Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleksandar Gogov

Krmiljenje mobilne platforme z računalnikom ODROID

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Danijel Skočaj

Ljubljana, 2018

Rezultati diplomskega dela so intelektualna lastnina avtorja. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Zmogljivi mobilni roboti so običajno dragi, zato se za množično poučevanje konceptov mobilne robotike pogosto uporabljajo cenovno bolj dostopne mobilne platforme kot je TurtleBot. TurtleBot krmilimo preko serijske povezave, običajno s prenosnim računalnikom pritrjenim na mobilno platformo. Kot alternativa se ponuja računalnik Odroid, ki je manjši, lažji in ima manjšo porabo energije, tako da je s teh vidikov bolj primeren za krmiljenje mobilnih robotov. V diplomski nalogi izvedite sistematično evalvacijo tega pristopa in preverite, če računska zmogljivost Odroida omogoča dovolj učinkovito izvajanje različnih tipičnih nalog mobilne robotike.

Za nasvete in pomoč pri izdelavi diplomske naloge bi se rad zahvalil mentorju izr. prof. dr. Danijelu Skočaju ter asistentu Mateju Dobrevskemu.

Kazalo

Povzetek

Naslov: Krmiljenje mobilne platforme z računalnikom ODROID

Avtor: Aleksandar Gogov

Cilj diplomskega dela je vzpostaviti sistem za krmiljenje mobilne platforme TurtleBot z računalnikom ODROID. Naredili smo sistematično evalvacijo mobilne platforme, katero sta krmilila prenosni računalnik in računalnik ODROID, na realnih nalogah, ki se pojavljajo v praksi in zajemajo celotni spekter kompleksnosti ter različne kombinacije razdelitev računskega bremena med procesnimi enotami. Merili smo procesno moč, ki je potrebna za ocenitev težavnosti nalog, in povprečni čas, ki je potreben za opravljanje določene naloge.

Ključne besede: Turtlebot, ODROID, ROS, evalvacija, procesna enota.

Abstract

Title: Controlling a mobile platform with a single-board computer ODROID

Author: Aleksandar Gogov

The goal of this undergraduate thesis is to establish a system for controlling a mobile platform TurtleBot with a single-board computer ODROID. We made a systematic evaluation of the mobile platform, controlled by a laptop and by ODROID, on real tasks that appear in practice, covering the entire spectrum of complexity and different combinations of the distribution of the calculation burden between process units. We measured the processing power required to estimate the difficulty of tasks and the average time needed to perform a specific task.

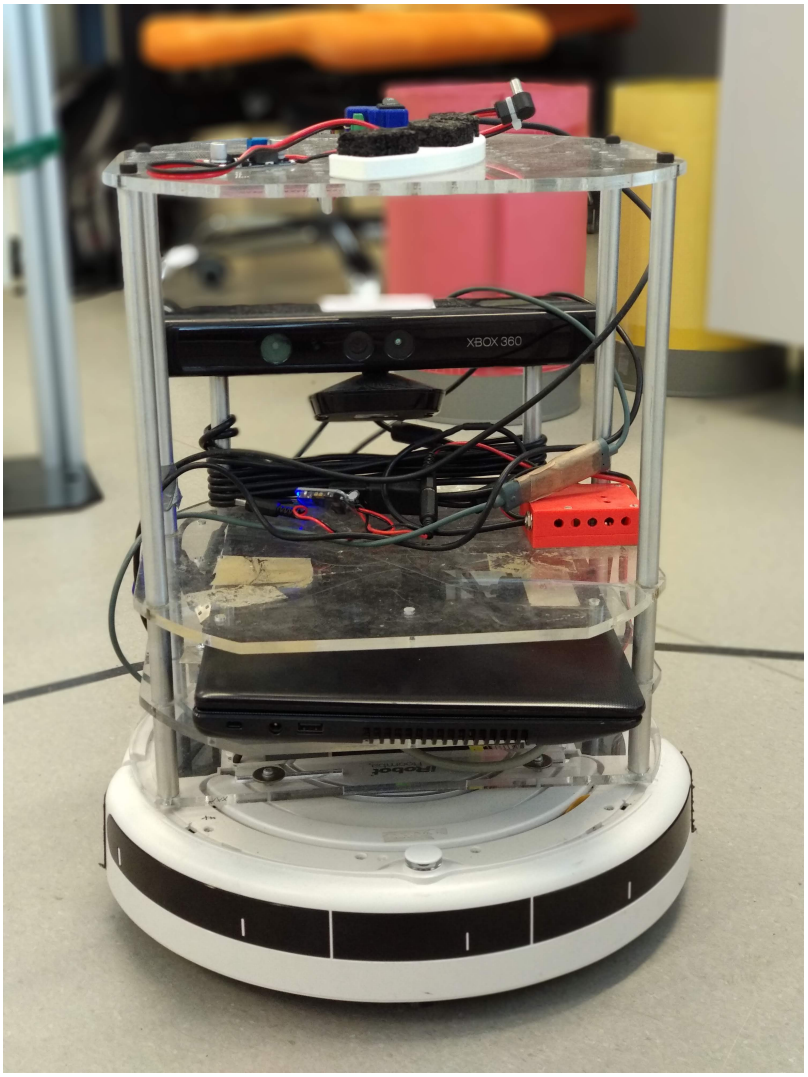
Keywords: Turtlebot, ODROID, ROS, evaluation, processing unit.

Poglavje 1

Uvod

Robotika predstavlja enega izmed vrhuncev tehničnega razvoja. Roboti imajo zmožnost prenašanja in preoblikovanja materiala, kar opravljajo z lahkoto. Danes roboti rudarijo minerale, sestavljajo obdelane materiale v avtomobilske komponente in te komponente sestavljajo v avtomobile. Na vidiku so avtonomni avtomobili, roboti, ki opravljajo gospodinjska opravila in sestavljajo specializiranje stroje na zahtevo. Če želimo še dlje, je robote treba narediti nizkocenovne in enostavno dostopne vsem, zlasti ljudem, ki želijo raziskati področje robotike. Poceni robotov je namreč veliko [17], vendar z vidika senzorjev, navigacije, računske moči in programske opreme niso primerjlivi z roboti, ki se uporabljajo v praksi. Dober kompromis med ceno in opremljenostjo ponuja mobilna platforma TurtleBot, prikazana na Sliki 1.1. Gre za robustnega in vzdržljivega robota, ki je dovolj natančen, zanesljiv ter enostaven za nadzor.

V pedagoške namene se pogosto uporablja TurtleBot opremljen z računalnikom za opravljanje različnih nalog. Imeti prenosni računalnik s procesno enoto TurtleBot, se je izkazalo za nekoliko nerodno zadevo. Vsakodnevna uporaba mobilne platforme zahteva pogosto polnjenje baterije procesne enote. To prispeva k hitremu zmanjšanju kapacitete do take točke, da ga ni več praktično uporabljati. Poleg baterije tudi velikost prenosnega računalnika otežuje delo in pri uporabi robota pride do številnih poškodb



Slika 1.1: Mobilna platforma TurtleBot.

samega računalnika. Zamenjava baterij in včasih celotnega prenosnika je stroškovno neučinkovita, zato rešitev problema vidimo v zamenjavi prenosnega računalnika z mini računalnikom. S tem se znebimo problemov baterije. Velikost in nizka poraba energije mini računalnikov omogočata postavitev v vgrajen in robusten sistem.

Nove različice TurtleBot oziroma TurtleBot3 [6] imajo vse te težave odpravljene, vendar je cenovno ugodneje zamenjati samo prenosnik. To nas

pripelje do problematike, ko se poraja vprašanje, kakšna procesna moč je potrebna za krmiljenje moblinega robota TurtleBot in kako naj bo porazdeljena. V nadaljevanju bomo odgovorili na ta vprašanja na podlagi sistematične evalvacije procesne enote in analize izvedbe različnih težavnostnih nalog in različnih konfiguracij.

Poglavje 2

Mobilna platforma

Celotni sistem je sestavljen iz baze robota, procesne enote, globinskega senzorja in kamere ter programskega okvira, ki povezuje vse komponente v enotni napravi. V tem razdelku bomo opisali vsako komponento mobilne platforme posebej ter alternativno procesno enoto – računalnik ODROID.

2.1 Robot Roomba

Za osnovo mobilne platforme smo uporabili sesalnik iRobot Roomba prikazan na Sliki 2.1. Roomba je član nove generacije avtonomnih robotov, ki opravljajo domača opravila, kot so košnja trave, sesanje ali pomivanje tal. Roomba ima nabor senzorjev, ki omogočajo navigacijo talne površine ter zaznajo prisotnost ovir, umazane pike na tleh in občutijo strme stene, da ne pade po stopnicah. Kasneje v razvoju so ustvarjalci Roombe izdali prirejeno verzijo. Odstranili so dele sesalca ter dodali baterijo in vtič za lažjo integracijo računalnika in senzorja Kintect, zaradi česar je robot zelo primeren za raziskovalne namene. Ta različica robota uporablja dve neodvisno upravljalni stranski kolesi, ki omogočata 360° zavoj. Za dodatno stabilnost baza vsebuje še eno kolo na zadnjem delu. Naprava vsebuje tudi notranjo baterijo, ki lahko zdrži približno 3 ure dela. Baza je pod nadzorom vgrajene procesne enote.

Slika 2.1: iRobot Roomba [10].

2.2 Barvno-globinska kamera Kinect

Primarni senzor mobilne platforme je prva generacija Microsoftove kamere Kinect vidna na Sliki 2.2. To je senzor za 3D zajem okolice, ki je bil izdelan leta 2010, in je namenjen upravljanju igralne konzole Xbox 360. Zaradi nizke cene in nabora senzorjev je postala naprava zelo zanimiva raziskovalcem. Za zajem slike skrbita dva senzorja: prvi je navadna kamera RGB (Red Green Blue) z ločjivostjo 640 x 840, drugi pa je globinski senzor, ki je sestavljen iz infrardečega laserskega projektorja v kombinaciji z enobarvnim senzorjem CMOS, ki snema video podatke v 3D. Zaradi ločljivost kamera Kinect ignorira objekte, ki so manjši od 5 cm, najbolje pa deluje z razdalje od 0,5 do 6,0 metra.

