Tallinna Tehnika Ülikool
Infotehnoloogia teaduskond
Arvutiteaduse instituut

MITME ROBOTI KOOSTÖÖSTSEENI JÄLGIMINE

Bakalaurusetöö

Üliõpilane: Elgar Lepp
Üliõpilaskood: 123858IAPB
Juhendaja: Gert Kanter

Tallinn
2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

_____ _____

(*kuupäev*) (*allkiri*)

Annotatsioon

Lõputöö üheks põhieesmärgiks oli robotite vaheline koostöö, andmete edastamine ja lugemine. Tagada olukord, et nad oleksid teadlikud üksteisest stseenis ning ka teadlikud objektist, mida nad saaksid üksteisele anda.

Töö põhiprobleemideks on robotite asukoha leidmine stseenist ning viisi leidmine, kuidas robotid saaksid omavahel suhelda. Lisaks oli probleemiks leida stseenist objekt, mida robotid saaksid üles korjata. Töö ülesandeks oli juba olemasolevate lahenduste uurimine ja nende vajadusepõhine kasutamine.

Töö tulemusena tuvastatakse stseenist mõlemad robotid kui ka objekt, mida robot peaks üles korjama. Lisaks sellele oli robotitel olemas teenus, kust mõlemad saavad pärida informatsiooni, kus asub objekt ja teine robot. Seda informatsiooni kasutades oli esimesel võimalik liikuda teise roboti juurde.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 7 peatükki, 10 joonist, 0 tabelit.

Abstract

Thesis' one main goal is cooperation between robots, exchanging and reading data. Ensure situation, where they are aware of eachother in the scene and aware about the object that they want to pass from one to another.

Work's main problem was locating robots position in the scene and finding a way for robots to exchange information between them. In addition to that, it was difficult to identify object from the scene that the robots could pick up. One of the thesis' tasks was to examine already existing solutions and using them when needed.

As a result both robots and the object, that will be picked up, are identified in the scene. Furthermore a service exists, where both robots can ask the distance between it and the second robot. Using that information first robot can move to the other one.

The thesis is in estonian and contains 31 pages of text, 7 chapters, 10 figures, 0 tables., etc.

Lühendite ja mõistete sõnastik

ROS	***The Robot Operating System*** ***Roboti operatsioonisüsteem***
PCL	***Pointcloud Library*** ***Punktipilve teek***
API	***Application Programming Interface*** ***Rakendusliides***
RPC	***Remote Procedure Call*** ***Kaugprotseduurikutse***
nD	***n-dimensional*** ***n-dimensiooniline***
OpenCV	***Open Source Computer Vision Library*** ***Pilditöötlusteek***
Rocon	***Robotics in Concert*** ***Robotid koostöös***
LAN	***Local Area Network*** ***Kohtvõrk***
RANSAC	***Random Sample Consensus*** ***Juhusliku Valimi Konsenus***

Jooniste nimekiri

Sisukord

1. Sissejuhatus

Käesoleva tööga üritatakse lihtsustada robotite omavahelist andmevahetust ja ülesannet mida iga robot peab täitma. Töö eesmärgiks on jõuda olukorda, kus kolmandal osapoolel on piisavalt informatsiooni, et juhendada mõlemat robotit. Informatsiooni mida kolmas osapool omab on pliiatsi, Nao ja Peoplebot-i asukoht. Koostöö üheks stsenaariumiks on objekti liigutamine põrandalt lauale. Sellise ülesandega ei tule robotid eraldi toime. Peoplebot roboti manipulaatorite tööulatus ei ole piisav, et põrandalt objekti haarata. Nao robot ei ulata aga pliiatsit laua peale panna.

1.1 Taust ja probleem

Robootika on arenev teadus- ja arendussuund, kuid kuivõrd robotplatvormid on väga erinevad, siis juhtimisalgoritmide ja -tarkvara standardiseerimine töötamaks kõikidel platvormidel on alles algfaasis. Lisaks eelnimetatule on vaja leida ka viisi kuidas üks robot saaks teada andmetest, mida omandab teine robot stseenis, kuna tavapäraselt on robotil ligipääs vahetult robotile monteeritud sensoritele. See muudab keeruliseks nende ülesannete lahendamise, kus on nõutud mitme roboti koostöö.

Käesolev bakalaurusetöö on vajalik Nao ja Peoplebot arendajatele. Selle arenduse vajalikkus seisneb selles, et siis ei pea kirjutama funktsionaalust, mille kaudu nad tuvastaksid roboti ümbrusest teise roboti asukoha. Töö tulemusena valminud tarkvaralahendust on võimalik rakendada ka Peoplebot-i robotplatvormil, sest Peoplebot-il on olemas sama kaamera, mis kolmandal osapoolel.

Eelnevalt kirjeldatud tarkvara arendatakse kasutades raamistikku *ROS*. *ROS* on vabavara ja *ROS*-is loodud paketid on reeglina samuti avatud lähtekoodiga. Avatud koodibaas tagab selle, et erinevad inimesed saavad seda täiendada ja panna oma arendatud liidese üles. See tagab jällegi võimaluse, et vajaminev liides on juba kellegi poolt loodud. Lahendamaks probleemi, et eraldiseisvad robotid, kus igaühes töötab eraldi ROS süsteem, suudaksid omavahel suhelda kasutatakse teeki *Rocon*, mille eesmärk on võimaldada robotitevaheline andmevahetus. Kolmanda osapoole sensorite poolt saadud punktipilve töötlemiseks kasutame teeki PCL ning pildi töötluseks kasutame teeki *OpenCV*. Nende teekide valikul mängis rolli nende avatud

lähtekood ning liidestus *ROS*-i raamistikuga.

Töö teostati 2015. aasta esimeses ja teises kvartalis Tallinna Tehnika Ülikooli Arvutiteaduse Instituudis.

1.2 Ülesande püstitus

Enne, kui sai alustada robotite tuvastamist stseenist, tuleb lahendada probleem, kuidas tagada osapoolte vaheline andmevahetus. Järgmine ülesanne on tagastada küsivale robotile vahekaugus selle ja teise roboti vahel. Tegemist on võrdlemisi mahuka ülesandega, mistõttu jagati see kaheks väiksemaks ülesandeks. Esiteks tuvastati punktipilvest Nao ja Peoplebot. Seejärel leitakse mõlema robot keskpunkt. Lisaks eelnevalt mainitud roboti asukohtadele on vaja tuvastada stseenist väiksem objekt mida robot saaks üles võtta. Tuvastatavaks objektiks sai valitud pliiats.

1.3 Metoodika

Robotitevahelise andmevahetuse jaoks on võimalik kasutada teeki *Rocon*. Mõlemat robotit algselt simuleeritakse demonstreerimaks, et selline andmevahetus on võimalik. Robotite vahekauguse leidmiseks koostatakse katse punktipilved. Mõlemad robotid paigutatakse erinevates positsioonides stseenile ja salvestatakse punktipilved. Rakendades *PCL* teegi algoritme saadud punktipilvedel, on võimalik määrata parameetrid, kuidas kumbagi robotit punktipilvest tuvastada ja nende keskpunkti leida. Pliiatsi tuvastamiseks on võimalik samuti koguda katseandmed. Saadud andmete järgi pannakse paika kindlad parameetrid, et tagada objekti tuvastamine. Objekti tuvastamiseks kasutatakse algoritme, mis on implementeeritud *OpenCV* teegis.

2. Valdkonna ülevaade

Selles peatükis antakse ülevaade varasematest tehtud töödest, arenduskeskkonnast, kus käesolev arendus tehti ning tähtsamatest teekidest, mida kasutati. Lisaks seletatakse „punktipilve" mõistet.

2.1 Varasemad lahendused

Kuna paljud robotiplatvormid ja robotite töökeskkonnad on erinevad, on paljud lahendustest eelmistest erinevad. Seetõttu on keeruline leida lahendust mis sobiks täielikult antud vajadusele. Küll aga on olemas madalama taseme funktsionaalsusi, mis paljudes olukordades jäävad samaks. Need on juba varasemalt arendajate poolt efektiivselt lahendatud ja teekidena internetist kättesaadavad. Käesolevas töös kasutatakse mitmeid selliseid teeke.

