TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut

Jürgen Ukkivi

132338IAPM

VISUAALSEL TAGASISIDEL PÕHINEV ROBOTMANIPULAATORI JUHTIMINE

Magistritöö

Juhendaja: Gert Kanter

Tallinn 2015

Deklaratsioon

Käesolevaga kinnitan, et esitatud töö

.....

.....

on minu isikliku töö tulemus. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kusagil mujal.

Kuupäev ……..

Üliõpilane ……

Nimi Allkiri

Jürgen Ukkivi

VISUAALSEL TAGASISIDEL PÕHINEV ROBOTMANIPULAATORI JUHTIMINE

Magistritöö

Annotatsioon

Eesmärk: Arendada tarkvara robotile, mis suudaks stseenist tuvastada objekti asukoha, juhtida manipulaator objekti juurde ning manipulaatoril oleva kaamera visuaalse info põhjal täpsustada objekti asukoht ning see haarata.

Lahendatavad probleemid: Kuidas luua tarkvara Roboti Operatsioonisüsteemis. Kuidas kolmemõõtmelisest ruumist leida üles vajaminev objekt. Kuidas manipulaatori kaamera kahemõõtmelisest pildist leida objekt. Kuidas saadud infot kasutada ning manipulaatoriga haarata objekt.

Tulemused: Loodi tarkvara, kasutades mitmeid erinevaid visuaalse tuvastamise algoritme, mis suudavad tuvastada objekti, seda jälgida ning selle info põhjal suudab robot objekti haarata.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 47 leheküljel, 8 peatükki, 39 joonist.

Jürgen Ukkivi

ROBOT MANIPULATOR CONTROL BASED ON VISUAL FEEDBACK

Master's thesis

Abstract

Purpose: To create software for a robot which would make it able to identify an object in the scene maneuver its manipulator's end effector to the object using visual feedback and grasp the object.

Problems: How to create software using Robot Operating System. How to find the needed object in three-dimensional space. How to find the object in a two-dimensional image. How to use the obtained information to grab the object.

Results: The software was created using different visual identification algorithms to detect the object and guide the manipulator based on visual feedback and based on that information the robot is able to grasp it.

Thesis is written in Estonian and there are 47 pages, 8 chapters, 39 figures.

Jooniste nimekiri

Sisukord

1. Sissejuhatus

Tehnika arenedes ja võimaluste laienemisel proovitakse meie elu muudkui paremaks ja efektiivsemaks muuta. Meie kõigi elu oleks lihtsam, kui meist endast oleks kaks või kolm või lausa mitu koopiat, kes oleks pühendunud samale eesmärgile. Kahjuks ei ole see veel võimalik, vaid on üks ilus unistus. Kui meist oleks üks koopia, siis suudaksime olla kuni kaks korda efektiivsemad ning see oleks väga suur võit. Alustada võiks isegi murdosa protsendist, mis muudab meie elu tõhusamaks. Kui me ei peaks kodus otsima autovõtmeid või rahakotti, vaid seda suudaks teha keegi teine. Kui me ei peaks töö ajal minema suupiste järgi või kui me ei peaks kontoris otsima vajalikke pabereid. Kui muutume eakaks või oleme võimetud neid ise tegema.

Kõik need pisitööd saaks teha robot ning kui me ei pea sellega tegelema, saame samal ajal tegeleda millegi olulisemaga. Roboti poolt üles korjatud objekt võib olla väike samm, kuid see võib olla alus millelegi palju suuremale.

1.1 Taust ja probleem

Lõputöö idee pakuti välja ülikooli poolt. Lõputöö on osa suuremast projektist. Antud lõputöö käsitleb ainult osa, kuidas robot identifitseerib objekti, töötleb visuaalseid andmeid ning selle põhjal haarab objekti. Projekti lõpptulemusena peab robot olema võimeline erinevates keskkondades tegevust kordama ning kasutama ka platvormi mobiilsust.

Robot paikneb Tallinna Tehnikaülikoolis Arvuteaduse instituudis, kus toimus ka lõputöö testimine.

1.2 Ülesande püstitus

Lõputöö eesmärgiks on luua tarkvara robotile, et see suudaks leida kolmemõõtmelisest ruumist vajaliku objekti. Robot peab manipulaatori peal oleva kaamera kahemõõtmelise pildi põhjal tuvastama objekti, et saavutada parem täpsus objekti haaramiseks. Lõpuks peab robot saadud andmeid kasutades haarama objekti.

1.3 Metoodika

Roboti tarkvara luuakse kasutades Roboti Operatsioonisüsteemi. Tarkvara on kirjutatud C++

keeles. Visuaalse info töötlemiseks kasutatakse mitmeid algoritme OpenCV ja Point Cloud Library teekidest.

1.4 Ülevaade tööst

Lõputöö jaguneb kolmeks põhiliseks osaks: esimeses osas kirjeldatakse, kuidas leiti kolmemõõtmelisest maailmast vajalik objekt ning selle koordinaadid.

Teises osas kirjeldatakse, kuidas tuvastati manipulaatori kaamera andmevoo sisendist objekt.

Kolmandas osas on välja toodud, kuidas kahes esimeses punktis saadud infot kasutatakse, et manipulaatoriga objet haarata.

Samuti on välja toodud kirjeldus põhilise töövahendi - Roboti Operatsioonisüsteemi kohta ning kasutatud riistvara ja algoritmide kohta.

2. Kasutatud tarkvara

2.1. ROS - Roboti Operatsioonisüsteem

Järgnev info on tõlgitud inglise keelest eesti keelde[1].

2.1.1. ROS-i eesmärgid

ROS ei ole traditsionaalses mõttes operatsioonisüsteem, mis haldab ja plaanib protsesse, pigem pakub see struktureeritud kommunikatsioonikihti olemasoleva operatsioonisüsteemile. ROS-i põhilised eesmärgid:

- Partnervõrk
- Mitmekeelne
- Tööriistapõhine
- Õhuke
- Tasuta ja avatud lähtekoodiga

A. *Partnervõrk*

Süsteem, mis on ROS-i kasutades ehitatud, koosneb mitmetest erinevatest protsessidest, potentsiaalselt mitme erineva peremehe peal, olles käivituse ajal ühendatud partnervõrgustikus. Partnervõrk vajab mehhanismi, mis võimaldab protsessidel üksteist üles leida. ROS-is nimetatakse seda *nimetamisteenuseks* või *ülemaks*.

B. *Mitmekeelne*

ROS on disainitud olema programmeerimiskeelte suhtes neutraalne. ROS toetab nelja erinevat keelt: C++, Python, Octave ja LISP ning mitu keeletuge on arendamisel.

ROS-i spetsifikatsioon on sõnumite tasemel, mitte kaugemal. Partnervõrgustiku suhtlus toimub formaadis XML-RPC, millel eksisteerivad implementatsioonid enamustes suurtes keeltes. Et toetada keeltevahelist arendamist, kasutab ROS lihtsat liidesemäärangu keelt (IDL), et kirjeldada moodulite vahel saadetud sõnumeid. Liidesemäärangu keel kasutab lühikesi tekstifaile, et kirjeldada iga sõnumi välju. Näide sõnumist:

```
Header header
Point32[] pts
ChannelFloat32[] chan
```

Koodigeneraatorid genereerivad iga toetatud keele jaoks vastava implementatsiooni, mis säästab

märgatavalt programmeerija aega ja vigu: eelmisest kolmest reast genereeritakse näiteks 137 rida C++ koodi, 96 rida Python, 81 rida LISP ja 99 rida Octave keeles. ROS-is on üle neljasaja sõnumitüübi, mis transpordivad andmeid sensoritest kuni objekti tuvastamisest kuni struktuurskeemideni. Lõpptulemusena on keeleneutraalne sõnumite töötlusskeem, kus on võimalik erinevaid keeli omavahel segada.

C. *Tööriistapõhine*

Et hallata ROS-i keerukust, on valitud mikrokerneli disain. Selles kasutatakse suurt arvu tööriistu, et kompileerida ja käivitada erinevaid ROS komponente selle asemel, et luua monoliitne arendus- ja käivituskeskkond.

Need tööriistad täidavad mitmeid eesmärke, näiteks navigeerivad lähtekoodi, sätivad konfiguratsiooni parameetreid, visualiseerivad partnervõrgu ühendusi, möödavad ribalaiuse kasutust, esitavad graafiliselt sõnumite andmeid jne.

D. *Õhuke*

Suurem osa eksisteerivatest robootika tarkvara projektidest sisaldavad draivereid või algoritme, mis küll võiksid olla taaskasutatavad ka väljaspool projekte. Kahjuks on mitmetel põhjustel see kood muutunud niivõrd seotuks vahevaraga, et on raske taaskasutada selle funktsionaalsust väljaspool projekti.

Et võidelda selle suunitlusega, julgustatakse ROS-i kasutajatel kõik draiveritega seotud tegevused teha eraldi teekidesse, millel ei oleks sõltuvusi ROS-iga. ROS-is kasutuses olev Cmake teeb lihtsaks „õhukese“ ideoloogia kasutamise. Viies peaaegu kogu keerukuse teekidesse ja luues ainult väikesed programmid, mis kasutavad teekide funktsionaalsust, muudab koodi kergemini taaskasutatavaks.

ROS taaskasutab koodi mitmetelt teistelt avatud lähtekoodiga projektidelt, näiteks draiverid, navigatsioonisüsteem ja simulaatorid Player projektilt, visualiseerimise algoritmid projektilt OpenCV ja planeerimisalgoritmid OpenRAVE-ilt. Igal juhul kasutatakse ROS-i ainult selleks, et avaldada erinevad konfiguratsiooni sätted ja suunata andmeid vastava tarkvara poole ning sellest välja suunata, lisades võimalikult vähe muutuseid. Et kasu saada pidevast kogukonna poolt tehtavatest parandustest, uuendatakse ROS-i automaatselt välistest andmehoidlatest.

E. *Tasuta ja avatud lähtekoodiga*

Kogu ROS-i lähtekood on avalikult kättesaadav. ROS-i loojad leiavad, et on äärmiselt oluline siluda vigu kõigil tarkvaratasemetel. See on vastab tõele, kui riistvara ja tarkvara disainitakse ja

silutakse paralleelselt. ROS on BSD litsentsi all, mis lubab nii kommertslike, kui ka mittekommertslike projektide arendust.

2.1.2. Nomenklatuur

ROS-i fundamentaalsed mõisted on sõlm, sõnum, teema ja teenus.

