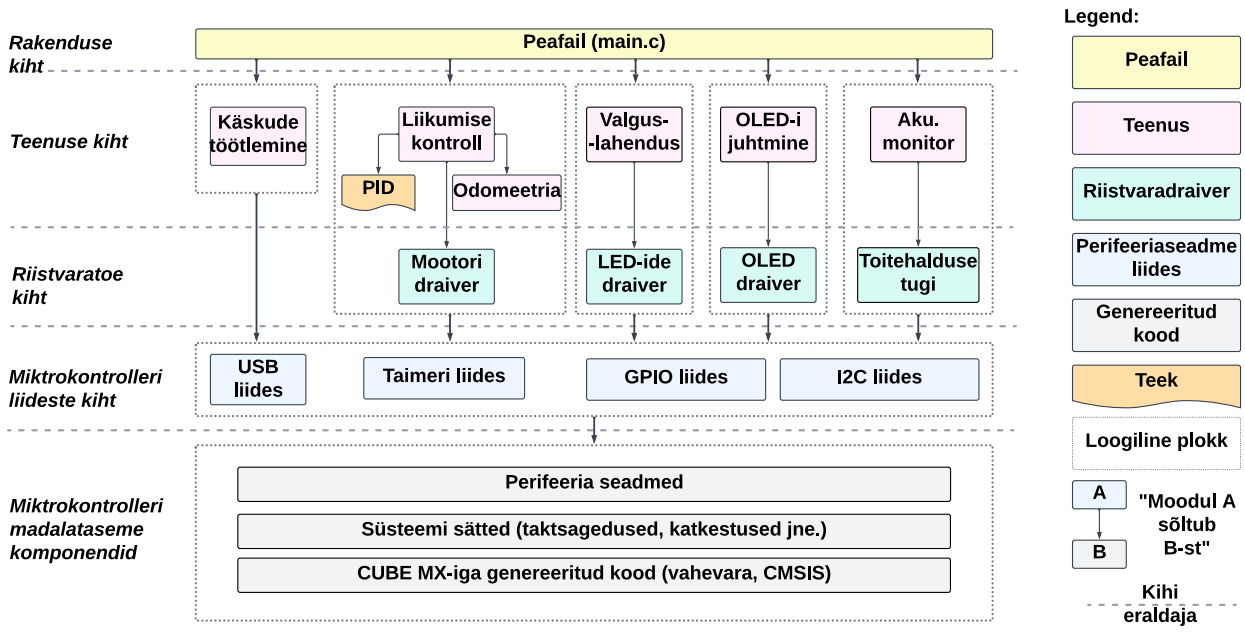
Eelmainitud reeglitest on antud töö raames loodud lihtsustatud kogum. Sellel on mitu põhjust. Mõned Barr-i standardis olevad reeglid on antud mitmel viisil, seega tuleb need eraldi täpsustada. Kuna Robotondi arendajate sihtgrupiks on tudengid, vähendati lihtsustatud kogumis vajalike reeglite arvu ja jäeti alles vaid kõige olulisemad reeglid.

Reeglid on nelja tüüpi. Üldreeglid hõlmavad põhilisi tavasid, nagu *include* käskude kasutamise järjekord ja tüüpide kasutus. Lausete ja väljendite reeglid määravad koodi struktuuri ja loogikat. Failivormingu reeglid kehtestavad komponentide järjekorra ja funktsioonide dokumentatsiooni vormingu. Stiilireeglid jõustavad nimeandmise tavasid. Kõik need on suunatud koodi loetavuse ja järjepidevuse parandamisele.

Reeglite mugavaks järgimiseks loodi tugi kahele programmile: clang-tidy ja clang-format. Kuigi suur osa reeglitest on juba kontrollitud clang-tidy ja clang-format tarkvara poolt, on oluline, et nende reeglite kogum oleks dokumenteeritud. Seega on kõik reeglid ning clang-tidy ja clang-format tarkvara sätted kirjeldatud Robotondi püsivara repositoriumis.

5.4 Arhitektuuri disain

Püsivara arhitektuuri disainimisel kasutati kihilise arhitektuuri lähenemist, kus püsivara on jagatud viieks abstraktsioonikihiks (Joonis 5.1): rakenduse kiht, teenuse kiht, riistvaratoe kiht, mikrokontrolleri liideste kiht ning kiht mikrokontrolleri madalataseme komponentidega. Mida kõrgem on kiht, seda vähem sõltub see riistvarast.



Joonis 5.1: Robotont 3.1 püsivara arhitektuur

5.4.1 Rakenduse kiht

Rakenduskihi eesmärk on määrata, kuidas kõik teenused toimivad koos. Näiteks, kui arendaja soovib rakendada sellist loogikat nagu "Kui aku monitor tuvastab ülepinge, siis robot ei tohiks liikuda", on see loogika implementatsiooniks sobilik kiht. Praegu on kogu Robotondi rakenduse kihti kuuluv loogika paigutatud ühte komponenti nimega peafail (*main.c*). Funktsionaalsuse lisandumisel võib kaaluda selle jagamist mitme komponendi vahel.

5.4.2 Teenuse kiht

Teenuste kihi eesmärk on määrata, milliseid ülesandeid Robotont peaks täitma. Iga ülesanne (või teenus) on individuaalne komponent, mis on võimalusel teistest sõltumatu. Suhtlemine teenustega peaks enamasti toimuma rakenduse kihi kaudu. Varasema peafaili koodi põhjal disainiti ja implementeeriti püsivarasse järgmised teenused:

1. Käskude töötlemise teenus – suunab ümber jadaliideste sõnumid vastavasse teenusesse.
2. Liikumise kontrolli teenus – arvutab iga mootori kiiruse soovitud roboti suuna ja kiiruse põhjal. Lisaks arvutab roboti koodrite põhjal odomeetriat ja edastab seda läbi jadaliideste.