2.2 *ROS*-i raamistik

ROS on paindlik raamistik robotitele tarkvara loomiseks. See kogumik tööriistu ja teeke on mõeldud keeruka ja töökindla roboti käitumise loomise lihtsustamiseks erinevatel platvormidel [1]. *ROS*-i tööpõhimõte põhineb protsesside võrdõigusvõrgustikul, mis on üldiselt omavahel ühendatud kasutades *ROS*-i kommunikatsiooni infrastruktuuri. *ROS* rakendab mitut erinevat kommunikatsiooni viisi. Sinna hulka kuulub sünkroonne *RPC*-stiilis kommunikatsioon üle teenuste, asünkroonne andmete voog üle sõlmede, mis vahetavad omavahel sõnumeid, ning ka andmete talletamine parameetri serveris [2]. Parameetri server on jagatud mitme muutujaga massiiv, kust on võimalik *API*-idel üle võrgu andmeid küsida [3].

2.3 *PCL* teek

PCL on laiaulatuslik teek, mis hõlmab arvukalt algoritme 2D/3D piltide ja punktipilvede töötluseks. Selles teegis on implementeeritud mitmeid tipptaseme algoritme. Näidetena võib välja tuua algoritmid, mis tegelevad filtreerimise, pinna rekonstruktsiooni või pinna osadeks jagamisega. Teegis olevate funktsioonidega on võimalik näiteks eemaldada võõrväärtused

mürarikkast andmetest, ühendada mitu 3D punktipilve omavahel kokku, luua pindu punktipilvedest ja neid visualiseerida [4].

2.4 *OpenCV* teek

OpenCV on vabavaraline pilditöötlus ja masinõppe tarkvara teek. *OpenCV* töötati välja, et tagada ühine infrastruktuur pilditöötlus rakendustele ja kiirendada tehisnägemise kasutamist äritoodetes. Selle teegiga saab näiteks tuvastada ja ära tunda nägusid, jälgida kaamera liikumist ja liikuvaid objekte, eraldada 3D mudeleid objektidest, otsida sarnaseid pilte piltide andmebaasist [5].

2.5 *Rocon* teek

Teegiga *Rocon* üritatakse lihtsustada mitme eraldi seisva *ROS* süsteemi omavahelist andmevahetust. *Rocon* teek on *ROS*-i kommunikatsioonikihti kapseldav tsentraliseeritud mitut *ROS*-i ühendav tarkvarasüsteem. *Rocon* teek võimaldab tekitada silla mitme eraldiseisva süsteemi vahel [6].

2.6 Punktipilv

Punktipilv on andmestruktuur mida rakendatakse mitmemõõteliste punktide esitamiseks. Tihti kasutatakse punktipilve selleks, et esitada kolme mõõtmelisi andmeid. Sellises 3D punktipilves hoiab iga punkt endas vaadeldava aluspinna X, Y ja Z ruumilisi koordinaate. Lisades punktipilve igale punktile värviinformatsioon, muutub punktipilv nelja mõõtmeliseks. Punktipilvi saab salvestada kasutades stereo kaameraid, 3D skannereid või siis genereerida programmidega tehislikult [4].

2.7 RANSAC

RANSAC on iteratiivne algoritm, mis on suuteline hindama matemaatilise mudeli parameetreid punktide hulgast, kus esineb võõrväärtusi. Võõrväärtused on punktid, mis arvuliselt erinevad ülejäänud andmetest. RANSAC ei ole deterministlik algoritm, seega tulemused genereeritakse õigesti ainult teatud tõenäosuse piires.

RANSAC-i algoritm valib iteratiivselt juhusliku alamtüübi sisendpunktidest ja tuletab nendest

matemaatilise mudeli parameetrite hinnangu. Leitud mudelile antakse hinnang testides seda sisendandmete vastu ja ka hiljem välja arvutatud punktide järgi, mille kuulumine skeemi on seletatav [16].

2.8 KD puu

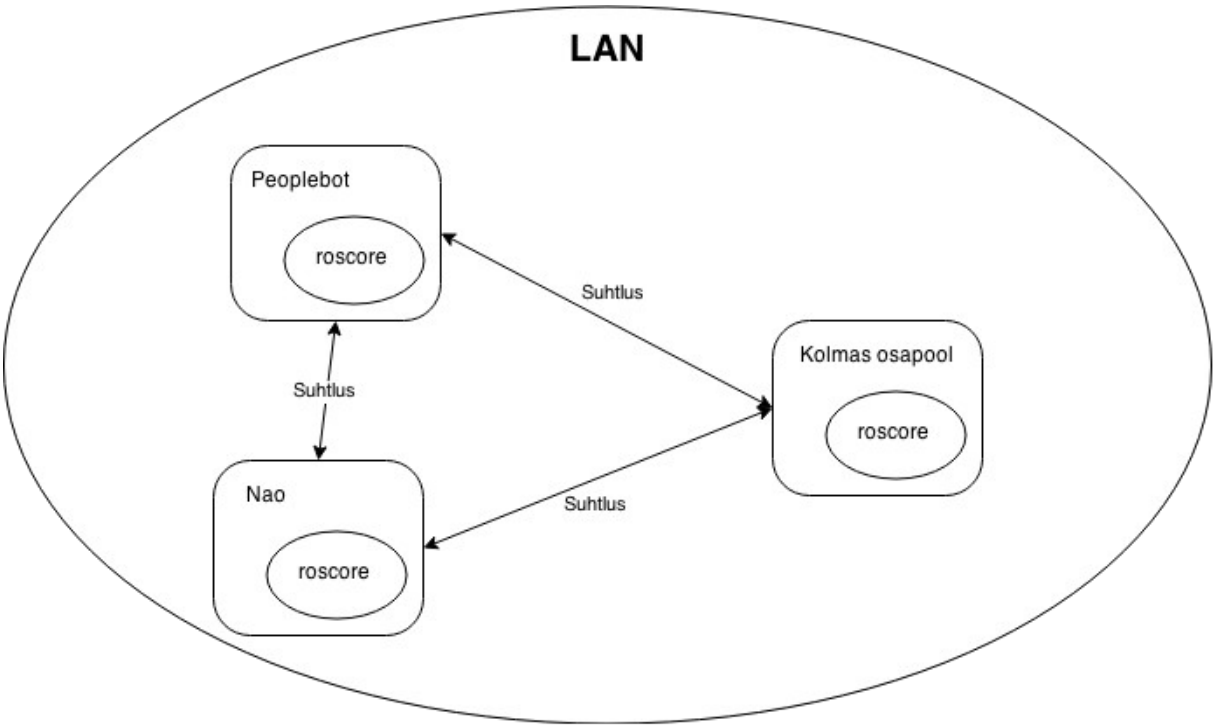
KD puu on andmestruktuur, mida kasutatakse lõpliku arvu punktide hoiustamiseks k-mõõtmelisest ruumist [17]. Oma olemuselt kd puu on binaarne andmestruktuur. Esimesel tasemel andmed jagatakse kahte ossa. Poolitajaks oleks (k-1)-mõõtmeline kindla künnisväärtusega risttasand. Üldjuhul poolitatakse andmed andmehulga mediaanil. Võrrelda käesolevat väärtust väärtusega, millega poolitus tehti, on täpselt teada, kuhu käesolev väärtus kuulub. Kummagi poolega tehakse rekursiivselt läbi samamoodi, et koostada tasakaalustatud kahendpuu [18].

3. Suhtlus erinevate robotite vahel

Käesolevas töös realiseeritakse suhtlus kahe roboti (Nao ja Peoplebot-i) ning kolmanda osapoole vahel. Kolmandaks osapooleks on arvuti, mille külge on ühendatud kaamera, mis jälgib koostööstseeni. Esialgselt toimuks andmevahetus ainult roboti ja kolmanda osapoole vahel. Robotid suhtlevad omavahel ainult väga lihtsate käskudega, mis annavad märku, et kumbki robot võib tegevusega alustada.