Sõlmed on protsessid, mis tegelevad arvutamisega. ROS on disainitud olema modulaarne: süsteem koosneb tavaliselt mitmest sõlmest. Antud kontekstis on „sõlm“ võrdeline „tarkvara mooduliga“. „Sõlme“ termin tuleneb sellest, et kui visualiseerida ROS-i süsteeme, siis kui mitu sõlme jookseb, ning partnervõrku näidata graafina, siis protsessid on graafi sõlmed ning partnervõrgu lülid on kaared.

Sõlmed suhtlevad omavahel edastades **sõnumeid**. Sõnum on andmestruktuur. Standardsed primitiivi tüübid on toetatud, kui ka massivid ja konstandid.

Sõlm saadab sõnumi, avaldades kindlasse **teemasse(topic)**, mis on string ja käitub nagu struktuurskeem. Sõlm, mis on kindlatest andmetest huvitatud, tellib vastavalt teemalt. Ühel teemal võib olla mitu avaldajat ja tellijat ning üks sõlm võib avaldada ja tellida kindlalt teemat. Üldiselt ei ole avaldajad ja tellijad teadlikud üksteise olemasolust.

Kuigi teemapõhine avalda-telli mudel on paindlik kommunikatsioonimuster, siis ei sobi see hästi sünkroonseteks ülekanneteks, mis võib lihtsustada mõne sõlme disaini. ROS-is on selle jaoks **teenus**, mis on defineeritud stringi nimega ja sõnumipaariga: üks päringu ja teine vastuse jaoks. Erinevalt teemadest, saab ainult üks sõlm avaldada kindla nimega teenust.

2.1.3. Kasutusjuhud

A. Sõlme silumine

Robootika valdkonnas on uurimistöö tavaliselt süsteemi spetsiifilises osas, näiteks sõlm, mis kas planeerib, kaalub või tuvastab midagi. Et saada robotisüsteem valmis katsetusteks, on vaja palju suuremat tarkvara ökosüsteemi. Näiteks, et teha visualiseerimise baasil objekti haaramiseks katseid, on vaja valmis draivereid kaamera ja manipulaatori jaoks ning mitmeid vahendavaid töötlussõlmi (objekti ära tundmiseks, poosi märkamiseks, trajektoori planeerimiseks). See muudab roboti uurimistöö integreerimise märgatavalt raskemaks.

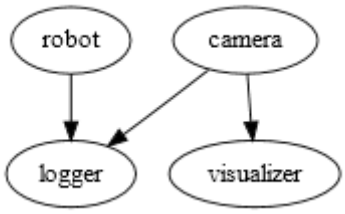
ROS-is on silumine tehtud võimalikult kergeks, kuna selle modulaarne struktuur laseb kasutada eelnevalt eksisteerivate ja hästi silutud sõlmede kõrval ka arendatavaid sõlmi. Kuna käivitamisel saavad sõlmed üksteisega ühenduda, siis saab sõlmegraafi dünaamiliselt modifitseerida. Eelmises visuaalse info põhjal manipulaatori kasutamise näites on vaja graafi mitme sõlmega, et tekiks infrastruktuur. Selle "infrastruktuuri" graafi võib panna käima ja jätta käima terveks

eksperimendiks. Ainult sõlmi, millel on vaja muuta lähtekoodi, tuleb taaskäivitada, mille ajal ROS muudab vaikselt sõlmegraafi. Tulemusena suurendab see produktiivsust, eriti kui robotisüsteemid muutuvad suureks ja keerukaks.

B. Logimine ja taasesitus

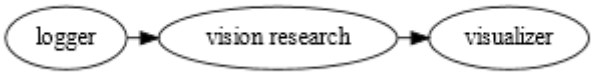
Uurimistöö robootika visualiseerimise valdkonnas on sageli tehtud sensori andmetega, mis on eelnevalt logitud, et oleks võimalik võrrelda erinevaid algoritme ning lihtsustada eksperimente. ROS toetab seda, pakkudes üldist logimist ning taasesituse funktsiooni. Kõikide sõnumite andmevoogu on võimalik salvestada kettale ning hiljem taasesitada. Seda on võimalik teha käsurealt ning ei vaja mingit muudatust lähtekoodi.

Näiteks on võimalik käima panna järgnev graaf, et koguda andmeid visuaal-odomeetria uurimistöö jaoks:



Joonis 1: Logimise graaf

Välja tulevaid sõnumeid on võimalik taasesitada teise graafi, mis sisaldab arenduses olevat sõlme:

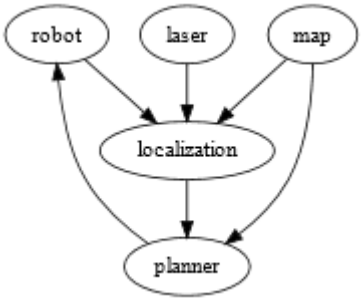


Joonis 2: Logimise taasesitus

Et kergendada logimist ja seiramist süsteemides, mis on jaotatud mitme hosti peal, on *rosconsole* teek ehitatud üles Apache projekti *log4cxx* süsteemil, et pakkuda mugavat ja elegantset logimisliidest, lubades printf stiilis diagnostilisi sõnumeid suunata läbi võrgustiku ühte ainsasse voogu nimega *rosout*.

C. Valmispakitud alamsüsteemid

Mõned valdkonnad robootikas, näiteks roboti navigeerimine siseruumis, on jõudnud sellisesse punkti, kus eelnevalt valmis tehtud algoritmid võivad töötada piisavalt hästi. ROS kasutab algoritme, mis on implementeeritud Player projektis, et pakkuda navigatsioonisüsteemi, luues sellise graafi:



Joonis 3: Navigatsiooni graaf

Iga sõlme võib käivitada käsurealt, kuigi korduvalt kirjutada käske, et protsesse käima panna, võib muutuda tüütuks. Et oleks olemas "pakitud" funktsionaalsus, nagu navigatsioonisüsteem, siis ROS pakub tööriista nimega *roslaunch*, mis loeb graafi XML kirjeldust ning käivitab selle. See funktsionaalsus võib märgatavalt aidata integreeritud robootika uurimistöö jagamist ning taaskasutamist, kuna suurte süsteemide käivitamist võib kergelt korrata.

D. Meeskonnapõhine arendus

Et toetada meeskonnaga arendust, on ROS-i tarkvarasüsteem organiseeritud pakettideks. ROS-i pakett on lihtsalt kataloog, mis sisaldab XML faili, mis kirjeldab paketti ja sõltuvusi.

ROS pakettide kollektsioon on puu, milles lehtedeks on ROS-i paketid: ROS paketi andmehoidlad võivad seega sisaldada komplekseid alamkataloogide skeeme. Näiteks ühes ROS andmehoidlas on juurkataloogid "nav", "vision" ja "motion_planning", igaüks neist sisaldab mitmeid pakette alamkataloogidena.

ROS-i pakettide avatud loomus võimaldab suurt variatsiooni nende struktuuris ja eesmärgis: mõned ROS paketid kasutavad eksisteerivat tarkvara, nagu Player ja OpenCV, automatiseerides neid ja eksportides nende funktsionaalsust. Mõned paketid kompileerivad sõlmi ROS graafides kasutamiseks, teised paketid pakuvad teeke ja eraldi programme ning mõned koosnevad skriptidest, et automatiseerida demonstratsiooni ning teste. Pakettide süsteem on mõeldud selleks, et saaks ROS-i põhist tarkvara jaotada väikesteks tükkideks, mida saab arendada ja hooldada oma graafiku põhjal erinevad arendajate meeskonnad.

E. Visualiseerimine ja seire

Disainides ja siludes robootika tarkvara on sageli vajalik jälgida mingit olekut samal ajal, kui süsteem töötab. Kuigi *printf* on tuttav tehnika programmide silumiseks üksikul masinal, siis see tehnika või muutuda keeruliseks suuremates jaotatud süsteemides.

Selle asemel võib ROS-is kasutada ühendusgraafi dünaamilist loomust, et kuulata ükskõik millist sõnumivoogu. Avaldajate ja tellijate kasutamine võimaldab kasutada üldotstarbelisi visualiseerijaid. Näiteks laseri skaneerimise või piltide jaoks on võimalik kirjutada lihtsaid programme, mis tellivad kindlalt teemalt ning visualiseerivad saadud infot. ROS-is leidub ka võimas visualiseerimiseks mõeldud programm rviz. Rvizis on võimalik visualiseerida mitmeid erinevaid andmetüüpe, nagu pildid, punktipilved, geomeetrilised primitiivid(nagu objekti tuvastuse tulemused) ning kuvada roboti poose ja trajektoore. Võimalik on kirjutada lisaprogramme, et näidata rohkem andmetüüpe.

F. Funktsionaalsuse komponeerimine

ROS-is koosneb suurem tarkvara mitmest erinevast sõlmest, nagu eelnevas navigatsiooni näites. Nagu eelnevalt on kirjeldatud, siis võib ROS käivitada kõik sõlmed ühe käsuga, kui sõlmed on kirjas XML failis. Kuigi mõnikord on vajalik klastri mitu käivitamist. Näiteks multiroboti katsetel on vaja käivitada navigatsioon igas süsteemis olevad robotis ning robotid, millel on humanoidi torso, peavad käivitama kaks käe kontollerit. ROS toetab seda, andes kattuvatele sõlmedele oma nimeruumi, kindlustades, et ei tekiks nimede vastuolu. Selle saavutamiseks lisatakse string kõikidele sõlmedele, teemadele ja teenuse nimedele, ilma, et oleks vaja sõlmede koodi muuta.

G. Teisendused

Robootika süsteemid peavad sageli jälgime ruumilisi suhteid mitmetel põhjustel: mobiilse roboti ja mingi fikseeritud asukoha viite lokaliseerimiseks, sensori ja manipulaatori suhe.

Et lihtsustada ja ühitada ruumilise raame, siis on ROS-is kirjutatud teisenduste süsteem nimega tf. Süsteem tf konstrueerib dünaamilise teisenduste puu, mis ühendab kõik raamid antud süsteemis. Kui info tuleb sisse erinevatelt roboti alasüsteemidelt, siis tf süsteem võib luua teisenduste voo sõlmede puu vahel, konstrueerides tee vajalikust sõlmest teiseni ning tehes vastavad arvutused.

Näiteks võib tf süsteemi kasutada, et luua punktpilved paigalolevas raamis, mis on saadud liikuva roboti kallutatud laserite skaneeringust.