Antud töö pakub disaini ka teiste teenuste nagu valguslahenduse, OLED ekraani ning aku monitori juhtimiseks kuid nende implementatsioonid valmivad teiste tudengite tööde osadena.

5.4.3 Riistvaratoe kiht

Seda kihti võib ka nimetada riistvaradraiveri kihiks. Antud kiht sisaldab püsivara draivereid Robotondi riistvara jaoks. Selle töö raames viidi sisse parandusi mootori püsivara draiveri juhtimisloogikasse. Siia kihti paigutuvad ka valguslahenduse draiver ja toitehalduse tugi, mida arendavad paralleelselt teised tudengid. I2C liideste testimiseks kasutati robotile paigaldatud OLED ekraani ja avatud litsensiga OLED draiverit [73].

5.4.4 Mikrokontrolleri liideste kiht

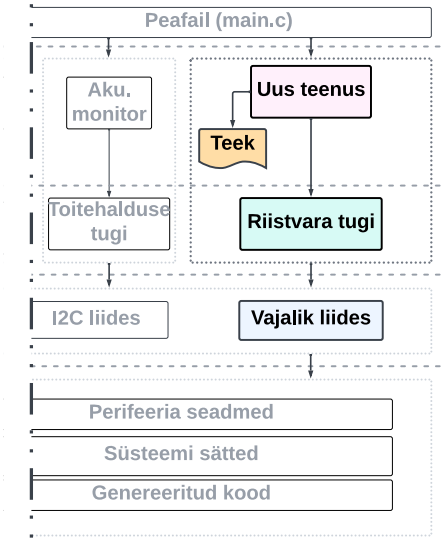
Antud kiht on ümbris (*wrapper*) STM32F4 HAL funktsioonidele. Kihi eesmärk on isoleerida Robotondi rakendusest STM32F4 HAL funktsioonid, mida kasutatakse STM32 perifeeriaseadmetega suhtlemiseks. See lähenemisviis annab ühildavuse osas märkimisväärse eelise. Kui mikrokontrolleri perekonda või STM32F4 HAL-i raamistikku muudetakse, peavad arendajad muutma ainult liidese kihi faile, ilma ülemise kihi faile muutmata. Lõputöö käigus arendati välja kõikide liideste disainid ja implementatsioonid, seal hulgas USB, GPIO, I2C ja taimeri liidesed.

5.4.5 Mikrokontrolleri madalataseme komponendid

Madalataseme komponentide kihti paigutati kood, mis genereeriti programmi STM32CubeMX abil. Mikrokontrolleri väljavahetamisel tuleb arendajal muuta selles kihis asuvat STM32 konfiguratsiooni.

5.4.6 Funktsionaalsuse lisamine

Kuna teenused on olemuselt isoleeritud ülesanded, siis nende lisamine toimub nn „horisontaalselt“ (Joonis 5.2). Uue teenuste lisamine ei mõjuta eelnevate teenuste tööd ja sama kehtib ka olemasolevate teenuste muutmisel.



Joonis 5.2: Funktsionaalsuse lisamise näide

5.5 Arhitektuuri implementeerimine

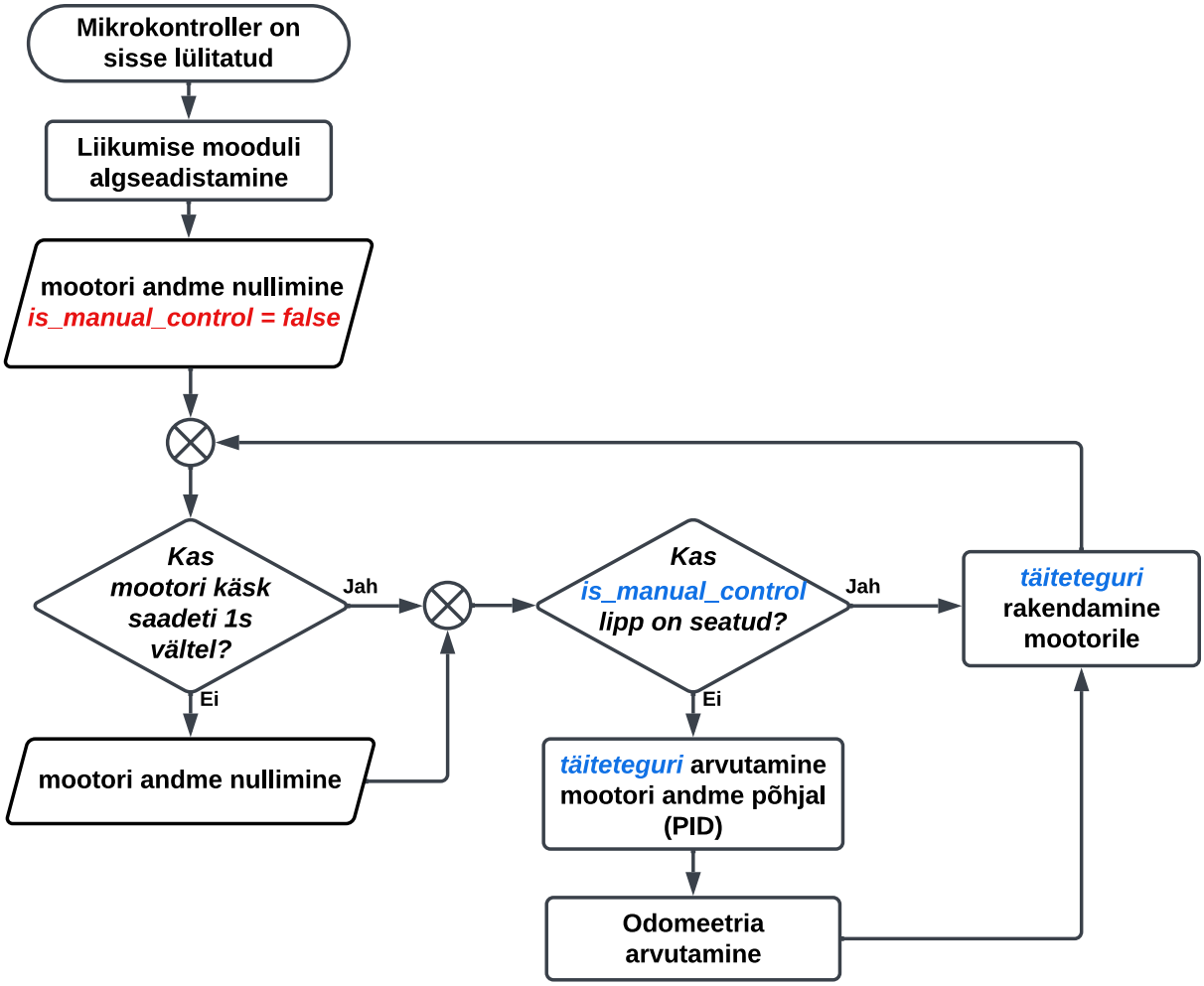
Selleks, et rakendada eelnevalt välja pakutud arhitektuuri disaini, võeti aluseks Robotont 3.0 püsivara ja saavutati järgmised tulemused:

1. Robotondi arendajatele on valminud juhendid, mis sisalduvad vormistusreeglid ja arhitektuuri ülevaade koos näidetega.
2. Failid on paigutatud kaustadesse vastavalt kihtidele.
3. Olemasolevaid komponente on muudetud ja täiendatud, et need oleksid kooskõlas uue arhitektuuriga ning oleksid jälgitavad ja arusaadavad.
4. Uue arhitektuuri jaoks on arendatud välja vajalikud komponendid, sealhulgas kõik liidese kihi ja valitud rakenduskihi komponendid.
5. Kõik implementeeritud komponendid on koodis kommenteeritud.

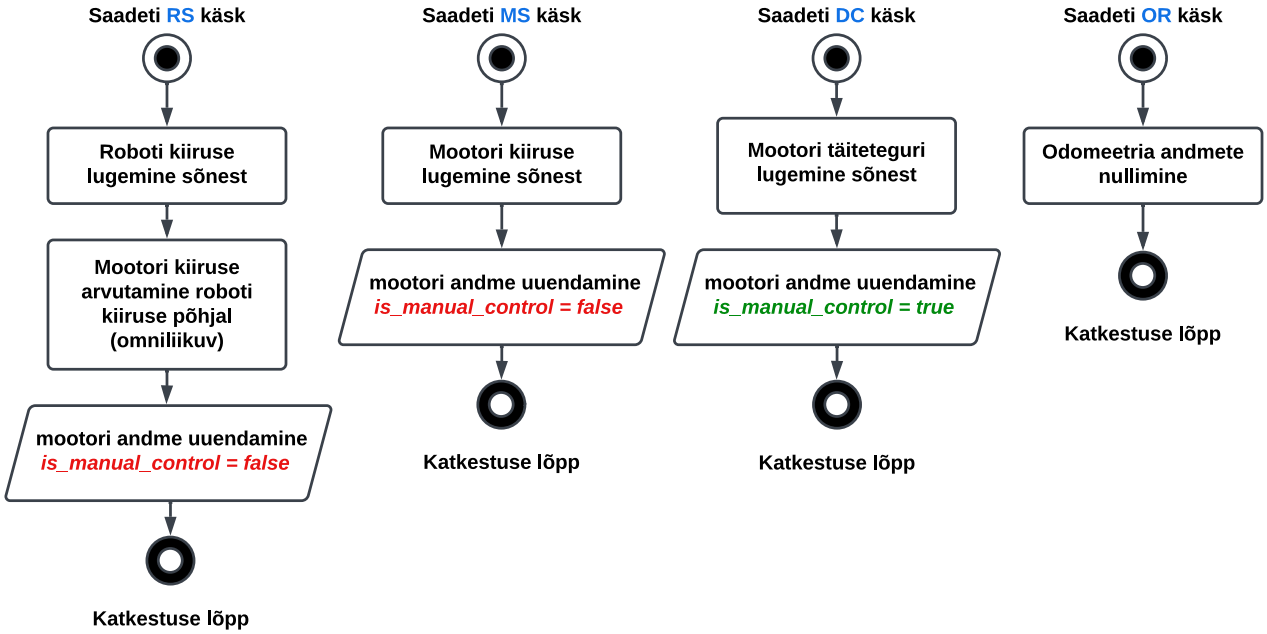
5.5.1 Liikumise teenuse kohandamine

Liikumise teenuse arendamisel võeti aluseks olemasoleva liikumise algoritmi ja kohandati uuele arhitektuurile vastavaks. Algoritm koosneb kahest osast: teenuse põhitsükkel (Joonis 5.3) ja kiiruse uuendamise katkestus (Joonis 5.4). Põhitsüklis kontrollitakse, kas viimase sekundi vältel on vastuvõetud uus mootori juhtimise käsk. Kui uut käsku pole etteantud perioodi vältel saabunud, siis mootorite kiirused algväärtustatakse ja robot jääb seisma.