3.1 Andmevahetuse realiseerimine Rocon-iga

Mõlemad robotid ning kolmas osapool on täiesti eraldi seisvad süsteemid, nende vahel puudub andmevahetus. Igal süsteemil töötab iseseisev *roscore* (*ROS* raamistiku tuumikprogramm). See on kogumik programme ja sõlmi, mis on hädavajalikud *ROS*-i põhisele süsteemile. Ilma selleta ei saa teised *ROS*-i sõlmed omavahel andmeid vahetada [7]. Selleks, et erinevad *ROS*-i süsteemid saaksid omavahel erinevad sõnumeid saata ja teenuseid välja kutsuda kasutatakse teeki *Rocon*. Suhtlus robotite vahel käib seda teeki kasutades üle

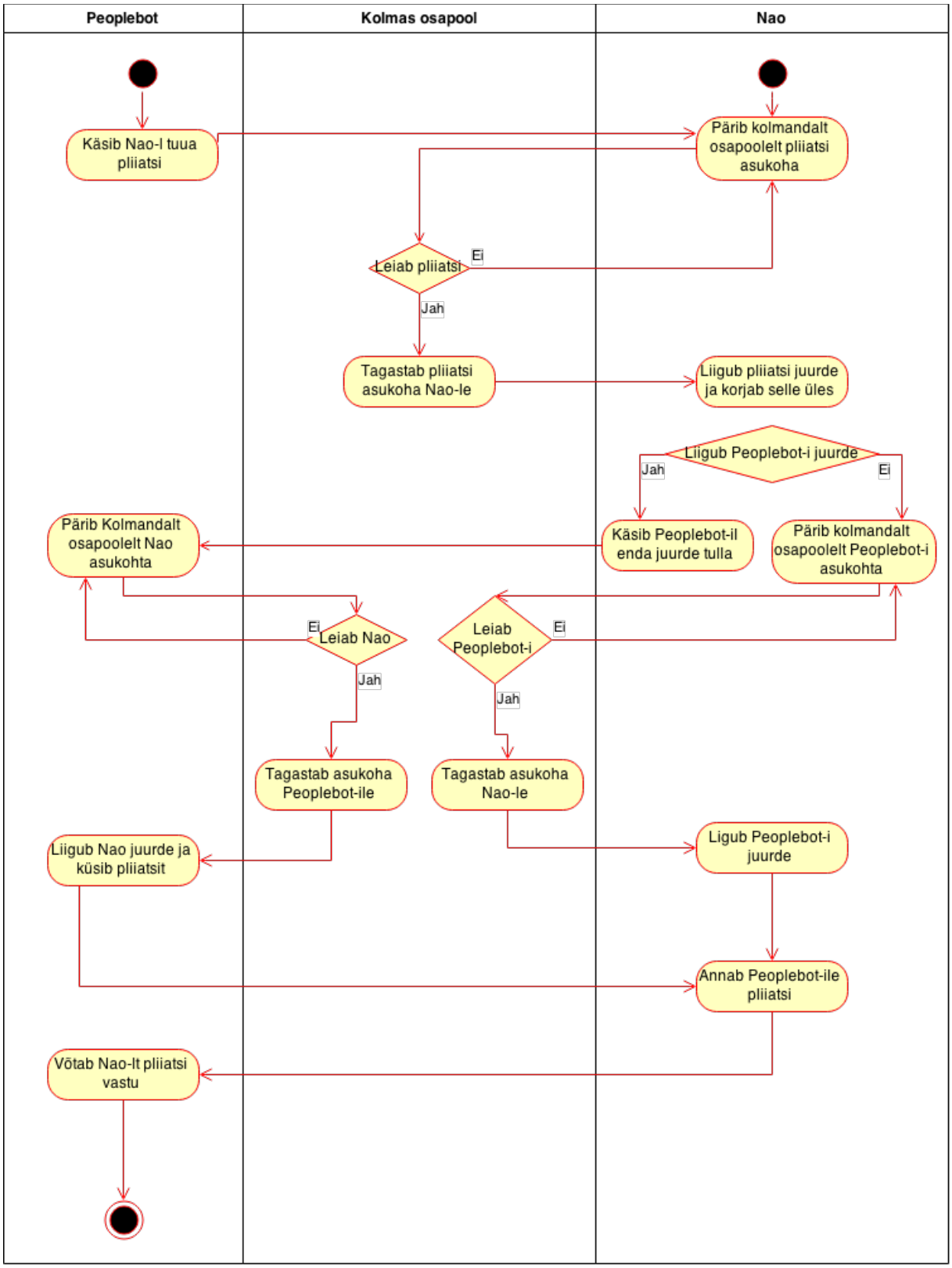


Joonis 1: Rocon-i struktuur

LAN-i, eelnevalt määratud pordil. Vastavat struktuuri illustreeritakse joonisel 1. Sõnumite eduakaks vahetamiseks võetakse *Rocon*-is kasutusele *hub*. Eelpool nimetatud rakendus kujutab endas üht tsentraliseeritud andmetalletuskohta kuhu saavad mitmed erinevad ROS süsteemid andmeid paigutada ja pärida. Andmete hoidmine käib seal võti-väärtus stiilis [8].

3.2 Andmevahetusprotokoll

Käesoleva töö käigus mõlemad robotid suhtlevad omavahel väga lihtsate käskudega. Kogu andmetega suhtlus käib läbi kolmanda osapoole. Valik sai tehtud kuna kolmandal osapoolel on kõige rohkem informatsiooni nii mõlema roboti kui ka pliiatsi kohta. Vastavalt kumma roboti poolt suhtlust alustatakse, jookseb sündmuste jada erinevalt. Alustades Peoplebot-iga, annab robot Nao-le teada, et viimane ulataks Peoplebot-ile pliiatsi. Nao omakorda küsib kolmanda osapoole käest pliiatsi asukohta temast. Juhul kui päringule vastust ei tule, siis tehakse see teatud aja möödudes uuesti. Lõpuks kui saadakse vajalik informatsioon, liigub Nao pliiatsi juurde ja võtab selle kätte. Seejärel pärib robot kolmandalt osapoolelt Peoplebot-i asukohta või siis annab Peoplebot-ile teada, et pliiats on käes ja Peoplebot robot võiks liikuda Nao juurde. Juhul kui Peoplebot peab Nao juurde minema, pärib suurem robot väiksema asukohta. Samamoodi nagu varem, kui robotid oma päringule vastust ei saa, teevad nad selle mõne aja pärast uuesti, korrates päringut seni kuni vastus saadakse. Peale päringu tegemist liigub andmed pärinud robot teise juurde ja Nao ulatab Peoplebot-ile pliiatsi. Sellega lõppeb robotite vaheline suhtlus. Täpne ja lõplik sündmuste ahel on välja toodud joonisel 2.



Joonis 2: Osapoolte vaheline sündmuste ahel

3.3 Andmevahetus *ROS*-i kaudu

ROS-i siseselt käib kogu andmevahetus mööda kanaleid, mida nimetatakse ROS-i terminoloogias *topic*-uteks (ingl teema). Edaspidi nimetame neid lihtsalt kanaliteks. Kanalid on ära nimetatud siinid, mis vahetavad sõnumeid erinevate sõlmede vahel [13]. Punktipilve edastatakse sõnumitena pidevalt, seda ka juhul, kui sõnumeid vastu ei võta. Sõnumid on siin kontekstis lihtsad andmestruktuurid, mis sisaldavad kindla tüübiga välju. Väljade hulka kuuluvad primitiivsed andmetüübid ja massiivid, mis sisaldavad primitiivtüüpe [14]. Kaamera poolt saadetakse välja sõnum, mis on *ROS*-i poolt eelnevalt defineeritud *PointCloud2*. Käesoleva töö käigus kirjutati vastuvõtja, mis võtab vastu saadetud sõnumid. Selleks vastuvõtjaks oli algoritm, mis otsib punktipilvest üles roboteid.

Robotite üksteise asukoha vahe pärimiseks on ära defineeritud teenused kindlatele kanalitele. Teenus kujutab endas päring-vastus andmevahetusprotokolli stiili, kus on eraldi ära defineeritud kaks sõnumit - üks päringu ja teine vastuse jaoks. Teenuse välja kutsumiseks saadetakse vastavale kanalile päringu sõnum ja jäädakse vastust ootama [15]. Kui teenus on oma töö ära teinud, siis pärijale saadetakse vastuse sõnum. Seejärel jääb teenuse pakkuja uut päringu sõnumit ootama. Töös kasutatakse kahte erinevat teenust, kus üks on mõeldud Nao-le vahe pärimiseks ja teine Peoplebot-ile. Teenuste kanalite nimed on vastavalt *distance_from_peoplebot* ja *distance_from_nao*. Seda oleks võimalik lahendada ka ühe teenusega, aga luues kaks erinevat teenust on võimalik päringu sõnum jätta tühjaks ja vältida lihtsalt tõeväärtus sõnumi saatmist, et näidata kumma roboti poolt päring tuleb. Vastuse sõnumiks on *ROS*-i poolt ette defineeritud sõnum *Pose* (ingl poos, asend), kus saab määrata väärtused X, Y, Z ning orientatsiooni. Hetkel kasutatakse ainult väärtusi X, Y ja Z, kuna suuna tuvastamine jäi käesoleva töö skoobist välja.