2.2 OpenCV

OpenCV on avatud lähtekoodiga C/C++ teek, mis on mõeldud tehisnägemisega seotud ülesannete lahendamiseks. OpenCV on kättesaadav olnud aastast 2000 ning on BSD litsentsi all. OpenCV on mõeldud programmeerijatele ja kasutajatele, kes lahendavad probleeme robootika valdkonnas. See sisaldab erinevaid madala tasemega algoritme nagu näotuvastus, jalakäija tuvastamine, tunnusjoonte sobitamine ja jälgimine.

OpenCV on projektis kasutusel, sest pakub just projektis vajalikku kahemõõtmelise pildi töötlust. Manipulaatori kahemõõtmeliselt pildilt oli vaja tuvastada objekt, saada selle asukoht ning veenduda, et pildi liikumisel suudetakse jätkuvalt jälgida objekti. Kasutusel on kontuuride ja nurkade avastamine. Kasutamisel sai ka määravaks see, et OpenCV on avatud lähtekoodiga ja tasuta ning sellel on loodud tugi ROS-is kasutamiseks. Näiteks on loodud vahendusteek, millega saab OpenCV pildiformaat muuta ROS-is kasutatavaks formaadiks ning vastupidi.

2.3 PCL – Point Cloud Library

PCL on kollektsioon tööriistadest ja algoritmidest, mis on mõeldud kolmemõõtmeliste andmete töötlemiseks. Teek on avatud lähtekoodiga ja on BSD litsentsi all ja väljastati aastal 2010. PCL-i tuum on struktureeritud väiksemateks teekideks, mis pakuvad algoritme ja tööriistu kindlates kolmemõõtmeliste andme- töötluse valdondades, mida on võimalik omavahel kombineerida, et lahendada tavalisi probleeme nagu objekti tuvastus, andmepilvede segmenteerimine ning pinna konstrueerimine.[2]

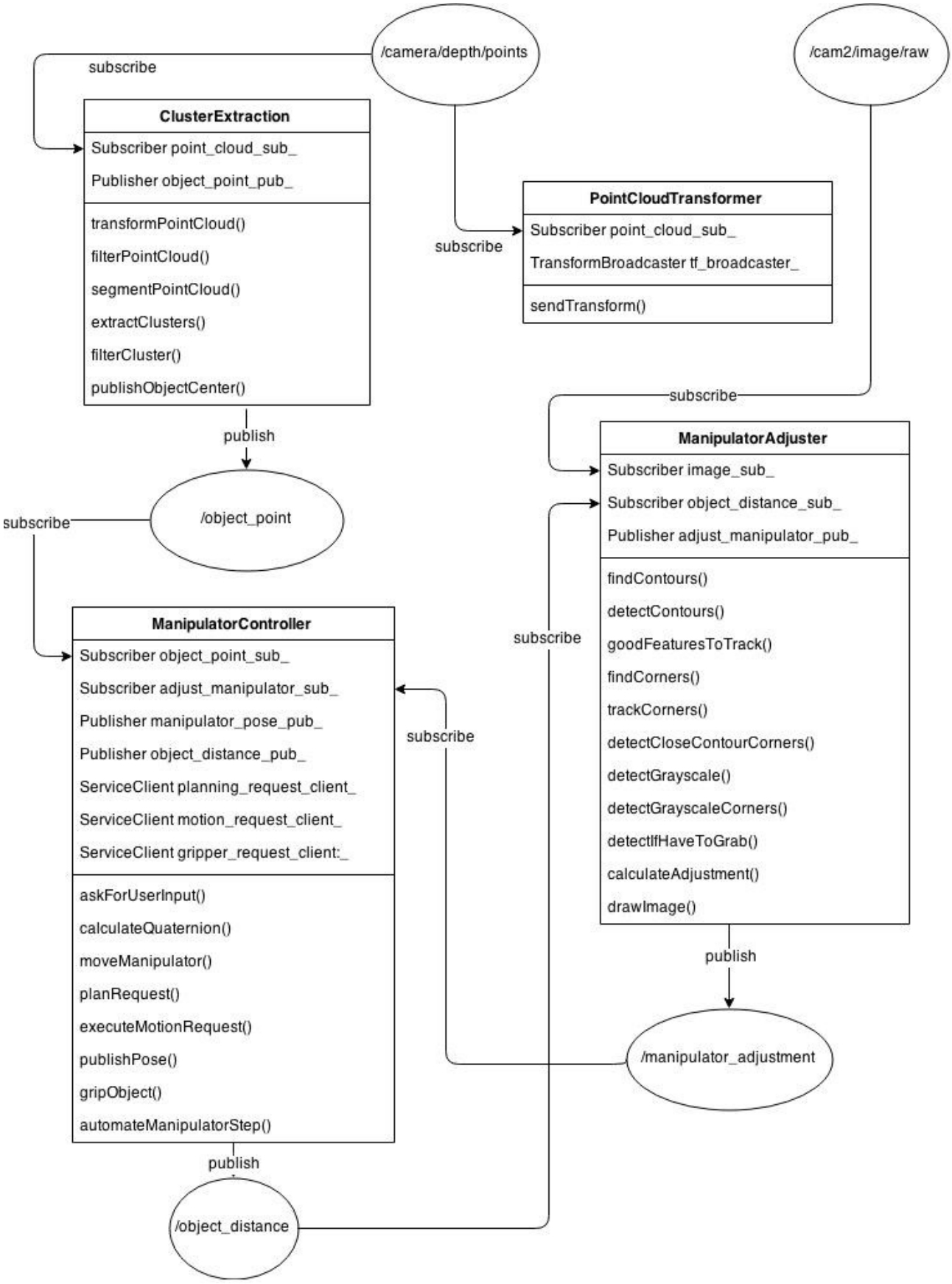
PCL-i kasutatakse antud lõputöös, sest PCL on tasuta ja avatud lähtekoodiga. Samuti on ROS-is loodud tugi PCL kasutamiseks. Näiteks on olemas teisendused ROS sõnumitest PCL sõnumiteks ning vastupidi.

2.4 ViSP - Visual Servoing Platform

ViSP on avatud lähtekoodiga teek, mille tegevusvaldkond on keskendatud visuaalsele tuvastamisele, jälgimisele ning servomootorite liigutamisele selle põhjal. ViSP on kirjutatud C++ keeles ning on GNU GPLv2 litsentsi all. ViSP-i on arendatud aastast 1999.

ViSP-i on kasutatud antud projektis, kuna pakub võimalusi, mida OpenCV ei pakkunud, näiteks laigu tuvastamine. Samuti on ViSP avatud lähtekoodiga ning on loodud tugi ROS'is kasutamiseks.

2.5 Programmi arhitektuur



Joonis 4: Programmi arhitektuur

Pildil olevad mullid on teemad, mida kasutatakse ROS-is et sõlmed saaksid omavahel suhelda. Kõik klassi muutujad ning meetodid ei ole kirjas, et kogu arhitektuur pildile mahuks. Teemasse /camera/depth/points saadab andmeid 3D kaamera ning teemasse /cam2/image/raw saadab andmeid manipulaatoril olev kaamera. Käsud robotile annavad ServiceClient objektid.

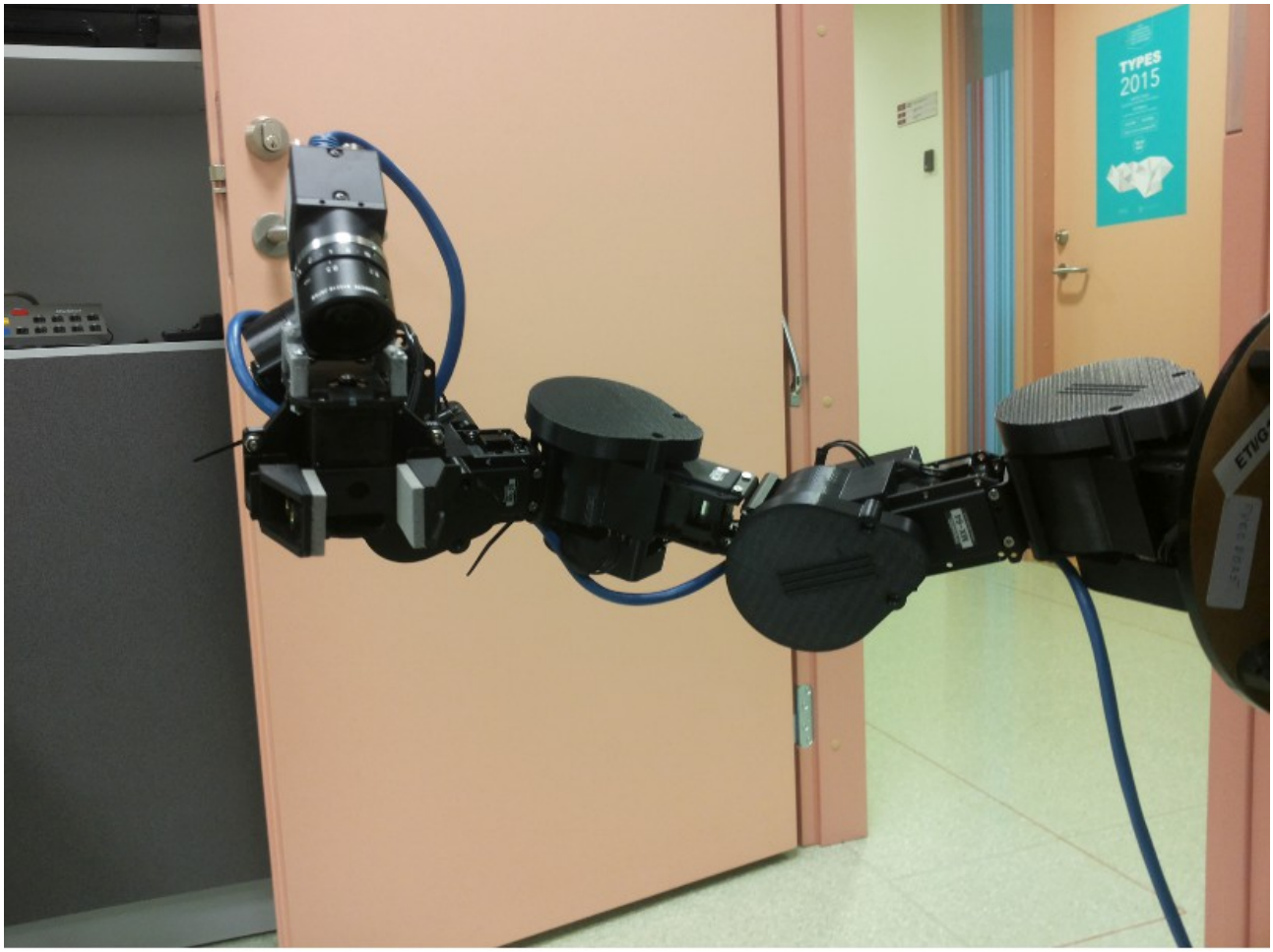
3. Kasutatud riistvara

3.1 PeopleBot

Joonis 5: PeopleBot

Lõputöös on kasutatud mitmeid erinevaid riistvara komponente ning nad kõik on ühendatud PeopleBoti külge. PeopleBot robot on põhiliselt mõeldud uurimistöö ja arenduse jaoks. Roboti tarkvara võimaldab kasutajal käivitada C/C++ programme, tänu millele on robot väga mitmekülgne. Lisaks järgnevalt mainitud komponentidele on robotil olemas ka laserkaugusmõõtja ja stereokaamera.