2.3 Prenosni računalnik

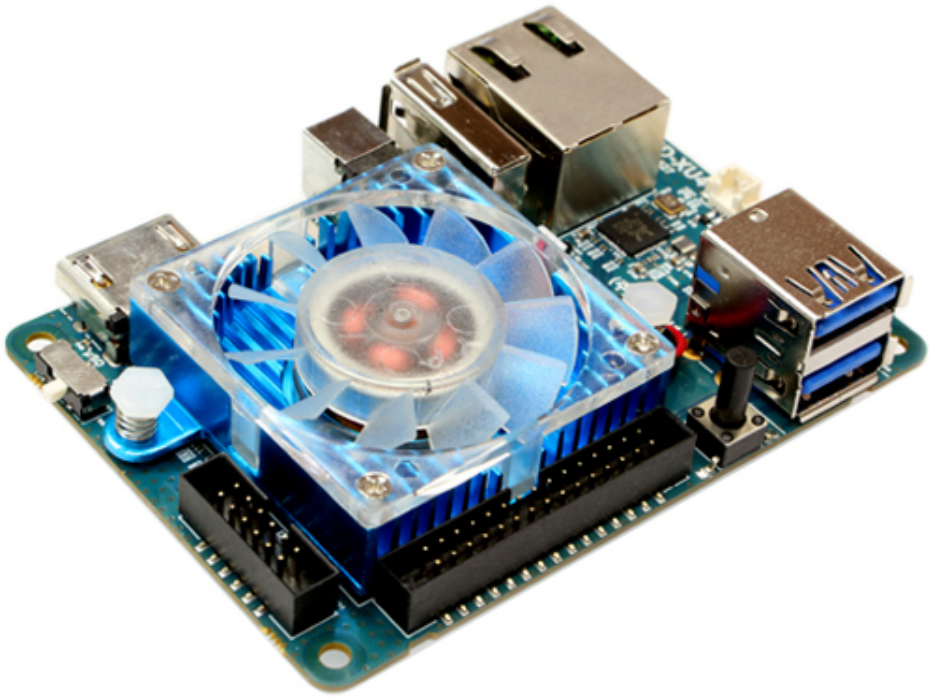
Centralna procesna enota TurtleBota, ki ga uporabljamo, je Asus X301A, prenosni računalnik z operacijskem sistemom Ubuntu. Ima dovolj procesorske moči za delovanje razvojnega operacijskega sistema ROS ali za obdelavo

Slika 2.2: Kinect [12].

podatkov, kot so točkovni oblaki, ki jih zazna 3D senzor Kinect. Vsebuje dvojedrni procesor Intel core i3-2350M 2.30 GHz, ima 120 GB SSD za shranjevanje podatkov in 4 GB RAM-a. Prenosni računalnik vsebuje dva priključka USB, ki zadostujeta za osnovno uporabo. Poleg neposrednega dostopa vgrajena WiFi antena dovoljuje dostop preko omrežja in omogoča porazdelitev delovnega bremena na več računalnikov. Vgrajena baterija ponuja približno 4 ure avtonomije (odvisno od računske obremenitve).

2.4 ODROID

Za rešitev našega v 1. poglavju predstavljenega problema smo se odločili uporabiti nizkocenovni, nizkoenergetski računalnik ODROID-XU4, prikazan na Sliki 2.3. Osemjedrni processor Samsung Exynos 5422 2.1GHz Quad-Core (Cortex-A15) + 1.4GHz Quad-Core (Cortex-A7) in napredna mala grafična enota omogočata brezhibnno delovanje operacijskega sistema in obdelavo podatkov iz senzorjev. Arhitektura procesne enote ODROID temelji na tehnologijah big.LITTLE [8] in HMP(Heterogeneous Multi-Processing) [1]. V grobem procesor ODROID vsebuje dve skupini s po 4 jedri: ena je skupina z večjo učinkovitostjo in porabo moči, ki jo bomo poimenovali ODR HIGH, druga skupina pa dosega nižjo učinkovitost, vendar ima manjšo po-



Slika 2.3: ODROID-XU4 [9].

rabo energije, poimenovali jo bomo ODR LOW. Vgrajena ima Ethernet vrata z gigabitno hitrostjo 2GB, LPDDR RAM in 16GB MicroSD za shranjevanje. Žal naprava ne vsebuje vgrajene brezžične antene za povezavo v omrežje, ampak ker imamo tri vrata USB, lahko uporabimo ena za povezavo z zunajno brezžično anteno. V našem primeru imamo cenovno ugoden brezžični adapter USB TP-LINK s hitrostjo do 150 Mbps. Majhna poraba energije omogoča povezati ODROID z baterijo na mobilni platformi s katerem zagotovimo vgrajenost sistema.

2.5 ROS

Medtem ko Roomba, procesna enota in Kinect predstavljajo strojno opremo mobilne platforme, ta za delovanje potrebuje tudi programsko opremo. Računalnik ima sicer nameščen operacijski sistem Ubuntu, toda za nadzor in delo-

vanje strojne opreme potrebuje še dodatno ogrodje. V ta namen uporabljamo ROS (angl. Robot Operating System) [15], ki je metaopreacijski sistem za robote, ki se razprostira od strojne opreme za različne robote ter senzorje do visokonivojskih algoritmov za navigacijo, računalniškega vida in ostalih opravil. Za bolj varno testiranje in izogibanje poškodbam strojne opreme ROS podpira simulator, v katerem lahko testiramo delovanje robota.

Gradniki ROS-a so t. i. *paketi (ang. package)*, ki predstavljajo zaključen del programa za neko opravilo. Komunikacija med *paketi* se izvaja preko t. i. *sporočil (ang. message)*, ki so v podatkovni strukturi napolnjeni s podatki, in ROS omogoča ustvarjanje svojih sporočil. *Sporočila* potekajo preko logičnih kanalov imenovanih *teme (angl. topic)*. Vsaka *tema* ima svoje ime in vrsto *sporočila* in nanjo se lahko priključi več *polsušalcev (angl. listener)* in *pošiljateljev (ang. publisher)* istočasno. Uporabna orodja (programi) se v ROS-u imenujejo *servisi (ang. services)*. *Servisi* so deli *paketov*, od katerih lahko zahtevamo neko storitev (podobno kot klicanje metod, vendar za razliko od njih niso omejeni le na razred, v katerem se nahajajo, temveč delujejo v sklopu celotnega sistema ROS). Tako v *servisih* kot v *temah* so tudi *sporočila*.

Roboti, ki so v interakciji z okoljem, dobivajo veliko šumov preko senzorjev, kar onemogoča ustvariti iste pogoje za ponovitev določene naloge ali odpravljanje napake programske opreme. Zato ROS podpira sistem, ki posname vse podatke na *temah*, nato pa jih kasneje (vse ali selektivno) v enakem zaporedju predvaja v simulatorju. S tem lahko ponovimo določeno obnašanje robota neomejenokrat.

Poglavje 3

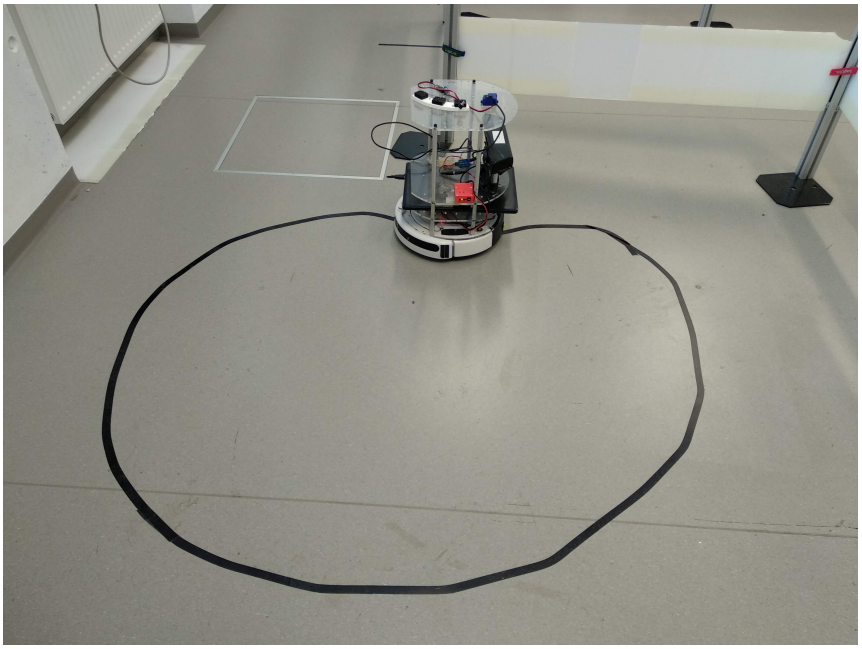
Naloge za evalvacijo

Da bi pokazali zmogljivost računalnika ODROID in ga primerjali s prenosnikom, moramo evalvacijo opraviti na realnih nalogah, ki se pojavljajo v praksi. Hkrati morajo biti naloge različnih težavnosti in s tem lahko ocenimo delovanje sistema na celotnem sprektru kompleksnosti nalog. Odločili smo se za 4 različne naloge, ki se razlikujejo po težavnosti. Zajemajo enostavno premikanje robota, premikanje in postavljanje ciljev po znanem zemljevidu, prepoznavanje obrazov ter segmentiranje valjev s pomočjo točkovnega oblaka. V naslednjem poglavju bomo opisali strukturo in kompleksnosti za vsako nalogo posebej.