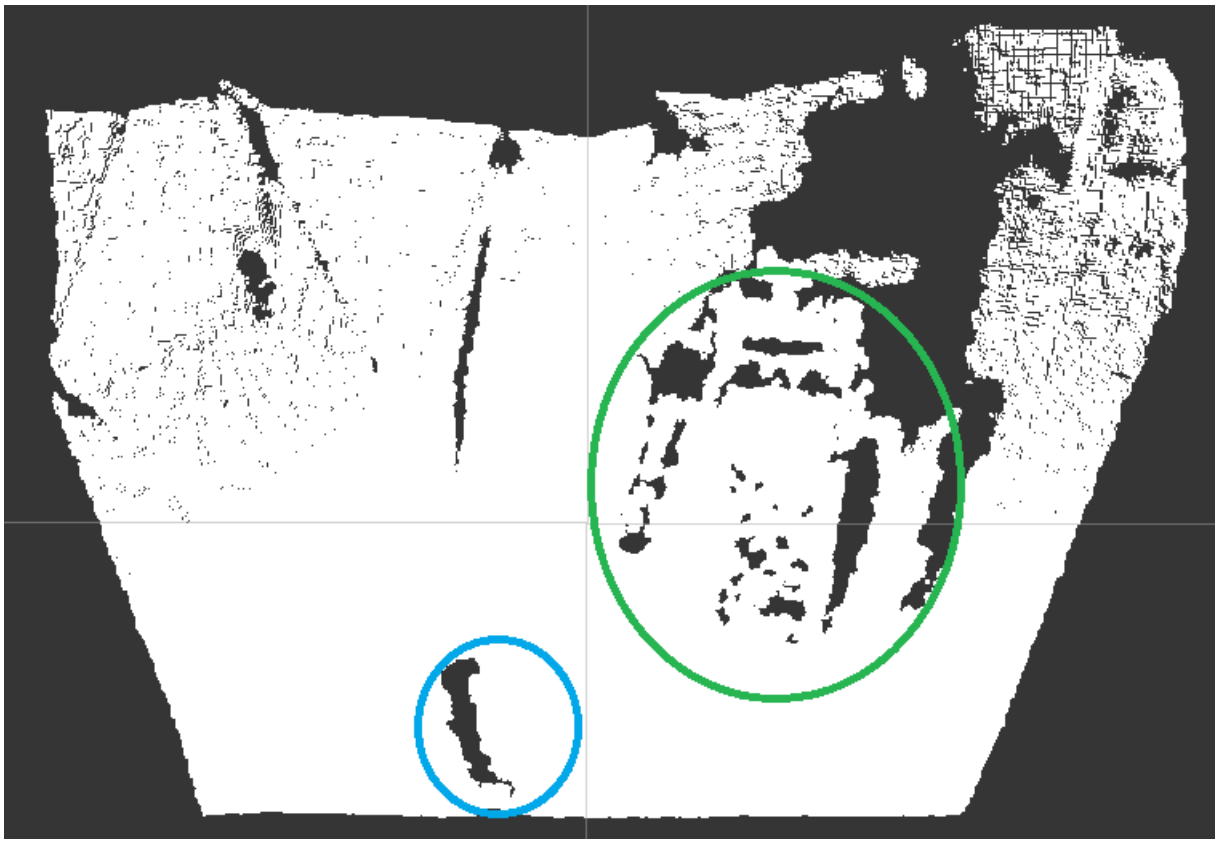
4. Robotite asukoha tuvastamine teineteisest punktipilves

Etteantud töös on punktipilve parseriks kolmas osapool. Kolmandaks osapooleks on hetkel valitud arvuti, mis tagab suurema töötlusvõime kui mõlemad robotid. Seega suudab kolmas osapool kiiremini tuvastada robotid ning anda olukorrast kiiremini, värskeima oleku. Arvuti saab andmed selle külge ühendatud Asus Xtion Pro Live kaamerast.

Lahendusel on hetkel üks eelis selle ees kui robot olukorda parsiks ning üks põhimõtteline puudus. Eeliseks oleks asjaolu, et statsionaarne ja hästi paigutatud kaamera näeb korraga tervet koostööstseeni. Peoplebot suudab kaamerat keerata ja näha ka selja taha, aga kaamera vaateväli ei ulatu roboti keha tõttu põrandani. Selja taga oleva põranda nägemiseks peaks ta tegema 180 kraadise pöörde. See muudaks robotite tegevuse aeglasemaks. Üleüldiseks puuduseks on robotite suuna kindlaks määramine. Nimelt lihtsalt parsimise kaudu on raske tuvastada, millises suunas robot teatud hetkel oma esiküljega on. Üheks võimaluseks on seda probleemi lahendada koodisiltidega. Koodisilt on kahemõõtmeline maatriks triipkood, mis on loetavad kas koodisildi triipkoodi lugejatega või tavaliste telefoni kaameratega. Nad suudavad hoida endas rohkem informatsiooni (kuni 4000 tähemärki ühes koodis), kui tavalised triipkoodid, mida kasutatakse tänapäeva kaubanduses [9]. Käesoleva töö raamesse see eesmärk ei mahu.

4.1 Punktipilve vähendamine

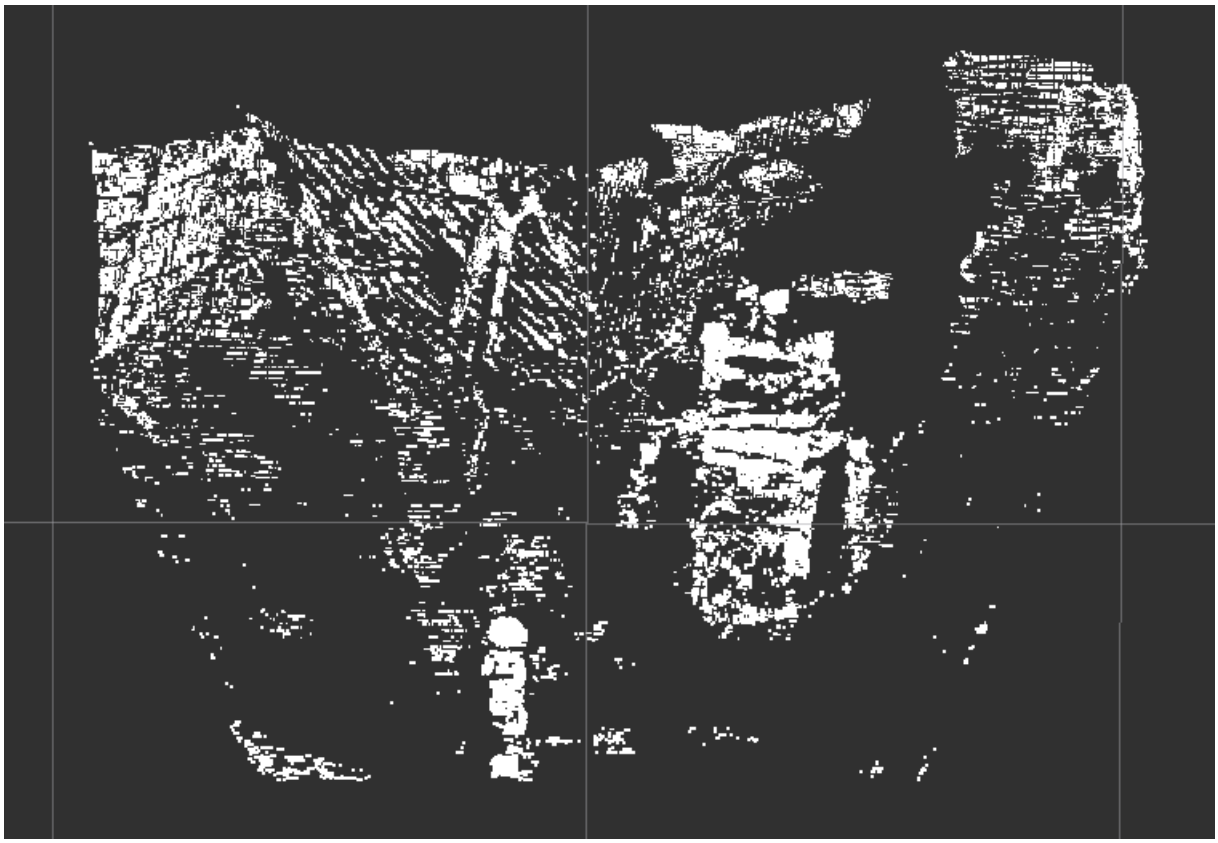
Esimene meetod, mida kasutatakse punktipilve vähendamiseks, on *PCL* teegi klass *VoxelGrid.* Voksel tähistab kuubikmahu väärtuse ühikut, mille keskpunkt on tervikliku võrgustiku üks punkt [10]. *VoxelGrid* moodustab etteantud punktipilvest vokselite võrgustiku. Seejärel leitakse igas vokslis selle massikeskpunkt. Uues punktipilves on ainult saadud massikeskpunktid. Selline lähenemine on küll natukene aeglasem, kui lihtsa keskpunkti leidmine, aga see kujutab pinda täpsemini [11]. Võimalik on veel määrata, mis mõõtmetega vokslit on soov luua. Hetkeses töös andis parima tulemuse siis kui kasutada voksleid, mille pikkus, laius ja kõrgus on 1cm. Punktipilve vähendamise tulemusi on näha joonistelt 3 ja 4. Kolmas joonis on sissetulev punktipilv, kus rohelise ringiga on tähistatud Peoplebot ja sinise ringiga Nao. Joonis 4 on punktipilv peale vähendamist.