3.2 Cyton Gamma 1500



Joonis 6: Cyton Gamma 1500 robotkäsi

Cyton Gamma 1500 on robotkäsi, mis töötab liikumise poolest nagu inimese käsi. Robotkäe seitse erinevat servomootorit võimaldavad sisse võtta peaaegu ükskõik millise poosi. Antud lõputöös kasutatud PeopleBoti külge on mõlemale küljele kinnitatud robotkäsi. Robotkäsi on võimeline kandma täiesti välja sirutatud käega pooleteise kilogrammi suurust raskust ning pooleldi välja sirutatud käega kahe kilogrammi suurust raskust. Robotkäe haarde ulatus on 68 cm 0.5 mm täpsusega. Robotkäel on kahe sõrmega haarats, mille haarde laius on 3.5 cm.

3.3 Flea3 kaamera

Joonis 7: Flea3 kaamera

Flea3 kaamera on kinnitatud manipulaatorite peale, et oleks võimalik manipulaatorit liigutada kaameralt saadud pildi põhjal . Kaamera on mõõtudega 29 x 30 x 30 mm. Kaamera loob pildi suurusega 1280 x 900 ning horisontaalse nurga suurus on 25.9 kraadi ja vertikaalse nurga suurus 16 kraadi.

3.4 Xtion Pro LIVE

Joonis 8: Xtion Pro LIVE

Xtion Pro Live on 3D-kaamera. Kaamera kasutab infrapuna sensoreid, andmete sügavust ja värvilist pilti tuvastavat tehnoloogiat, et jälgida kasutaja kujutist, liikumist ja on võimeline ka jälgima heli reaalajas. Xtion Pro Live on kinnitatud Bumblebee2 peale ja on seega kõige kõrgem PeopleBoti komponent.

4. Kasutatud algoritmid

4.1 Random Sample Consensus (RANSAC)

RANSAC-i[3] kasutatakse sageli mudeli parameetrite leidmiseks, kasutatades sisendandmeid, mille seas võib leiduda mittevajalikke andmeid. RANSAC hindab andmete globaalset suhet, samal ajal klassifitseerides andmed mudelisse kuuluvateks ning mittekuuluvateks. RANSAC on levinud, kuna see eemaldab suure osa vöörväärtustest.

RANSAC opereerib oleta-ja-verifitseeri raamistiku põhjal: minimaalne osa sisendandmete punktidest valitakse suvaliselt ja mudeli parameetrid hinnatakse sellest alamhulgast. Mudelit hinnatakse siis terve andmehulga peal ning leitakse sellesse kuuluvad punktid. Protsessi korratakse, kuni parema mudeli leidmise tõenäosus praegusest parimast mudelist on kindlast piirist madalam (1% - 5%). RANSAC leiab sageli õige tulemuse, isegi siis, kui on tegemist suure hulga mudelisse mittekuuluvate punktidega, kuigi katsetuste arv suureneb eksponentsiaalselt ning arvutuskulu on märgatavalt suurem.

Kui on olemas andmehulk U , siis RANSAC proovib läbi andmete alamhulki, suurusega m , kus m on minimaalne number katseid, mida on vaja, et arvutada lahendus, mis on võrdeline mudeli keerukusega. Kui mudel on hüpotiseeritud sellest alamhulgast, siis mõõdetakse selle kuuluvust kõikide andmehulka U kuuluvate punktidega ning võimalik lahend leitakse.

Seda protsessi korratakse, kuni lõpetuskriteerium saavutatakse. Standardne kriteerium RANSAC-is põhineb minimaalsel proovide arvul, mida on vaja, et veenduda mingil tasemel kindlusega η_0 , et vähemalt üks valitutest minimaalsetest alamhulkadest on võõrväärtusteta. Võttes arvesse, et mudelisse kuuluva punkti suhe on $\mathcal{E}$, siis tõenäosus, et valitakse võõrväärtusteta proov, millel on m mudelisse kuuluvat punkti, on $\mathcal{E}^m$. Tõenäosus, et valitakes k proovi, millest igaüks on sisaldab vähemalt ühte võõrväärtust, on $\left(1-\mathcal{E}^m\right)^k$. Seega miinimumarv proove, mida tuleb teostada, et veenduda, et tõenäosus on alla $1-\eta_0$ piiri, on:

$$k \geq \frac{log(1-\eta_0)}{log(1-\varepsilon^m)}$$

Kuna tõeline mudelisse kuuluvate punktide suhe $\mathcal{E}$ ei ole varasemalt teada, siis on võimalik leida selle suhte alampiir, kasutades proovi, millel on suurim mudel. Seda suhet uuendatakse, kui algoritmi edasi kasutatakse.

4.2 Canny algoritm

Canny[4] algoritmi eesmärk on rahuldada kolme põhilist kriteeriumit:

- Madal eksimus - algoritm peab märkama ainult eksisteerivaid ääri ning võimalikult palju
- Kaugus märgatud ääre pikslite ja reaalse ääre pikslite vahel peab olema võimalikult väike
- Ühte kontuuri äärt tohib tuvastada ainult üks kord

Canny algoritmi esimene samm on filtreerida pildilt müra. Selleks kasutatakse Gaussi filtrit. Näide võimalikust Gaussi filtrist suurusega 5:

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Järgmisena leitakse pildi intensiivsuse gradient. Selleks rakendatakse konvolutsioonimaatriks paare x- ja y-suunas:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Siis leitakse gradiendi suurus ja suund:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan(\frac{G_y}{G_x})$$

Suund ümardatakse ühest neljast võimalikuks nurgaks (0, 45, 90 või 135).

Eemaldatakse kõik pikslid, mida ei peeta osaks äärest. Jäävad alles ainult peenikesed jooned.

Lõpuks kasutab Canny pikslite sõelumiseks gradiendi ülemist ja alumist piiri

Kui piksli gradient on kõrgem kui ülemine piir, siis koheldakse teda kontuurina.

Kui gradient on allpool alumist piiri, siis seda ei võeta arvesse.

Kui piksel jääb piiride vahele, siis koheldakse teda kontuurina, kui ta on valitud piksli naaber.

4.3 Good features to track

Algoritm Good features to track[5] jälgib pildi regioone, kus on iga suunas suur intensiivsuse erinevus. Algoritm leiab nihke (u, v) intensiivsuse erinevuse igas suunas valemiga:

$$E(u,v) = \sum_{x,y} \quad w(x,y) \quad [I(x+u, y+v) - I(x,y)]^2$$

Kus *w(x, y)* on funktsioon, mis on kas nelinurkne aken või Gaussi aken, mis annab pikslitele kaalu. *I(x + u, y + v)]* on nihutatud intensiivus ning *I(x, y)* on intensiivsus.

Järgnevalt maksimeeritakse funktsiooni *E(u, v)* efektiivsust nurkade märkamiseks:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

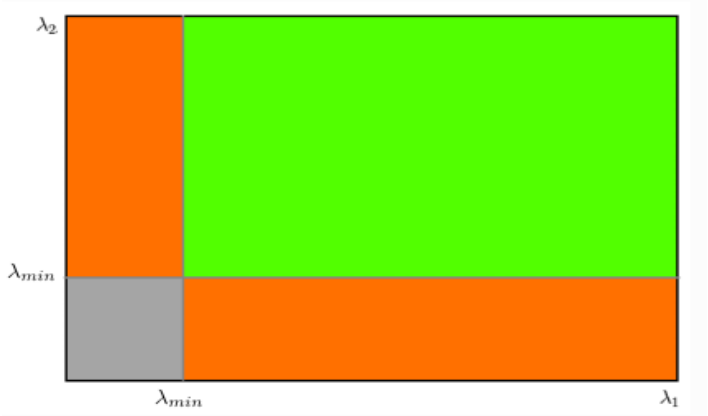
Kus

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Kus I_x ja I_y on pildi tuletised x- ja y-suunas vastavalt. Pärast seda otsustatake kas aken sisaldab nurka või mitte.

$$R = min(\lambda_1, \lambda_2)$$

Kui väärtus on suurem kui piiri väärtus, siis peetakse seda nurgaks. Kui panna see ruumi, siis saab järgneva pildi:



Joonis 9: Nurga määramine

Kui λ_1 ja λ_2 on suuremad kui miinimumväärtus λ_{min}, siis peetakse teda nurgaks (suur ristkülik).