3.1 Sledenje črte

Za prvo in najbolj enostavno nalogo smo izbrali preprosto sledenje črni črti na tleh, ki je dejansko povezana v krog, kot je videti na Sliki 3.1. Ta naloga zahteva samo branje štirih IR senzorjev za previse, ki so razporejeni na spodnjem delu Roombe, in sicer: levi senzor, sprednji levi, sprednji desni in desni senzor. Ko vrednost senzorja presega vnaprej določen prag, pomeni, da se na tej strani robota nahaja naša črna črta. Zato ustrezno obrnemo robota tako, da bo črta vedno na sredini oziroma med sprednjim levim in sprednjim desnim senzorjem. S takšnim obnašanjem robota zagotovimo konstantno

sledenje črte.



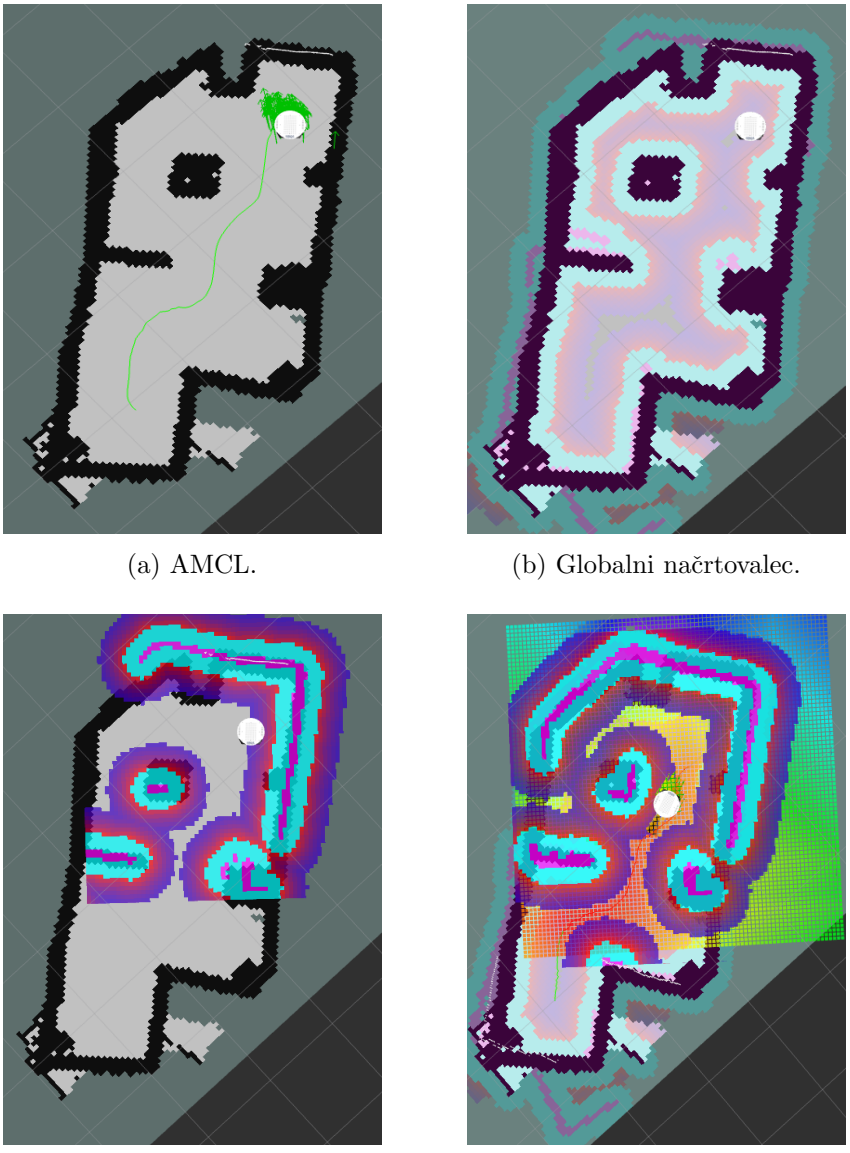
Slika 3.1: Poligon za sledenje črte.

3.2 Navigacija po znanemu zemljevidu

Avtonomna navigacija prek znanega zemljevida je zahtevnejša naloga, potreba po procesni moči je znatno večja. Poleg senzorjev iz baze robota tukaj potrebujemo tudi Kinect.

Najprej mora naš robot zgraditi zemljevid in za to mora poznati svojo točno lego, po drugi strani pa za določanje lege potrebuje zemljevid. Rešitev problema leži v ROS – *paket Gmapping* – ki vsebuje algoritma SLAM [2] in fastSLAM [14] za gradnjo zemljevida danega okolja. SLAM je kratica za *sočasno lokalizacijo in kartiranje (angl. simultaneous localization and mapping)* in je tehnika sestavljanja zemljevida v neznanem okolju, ko se po njej gibljemo in beležimo, kje se nahajamo. Z drugimi besedami povedano je SLAM proces sestave novega ali posodobitev obstoječega zemljevida v ne-

znanem okolju ter sočasno spremljanje leg naprave v obstoječem okolju. V



(a) AMCL. (b) Globalni načrtovalec.

(c) Lokalni načrtovalec. (d) Celotni navigacijski nabor.

Slika 3.2: 2D vizualizacija navigacijskega nabora. Slika (a) prikazuje verjetnostni sistem lokalizacije robota AMCL. Na sliki (b) vidimo zgled globalnega načrtovalca ter na sliki (c) lokalnega načrtovalca. Na sliki (d) vidimo vizualizacije celotnega navigacijskega nabora.

algoritmu ločimo dva problema: kartiranje, ki odgovarja na vprašanje, kako zgleda svet, in lokalizacijo, ki odgovarja na vprašanje, kje v svetu se nahajamo. Funkcija kartanja je sestava zemljevida oziroma predstavitev okolja in interpretacija podatkov s senzorja. Zemljevid se uporablja za določanje položaja in orientacije robota znotraj nekega okolja ter prikaz okolja z namenom planiranja in navigacije. Na drugi strani lokalizacija ocenuje lego, torej položaj in orientacijo robota glede na dani zemljevid in okolico. Sledenje legi je rešitev problema lokalizacije, kje se nahaja naprava oziroma, za koliko se je premaknila, pri čemer mora biti znana začetna lega naprave.

Po izdelavi zemljevida mora biti robot sposoben oceniti položaj na že znanem zemljevidu, zato uporabljamo verjetnostni sistem lokalizacije robota, ki se premika v 2D prostoru AMCL [4] ali *prilagodljiva Monte Carlo lokalizacija (angl. Adaptive Monte Carlo Localization)* prikazan na Sliki 3.2(a). Algoritem uporablja filter delcev za predstavitev porazdelitev verjetnih stanj, pri čemer vsak delec predstavlja možno stanje oziroma hipotezo o tem, kje se robot nahaja. Algoritem se začne tako, da nima informacij, kje se robot nahaja, in predvideva, da je enako verjetno, da bo na kateri koli točki v prostoru. Ko se robot premika, premika delce, da napove svoje novo stanje po gibanju. Kadarkoli robot zazna nekaj, se delci ponovno vzorčijo na podlagi tega, kako dobro se dejanski zaznani podatki ujemajo s predvidenim stanjem. Ultimativno se morajo delci približati dejanskemu položaju robota.

Zdaj ko že imamo zemljevid in poznamo položaj robota, je na vrsti načrtovanje poti in navigacije robota po prostoru do zastavljenega cilja. Za to je odgovoren ROS Move Base. Ko prejme cilj, prevzame nadzor nad robotom in ga poskuša pripeljati do zastavljenega cilja. To doseže s povezovanjem lokalnega in globalnega načrtovalca ter popravnim ravnanjem (angl. recovery behaviour). Globalni načrtovalec, ki je viden na Sliki 3.2(b), deluje optimistično. Odgovoren je za ustvarjanje visokonivojskega načrta, ki mu robot sledi. Predvideva, da je robot okrogle oblike, in ne upošteva njegove dinamike. Za dani oddaljeni cilj si ustvari površni načrt potovanja, ki mu lokalni načrtovalec nato poskuša slediti. Z uporabo zemljevida lokalni

načrtovalec, viden s Slike 3.2(c), ustvari funkcijo v obliki mreže, s katero vkodira ceno potovanja skozi celice te mreže. Krmilnik na podlagi vrednosti teh celic določi linearno in kotno hitrost robota, tako da ju diskretno vzorči. Na ta način za vsako vzorčeno hitrost simulira prihodnje stanje robota glede na trenutno stanje oziroma določi, kje bi se robot nahajal, če bi nekaj časa uporabljal določeno hitrost. Na koncu oceni vsako dobljeno trajektorijo, pri čemer upošteva bližino cilja in prepreke. Če robot ne more najti poti do danega cilja ali se je zaletel v oviro, poskuša popraviti predstavo sveta, za katero misli, da je napačna, s popravnim ravnanjem. Najprej poskusi na zemljevidu odstraniti ovire iz okolice, nato z obratom na mestu počisti in ponovno zazna okolico. Če še vedno misli, da je zataknjen, ponovi postopek bolj agresivno. Če je uspešno opravil ravnanje, nadaljuje z navigacijo do cilja, sicer se potovanje do izbranega cilja opusti. Za lažjo predstavitev celotni nabor navigacije vidimo na Sliki 3.2(d).