Joonis 3: Otse kaamerast tulev punktipilv

Joonis 4: Punktipilv peale vähendamist kasutades VoxelGrid-i

Järgmisena kasutatakse algoritmi, mis eemaldab punktipilvest põranda. Selleks tuleb käesolev punktipilv osadeks jagada. Tegevuse juures saab ära määrata millist tüüpi mudeli ja meetodi järgi soovitakse osadeks jagada. Mudeliks kasutatakse *SACMODEL_PLANE*-i, mis on mõeldud tasapindade osadeks jagamiseks. Meetodiks valiti *RANSAC*, mis aitab tuvastada tasapinda isegi juhul, kui punktipilves kõik punktid ei ole ühel joonel. Lisaks tuleb ära määrata tasapinnast eemal oleva punkti kaugus ehk vea künnis, mis on lubatud, et punkti veel tasapinna hulka kaasata. Esimese osadeks jagamisega ei pruugi kõiki tasapinna punkte üles leida, mistõttu tehakse seda tegevust tsüklis seni kuni alles jääb teatud protsent algsest punktipilvest. Käesoleva töö puhul andis parima tulemuse, kui algsest punktipilvest jäi alles ainult viiendik või vähem. Peale põranda eemaldamist jääb punktipilves alles ainult sein ja robotid, nagu seda on näha jooniselt 5.



Joonis 5: Punktipilv peale põranda eemaldamist

4.2 Punktipilvest robotite tuvastamine

Robotite tuvastamiseks kasutatakse töös eukleidilist klasterdamist. Selle üheks põhimõtteks

oleks punktipilve jagamine väiksemateks klastriteks, et kogu punktipilve protsessimine oleks vähem kulukas. Teine põhjus oleks, et õigete parameetrite valimisel on võimalus kohe leida vajalikud objektid. Lihtsa andmete klasterdamise eukleidilises mõttes saab implementeerida kasutades ära kolme dimensioonilise võrgustiku ruumi alajaotusi, mis kasutavad fikseeritud laiusega kastikesi. Üldisemalt võib öelda, et see klasterdamine kasutab kaheks alamsõlmega puu andmestruktuuri. Kasutades eukleidilist klasterdamist, käiakse läbi tulemuse saavutamiseks järgnevad punktid:

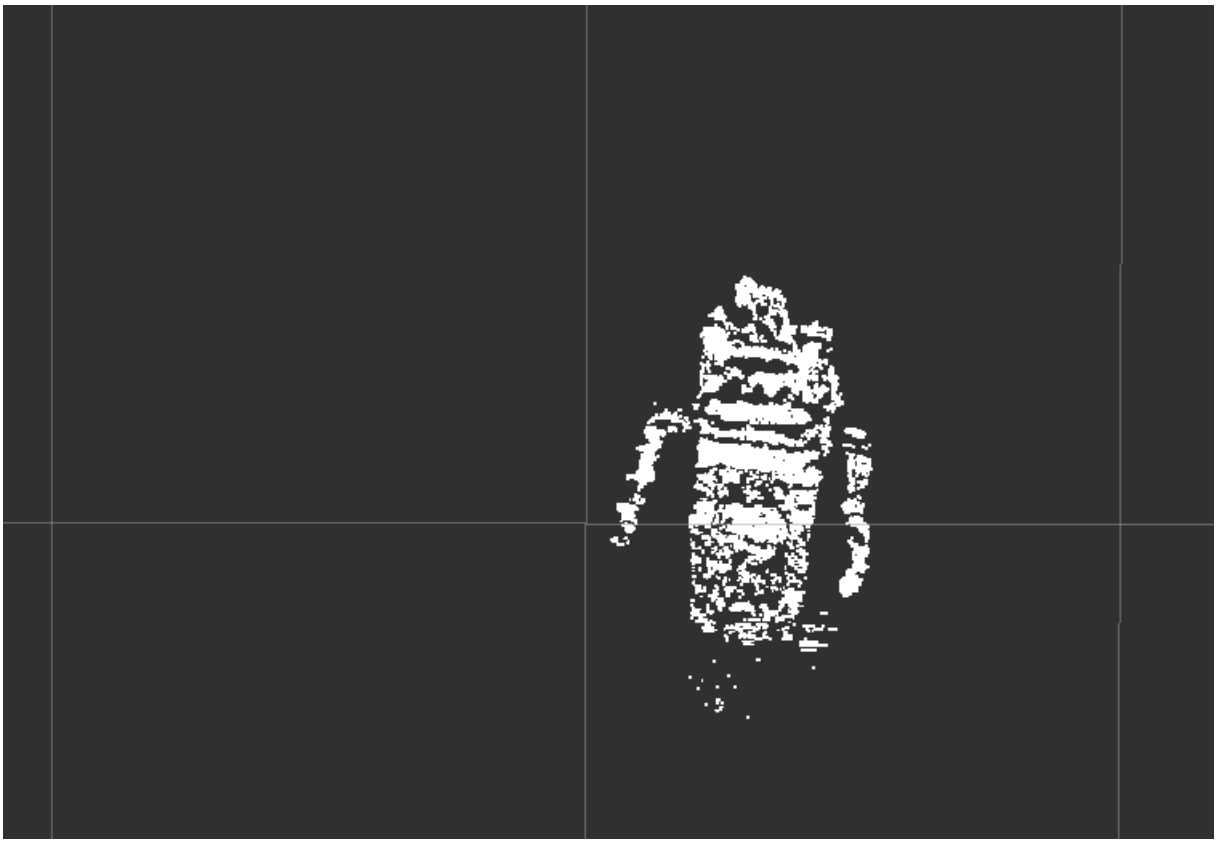
- Luuakse Kd-puu kujutus etteantud punktipilvest P.
- Tehakse tühi massiiv M klastrite jaoks ja järjekord J punktide jaoks, mida on vaja läbi vaadata.
- Seejärel iga punkti p_i,mis kuuluvad punktipilve P, tehakse läbi järgnevad sammud:
    - Lisatakse punkt p_i järjekorda J
    - Iga punkti p_i kohta järjekorras J läbitakse järgnevad sammud
        - Otsitakse punkti p_i naabrite kogum p_{ik}, mis asuvad raadiuses r
        - Iga naabri, mis kuulub punktipilve P, puhul tehakse kindlaks, kas seda on juba vaadatud. Juhul kui ei ole, siis lisatakse see järjekorda J.
    - Kui kõik punktid järjekorras J on läbi vaadatud, lisatakse J massiivi M. Järjekord J samal ajal tehakse tühjaks
- Algoritm lõpetab töö, kui kõik punktid punktipilves P on läbi vaadatud ja on osa mingist klastrist massiivis M [12].