5. Objekti tuvastamine kolmemõõtmelises ruumis

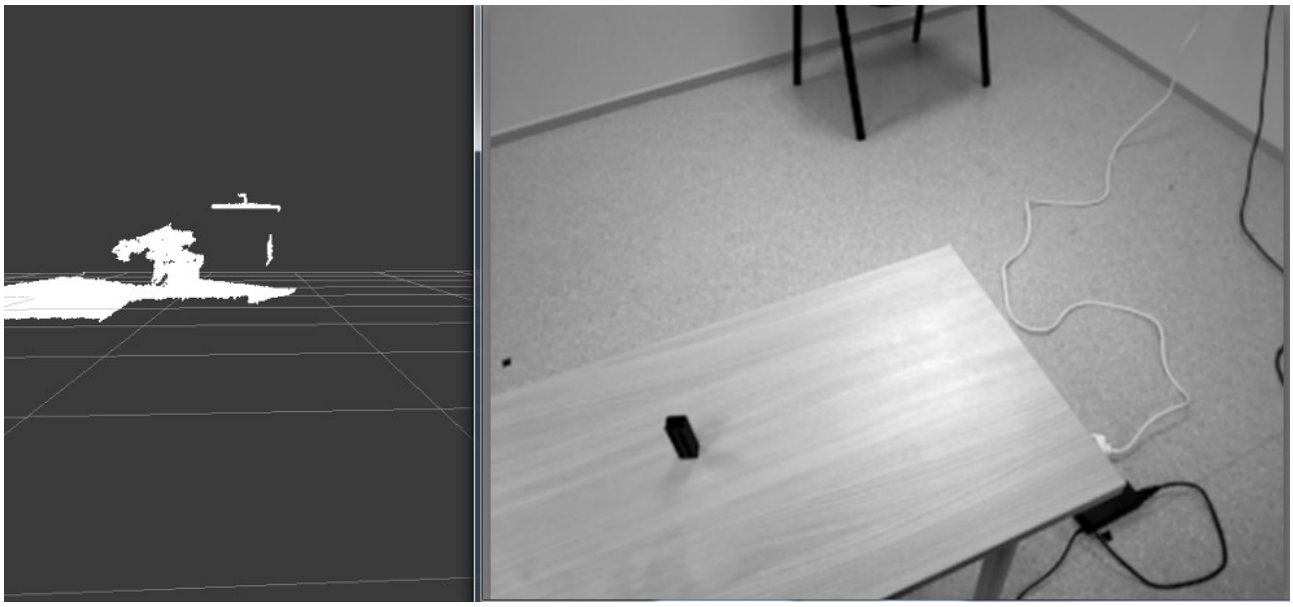
5.1 Kolmemõõtmelise ruumi teisendamine

Kolmemõõtmeline ruum luuakse kasutades roboti peal olevat 3D-kaamerat. Kuna objekt on roboti ees asuva laua peal, siis et manipulaatoriga oleks võimalik sellest haarata, peab robot olema sellele väga ligidal ja seega peab olema ka kaamera suunatud allapoole. Kui kaamera oleks suunatud otse, siis objekt ei jääks vaatevälja.

See tekitab probleemi, sest kõik punktipilve sisestatud objektid on ka nüüd sama kaldega, nagu seda on kaamera. Analüüsida andmeid sellise punktipilve peal, kus on koordinaadid nihkega on väga keeruline.

Joonis 10: Teisendamata punktipilv

Kuna robotil on mitmeid komponente, siis tuleb neid suunata kindla referentsruumistiku järgi ning kõikidel lülidel peab olema valmis arvutatud teisendus. Kui mingile lülile antakse käsk, siis see antakse referentsruumistiku koordinaadina, mis teisendatakse lüli koordinaadistikku ning sooritatakse käsk. Samuti on vaja punktipilvele teisendust, mis võtaks kaamerast loetud andmed ning teisendaks kolmemõõtmelise ruumi referentsruumistikku. Kuna robotil oli juba olemas teisendused "maailma" ja 3D-kaamera vahel, siis oli võimalik seda kasutada.



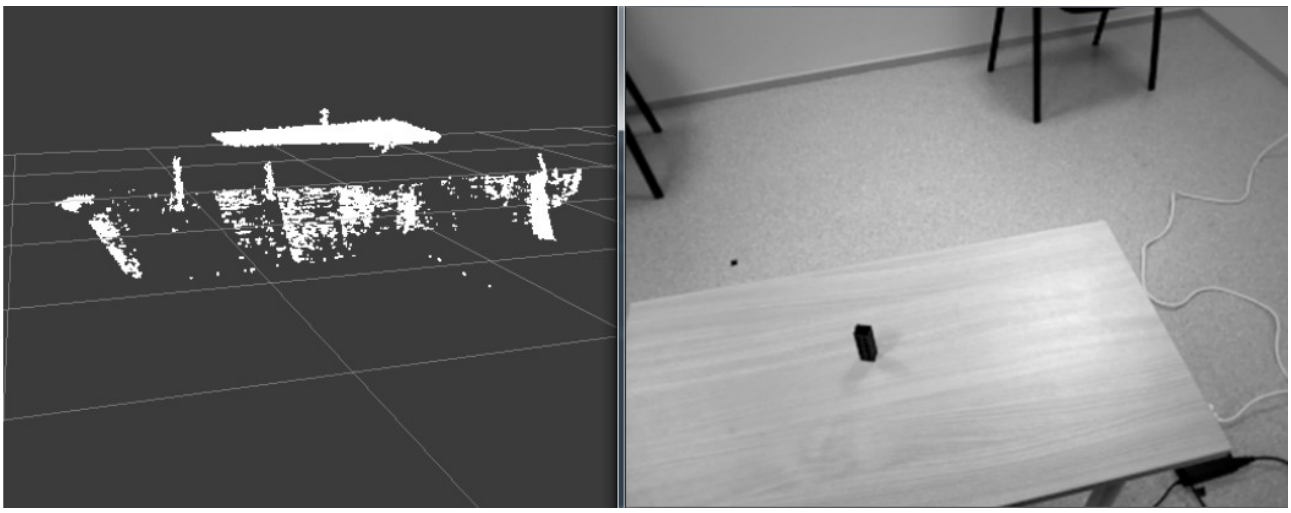
Joonis 11: Teisendatud punktipilv

5.2 Objekti leidmine punktipilvest

Objekti töötlemisel kolmemõõtmelises ruumis on kasutatud PCL teeki (Punkt 2.3). PCL-is on olemas mitmeid algoritme punktipilvede töötlemiseks. Objekti leidmiseks punktipilvest kasutatakse PCL-i klastrite eemaldust.

Kui kolmemõõtmelise ruumi andmed on olemas, siis on vaja kogu andmehulgast välja filtreerida objekt ning arvutada selle asukoht. Selleks on vaja eemaldada mitteolulised andmed. Kõigepealt eemaldatakse ruumist kõik andmed, mis ei ole laua pinnaga seotud.

Enne andmete eemaldamist valmistatakse ette kolmemõõtmeline ruum. Kolmemõõtmelises ruumis on punktid üksteise kõrval väga tihedalt ja andmetöötlus on mahukas. Et vähendada andmetöötlusele kuluvat aega, eemaldatakse ruumist osa punktidest, mis tekitavad müra ja mida ei ole vaja, et ära tunda erinevad punktikogud.



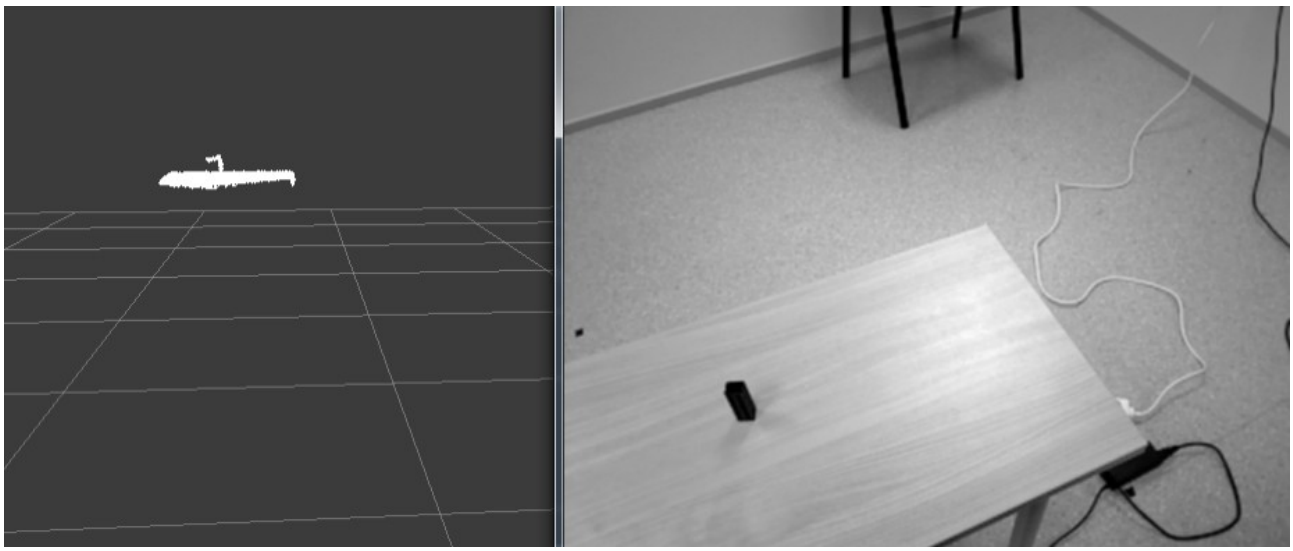
Joonis 12: Eemaldatud tasapinnaga punktipilv

Kui punktide tihedust on vähendatud, siis tuleb eemaldada kõige suurem tasapinnaline punktikogu, kuna mõnel juhul on raske tuvastada, kas eraldi olevad punktikogud on osa tasapinnast või mitte. Tasapinna tuvastamiseks kasutatakse RANSAC algoritmi (Punkt 4.1).

Eelnevalt pildilt on näha, et suurem osa pinnast sai eemaldatud, kuid palju jäi siiski alles. Antud juhul on punktipilvest eemaldatud 30% punktidest. Tulemused oleksid täpsemad, kui 3D-kaamera suudaks paremini sisse skaneerida kogu punktipilve. Võimalik oleks ka eemaldada suurem osa punktipilvest, kuid siis muutuks mingil hetkel kõige suuremaks tasapinnaks laud ise, mida pole vaja, sest paremaks objekti tuvastuseks on parem, kui laud on ühes tükis ja kergesti ära tuntav kui üks andmekogudest. Algoritmi eesmärk oli vähendada puntke, et suurendada tuvastusvõimet ja vähendada punktide koguhulka. Laud on hästi tuvastatav ja ühes tükis ning punktide arv on vähendatud, seega võib algoritmiga rahule jääda.