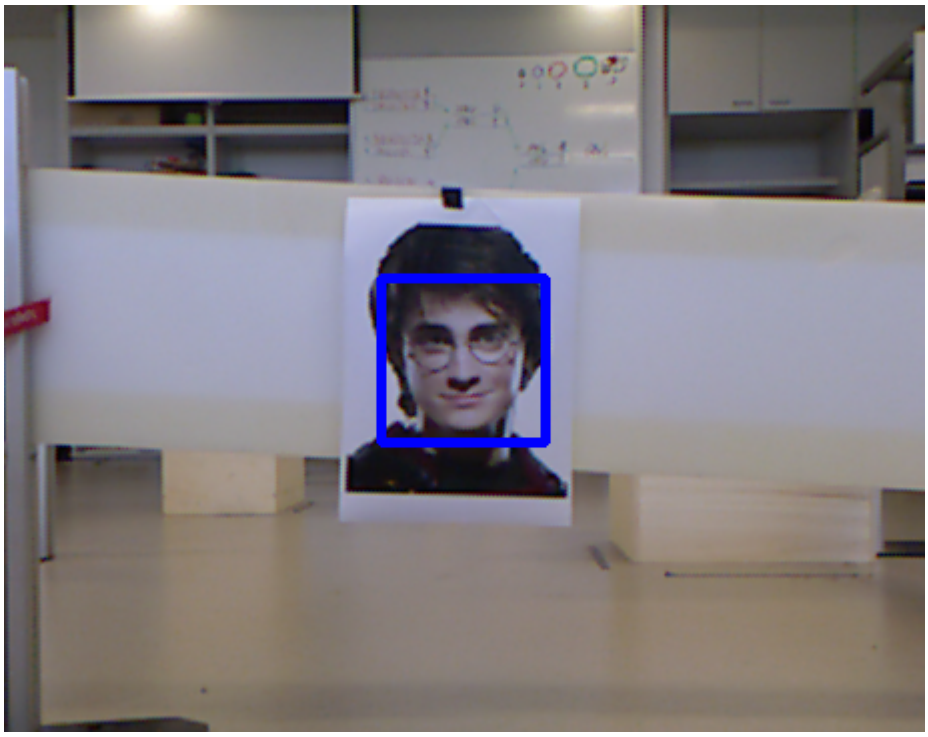
3.3 Prepoznavanje obrazov

Ena izmed najbolj zanimivih stvari je naučiti robota prepoznavati obraze. Poleg tega je zelo dober način oceniti zmogljivost procesne enote, ker je to računsko intenzivna naloga. Za to nalogo potrebujemo samo Kinect oziroma kamero RGB. Problematiko lahko porazdelimo na tri dele: zajem slike, zaznavanje obrazov in prepoznavanje obrazov. Slike dobivamo iz Kinecta oziroma iz navadne kamere RGB, nato se na sliki požene algoritem za zaznavanje obrazov z uporabo Haarjevih klasifikatorjev [18]. To je pristop, ki temelji na strojnem učenju. Na začetku algoritem potrebuje veliko pozitivnih slik (slike obrazov) in negativnih slik (slike brez obrazov). Klasifikator, in sicer kaskada povečanih klasifikatorjev, ki se uporabljajo s Haarjevimi lastnostmi, se nauči z nekaj sto vzorčnimi pogledi določenega predmeta, v našemu primeru obraza, imenovanimi pozitivni primeri, ki so velikosti recimo 20 x 20, in negativni primeri – poljubne slike enake velikosti. Ko je klasifikator zgrajen, ga lahko uporabimo za razvrščanje neke slike. Klasifikator sporoči, ali je slika

obraz ali ne. Če želite poiskati obraze na celotni sliki, lahko premikate iskalno okno po sliki in preverite vsako lokacijo z uporabo klasifikatorja. Klasifikator je zasnovan tako, da je mogoče preprosto „spremeniti“ njegovo velikost, da bi lahko našli obraze različnih velikosti, kar je učinkovitejše od spreminjanja velikosti slike. Če torej želite poiskati predmet neznane velikosti na sliki, je treba postopek skeniranja opraviti večkrat na različnih skalah. Klasifikatorju rečemo, da je kaskaden, zato ker je sestavljen iz več preprostih klasifikatorjev ali stopenj, ki se naknadno uporabijo na regiji interesa, dokler ne pridemo do točke zavrnitve kandidata ali zaznave obraza.

Slika z zaznanim obrazom, kot je videti na Sliki 3.3, je zdaj poslana, da analiziramo, čigav obraz smo pravkar odkrili. To nas vodi do tretjega dela problematike, prepoznavanja obrazov. Zato uporabljamo sistem za prepoznavanje obraza ali algoritem, ki se imenuje Fisherfaces [13]. Prepoznavanje opravi tako, da primerja obraz, ki ga je treba prepoznati, z določeno učno množico znanih obrazov. V sklopu učenja algoritmu podamo obraze in mu povemo, komu ti obrazi pripadajo. Ko se od algoritma zahteva, da prepozna neznani obraz, uporablja učno množico obrazov za prepoznavanje. Da bi razumeli postopek učenja obrazov, se moramo spomniti linearne algebre – vsak vektorski prostor ima ortogonalno bazo. Z združevanjem elementov te baze lahko sestavimo vsak vektor v tem vektorskem prostoru in obratno, vsak vektor v vektorskem prostoru se lahko razgradi na elemente iz te baze. Kjer vemo, da so slike matrike, sestavljene iz sl. elementov, ki predstavljajo vrednosti intenzitete, jih lahko preoblikujemo v vektorje. Ker na učni množici gledamo vektorje iste velikosti, lahko izvedemo algoritem PCA [11] (angl. Principal Component Analysis) in pridobimo lastne vektorje, ki tvorijo osnovo vektorskega prostora. Ti lastni vektorji predstavljajo najpomembnejše lastnosti obrazov iz učne množice. Ni nujno, da lastnosti, ki jih identificira PCA, vsebujejo diskriminantne informacije. Zato izvedemo še LDA [13] (angl. Linear Discriminant Analysis) s katerim dobimo lastnosti, ki najbolje ločujejo različne obraze med seboj. Ko robot dobi nov neznan obraz, ga razstavi na elemente iz vnaprej zgrajene baze. Pogleda katere lastnosti

najbolje opišejo obraz, in tako določi kateri osebi pripada.

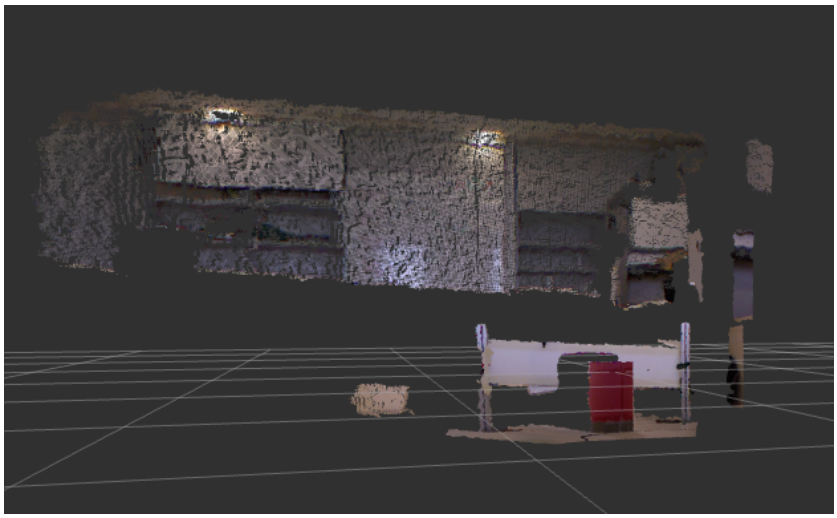


Slika 3.3: Zaznani obraz na sliki.

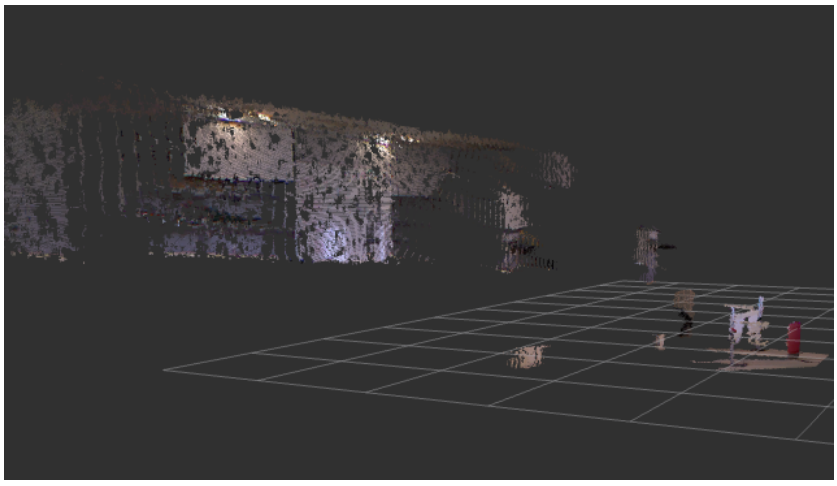
3.4 Segmentacija valjev iz točkovnega oblaka

Da bi predstavili to nalogo, moramo razumeti, kaj sta točkovni oblak ali PCL (angl. Point cloud Library) [16] in segmentacija objektov z uporabo naključnega vzorčenja RANSAC (angl. Random Sampling Consensus) [3]. Kot je opisano v 2. poglavju, iz barvno-globinske kamere Kinect dobimo 2 vrsti informacij: 1. – slika RGB kot običajna kamera, 2. – globinska slika je običajno kodirana kot sivinska slika, vendar namesto intenzitete vsaka sivinska vrednost predstavlja merjeno razdaljo do kamere. Globinska slika je lahko predstavljena kot točkovni oblak, to je urejena množica 3D točk v koordinatnemu sistemu kamere. Če sta slika RGB in globinska slika kalibrirani med

seboj, iz kamere dobimo obarvan točkovni oblak, kot je viden na Sliki 3.4.



(a) Točkovni oblak od spredaj.