Raadiuseks, mille ümbrusest otsitakse klastrisse kuuluvaid punkte, valitakse 11 sentimeetrit, nii on mõlemad robotid eraldi ühes klastris ning nende küljes ei ole võõrkpunkte. Raadiuse valiku puhul peab olema ettevaatlik juhul kui raadius valitakse liiga väike, võib juhtuda, et klastrid, mis peaksid olema koos, lahutatakse (Näiteks Peoplebot-i puhul väike raadius eraldas ta alam- ja ülemosa kaheks erinevaks klastriks). Liiga suure raadiuse puhul võib lisanduda robotite juurde võõrpunkte, mis võivad edasist tööd segada [12].

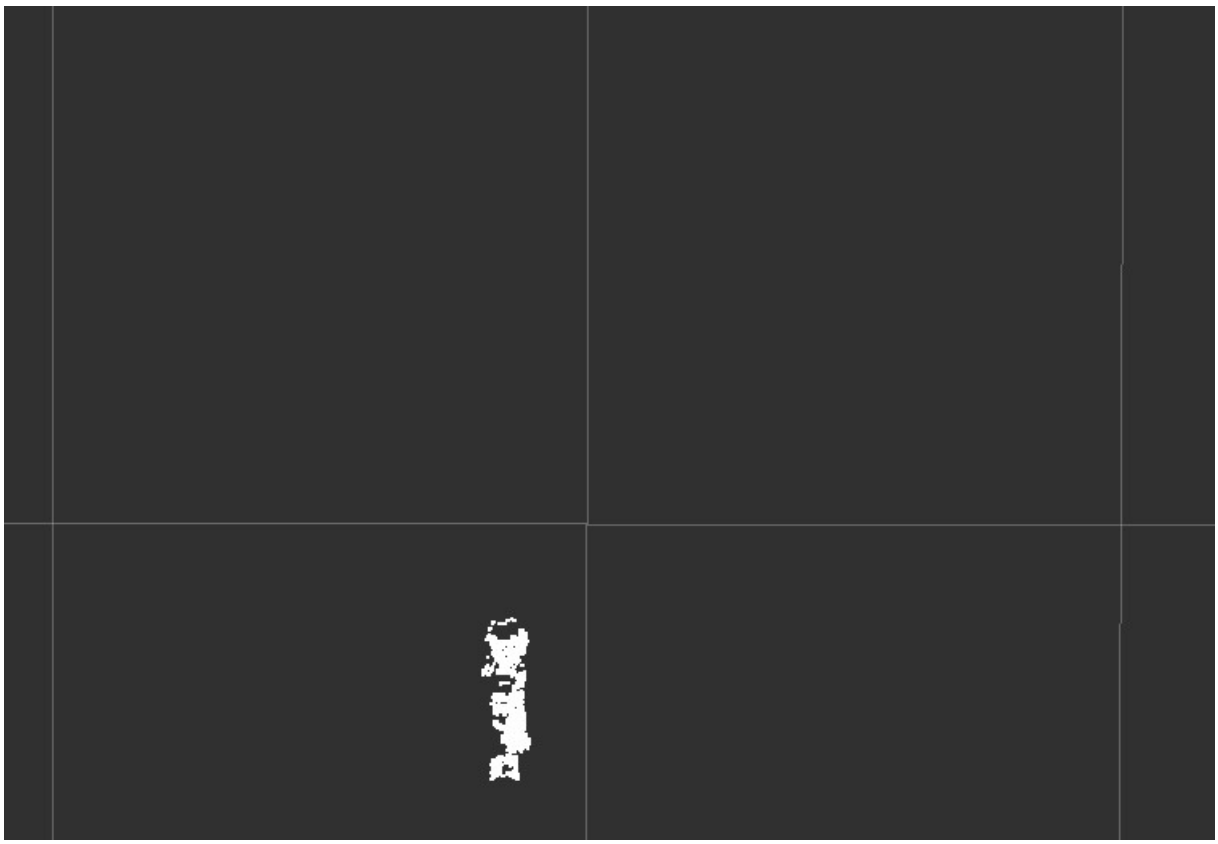
Defineeritakse ära kindlad klastrite suurused, mida tagastatakse, et ära hoida klastrite uurimise, mis on liiga suured või väiksed, et meile sobida. Alampiiriks käesolevas töös

valitakse 500 punkti klastris. Ülempiiriks jääb 10000 punkti klastris.

Klastrite läbivaatamisel pannakse paika mõlemate robotite kindlad klastrite suuruse vahemikud, et tuvastada kumba robotit nähakse. Nao puhul jääb nendeks parameetriteks 500-1100 punkti klastris. Peoplebot-i puhul 4000-9000 punkti klastris. Saadud lõpptulemused kummagi roboti klastrist on näha joonistel 6 ja 7. Leitud punktipilved salvestatakse mällu. Punktipilve parsimine käib käesolevas töös 30 korda sekundis. Juhul kui mõni kord ei suudeta leida punktipilvest robotit, kasutatakse mälus olevat punktipilve robotist. Sellist käitumist saab hetkel kasutada, kuna käesolevad robotid ei liigu kiiresti.



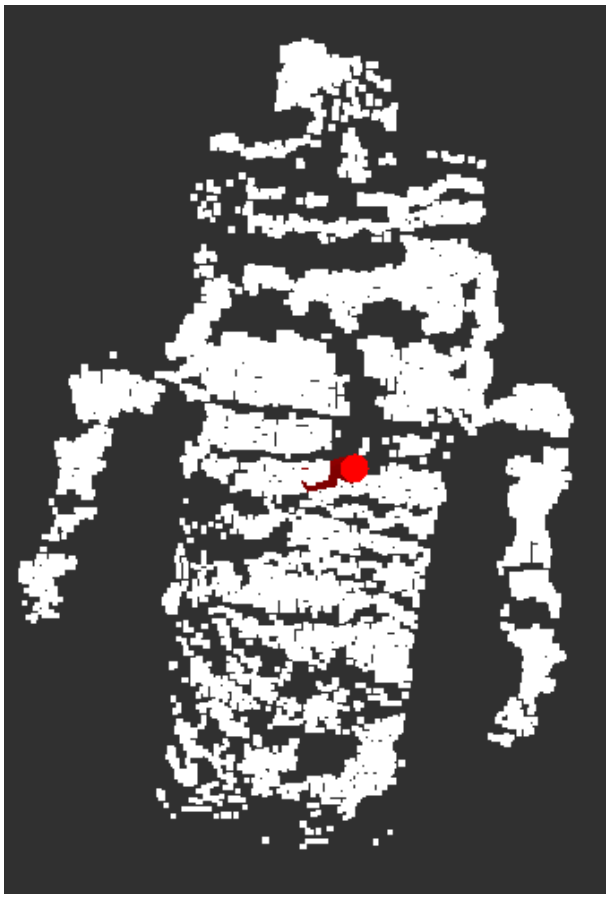
Joonis 6: Punktipilvest leitud Peoplebot



Joonis 7: Punktipilvest leitud Nao

4.3 Robotite keskpunkti leidmine

Massikeskme arvutamiseks kasutatakse *PCL* teegi funktsiooni *compute3DCentroid*. Funktsioon kasutab parameetritena viidet punktipilvele ja viidet *Eigen*-i teegi maatriksile, millel on 4 rida ja 1 tulp. Funktsioon küll tagastab keskpunkti leidmisel kasutatud punktide arvu, aga õige tulemus pannakse parameetris viitena kaasa antud maatriksisse. Maatriksi puhul ei ole oluline, kas see on initsialiseeritud või kõik väärtused on nullid, kuna funktsioon teeb seda oma töö alguses. Algul itereeritakse üle etteantud punktipilve. Iga punkti juures liidatakse punkti X, Y ja Z osa vastavasse maatriksi elementi. Neljas maatriksielement pannakse alati võrduma nulliga. Seejärel luuakse punktipilve punktide arvust skalaar ja jagatakse saadud maatriksiga. Kummagi roboti keskpunkte on võimalik näha märgistatuna joonistelt 8 ja 9.