Uus algoritm jäi peaaegu samaks võrreldes Robotont 3.0 versioonidega. Robotil arendati välja uus režiim, mis "DC" käsu vastuvõtmisel koos täiteteguriga rakendab täitetegurit niikaua, kuniks kasutaja saadab uuesti "RS" või "MS" käsu. Selles režiimis ei arvutata PID-i väärtust ega odomeetriat.



Joonis 5.3: Robotont 3.1 liikumise algoritmi põhistsükkel



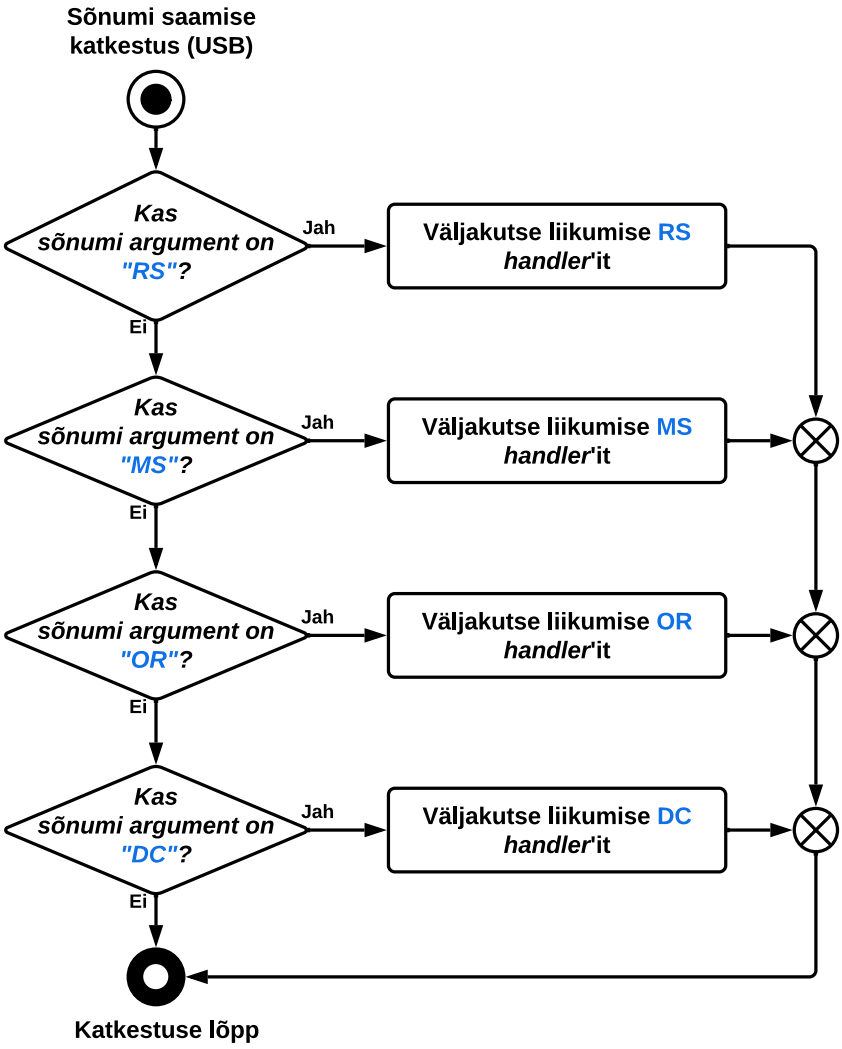
Joonis 5.4: Robotont 3.1 liikumise algoritmi katkestus

5.5.2 Käsu lugemise teenuse arendamine

Saabuva sõnumi lugemiseks ja analüüsimiseks töötati välja eraldiseisev käsu töötlemise teenus. Antud teenus võtab vastu USB sõnumi, lõikab ära sõnumi eespool olevat argumendi ja suunab andmed edasi vastava teenusele. Loodud teenus suurendab koodi loetavust ja implementatsioon võimaldab lihtsasti lisada täiendavaid käske.

Võrreldes eelmise käsustikuga (kuni Robotont 3.0), on uues käsustikus mõningaid muudatusi. Esiteks, käsk "EF" (*effort*) nimetati ümber "DC"-ks (*duty cycle*), kuna see kirjeldab paremini käsu olemust. Teiseks otsustati edaspidi määrata käsu argumendid rangelt kahebaidisteks. See võimaldas optimeerida sõnede töötlemist. Seega, käsk "ODOM" (*odometry*) nimetati ümber "OD"-ks.

Kuna Robotont 3.1 arendusega koos viiakse platvorm üle ROS 2 peale, arendatakse ka uus ROS-i draiver sõnumi töötlemiseks. Seega, protokolli muutmine ei mõjuta oluliselt tagasiühilduvust. Joonisel 5.5 on kirjeldatud käsu lugemise teenuse loogika.



Joonis 5.5: Robotont 3.1 käsu lugemise katkestus

5.5.3 Mootor tarkvara draiveri kohandamine

Selle töö raames kirjutati täiesti ümber mootori draiver, et ta oleks kooskõlas antud nõuetega. Isoleeriti võimalikult palju muutujaid, et vältida juurdepääsu teisest programmiosadest. Kõik mootoriga seotud struktuurid (*struct*) kirjutati ümber nii, et nende sisu oleks lihtsam jälgida. Mõned muutujad on nüüd esitatud *define*-na. Funktsioonid on lihtsustatud ja kommenteeritud, et neid oleks lihtsam jälgida. PWM (*Pulse-Width Modulation*) juhtsignaali töötsükli (*duty cycle*) piirang on nüüd seadistatav selleks ettenähtud funktsiooni kaudu.