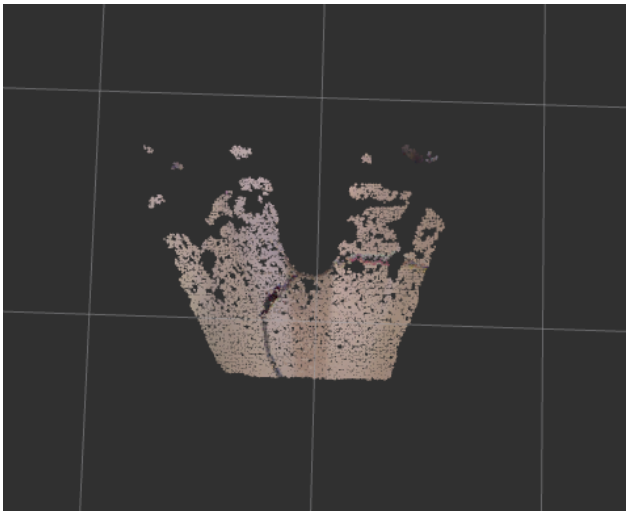
(b) Točkovni oblak viden s strani.

Slika 3.4

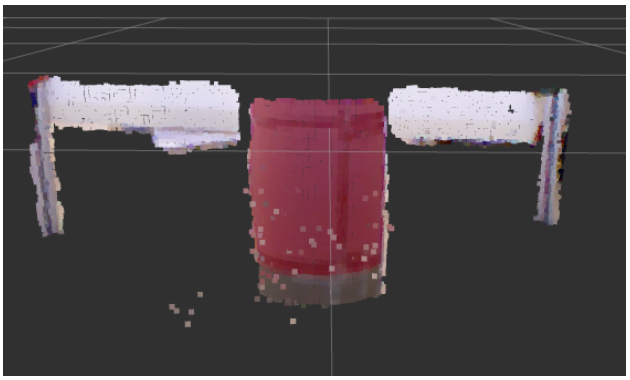
Naloga zahteva izločiti valje iz točkovnega oblaka, zato uporabljamo zgoraj omenjeni algoritem za naključno vzorčenje. To je iterativna metoda za ocenjevanje parametrov matematičnega modela iz množice podatkov, ki vsebuje veliko šuma. Algoritem predpostavlja, da so vsi podatki, ki jih opazujemo, sestavljeni iz zunaj in znotraj ležečih podatkov. Notranje ležeče

podatke lahko opišemo z modelom s posebno množico vrednosti parametrov, medtem ko zunaj ležeči podatki ne ustrezajo temu modelu. Druga nujna predpostavka je, da je na voljo postopek, ki lahko optimalno oceni parametre izbranega modela iz podatkov. RANSAC doseže svoj cilj z iterativno izbiro naključne množice iz originalnih podatkov. Ti podatki so hipotetično znotraj ležeči in ta hipoteza se preizkusi. Postopek se ponovi fiksno številokrat. Vsakič se proizvaja bodisi model, ki je zavrnjen, ker je premalo notranje ležečih točk, bodisi refiniran model skupaj z oceno napake. Obdržimo tisti model, ki ima najnižjo oceno napake.

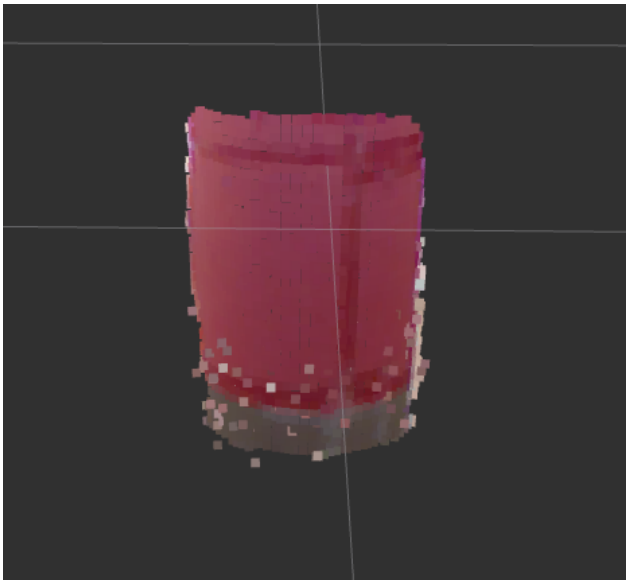
Ta naloga je sestavljena iz dveh stopenj: najprej se izvaja pridobivanje in filtriranje točkovnega oblaka, nato pa segmentiranje ali odkrivanje valjev. V obeh fazah uporabljamo RANSAC. V prvi fazi po zajemu točkovnega oblaka zaradi računske kompleksnosti oblak na začetku filtriramo in s tem zmanšujemo število podatkov za nadaljnjo obdelavo. Nato iz točkovnega oblaka skušamo segmentirati talno ravnino, ki je prikazana na Sliki 3.5 (a). Potem iz originalnega točkovnega oblaka odstranimo talno ravnino vidno na Sliki 3.5 (b) in te podatke predamo drugi fazi. V drugi fazi dobimo točkovni oblak, ki je že filtriran in brez talne ravnine. Sedaj iz takega oblaka s pomočjo RANSAC-a segmentiramo še valj, kot je videti na Sliki 3.5 (c).



(a) Talna ravnina.



(b) Točkovni oblak brez talne ravnine.



(c) Zaznani valj.

Slika 3.5

Poglavje 4

Eksperimentalna evalvacija

Naša predpostavka je, da ODROID lahko nadomesti Asus kot glavno procesno enoto mobilne platforme. Ta predpostavka temelji na dejstvu, da ima ODROID dovolj visoko računsko moč, vsa vhodna in izhodna vrata, ki jih potrebujemo za nadzor TurtleBota, in visoke hitrosti prenosa podatkov. Velika prednost je tudi arhitektura procesne enote, ki zagotavlja nizko porabo energije. V tem poglavju bomo predstavili evalvacijo mobilne platforme opisane v 2. poglavju na nalogah opisanih v 3. poglavju ter testirali zgoraj opisano hipotezo.

4.1 Okvir eksperimenta

ROS je zasnovan tako, da omogoča porazdeljeno obdelavo podatkov. To pomeni, da se lahko porazdeli delovno breme na različne računalnike v istem omrežnem prostoru. To nam omogoča, da opazujemo Asus in ODROID, ko delata skupaj, da bi dosegli isto nalogo. Seveda, da bo to uspešno, moramo naloge porazdeliti na podnaloge. Kriterij za porazdelitev nalog pravi, da mora vsaka podnaloga opraviti del celotne naloge in se razlikovati od ostalih podnalog. Po tem kriteriju in preprostosti naloge Sledenje črte porazdelitev na podnaloge ni bila praktična, ostale naloge smo porazdelili na tri podnaloge in to nam prinese 8 kombinacij porazdelitev bremena, kot je videti v

Tabeli 4.1, za vsako nalogo, ki omogoča porazdelitev.

Tabela 4.1: Kombinacije porazdelitev delovne obremenitve

Podnaloge.		
Prva	Druga	Tretja
ODR	ODR	ODR
ODR	PC	ODR
ODR	ODR	PC
ODR	PC	PC
PC	ODR	ODR
PC	PC	ODR
PC	ODR	PC
PC	PC	PC

4.2 Obremenitev procesorja

Procesna moč je glavna referenčna točka, na podlagi katere primerjamo in ocenjujemo naše procesne enote. Izmerili smo porabo CPE kot odstotek zasedenosti v vseh nalogah. V primeru Asusa je bilo merjenje izvedeno kot povprečje povprečja vseh jeder. V primeru računalnika ODROID pa zaradi arhitekture procesorja, opisane v 2. poglavju, namesto povprečja povprečja vsakega jedra vsako jedro opazujemo ločeno, kot kaže Tabela 4.2. Povprečimo jedra znotraj v skupini, kot je prikazano v Tabeli 4.3, ker jedra v svoji skupini delajo na isti frekvenci. Za boljši vpogled v razliko med procesnimi enotami merimo še čas, ki je potreben za izvedbo določene naloge. Ta meritev se izvaja enako na obeh procesnih enotah, ker uporabljata isto programsko kodo. Čas izvedbe se je izkazal kot dobro merilo uspešnosti pri vseh kombinacijah delovne obremenitve, ki jih vidimo v Tabeli 4.1, na vseh nalogah, ki dovolju-

jejo porazdelitev.

V nadaljevanju bomo opisali porazdelitev nalog na podnaloge, merjenje ter komentar rezultatov za vsako nalogo posebej.

Tabela 4.2: Povprečna poraba jeder v odstotkih na napravi ODROID za vsako nalogo posebej.

Jedra	**Idle**	**Naloga 1**	**Naloga 2**	**Naloga 3**	**Naloga 4**
CPU 0	1,36 %	16,26 %	44,64 %	34,87 %	36,42 %
CPU 1	1,31 %	16,28 %	36,67 %	22,69 %	30,14 %
CPU 2	1,37 %	17,12 %	36,60 %	22,34 %	28,53 %
CPU 3	1,53 %	17,43 %	36,13 %	21,85 %	28,54 %
CPU 4	0,11 %	0,49 %	11,48 %	23,60 %	65,14 %
CPU 5	0,03 %	0,34 %	11,56 %	17,81 %	64,72 %
CPU 6	0,01 %	0,37 %	10,15 %	25,45 %	71,00 %
CPU 7	0,05 %	0,29 %	10,59 %	30,95 %	62,13 %

Tabela 4.3: Povprečna poraba procesorja v odstotkih za vsako nalogo posebej.

Naloge	**Asus 2.30GHz**	**ODR LOW 1.4GHz**	**ODR HIGH 2.1GHz**
Idle	3,56 %	1,39 %	0,05 %
Sledenje črte	4,25 %	16,77 %	0,37 %
Navigacija po mapi	39,36 %	38,51 %	10,94 %
Prepoznavanje obrazov	55,65 %	25,44 %	24,45 %
Segmentacija valjev	75,48 %	30,91 %	65,75 %

4.3 Sledenje črte

Zaradi preprostosti te naloge ni bilo smiselno porazdeliti obremenitev in merjenje je bilo izvedeno na obeh procesnih enotah ločeno. Merili smo čas, ki je potreben za vožnjo enega polnega kroga, ter potrebno procesno moč za opravljanja te naloge. Za zanesljivejše rezultate smo to nalogo ponovili tolikokrat, dokler nismo prišli do 10 uspešnih poskusov na obeh računalnikih. Uspešen poskus pomeni vožnjo celega polnega kroga.