Joonis 8: Peoplebot-i massikese

Joonis 9: Nao roboti massikese

4.4 Robotite omavahelise kauguse leidmine

Robotite vahe leidmiseks peab teada olema mõlema roboti keskpunkt. Vastasel juhul tagastatakse robotile väärtus 0. Seda väärtust võib julgelt tagastada, kuna ei ole võimalust et üks robot saab olla täpselt samasuguse keskpunktiga nagu teine. Juhul kui mõlemad keskpunktid on olemas, siis vahe leidmine on üpriski lihtne. Nimelt saab kasutada täiesti tavalist maatriksite lahutamist, et jõuda lõpptulemuseni. Kui leitakse Nao-le kaugus Peoplebot-ist, siis viimase keskpunktist lahutatakse Nao keskpunkt. Peoplebot-i kauguse leidmiseks Nao-st kasutatakse vastupidist meetodit. Saadav vastus on 4 rea ja 1 tulbaga maatriks, kus tulba esimene väärtus on X, teine väärtus on Y, kolmas väärtus on Z ning neljas väärtus on alati 0 nagu eelnevas peatükis kirjeldati. Antud maatriks peaks tähistama vektorit, mida läbides vastav robot jõuab teiseni.

5. Pliiatsi kontuuri tuvastamine

Pliiatsi kontuuride tuvastamiseks kasutatakse kaamerast tulevat punktipilve ja tavalist pilti. Antud töös kasutatakse pildi ja punktipilve parsimiseks taaskord kolmandaks osapooleks olevat arvutit, kuna see tagab meile täpsema hetke oleku ja kiirema töötlemisvõime. Arvuti saab oma andmed Asus Xtion Pro Live kaamerast.

Kaamera suudab korraga suurema osa stseenist ära töödelda kui mõlemad robotid. Pliiatsi otsimisel võib tekkida olukordi, kus arvuti ei suuda tuvastada pliiatsit, aga robot suudab. Näiteks, kui pliiats asub roboti või võõrobjekti taga. Kokkuvõttes on arvutil laiem ülevaade stseenist kui robotitel.

5.1 Pildi vähendamine

Pildilt eemaldatakse piksleid vältimaks olukorda, kus tekivad ebavajalikud kontuurid, näiteks sein või robotid. Pildilt on raske universaalselt põrandat välja filtreerida, seetõttu kasutame põranda leidmiseks punktipilve ja peatükis 4.1 seletatud tehnikat. Erinevus seisneb selles, et enne põranda leidmist ei kasuta me *VoxelGrid* funktsioone punktipilve vähendamiseks.

Punktipilvest saadud põrandat kasutatakse punktipilve punktide ja pildi pikslite omavahel vastavusse panemiseks. Punktipilve punktide mitte vähendamine oli vajalik, et jääks võimalikult palju punkte alles, mida pikslitega vastavusse panna, mis omakorda annab täpsema tulemuse. Punktipilve punktide ja pildi pikslite vastavusse panemiseks kasutakse järgnevat valemit: $h_p * w_c + w_p$. Valemil tähistab h_p piksli asukohta pildi y-teljel, w_c on punktipilve laius ja w_p piksli asukoht pildi x-teljel. Juhul kui vastavale pikslile ei ole vastet punkti näol punktipilves, siis piksel pannakse võrduma nulliga ehk piksli värv muudetakse mustaks.

5.2 Pliiatsi leidmine

Pliiatsi tuvastamise esimeseks sammuks on värvilise pildi halliks muundamine ja seejärel pildilt servade leidmine. Selleks kasutatakse *Canny* servade leidmise algoritmi. Saadud servi kasutatakse pildilt kõikide kontuuri joonte leidmiseks. Kontuuride salvestamisel jäetakse alles

väliseimate kontuuride jooned, kuna on vaja teada pliiatsi asukohta ning sisemised kontuurid ei ole abiks. Sedasi saab kokku hoida töötlemise aega ja arvuti mälu. Kontuuride salvestamisel leitakse üles kohad, kus punktid moodustavad vertikaalsed, horisontaalsed või diagonaalsed jooned. Punktidest koosnevatest joontest jäetakse alles otsmised punktid. Teisi punkte edasises töös vaja ei läinud.

Leitud kontuuridele otsitakse lisapunktid, mis moodustaksid kontuuride ümber minimaalsed nelinurgad. Nelinurkade abil määrati kindlaks suurused, mille abil on võimalik liiga suured ja väiksed kontuurid eemaldada. Samuti leitakse kontuuride seast sirged jooned, kasutades *Hough* teisendusalgoritmi. *Hough* teisendused on tehnika, mida rakendatakse piltidelt kindlate kujundite omaduste isoleerimiseks. Klassikalist *Hough* teisendust kasutatakse erinevate kõverate leidmiseks, näiteks jooned, ringid ja ellipsid [19]. Pliiats on oma kujult piklik objekt, seega joonte abil saab kontuuride arvu vähendada.

Kontuure ümbritsevate nelinurkade suuruste ja sirgete joonte abil on võimalik leida üles pliiatsi kontuurid. Seejärel tuvastatakse pliiatsit ümbritseva nelinurga keskpunkt. Punktile pannakse vastavusse eelmainitud valemi abil punkt punktipilves. Punktipilve punkt kujutab kindlat punkti ruumis ja selle abil leitakse pliiatsi asukohta ruumis.

6. Tulemuste analüüs

Käesolevas töös kasutati robotite leidmiseks 7 erinevat eelsalvestatud punktipilve, kus robotid olid erinevates positsioonides ja kaugustes üksteise suhtes. Punktipilvedest tuvastas algoritm Nao ja Peoplebot-i asukoha stseenis. Harva oli juhuseid, kus algoritm ei suutnud robotit leida ja selleks on eelpool välja toodud lahendus, kus varem leitud roboti punktipilv jäetakse meelde. Tekkis olukordi, kus roboti klastris oli juures võõrpunktid, mis ei kuulunud robotite juurde. Sellised võõrväärtused võivad segada robotite punktipilvedega edasi töötamist, näiteks massikeskme leidmist, kuid see nihutab massikeset niivõrd vähe, et ei mõjuta robotite reaalset asukohta.

Saab öelda, et algoritmi tulemustega võib rahul olla. Joonistelt 6 ja 7 on punktipilvedest selgelt aru saada, et tegemist on Nao ja Peoplebot-iga. Kindlasti on olemas võimalusi algoritmi paremaks teha, kuid saja protsendilist täpsust on raske saavutada, kuna alati on olukordi või positsioone, kus algoritm ei ole suuteline leidma vastavat punktipilve kogumit. Seega tuleb arvestada vahepealsete vigadega.

Pildilt ebavajaliku informatsiooni eemaldamisel jäi igast objektist teatud osa äärtest alles. Arvatavaks põhjuseks on sajaprotsendilise täpsuse puudumine punktipilve ja piksli vastavusse panemisel. Alles jäänud objektide ääred aga tekitavad piisavalt väikseid ja kõveraid kontuure, et ei sega sirge pliiatsi otsimist.

Pliiatsi otsimisel suurem osa kordadest see ka leiti kasutades kontuure ja jooni. Juhtus olukordi, kus tagastati vale punkt, kuna võõrkontuur imiteeris täpselt otsitavat pliiatsit (nt põrandal linoleumide ühendusserva kontuuritükike). Taaskord saab öelda, et selliseid juhtumeid oli katsetustes piisavalt vähe, et mitte nendele tähelepanu pöörata käesoleva töö käigus. Käesolevas töös kasutatava meetodiga pliiatsi leidmisel on üks miinus. Nimelt kui viia pliiats seina äärde, kus ei pruugi palju punktipilve punkte olla, siis pliiatsi asukoht võidakse lihtsalt pildilt eemaldada, kuna see arvatakse seina hulka.

7. Kokkuvõte

Käesolev töö koosnes kolmest erinevast põhieesmärgist. Esmalt oli vaja saavutada olukord, kus kaks robotit ja kolmas osapool suudaksid omavahel andmeid vahetada. Seejärel tuli tagastada küsivale robotile, tema vahekaugus teisest robotist. Enne seda oli vaja tuvastada robotid punktipilvest ning leida nende keskpunktid. Lõpuks tuli leida põrandalt pliiats, et Nao teaks, kuhu liikuda.

Töö tulemusena leitakse punktipilvest suure tõenäosusega mõlemad robotit ning nende massikeskmed. Seda ka juhtudel, kui robotid olid erinevates asendites ja kaugustes üksteisest. Sellest võib järeldada, et algoritm on piisavalt universaalne, tagamaks efektiivse töö robotitega, kui need stseenis ringi liiguvad. Edukalt tuvastatakse pliiatsi kontuurid ja leitakse üles pliiatsi keskpunkt punktipilves ning seeläbi asukoht stseenis, mis muudab Nao jaoks pliiatsi leidmise lihtsamaks.