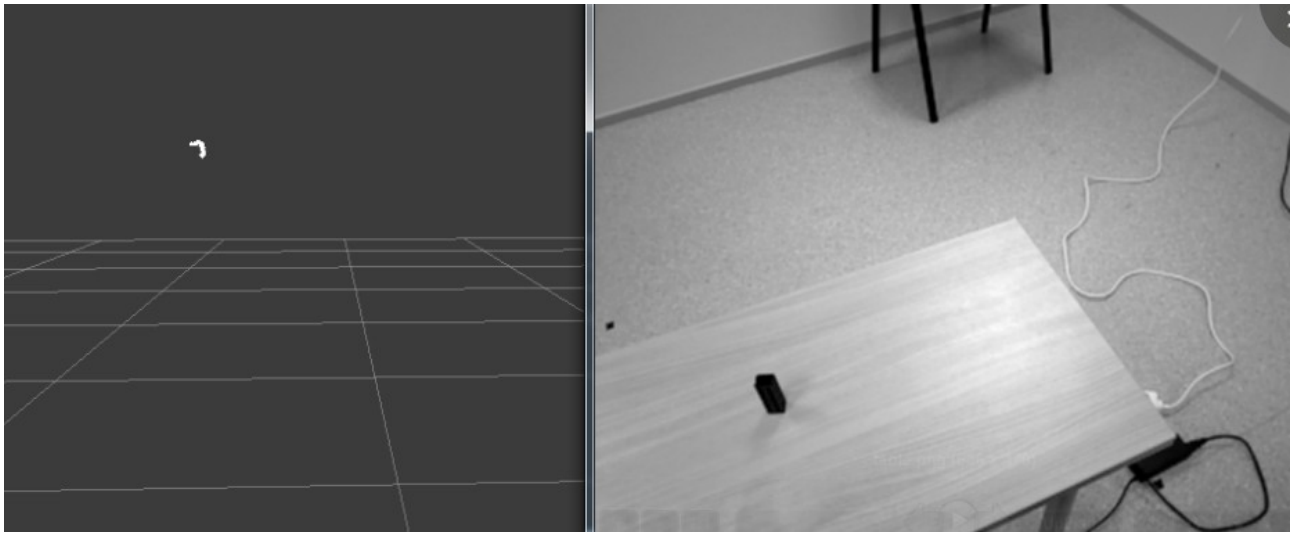
Nüüd kui punktide arvu on vähendatud ja on loodud eelised laua andmekogu tuvastamiseks, siis leitakse üles kõik erinevad ühte tükki kuuluvad andmekogud. Nende seast on vaja tuvastada laua punktikogu.

Laua punktikogu leidmiseks kasutatakse ära seda, et on teada reaalne laua kõrgus ning seda on võimalik võrrelda kaameraga loodud andmekogudega. Arvutatakse kõikide andmekogude keskmine z-koordinaat ning võrreldakse seda laua reaalse kõrgusega, et kindlam olla, siis lisatakse ka miinimumarv punkte, mis peab otsitavas punktikogus olema. See elimineerib võimaluse, et juhuslikku müra võidakse pidada otsitavaks lauaobjektiks. Tulemusena eraldatakse punktipilvest kõik muu ning saadakse vaid laua pind ning sellel olev objekt.



Joonis 13: Punktipilvest eemaldatud andmekogu koos laua ja objektiga

Nüüd on vaja eemaldada pildilt eemaldada laud ning alles peab jääma ainult otsitav objekt. Selle saavutamiseks kasutatakse jälle ära laua reaalset kõrgust, et eemaldada punktikogust punktid, mis jäävad all kindla z-koordinaadi. Kuna 3D-kaameraga laud ei ole täiesti tasapinnaline, siis tuleb kõrgusele lisada konstant, et kogu laud kaotada, vastasel juhul jääb osa kaldes pinnast alles. Lõpuks jääb ainult objekt.



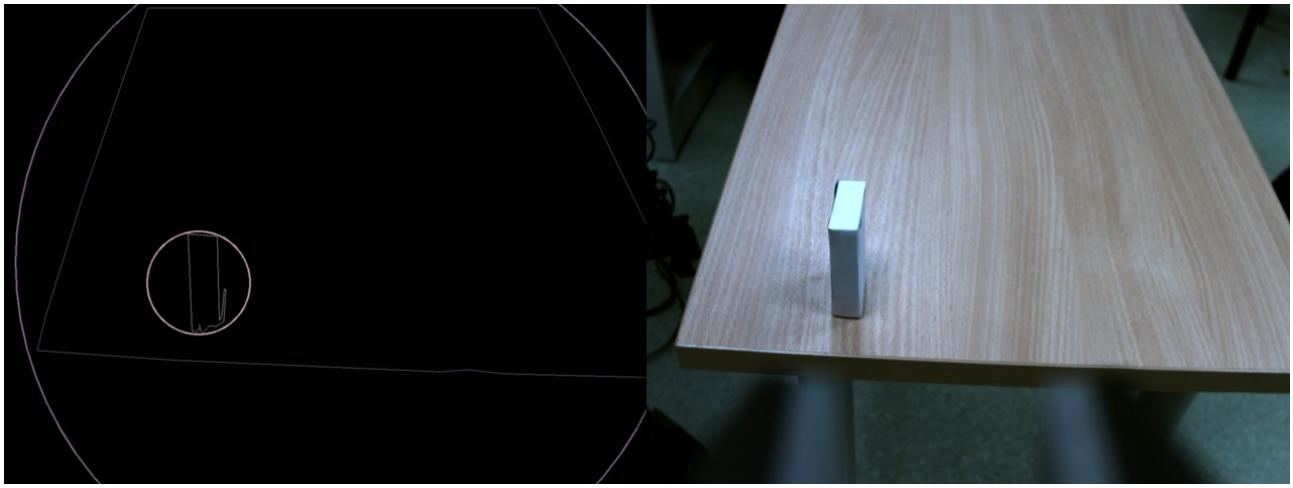
Joonis 14: Punktipilvest eemaldatud objekt

6. Objekti tuvastamine kahemõõtmeliselt pildilt

Kolmemõõtmelisest ruumist on võimalik objekti koordinaat kätte saada ning sellest juba piisaks, et manipulaator liigutada objektini. Kuid kolmemõõtmeline ruum ei ole nii täpselt kaardistatud, et oleks võimalik objektist haarata. Kolmemõõtmelisest ruumist saadud koordinaat annab ebatäpse asukoha ning et objektist haarata on vaja manipulaatorit palju täpsemalt juhtida. Kui kasutada ainult koordinaati, siis võib juhtuda, et manipulaatori liigutamisel ei lähe manipulaator objekti vastugi. Nii suur ebatäpsus nõuab, et kasutataks kaamerat manipulaatori peal, et haarata objekt. Edukas algoritm peab olema võimeline jälgima objekti manipulaatori liikumise ajal ning peab olema võimalik leida objekti kahe ääre vahel olev keskkoht. Järgnevalt kirjeldatakse erinevaid objekti tuvastusviise kahemõõtmeliselt pildilt.

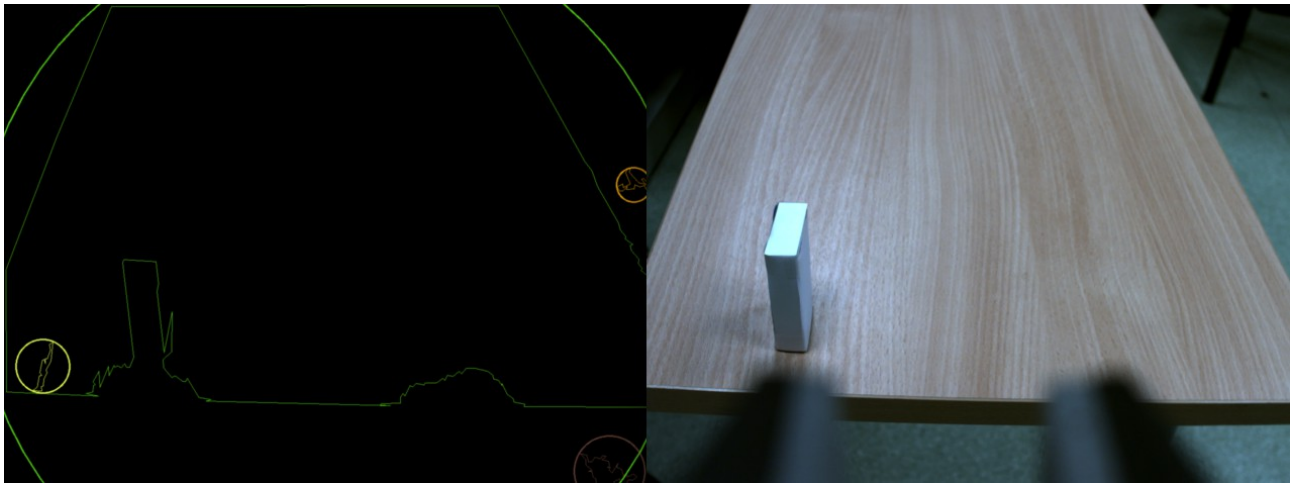
6.1 Kontuuride tuvastamine

Kontuuride tuvastamisel kasutatakse teeki OpenCV (Punkt 2.2). Et kasutada kontuuride tuvastust, muudetakse kaamerast saadud värviline pilt halliks. Kontuuride leidmiseks kasutatakse Canny algoritmi (Punkt 4.3).



Joonis 15: Kontuuri tuvastus kaugelt

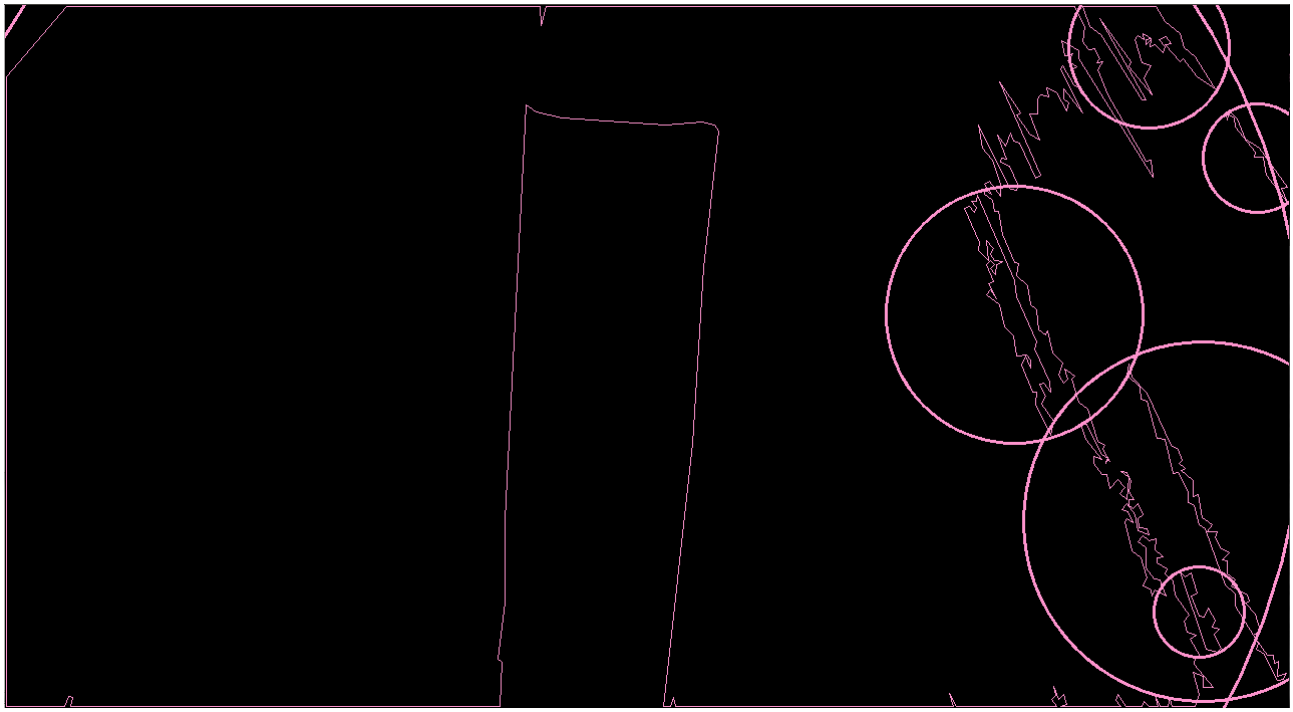
Pildilt on näha, et kontuuride tuvastus töötab väga hästi, ning objekt on tuvastatud. Kuid kui manipulaatorit liigutada ligemale, siis tekib probleem. Nimelt manipulaatori haaratsid on samuti pildil ning neid nähakse samuti kontuuridena. Hetkel on haaratsid laua servast piisavalt kaugel ning haaratseid ei arvestata laua kontuuri osana.