Tabela 4.4: V sekundah merjen čas za vsako ponovitev.

Ponovitev	Asus	ODROID
1	36,88	35,01
2	35,56	35,61
3	33,85	36,19
4	35,05	35,96
5	33,94	36,97
6	34,61	32,28
7	35,15	35,10
8	33,84	34,21
9	34,70	35,05
10	35,00	34,28
Povprečje	34,86	35,07
Standardni odklon	0,88	1,23

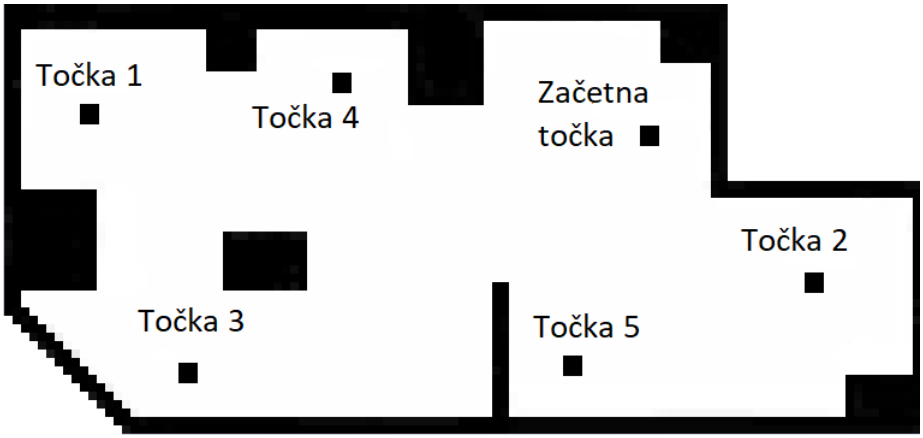
Kot pričakovano sta obe procesni enoti opravili nalogo brez težav, kar pomeni, da ni bilo neuspešnih poskusov. Iz tabele 4.4 razberemo za vsako ponovitev potreben čas, povprečni čas za vožnjo enega kroga ter standardni odklon. Asus povprečno rabi 34.86 sekund, standardni odklon je 0.88 sekund, medtem ko rabi ODROID za vožnjo enega polnega kroga 35.07 sekund s standardnim odklonom 1.23 sekund. Trditev, da ima pri nalogi Sledenje črte manjša procesna moč relativno velik vpliv na izvedbo bolj zahtevnih nalog,

podpira merjenje moči pri vseh nalogah, katerih rezultate so razvidni v Tabeli 4.3. Asus rabi 4,25 % procesne moči, ODROID pa rabi samo 16.77 % moči iz skupine nizkofrekvenčnih jeder.

Iz tega lahko razberemo, da poleg razlike v povprečnemu času vožnjo enega polnega kroga, ki iznaša 210 milisekund, je ODROID zmožen tudi opraviti naloge, ki zahtevajo enostavno branje infrardečih senzorjev in premikanje mobilne platforme, z enako učinkovitostjo kot Asus. Glede na to da potrebuje le nizkofrekvenčna jedra, ima ODROID tudi manjšo porabo energije za podobne naloge.

4.4 Navigacija po zemljevidu

Celotna delovna obremenitev opisana v 3. poglavju je porazdeljena na tri podnaloge: prva podnaloga skrbi za zagon osvnovne funkcije TurtleBota ter za zagon 3D senzorja iz Kinect, druga zajema algoritem za lokalizacijo AMCL, tretja pa skrbi za načrtovanje poti in navigacijo robota.



Slika 4.1: Zemljevid uporabljen pri avtonomni navigaciji.

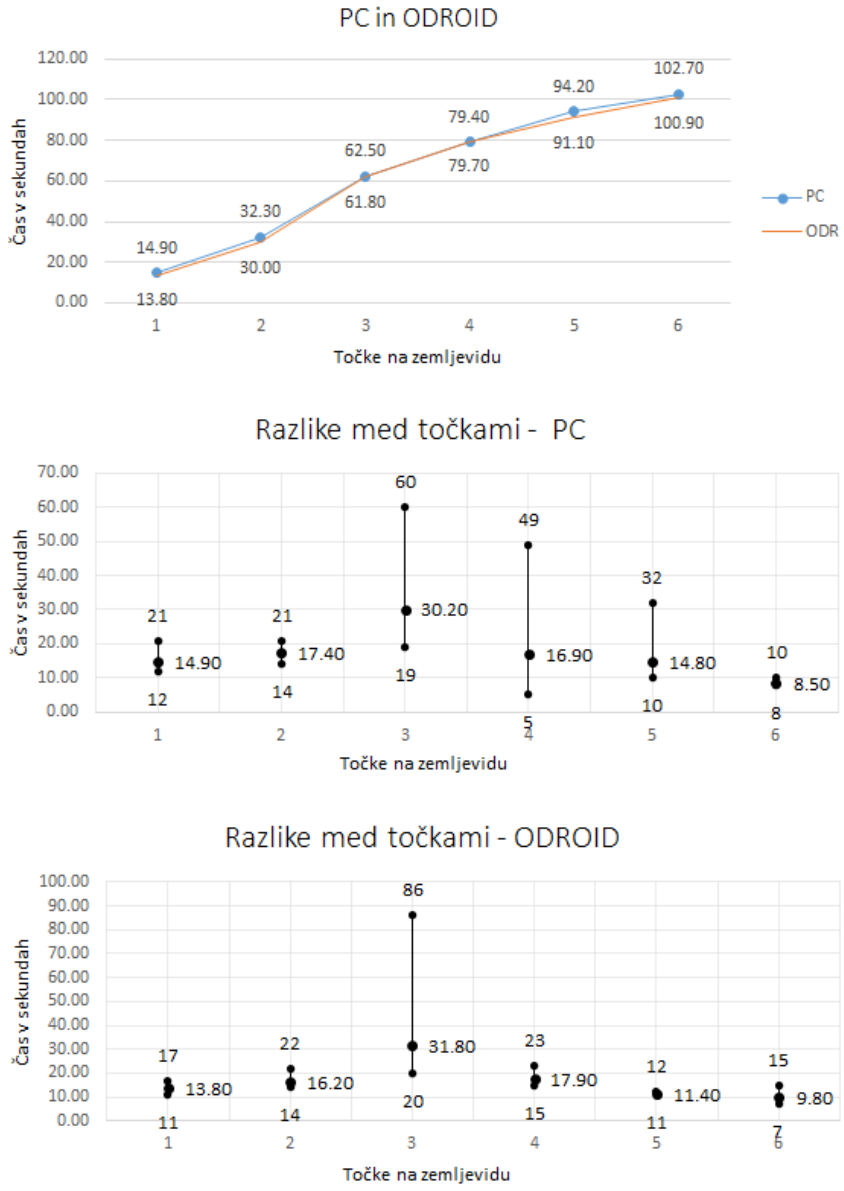
Testiranje je zastavljeno tako, da mobilna platforma obiskuje 5 različnih statičnih točk na zemljevidu, kot je prikazano na Sliki 4.1. Vedno začne iz iste točke, obišče vse ostale točke v istem vrstnem redu in se vrača na začetno

točko. Čas, ki je potreben za obisk vseh točk, merimo tako, da merimo čas od ene do druge zaporedne točke na vseh kombinacijah prikazanih v Tabeli 4.1. Poleg časa smo tukaj merili tudi stopnjo uspešnosti, kar je odstotek, ki smo ga dobili tako, da smo pogledali odstotek uspešnih poskusov iz vseh narejenih poskusov. To nam kaže, kako dosledno določena kombinacija izvaja nalogo. Za vsako kombinacijo smo merjenje ponovili tolikokrat, dokler nismo dobili 10 uspešnih poskusov, in končni rezultat je povprečje vseh uspešnih poskusov. Uspešen poskus pomeni, da robot obišče vse točke v pravilnem vrstnem redu, oziroma neuspešen poskus pomeni, da se robot zatakne.

Tabela 4.5: Rezultati avtonomne navigacije v vseh kombinacijah.

Kombinacije					
Zagon robota in senzorjev	AMCL	Načrtovanje in navigacija	Čas v sekundah	Standardni odklon	Stopnja uspešnosti v odstotkih
ODR	ODR	ODR	100,90	20,47	45,45
ODR	PC	ODR	107,80	15,09	83,33
ODR	ODR	PC	117,30	39,36	45,45
ODR	PC	PC	91,90	11,18	90,91
PC	ODR	ODR	152,80	22,52	52,63
PC	PC	ODR	169,10	39,23	40,00
PC	ODR	PC	102,10	18,14	83,33
PC	PC	PC	102,70	23,48	52,33

Za našo osnovo primerjanja bomo pogledali merjenja brez porazdelitev delovne obremenitve. Na Sliki 4.2 vidimo povprečni čas, ki je potreben za obisk vseh točk, in sicer je Asus potreboval 102,70 sekund, ODROID pa 100,90 s. Skupni čas kaže, da praktično ni nobene razlike med procesorji. Na isti sliki so prikazane povprečne razlike v času med točkami in njihovega maksimuma ter minimuma. Tu vidimo, da se Asuseva izvedba razlikuje od izvedbe računalnika ODROID, kajti pri Asusu pri 3., 4. in 5. točki



Slika 4.2: Primerjava med Asus in ODORID brez porazdelitev obremenitve. Na prvemu grafu je prikazan skupni čas potreben za obisk vseh točk na zemljevidu na Sliki 4.1. Na drugem in tretjem grafu so prikazani povprečje, minimum ter maksimum časa, ki je potreben od ene do druge točke

opazimo veliko odstopanje od povprečja, medtem ko ODROID odstopa samo na 3. točki. Na zemljevidu, ki je prikazan na Sliki 4.1, vidimo, da ima pot od točke 2 do točke 3 veliko ovir in je relativno dolga. To prinese veliko šuma za vse senzorje, zato vidimo odstopanje na tej točki pri obeh procesnih enotah. Odstopanje pomeni nedoslednost obiska teh točk, ki kaže, da je ODROID izvedel nalogo boljše. Naše meritve procesorja podpirajo zaključek, saj je ODROID razporedil breme na obe skupini jeder. V Tabeli 4.3 vidimo, da zasedenost nizkofrekvenčnih jeder znaša 38,51 % ter visokofrekvenčnih 10,94 %, medtem ko Asus izkorišča 39,36 % procesorja.