5.5.4 Koodri kohandamine

Varasemates versioonides (kuni Robotont 2.0 kaasarvatud) ei toetanud mikrokontroller taimeri koodri režiimi. Seetõttu oli mootori kiiruste koodri põhjal arvutamine realiseeritud tarkvaraliselt. Kuna uus mikrokontroller seda juba toetab, seadistati kooder ümber, et loendada pulsse

riistvaraliselt. Esiteks genereeriti uus seadistud STM32CubeMX tarkvara abil, millega lülitati sisse taimeri koodri režiim. Genereeritud koodi osad kopeeriti Robotondi koodi. Viimaseks kirjutati taimeri liidese jaoks funktsioon, mis võimaldab lugeda ja algväärtusta taimeri koodri väärtust. Sellega vähenes koodi keerukus ja kasutatakse rohkem ära mikrokontrolleri võimekust.

5.5.5 Uus failistruktuur

Selleks, et failid oleksid kooskõlas uue arhitektuuriga, sooritati oluline muudatus failide struktuuris (Joonis 5.6).

Joonis 5.6: Robotont 3.1 püsivara failipuu

Kõigepealt korrastati mikrokontrolleri madalataseme komponendid ning kogu STM32CubeMX poolt genereeritud koodpaigutati *stm-core* kausta. Mikrokontrolleri liideste komponendid paigutati eraldi *mcu* kausta. Sinna kuuluvad autori poolt arendatud liidesed I2C, USB, Taimeri ning GPIO juhtimiseks. Riistvaratoe kihil ja vastavalt *hw* (*hardware*) kaustas asuvad Robotondi riistavara draiverid. Teenuse kihi failid pandi *svc* (*service*) kausta. Selle töö raames kohandati liikumise teenus ja sõne töötlemise teenus. Rakenduse kihi failid paigutati *app* (*application*) kausta, ning hetkel seal on ainult üks komponent: peafail (*main*).

5.5.6 Peafaili kohandamine

Kuna suurem osa funktsionaalsusest uues arhitektuuris eelnevalt kirjeldatud alamkihtides, siis peafaili eesmärgiks on vaid initsialiseerida süsteemi sätted ning juhtida iga teenuste tööd. Selle faili eesmärk on määrata aja nõuded teenusele ja kirjeldada, kuidas teenused omavahel suhtlevad (Joonis 5.7).

```
while (true)
{
    current_tick = system_hal_timestamp();
    if (current_tick >= last_tick + MAIN_LOOP_DT_MS)
    {
        last_tick = current_tick;
        debug_counter++;

        // Service Layer modules update
        movement_update();
        led_update();
        menu_update();

        /*
        status = battery_monitor_getStatus();
        if (status == STATUS_12V_OVERVOLTAGE)
        {
            led_blinkRed();
            movement_stop();
        }
        */

        // Debug
        if (debug_counter % DEBUG_LED_PERIOD_PRESCALER == 0)
        {

            toggleDebugLeds();
        }
    }
}
```

Joonis 5.7: Põhitsükkel koos näitega, kuidas teenused suhtleksid

5.5.7 Vormistureeglite ja stiilireeglite rakendamine

Kogu arendusprotsessi vältel kohandati funktsioone ja muutujaid nii, et nad oleksid kooskõlas uue vormistus -ja stiilireeglitega. Mõned funktsioonid ja muutujaid nimetati ümber, mõned lisati, mõned kustutai ära. Lisati niipalju kommentaare, et kõik käsitlevad komponendid oleksid arusaadavad teistele arendajale. Kommenteeriti nii funktsiooni päised, kui ka mõnede funktsioonide sisu ning muutujad.

5.5.8 Piirangud

Hetkel pole igale teenusele planeeritud eraldi aega koodis. Selle asemel värskendatakse kõiki teenuseid ükshaaval põhitsüklis. Põhitsükli (Joonis 5.7) periood on 20 millisekundit. Kui kõik teenusevärskendused sellesse ajaaknasse ei mahu, siis Robotondi jõudlus langeb. Kõige kriitilisemalt või see mõjutada mootori draiveri PID-kontrolleri värskendamist.

Selle probleemi lahendamiseks vajame viisi, kuidas määrata igale teenusele kindel täitmisperiood. Selleks sobib RTOS (*Real Time Operating System*). RTOS võimaldab iga teenust käsitleda ülesandena (*task*) ning määrata neile prioriteedid. Prioriteedi järgi saavad tähtsamad ülesanded eelise ning tagatakse vajalikud ajalised nõuded.

RTOS-i implementeerimine jääb käesoleva töö raamidest välja, kuid autor soovitab seda kaaluda Robotondi tulevastes versioonides, et vältida probleeme aja planeerimisega.

Kokkuvõte

Käesolevas bakalaureusetöös keskenduti kolmanda põlvkonna Robotont haridusrobotile uue püsivara arhitektuuri kavandamisele ja selle rakendamisele. Töö käigus uuendati kogu repositooriumi struktuur, loodi kaks uut teenust, neli liidest ning viidi sisse muudatusi kõikidesse projekti failidesse. Uus arhitektuur sai edukalt testitud ja on käesolevaks hetkeks teise tudengite poolt aktiivselt kasutusel.

Valitud tööriistad vastavad arenduskeskkonna nõuetele - arendusvahendid on tasuta kättesaadavad ning ühilduvad nii Windowsi kui ka Linuxi operatsioonisüsteemidega.