Olemas on kaks võimalikku viisi, kuidas tööga edasi minna. Esimese sammuna oleks võimalik objektide tuvastamise algoritme paremaks teha, et vältimaks olukordi, kus ei suudeta objekti tuvastada või otsitava objekti asemel leitakse juhuslik punktide kogum. Teisena oleks võimalik arendusest eemaldada kolmanda osapoolena olev arvuti ning kogu töö kohandada Peoplebot-i robotile. Seda muidugi juhul, kui Peoplebot suudab töö teha ära peaaegu sama kiiresti.

1 Hinnang eesmärgi saavutamisele

Hetkeses lahenduses teeb kogu jõudlusrohke töö ära kolmas osapool, milleks on arvuti, mis on võimsam kui robotid. Informatsioon stseenist tuleb kaamerast, mis on toanurgas, nii on võimalik saada täielik ülevaade stseenist. Kolmanda osapoole poolt töödeldud informatsiooni saavad mõlemad robotid igal ajahetkel küsida. Kokkuvõtvalt saab öelda, et töö alguses välja toodud eesmärk ja probleemid said lahendatud.

2 Põhitulemuste loetelu

Robotid pandi omavahel andmeid vahetama läbi kolmanda osapoole, kasutades teeki *Rocon*. Mõlemad robotid saavad töötavalt teenuselt igal ajahetkel pärida informatsiooni teise roboti kohta.

Punktipilvest mõlema roboti tuvastamiseks kasutatakse teegi *PCL* funktsioone. Esmalt vähendatakse punkitpilves olevaid punkte, seejärel eemaldatakse põrand ning lõpuks jagatakse punktipilv klastritesse, mille abil on võimalik tuvastada Nao ja Peoplebot. Massikese leitakse kasutades *PCL* teegi funktsiooni, mis annab roboti keskpunktis oleva vektori. Robotite vahelise kauguse leidmiseks lahutatakse ühe roboti massikeskme vektor teise roboti vektorist.

Punktipilvest pliiatsi leidmiseks pannakse vastavusse kaamerast saadud pilt ja punktipilv. Punktipilvest leitakse põrand ja selle abil eemaldatakse pildist ebavajalik informatsioon. Seejärel tuvastatakse kontuuride ja joonte abil pildilt pliiats ja pannakse vastavusse pildil oleva pliiatsi keskpunkti piksel punktipilve punktiga, mille abil leitakse pliiatsi asukoht stseenis.

3 Hinnang eesmärkide saavutamisele

Eesmärgid suudeti saavutada kasutades juba varasemalt vabavarana olemasolevaid vahendeid. Lahendada tuli spetsiifilisi olukordi ja proovida erinevaid parameetreid kasutuses olevatele funktsioonidele, jõudmaks olukorrani, mis tagab parima tulemuse. Eesmärkide saavutamisel lähtuti järgmistest põhimõtetest - üritati andmete mahtu vähendada ja saadud andmete hulgast vajaliku informatsiooni leida.

Käesolevas lahenduses prooviti saada parim tulemus probleeme võimalikult lihtsalt lahendades. Kindlasti on probleemile töökindlamaid ja täpsemaid lahendusi, aga sellega kaasneb järsk keerukuse tõus.

Summary

Current work contains three different main goals. Firstly, a solution for communication between robots was required for robot collaboration. This is important because robot needs to know the distance between, it and the other robot. Before this could be done, robots should be identified from a pointcloud and their centroid should be calculated. Finally, the grasp target object (i.e. pencil) is detected from the image.

As a result both robots are identified from the pointcloud and their centroids calculated with high probability. Even when the robots were in different poses and distances from eachother. From that it can be concluded that the algorithm is robust enough to ensure efective work with robots that move around the scene. Pencils contures were identified succesfully. Pencils centroid is also located from the pointcloud and thereby from the scene, which makes finding the pencil easier for Nao.

Kasutatud kirjandus

[1] ROS-i kodulehekülg *[WWW]* http://www.ros.org/about-ros/ (15.04.2015)

[2] ROS-i ametlik wiki lehekülg *[WWW]* http://wiki.ros.org/ROS/Introduction (15.04.2015)

[3] ROS-i ametlik wiki lehekülg *[WWW]* http://wiki.ros.org/Parameter%20Server (15.04.2015)

[4] PCL teegi kodulehekülg *[WWW]* http://pointclouds.org/about/ (15.04.2015)

[5] OpenCV teegi kodulehekülg *[WWW]* http://opencv.org/about.html (15.04.2015)

[6] Rocon-i leht ROS-i ametlikus wikis *[WWW]* http://wiki.ros.org/rocon (15.04.2015)

[7] roscore lehekülg ROS-i ametlikus wikis *[WWW]* http://wiki.ros.org/roscore (18.04.2015)

[8] rocon hub-i lehekülg ROS-i ametlikus wikis *[WWW]* http://wiki.ros.org/roscore (18.04.2015)

[9] Kella Price. QR codes for trainers – *Infoline,* 2013, 1301, 1 *[Online]* https://books.google.ee/books?id=zbZUAwAAQBAJ&pg=PA4&lpg=PA4&dq=qr+code+explanation+text&source=bl&ots=VLgS4wBMsN&sig=NifQxbIy58D1CXxM9E7_tigL-3o&hl=et&sa=X&ei=u2M5VfSJBoKtsgH5-IGIDg&ved=0CE4Q6AEwBjgK#v=onepage&q&f=false (24.04.2015)

[10] Arie Kaufman, Daniel Cohen, Roni Yagel. Volume graphics – *IEEE Computer,* 1993, 26, 51-64 [Online] http://labs.cs.sunysb.edu/labs/projects/volume/Papers/Voxel/ (26.04.2015)

[11] PCL teegi dokumentatsioon *[WWW]* http://pointclouds.org/documentation/tutorials/voxel_grid.php (26.04.2015)

[12] PCL teegi dokumentatsioon *[WWW]* http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php (27.04.2015)

[13] ROS-i ametlik wiki *[WWW]* http://wiki.ros.org/Topics (03.05.2015)

[14] ROS-i ametlik wiki *[WWW]* http://wiki.ros.org/Messages (03.05.2015)

[15] ROS-i ametlik wiki *[WWW]* http://wiki.ros.org/Services (03.05.2015)

[16] Radek Beneš, Martin Hasmanda, Kamil Riha. Non-linear RANSAC method and its utilization – *Elektrorevue*, 2011, 2, 7 *[Online]* http://www.elektrorevue.cz/en/download/non-linear-ransac-method-and-its-utilization/ (03.05.2015)

[17] Andrew W. Moore. 1991. An intoductory tutorial on kdtrees. *[Online]* http://www.autonlab.org/autonweb/14665/version/2/part/5/data/moore-tutorial.pdf (03.05.2015)

[18] Chanop Silpa-Anan, Richard Hartley. Optimised KD-trees for fast image descriptor matching – S*eeing Machines, Australian National University, NICTA [Online]* http://users.cecs.anu.edu.au/~hartley/Papers/PDF/SilpaAnan:CVPR08.pdf (03.05.2015)

[19] R. Fisher, S. Perkins, A. Walker, E. Wolfart. Hough Transform *[WWW]* http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm (17.05.2015)

Lisa 1 – *Rocon*-i *hub*-i seadistus

```
<launch>
  <arg name="hub_name" default="Rocon Hub" />
  <arg name="hub_port" default="6380" />
  <arg name="zeroconf" default="true" />
  <arg name="gateway_unavailable_timeout" default="30"/>
  <arg name="gateway_dead_timeout" default="7200"/>
  <arg name="hub_watcher_thread_rate" default="0.2"/>
  <arg name="external_shutdown" default="false" />

  <node pkg="rocon_hub" type="hub.py" name="hub">
    <rosparam command="load" file="$(find rocon_hub)/param/default.yaml" />
    <param name="name" value="$(arg hub_name)" />
    <param name="port" value="$(arg hub_port)" />
    <param name="zeroconf" value="$(arg zeroconf)" />
    <param name="external_shutdown" value="$(arg external_shutdown)" />
    <param name="gateway_unavailable_timeout" value="$(arg
gateway_unavailable_timeout)"/>
    <param name="gateway_dead_timeout" value="$(arg gateway_dead_timeout)"/>
    <param name="watcher_thread_rate" value="$(arg hub_watcher_thread_rate)"/>
  </node>
</launch>
```

Lisa 2 – Koodi *Git*-i repositoorium

https://bitbucket.org/elgarlepp/multirobot-collaboration/