Joonis 16: Manipulaatori positsioonist tingitud rikutud kontuur

Eelnevalt pildilt on näha, et kontuuri ei tuvastada enam ning objekt on ühinenud laua ning manipulaatori kontuuriga. Algoritm ei suuda enam iseseisvat objekti leida.

Kuna manipulaator liigub kogu aeg ligemale, siis võib juhtuda, et kindlal kaugusel töötab algoritm hästi, kuid teisel kaugusel mitte. Kuna manipulaatori kaamera pildi lõppfaasis on objekt suurelt näha, siis saab seal kasutada kontuuri tuvastust.



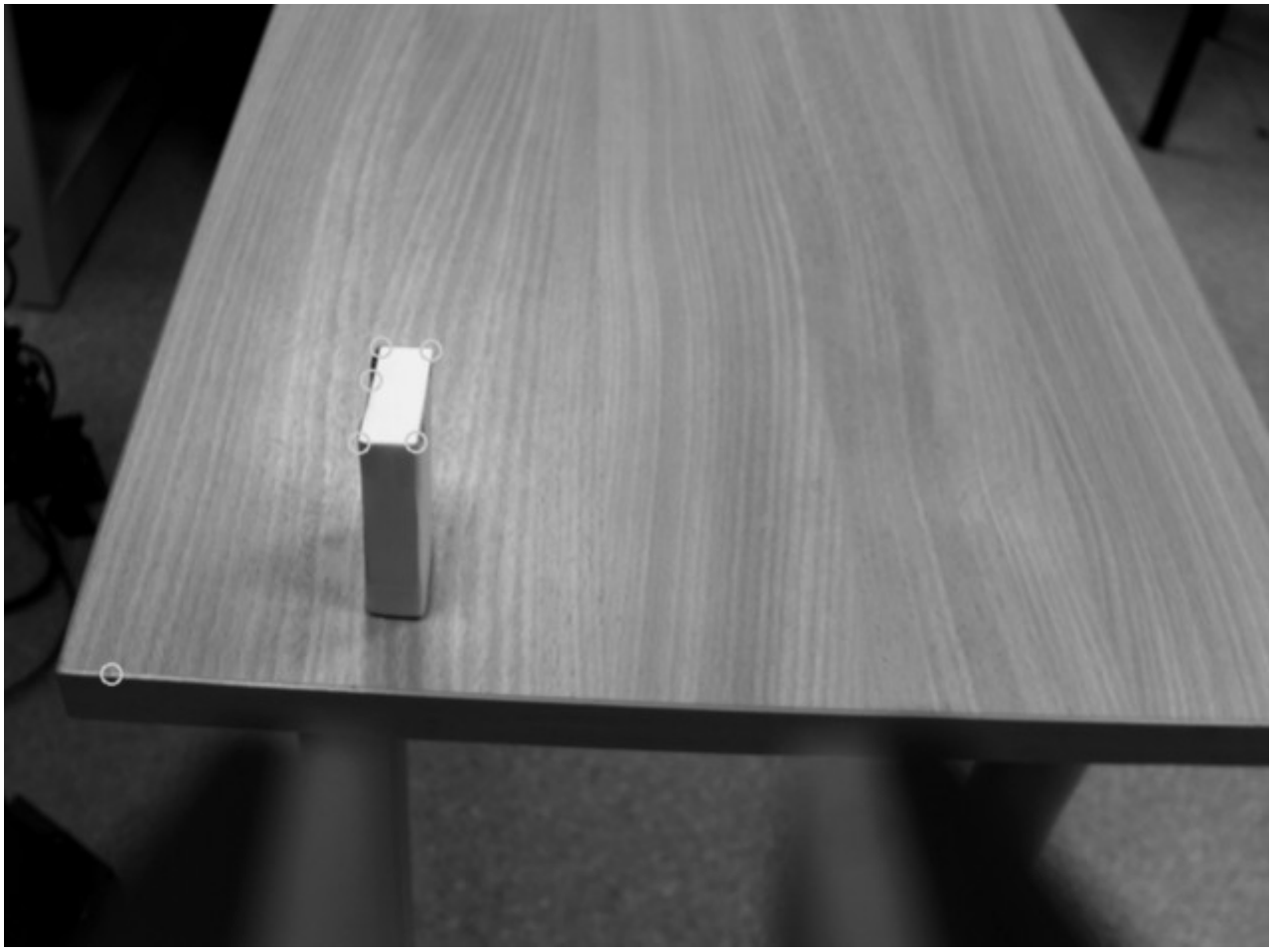
Joonis 17: Kontuurituvastus ligidalt

Sellel pildil on vaateväljast eemaldatud manipulaatorid, vältimaks kontuuride segamist. Antud pildil küll ei ole objekt tuvastatud kui iseseisva kontuurina, kuid kuna kontuuri jooned liiguvad otse alla, siis nad ühinevad äärtega ja moodustavad pildi äärtega ühe suure kontuuri, millest on võimalik

objekti kontuur kätte saada.

6.2 Nurkade tuvastamine

Nurkade tuvastamiseks kasutatakse samuti OpenCV teeki. Antud meetodit kasutades muudetakse sisendi pilt ka halliks. Nurkade märkamiseks kasutatakse Good features to track algoritmi(Punkt 4.3).

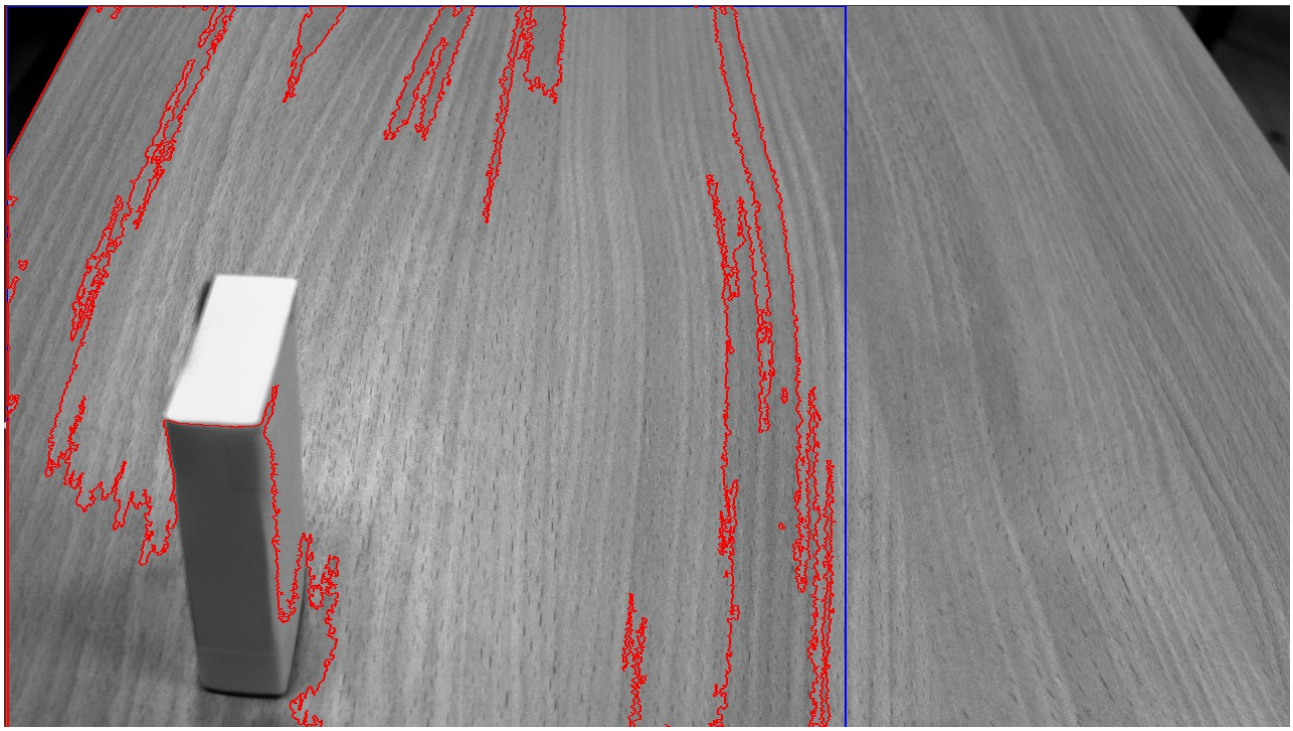


Joonis 18: Nurkade tuvastus

Pildil on pandud algoritm jälgima kuute nurka juhuks, kui leitakse midagi muud pildilt, mis meenutab nurka. Antud juhul on näha, et leiti kõik nurgad üles ning lisaks objekti üks äärekoht ning ka üks äärekoht laual. Algoritmiga tekkisid siiski probleemid, kui liigutada manipulaatorit ligemale. Suurema osa ajast suudeti nurki jälgida, kuid tuli ette kohti, kus nurgad kadusid ära. Vajalik on jälgida vaid kahte nurka, et saaks objekti asukoha ning laiuse teada, seega jälgiti kahte eesmist nurka.