Kihilise arhitektuuri lähenemine aitab tulevikus arendajatel paremini mõista, kuhu uut funktsionaalsust paigutada. Lisaks aitab see lähenemine täita varasemalt esitatud nõudeid. Disainitud arhitektuur on modulaarne ja võimaldab roboti funktsionaalsuse lisamist ja eemaldamist spetsiaalselt selleks ettenähtud teenuste kihi abil. Kuna komponendid on omavahel eraldatud ning andmevahetus toimub funktsioonide kaudu, siis komponentide lisamine ei mõjuta olemasolevat rakenduse koodi.

Mikrokontrolleri madalatasemelised komponendid on isoleeritud liideste kihi abil, mis võimaldab vähese vaevaga näiteks protsessori või muu riistvara väljavahetamist. Kui uus protsessor ei toeta STM32F4 HAL raamistikku, saab arendaja vajadusel madalatasemelisi komponente välja vahetada ilma ülemise kihi koodi muutmata, sest Robotondi rakenduse kiht ei sõltu konkreetsest raamistikust ega mikrokontrollerist.

Kood on dokumenteeritud nii kommentaaridena kui ka eraldiseisvate dokumentidena, mis õpetavad, kuidas uut arhitektuuri kasutada ja milliseid vormistusreegleid järgida. See suurendab Robotondi pikaajalist tuge ning on juba praegu osutunud kasulikuks tudengitele, kes tegelesid paralleelselt teiste Robotondi alamsüsteemide arendamisega.

Robotondi eesmärk on seni olnud õpetada robotite programmeerimist eelkõige kõrgemal (ROS-i) tasemel. Käesoleva töö raames väljatöötatud arhitektuur koos juhendmaterjalide ja implementatsiooniga pakub õppuritele vormistatud näitelahendust, mille põhjal süvitsi tutvuda ka manussüsteemide arenduspõhimõtetega.

Tänuavaldused

Tänan lõputöö juhendajat Veiko Vunderit ja vabaduse eest, mis mulle anti lahenduste valikul, pideva suhtluse eest kogu töö ajal ning samuti suurepärase panuse eest keeletoimetamisel! Samuti soovin tänada Karl Kruusamäed sisukate arutelude eest töö algusfaasis.

Lõputöös on teksti toimetamisel ja grammatika parandamiseks kasutatud ChatGPT 3.5 (OpenAI) tekstirobotit. Kasutatud sisendid (*prompt*) hõlmavad endas näiteks fraase „sõnasta ümber“ ja „paranda grammatilisi vigu“. Nende sisendite abil saadud väljundi abil kontrolliti ja parandati kogu lõputöö tekstis grammatika ja stiilivigu.

Viited

[1] R. Raudmäe, „Avatud robotplatvorm Robotont“, Tartu Ülikool, 2019. http://hdl.handle.net/10062/64341 (vaadatud: 26. aprill 2024).

[2] T. Bräunl, Toim, „Omni-Directional Robots“, *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*, Berlin, Heidelberg: Springer, 2008, lk 147–156. doi: 10.1007/978-3-540-70534-5_9.

[3] „ROS/Introduction - ROS Wiki“. https://wiki.ros.org/ROS/Introduction. (vaadatud: 4. aprill 2024)

[4] „An Introduction to Robot OS | Toptal®“, Toptal Engineering Blog. https://www.toptal.com/robotics/introduction-to-robot-operating-system. (vaadatud: 30. aprill 2024)

[5] „STM32F4 - ARM Cortex-M4 High-Performance MCUs - STMicroelectronics“. https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html. (vaadatud: 4. aprill 2024)

[6] J. L. Amoros, „Firmware Development Fundamentals | Krasamo“. https://www.krasamo.com/firmware-development-company/. (vaadatud: 4. aprill 2024)

[7] „Introduction to Computer Applications and Concepts. Module 3: System Software. Reading: Firmware“. https://www.coursehero.com/study-guides/zeliite115/reading-firmware/. (vaadatud: 4. aprill 2024)

[8] „GENERAL: Intel HEX File Format“. https://developer.arm.com/documentation/ka003292/latest/. (vaadatud: 28. aprill 2024)

[9] A. Tech, „GCC vs. Clang/LLVM: An In-Depth Comparison of C/C++ Compilers“, Medium. https://alibabatech.medium.com/gcc-vs-clang-llvm-an-in-depth-comparison-of-c-c-compilers-899ede2be378. (vaadatud: 30. aprill 2024)

[10] „Compilation In C: Detail Explanation Using Diagram & Example // Unstop“. https://unstop.com/blog/compilation-in-c. (vaadatud: 10. aprill 2024)

[11] „ST-LINK/V2 - ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32 - STMicroelectronics“. https://www.st.com/en/development-tools/st-link-v2.html. (vaadatud: 4. aprill 2024)

[12] „STSW-LINK004 - STM32 ST-LINK utility - STMicroelectronics“. https://www.st.com/en/development-tools/stsw-link004.html. (vaadatud: 4. aprill 2024)

[13] „GDB: The GNU Project Debugger“. https://sourceware.org/gdb/. (vaadatud: 28. aprill 2024)

[14] „Arm GNU Toolchain“. https://developer.arm.com/Tools%20and%20Software/GNU%20Toolchain. (vaadatud: 5. aprill 2024)

[15] „What is a Software Toolchain?“, Software Quality. https://www.techtarget.com/searchsoftwarequality/definition/software-toolchain. (vaadatud: 28. aprill 2024)

[16] „The GNU C Programming Tutorial“. http://crasseux.com/books/ctutorial/Compiling-multiple-files.html. (vaadatud: 22. aprill 2024)