Tabela 4.5 vsebuje povprečni skupni čas, ki je potreben za obisk vseh točk za vse kombinacije. Če pogledamo druge kombinacije, v katerih se zagon robota in senzorjev ter načrtovanje in navigacija izvajajo na isti procesni enoti, je skupni čas razmeroma blizu našemu osnovnemu merjenju pri obeh procesnih enotah. To je smiselno, ker ista procesna enota bere podatke in nadzoruje mobilno platformo. Pri kombinacijah, kjer ena procesna enota izvaja zagon robota in načrtovanje, druga pa lokalizacijo, imamo kar 83.33 % uspešnosti, kar potrjuje naš zaključek. Če pogledamo kombinacije, kjer se lokalizacija in načrtovanje izvajata na Asusu, vidimo, da je skupni čas približno isti osnovnemu merjenju, in sicer 102.70 sekund, ko Asus zaganja robota in sesnorje, ter 91.90 sekund, ko ODROID izvaja prvo podnalogo, kar je najhitrejši čas za opravljeno nalogo.

Ker vse podnaloge izmenujejo podatke med seboj, posebej podnalogi za lokalizacijo in za načrtovanje in navigacijo, lahko predpostavimo, da ima Asus hitrejši prenos podatkov. V primeru, ko ODROID opravlja zagon robota in senzorjev, ima Asus na voljo več procesne moči za opravljanje ostalih podnalog, kar pojasnjuje najhitrejši čas. Poleg najhitrejšega časa ima ta kombinacija tudi največjo stopnjo uspešnosti, ki znaša 90.91 %, kar potrjuje našo ugotovitev. Kombinacije, ki se jim hočemo izogniti, so tiste, ko uporabljamo Asus za zagon robota in ODROID za načrtovanje in navigacijo. V primeru, ko uporabljamo ODROID za lokalizacijo, je potrebni čas za opravljanje 152.80 sekund. Še slabše rezultate dobimo, ko isto podnalogo opravlja

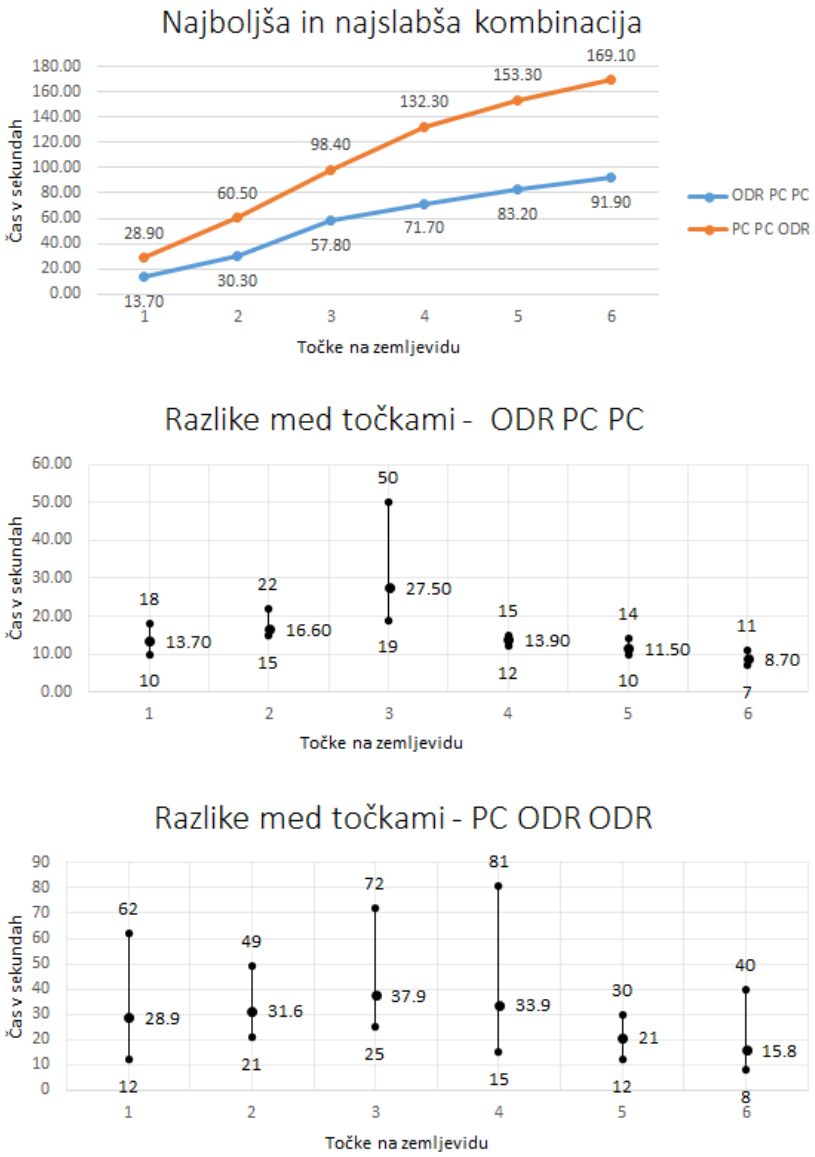
Asus, saj potrebuje 169.10 sekund. Zaradi širokega spektra senzorjev, potrebnih za izvajanje te naloge, pomanjkljivosti zemljevida ter nepravilnosti kolesa procesni enoti prejemata veliko šuma, zaradi česar je verjetnost, da pride do napake, velika. Zato v kombinacijah, pri katerih je stopnja uspešnosti okoli 50 %, ne moremo biti prepričani o rezultatih in sklepih, ki izhajajo iz le-teh.

4.5 Prepoznavanje obrazov

Struktura naloge opisana v 3. poglavju dovoli porazdelitev na 3 podnaloge: zajem slike, zaznavanje obrazov in prepoznavanje obrazov. Merili smo čas od zajema slike in detekcije obrazov na sliki do prepoznavanje obrazov na detekcijah in sporočanja rezultata nazaj. ROS omogoča pregled statistike o izmenjavi podatkov med dvema temama. To smo izkoristili za pridobitev povprečne starosti sporočila oziroma podatkov, koliko časa rabi sporočilo od ene do druge teme. V našemu primeru je to čas, ki je potreben za izvedbo ene podnaloge, in iz tega lahko dobimo potrebni skupni čas za izvedbo cele naloge, ki je vsota časa iz vsake podnaloge.

Tabela 4.6: Rezultati prepoznavanja obrazov v vseh kombinacijah.

	V milisekundah merjen čas			
Kombinacije	Zajem slike	Detekcija obrazov	Prepoznavanja obrazov	Seštevek časa
ODR ODR ODR	10	90	10	110
ODR PC ODR	700	80	10	790
ODR ODR PC	10	30	70	110
ODR PC PC	700	40	30	770
PC ODR ODR	1220	40	30	1290
PC PC ODR	0	150	30	180
PC ODR PC	1540	400	0	1940
PC PC PC	0	30	0	30



Slika 4.3: Primerjava med najboljša in najslabša kombinacija porazdelitev obremenitve. Na prvemu grafu je prikazan skupni čas potreben za obisk vseh točk na zemljevidu na Sliki 4.1. Na drugem in tretjem grafu so prikazani povprečje, minimum ter maksimum časa, ki je potreben od ene do druge točke.

Prva primerjava, ki jo hočemo narediti, je pogledati izvedbo računalnika ODROID v primerjavi z Asusom brez porazdelitvev delovnega bremena. Iz Tabele 4.6 lahko razberemo očitno razliko. Asus rabi 30 milisekund, medtem ko rabi ODROID 80 milisekund več časa, kar je skoraj 4-krat več. Tudi tukaj hočemo evalvirati sodelovanje med procesnami enotami. Podatke o tem izvemo iz rezultatov pri vseh ostalih kombinacijah razporeditev delovnega bremena. Če pogledamo kombinacije, ki imajo ODROID kot zajemalec slike, takoj opazimo časovne konice v kombinacijah, ki uporabljajo za detekcijo obrazov Asus. V obeh primerih čas detekcije znaša 700 milisekund. To ne pomeni, da rabi Asus več časa za detekcijo, temveč da izgubljamo čas pri pošiljanju podatkov iz ene v druge procesne enote. Enako sklepamo za kombinacije, ki imajo Asus kot jemalec slike. Tukaj je razlika še bolj očitna. Kadar pošiljamo podatke detektorju, ki se nahaja na napravi ODROID, v enem primer potrebujemo 1220 milisekund ter v drugemu premiru 1540 milisekund. To kaže, da imamo pri prenosu podatkov ozko grlo. Razlika v časovnih konicah med kombinacijami, ki imajo ODROID za zajemalec slike in tistimi, ki imajo Asus za zajemalec slike, implicira na to, da je podatkovni kanal ožji v smeri iz Asusa v ODROID. Rezultati nam pokažejo, ne glede na to katera procesna enota je na mobilni platformi – Asus ali ODROID, da dokler se zajem slike in detekcija obrazov izvedeta na istem računalniku, se celotni proces prepoznavanja obrazov izvede v zelo kratkem času.