[17] „Make - GNU Project - Free Software Foundation“. https://www.gnu.org/software/make/. (vaadatud: 10. aprill 2024)

[18] „CMake - Upgrade Your Software Build System“. https://cmake.org/. (vaadatud: 10. aprill 2024)

[19] N. Tsamba, „CMake vs. Make: What’s the Difference?“, Earthly Blog. https://earthly.dev/blog/cmake-vs-make-diff/. (vaadatud: 5. aprill 2024)

[20] „Makefile Tutorial by Example“, Makefile Tutorial. https://makefiletutorial.com. (vaadatud: 28. aprill 2024)

[21] „CMake Tutorial — CMake 3.29.2 Documentation“. https://cmake.org/cmake/help/latest/guide/tutorial/index.html. (vaadatud: 28. aprill 2024)

[22] „What is Code Editors?“ https://www.nobledesktop.com/learn/code-editors/what-is-code-editors. (vaadatud: 10. aprill 2024)

[23] A. C, „17 Best Code Editors for Developers in 2024 to Write Error-Free Code More Efficiently“, Hostinger Tutorials. https://www.hostinger.com/tutorials/best-code-editors. (vaadatud: 28. aprill 2024)

[24] „What is an IDE? - Integrated Development Environment Explained - AWS“, Amazon Web Services, Inc. https://aws.amazon.com/what-is/ide/. (vaadatud: 28. aprill 2024)

[25] „STM32CubeIDE - Integrated Development Environment for STM32 - STMicroelectronics“. https://www.st.com/en/development-tools/stm32cubeide.html. (vaadatud: 10. aprill 2024)

[26] „STM32CubeMX - STM32Cube initialization code generator - STMicroelectronics“. https://www.st.com/en/development-tools/stm32cubemx.html. (vaadatud: 10. aprill 2024)

[27] „Arduino Hardware“. https://www.arduino.cc/en/hardware. (vaadatud: 28. aprill 2024)

[28] „Software | Arduino“. https://www.arduino.cc/en/software. (vaadatud: 10. aprill 2024)

[29] „The Internet of Things with ESP32“. http://esp32.net/. (vaadatud: 28. aprill 2024)

[30] „Get Started - ESP32 - — ESP-IDF Programming Guide v5.2.1 documentation“. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html. (vaadatud: 10. aprill 2024)

[31] „32-bit Microcontrollers (MCUs) | Microchip Technology“. https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus. (vaadatud: 28. aprill 2024)

[32] „MPLAB® X IDE | Microchip Technology“. https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide. (vaadatud: 10. aprill 2024)

[33] „PlatformIO Home — PlatformIO latest documentation“. https://docs.platformio.org/en/latest/home/index.html. (vaadatud: 5. aprill 2024)

[34] „PlatformIO IDE - Visual Studio Marketplace“. https://marketplace.visualstudio.com/items?itemName=platformio.platformio-ide. (vaadatud: 28. aprill 2024)

[35] „Cloud & Desktop IDEs — PlatformIO latest documentation“. https://docs.platformio.org/en/latest/integration/ide/index.html#ide. (vaadatud: 5. aprill 2024)

[36] „Clang-Tidy — Extra Clang Tools 19.0.0git documentation“. https://clang.llvm.org/extra/clang-tidy/. (vaadatud: 11. aprill 2024)

[37] „ClangFormat — Clang 19.0.0git documentation“. https://clang.llvm.org/docs/ClangFormat.html. (vaadatud: 11. aprill 2024)

[38] „Visual Studio Code - Code Editing. Redefined“. https://code.visualstudio.com/. (vaadatud: 28. aprill 2024)

[39] J. Reid, „Visual Studio Code C++ December 2021 Update: clang-tidy“, C++ Team Blog. https://devblogs.microsoft.com/cppblog/visual-studio-code-c-december-2021-update-clang-tidy/. (vaadatud: 11. aprill 2024)

[40] „Features for editing and navigating C++ code in VS Code such as“. https://code.visualstudio.com/docs/cpp/cpp-ide. (vaadatud: 28. aprill 2024)

[41] G. Woke, „The difference between libraries and frameworks“, Simple Talk. https://www.red-gate.com/simple-talk/development/other-development/the-difference-between-libraries-and-frameworks/. (vaadatud: 28. aprill 2024)

[42] „Introduction to Embedded Systems - HAL“. https://sites.google.com/vt.edu/introduction-to-embeddedsystem/api/hal. (vaadatud: 28. aprill 2024)

[43] „Programming | Arduino Documentation“. https://docs.arduino.cc/programming/. (vaadatud: 9. aprill 2024)

[44] „inoplatforms/ino-hardware-package-list.tsv at master · per1234/inoplatforms“, GitHub. https://github.com/per1234/inoplatforms/blob/master/ino-hardware-package-list.tsv. (vaadatud: 25. aprill 2024)

[45] S. Chatterjee, „Why do We Use the Arduino Programming Language? How is it Helpful?“, Emeritus Online Courses. https://emeritus.org/blog/coding-arduino-programming-language/. (vaadatud: 9. aprill 2024)

[46] „STM32CubeF4 - STM32Cube MCU Package for STM32F4 series (HAL, Low-Layer APIs and CMSIS, USB, TCP/IP, File system, RTOS, Graphic - and examples running on ST boards) - STMicroelectronics“. https://www.st.com/en/embedded-software/stm32cubef4.html. (vaadatud: 10. aprill 2024)

[47] „Full API list - API references and tutorials | Mbed OS 6 Documentation“. https://os.mbed.com/docs/mbed-os/v6.16/apis/index.html. (vaadatud: 10. aprill 2024)

[48] „Intro to Real-Time Operating Systems (RTOS)“, Wind River. https://www.windriver.com/solutions/learning/rtos. (vaadatud: 10. aprill 2024)

[49] „ATtiny85“. https://www.microchip.com/en-us/product/attiny85. (vaadatud: 30. aprill 2024)

[50] B. Lamson-Scribner, „Bare Metal Programming: ATtiny85“, Medium. https://medium.com/@bradford_hamilton/bare-metal-programming-attiny85-22be36f4e9ca. (vaadatud: 30. aprill 2024)

[51] V. Bogdanov, „Bare Metal Firmware Development: What, When and How“, rinf.tech. https://www.rinf.tech/bare-metal-firmware-development-what-when-and-how/. (vaadatud: 9. aprill 2024)

[52] „Best Firmware Architecture Attributes - Tayyar GUZEL“. https://www.embeddedrelated.com/showarticle/690.php. (vaadatud: 10. aprill 2024)

[53] J. Beningo, „Concepts for Developing Portable Firmware“, *Reusable Firmware Development*, Berkeley, CA: Apress, 2017, lk 1–28. doi: 10.1007/978-1-4842-3297-2_1.

[54] „How to choose the right firmware architecture for your IoT device | LocoLabs“. http://locolabs.com/how-to-choose-the-right-firmware-architecture-for-your-iot-device-2/. (vaadatud: 3. aprill 2024)

[55] „MCU Motor Studio 3.0 | Toshiba Electronic Devices & Storage Corporation | Europe(EMEA)“. https://toshiba.semicon-storage.com/eu/semiconductor/product/microcontrollers/motor-studio.html. (vaadatud: 3. aprill 2024)

[56] „AUTOSAR Layered Architecture | Renesas“. https://www.renesas.com/us/en/products/automotive-products/autosar/autosar-layered-architecture. (vaadatud: 10. aprill 2024)

[57] „Core Partner AUTOSAR“. https://www.autosar.org/about/partners/core-partner. (vaadatud: 11. aprill 2024)

[58] „What is the OSI Model?“ https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/. (vaadatud: 11. aprill 2024)

[59] D. Fröhling, „Dirk-/WHSR“. 25. november 2021. https://github.com/Dirk-/WHSR (vaadatud: 3. aprill 2024).

[60] „Arduino Nano — Arduino Official Store“. https://store.arduino.cc/products/arduino-nano. (vaadatud: 11. aprill 2024)

[61] „ADC in Arduino | Arduino“. https://www.electronicwings.com/arduino/adc-in-arduino. (vaadatud: 11. aprill 2024)

[62] „Linorobot - Open Source ROS compatible robots“. https://linorobot.org/. (vaadatud: 11. aprill 2024)

[63] „linorobot/linorobot: Autonomous ground robots (2WD, 4WD, Ackermann Steering, Mecanum Drive)“. https://github.com/linorobot/linorobot/tree/master. (vaadatud: 3. aprill 2024)

[64] „Duckiebot (DB-J)“, the Duckietown® project store. https://get.duckietown.com/products/duckiebot-db21. (vaadatud: 18. aprill 2024)

[65] „Duckieboard (LED / Motor Hut)“, the Duckietown® project store. https://get.duckietown.com/products/duckietown-motor-led-board. (vaadatud: 18. aprill 2024)

[66] „duckietown/fw-device-hut: Firmware for the Duckietown Hat“. https://github.com/duckietown/fw-device-hut. (vaadatud: 15. aprill 2024)

[67] E. Mõtshärg, „3D-prinditava kere disain ja analüüs vabavaralisele haridusrobotile Robotont“, Tartu Ülikool, 2023. https://hdl.handle.net/10062/93818 (vaadatud: 4. aprill 2024).

[68] „AIRE - ROSi algkursus“, AIRE. https://aire-edih.eu/sundmus/ros-i-algkursus/. (vaadatud: 29. aprill 2024)

[69] „Odometry“, Game Manual 0. https://gm0.org/en/latest/docs/software/concepts/docs/software/concepts/odometry.html. (vaadatud: 29. aprill 2024)

[70] „STM32F407VG - High-performance foundation line, Arm Cortex-M4 core with DSP and FPU, 1 Mbyte of Flash memory, 168 MHz CPU, ART Accelerator, Ethernet, FSMC - STMicroelectronics“. https://www.st.com/en/microcontrollers-microprocessors/stm32f407vg.html. (vaadatud: 4. aprill 2024)

[71] „Embedded C Coding Standard | Barr Group“. https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard. (vaadatud: 11. aprill 2024)

[72] „CppStyleGuide - ROS Wiki“. https://wiki.ros.org/CppStyleGuide. (vaadatud: 11. aprill 2024)

[73] „afiskon/stm32-ssd1306: STM32 library for working with OLEDs based on SSD1306, SH1106, SH1107 and SSD1309, supports I2C and SPI“. https://github.com/afiskon/stm32-ssd1306. (vaadatud: 26. aprill 2024)

Lisad

Teistega koostöös arendatud Robotont 3.1 püsivara koos juhendiga ja dokumentatsioonidega Githubi koodihoidlas:

https://github.com/ut-ims-robotics/tsigrinski-thesis-2024-robotont-firmware

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Leonid Tšigrinski,

1. Annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose
   "**Õpperoboti Robotont püsivara arhitektuuri uuendamine**"

   mille juhendaja on Veiko Vunder

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teose üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace'i kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.
3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Leonid Tšigrinski

01.05.2024