Če postavimo vse kombinacije v perspektivo, je izvedbo računalnika ODROID brez porazdelitev delovnega bremena zadovoljiva. Na to kažejo tudi merjenja procesne moči. V Tabeli 4.3 opazimo, da ODROID potrebuje 25.44 % procesne moči iz nizkofrekvenčne skupine jedr in 24.45 % iz skupine visokofrekvenčnih jedr, medtem ko Asus zapravi malo več kot polovico procesne moči ozirmoa 55.65 %. Iz tega lahko brez skrbi sklepamo, da je ODROID zmožen izvajati naloge, ki potrebujejo matematično zahtevne operacije nad slikami RGB, kot so detekcija obrazov in prepoznavanje obrazov. Seveda moramo pri tem opozoriti, da bo delovanje časovno optimalno, če minimiziramo pošiljanje podatkov med procesnimi enotami.

4.6 Segmentacija valjev iz točkovnega oblaka

Segmentacija valjev je računsko najbolj zahtevna naloga, kajti obsega delo s točkovnim oblakom, ki predstavlja zbirko 3D točk. Tretja prostorska razsežnost uvaja dodatno breme procesorja in to lahko razberemo iz Tabele 4.3. Pri Asusu je obremenjenost zrastla za skoraj 25 % v primeru prepoznavanja obrazov, kar znaša 75.48 %, medtem ko se je obremenjenost pri napravi ODROID povečala pri obeh skupinah jedr, in sicer za 30.91 % pri nizkofrekvenčnih in 65.75 % pri visokofrekvenčnih jedrih. Struktura te naloge je enaka kot pri prepoznavanju obrazov, kar pomeni, da imamo isto porazdelitev podnalog: zajem točkovnega oblaka iz Kinecta, filtriranje oblaka in odstranitev talne ravnine ter segmentacija valjev iz filtriranega oblaka. Enako kot pri prepoznavanju obrazov je bilo merjenje časa izvedeno preko statističnih orodij za teme.

Tabela 4.7: Rezultati segmentacije valjev v vseh kombinacijah.

	V milisekundah merjen čas			
Kombinacije	Zajem oblaka	Filtriranje oblaka	Segmentacija valjev	Seštevek časa
ODR ODR ODR	200	900	240	1340
ODR PC ODR	9940	1000	170	11110
ODR ODR PC	200	960	200	1360
ODR PC PC	10190	240	270	10700
PC ODR ODR	15180	640	490	16310
PC PC ODR	60	810	380	1250
PC ODR PC	15620	1232	78	16930
PC PC PC	60	340	150	550

V Tabeli 4.7 vidimo, da glede na kompleksnost te naloge ODROID rabi 1340 milisekund časa, medtem ko Asus porabi 550 milisekund. Kljub temu da je Asus hitrejši, kar pomeni tudi močnejši, ODROID opravlja to nalogo

v dostojnem času. Ozko grlo pri prenosu podatkov iz ene na druge procesne enote pri tej nalogi še bolj izstopa, posebej v smeri iz Asusa proti napravi ODROID. Če izoliramo kombinacije, v katerih Asus pošilja nefiltriran točkovni oblak do računalnika ODROID, opazimo, da rabi več kot 15 sekund časa za pošiljanje točkovnega oblaka, kar je skoraj 77-krat počasneje kot če to izvede ODROID brez porazdelitev bremena, in približno 256-krat počasneje v primerjavi z izvedbo Asusa brez porazdelitev bremena. Seveda imamo tudi tukaj časovne konice v obratni smeri pošiljanja podatkov. Vendar tudi če je čas pošiljanja za približno 5 sekund krajši, so še vedno nepraktične konfiguracije porazdelitve bremena, še posebej takrat, če jih primerjamo s kombinacijami, ko se zajem slike in filtriranje oblaka izvajata na isti procesni enoti. V tem primeru je za izvedbo tega v povprečju potrebnih okoli 1316 milisekund.

Poglavje 5

Zaključek

V diplomskem delu smo na mobilni platformi TurtleBot, ki ima nameščen operacijski sistem Ubuntu in razvojno okolje ROS, evalvirali dve različni procesni enoti Asus in ODROID. Primerjali smo zmogljivosti in zanesljivosti ter sodelovanje procesnih enot v štirih nalogah, ki zajemajo celotni spekter kompleksnosti. Iz dobljenih rezultatov lahko ugotovimo, ali naša hipoteza, da je ODROID zmožen zanesljivo krmiliti mobilne platforme ter odkriti najboljšo porazdelitev delovnega bremena v primeru sodelovanja med procesnimi enotami, drži.

Preproste naloge, ki zahtevajo enostavno branje infrardečih senzorjev in premikanje mobilne platforme brez kakršnekoli navigacije ali lokalizacije, ne predstavljajo problema za ODROID. Iz rezultatov, pridobljenih pri merjenju naloge Sledenje črte, lahko razberemo, da ODROID opravlja nalogo z Asusu ekvivalentno učinkovitostjo. Navigacija po zemljevidu zahteva bistveno več dela, tukaj rabimo celotno odometrijo mobilne platforme in 3D senzorjev iz Kinecta. Poleg tega so potrebni lokalizacija in načrtovanje poti ter premikanje. Čeprav je ta naloga bolj zapletena in računsko zahtevna, je ODROID pokazal boljše rezultate, ki so vidni v času izvajanja in doslednosti obiskovanja točk. Na podlagi rezultatov smo opazili, da se pri navigaciji po zemljevidu splača porazdeliti delovno breme na obe procesni enoti pri določenih kombinacijah porazdelitve bremena. Pri kombinacijah, kjer se na eni procesni enoti

izvajata zagon robota in senzorjev ter načrtovanje in navigacija, na drugi pa lokalizacija, imamo poleg relativno dobrega časa izvajanja tudi visok odstotek stopnje uspešnosti, kar potrjuje konsistentnost izvajanja naloge. Daleč najbolši razultati dobimo, ko ODROID zaganja robota in senzorje, Asus pa skrbi za lokalizacijo ter načrtovanje in navigacijo. V tem primeru je naloga opravljena v najkrajšem času z najvišjo stopnjo uspešnosti.

Na podlagi merjenja smo ugotovili, da je ODROID zmožen izvajati naloge, ki so matematično zahtevne in potrebujejo operacije nad slikami RGB, kot je tretja naloga po težavnosti oziroma prepoznavanje obrazov. Sicer rabi za zajem slike in detekcijo ter prepoznavanje obrazov ODROID več časa, ampak če postavimo v perspektivo vse kombinacije, je ta razlika v času nepomembna. Pri merjenju časa v vseh kombinacijah smo opazili, da imamo ozko grlo pri prenosu podatkov iz ene na druge procesne enote, posebej v smeru iz Asusa v napravo ODROID. Zato se, če se zajem slike in detekcija obrazov izvajata na isti procesni enoti, izognemo temu in zmanjšamo čas za izvajanje naloge. Iz merjenja pri četrti nalogi zaključimo, da je ODROID zmožen opraviti naloge, ki zahtevajo delo in matematične operacije pri točkovnem oblaku, čeprav za to rabi več časa kot Asus. Pri tej nalogi ozko grlo pošiljanja podatkov še bolj izstopa in razlika je še očitnejša. To pomeni, če zajem točkovnega oblaka in filtriranje oblaka opravimo na isti procesni enoti, da se čas izvedbe bistveno zmanjša. Ugotovili smo, da porazdaljevanje delovnega bremena pri prepoznavanju obrazov in segmentaciji valjev ne zmanjša časa izvajanja. Velikost podatkov, ki se pošiljajo med procesnimi enotami, je bistveno večja kot pri navigiranju na zemljevidu, zato smo tukaj opazili ozko grlo.

Čas in doslednost izvajanja nalog bi se dalo izboljšati tako, da bi postavili boljšo in hitrejšo brezžično anteno na napravi ODROID. S tem bi se izognili časovnim konicam pri pošiljanju podatkov in zmanjšali bi skupni čas za izvedbo nalog. Poleg tega bi lahko izkoristili večjedrnost procesne enote ODROID, če bi zasnovali naloge tako, da bi omogočale paralelno procesiranje na več jedrih istočasno. S tem bi se čas izvajanja bistveno zmanjšal.

Literatura

[1] Hongsuk Chung, Munsik Kang, and Hyun-Duk Cho. Heterogeneous multi-processing solution of exynos 5 octa with ARM big. little^{TM} technology.

[2] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, June 2001.

[3] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[4] Dieter Fox. Adapting the sample size in particle filters through KLD-Sampling. *The International Journal of Robotics Research*, 22(12):985–1003, 2003.

[5] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[6] Willow Garage. Turtlebot. *Website: http://turtlebot.com/*, 2011.

[7] Cyrill Stachniss Giorgio Grisetti and Wolfram Burgard. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23:34–46, 2007.

[8] Marcus Hähnel and Hermann Härtig. Heterogeneity by the numbers: A study of the ODROID XU+ e big. little platform. In *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*, 2014.

[9] Hardkernel. Odroid-xu4. `https://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825`.

[10] iRobot. Roomba. `https://en.wikipedia.org/wiki/Roomba`.

[11] Ian Jolliffe. Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer, 2011.

[12] Microsoft. Kinect. `https://en.wikipedia.org/wiki/Kinect`.

[13] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural networks for signal processing IX, 1999. Proceedings of the 1999 IEEE signal processing society workshop.*, pages 41–48. Ieee, 1999.

[14] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[15] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[16] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). pages 1–4, May 2011.

[17] Marco Ruzzenente, Moreno Koo, Katherine Nielsen, Lorenzo Grespan, and Paolo Fiorini. A review of robotics kits for tertiary education. In *Proceedings of International Workshop Teaching Robotics Teaching with Robotics: Integrating Robotics in School Curriculum*, pages 153–162. Citeseer, 2012.

[18] Phillip Ian Wilson and John Fernandez. Facial feature detection using haar classifiers. *J. Comput. Sci. Coll.*, 21(4):127–133, April 2006.