6.3 Laigu tuvastamine

Laigu tuvastamiseks kasutati teeki ViSP. Jällegi muudeti sisendpilt halltoonideks, et oleks algoritmi võimalik kasutada. Laigu tuvastuse algoritm on senistest lihtsaim. See vajab sisendiks koordinaati. Leitakse koordinaadi halltoon ning leitakse, kas naaberpunktid on sarnase halltooniga või mitte ning kordab seda, kuni kõik piir ümber laigu on leitud.



Joonis 19: Laigu tuvastus

Pildil kasutati objekti pealse valge pinna koordinaati, et leida halltoon, kuid algoritm ei suutnud seda tuvastada. Antud lõputöös on taustaks kasutatud puidust lauda ning selle jooned segavad mitmeid tuvastusalgoritme. Nagu näha, siis segavad need jooned halltoon piiri leidmist. Kuna lõputöös on selline laud kasutusel, siis seda algoritmi kasutada ei olnud võimalik.

6.4 Värvi tuvastamine

Kuna olemasolevad algoritmid töötasid hästi ainult kindlal kaugusel ja positsioonil, siis oli vaja midagi lisaks. Ligidale liikudes ei suutnud ükski olemasolevatest algoritmidest objekti hästi jälgida. Selle jaoks oli vaja midagi ise luua. Kõige paremini silmaga tuvastatav erinevus on objekti värv pealmisel küljel. Seega peab seda ka värvi järgi olema võimalik eristada. Värvi tuvastamiseks kasutati kindlat halltoon.

Joonis 20: Halltooni tuvastus

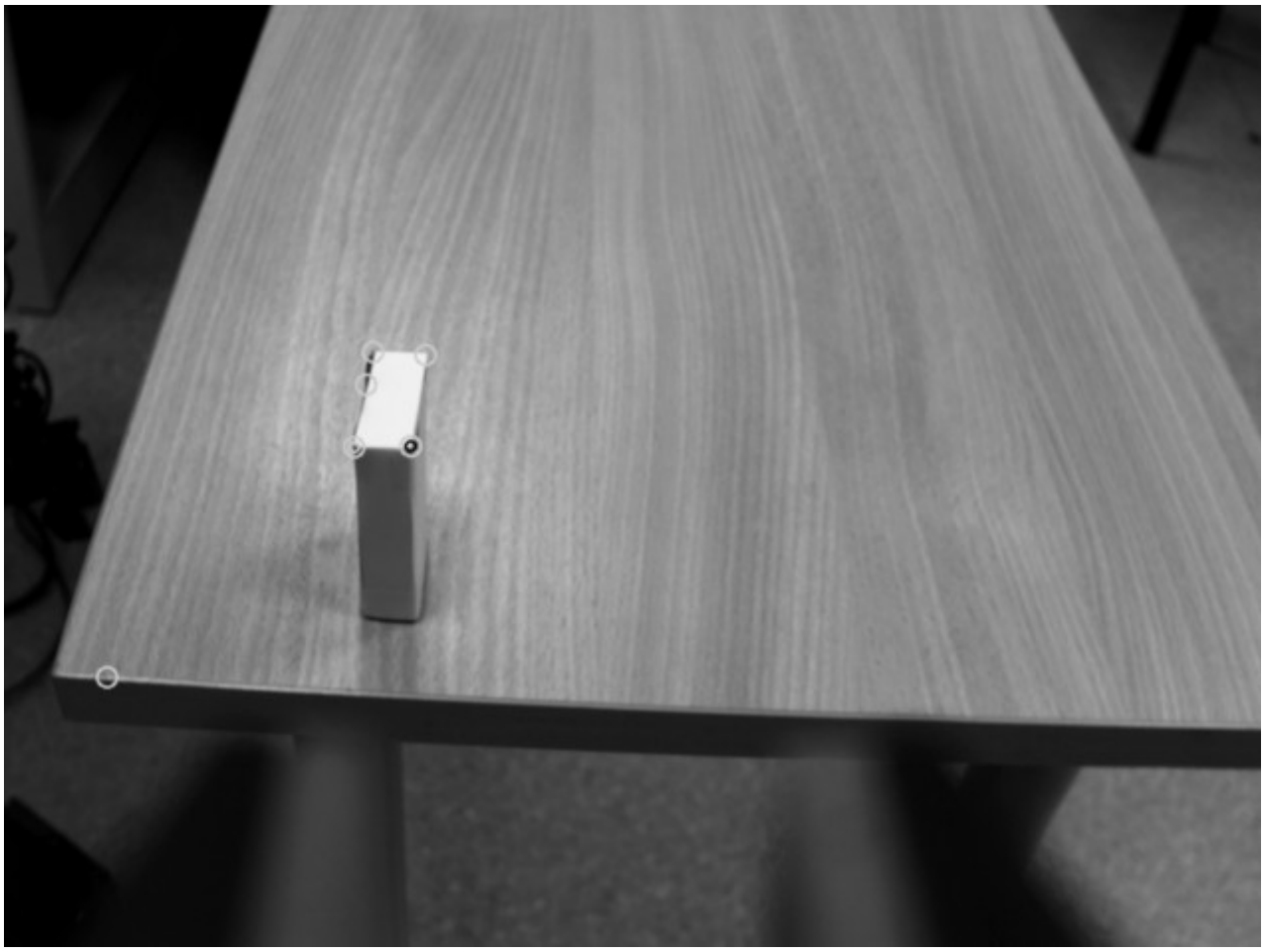
Antud pildil jälgitakse halltooni vahemikus 225-255. Halltooni jälgimine töötab väga hästi ja suudab jälgida kogu objekti pealmist pinda. Ligidal olles on see väga töökindel lahendus ning seda saab ka kaugelt jälgides kasutada.

6.5 Kombineeritud lahendus

Igal algoritmil on tugevust ja nõrkused ning kogu objekti jälgimist manipulaatori liikumise ajal ei ole ainult ühe algoritmiga võimalik. Lõpplahendusena kasutati kontuuride ja nurkade jälgimist. Kuna pildilt peab tuvastama objekti keskkoha, siis on vaja teada, milline leitud nurkadest on eesmised ülemised nurgad. Võib lihtsalt määrata mingi konstandi, et sealt jälgida nurki, kuid see ei ole nii kindel lahendust.

Kuna kontuurituvastus leiab algselt üles objekti, siis saab seda kasutada, et leida objekti y-koordinaat ning jälgida neid nurki, mis on selle y koordinaadi läheduses. Tulemusena leitakse vasak ning parem nurk. Kui vasak ja parem nurk on leitud, siis kasutatakse jälgimisel ainult nurkade

tuvastust. Järgneval pildi on parem nurk must ning vasak valge



Joonis 21: Nurkade tuvastus koos vasaku(valge) ja parema(must) nurgaga

Kuni manipulaator on veel kaugemal, siis töötab nurkade tuvastus hästi ning nurgad on alati jälgitud. Kui palju ligemale liikuda, siis ei suuda nurkade tuvastus õigeid nurki enam leida. Kuna kontuurituvastusega on võimalik ligidalt ka objekti kindlalt jälgida, siis ligidale jõudes kasutatakse kontuurituvastust.

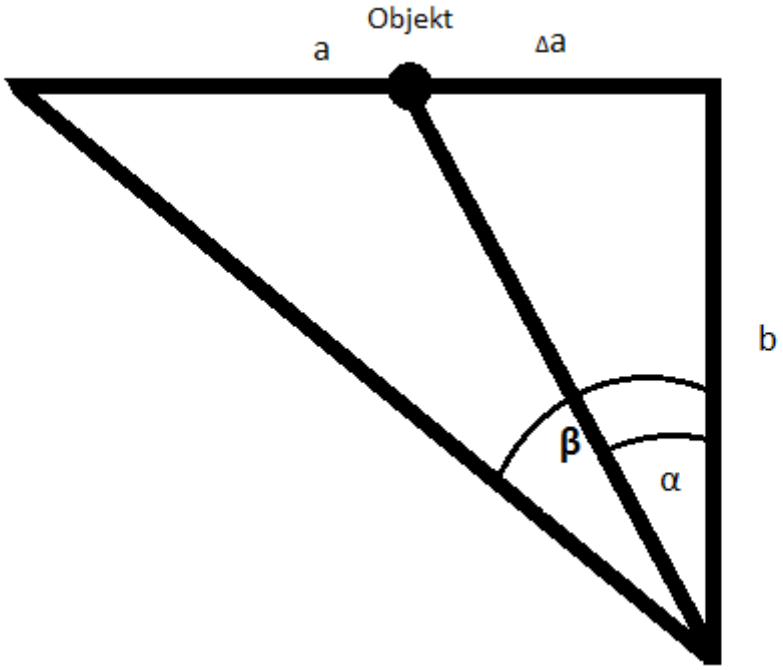
Seega lõpliku lahendusena kasutatakse alguses kontuurituvastust, et leida üles kaamerapoolse külje y-koordinaat, hakatakse jälgima nurki. Kui kaamera jõuab ligidale, lõpetatakse nurkade jälgimine ning kasutatakse kontuurituvastust. Lisaks kontuurituvastusele on kasutusel ligidal olles ka värvi tuvastus. Kontuuri jälgitakse seni, kuni pildilt ei ole leida otsitavat halltooni. See lisab kindlust, kui manipulaator peaks liikuma nii madalale, et valget värvi pole pildilt märgata, siis saab kontuure tuvastades edasi liikuda, kuni valge värv on jälle nähtav.

7. Objekti haaramine

Olles saanud punktipilvest objekti koordinaadid on võimalik manipulaatorit liigutada. Manipulaatori liigutamiseks on vaja koordinaati kui ka kvaterniooni. Kvaterniooni arvutamine sõltub samuti objekti asukohast. Kvaterniooni on vaja, et oleks teada, kuhu poole on manipulaatori haarats suunatud.

Esialgu on manipulaator vaja liigutada objektist kaugemale, et see tuvastada. Kuna manipulaatori peal olev kaamera on suunatud 35 kraadi allapoole, siis lisatakse see arvutatud kvaternioonile, et objekt oleks vaatevälja keskel. Kui kvaternioon on arvutatud, siis saab alustada liigutuse planeerimist. Kui liigutus on planeeritud, siis on see võimalik ka läbi viia.

Antud piltidel on manipulaator liigutatud 20 cm kaugusele objektist ning 20 cm kõrgemale. Objekt tuvastatakse. Seejärel hakatakse manipulaatorit liigutama objektile üha ligemale. Enne lähemale liigutamist on vaja arvutada nihe, mis tehakse manipulaatori kaamera põhjal. Järgnev kujund illustreerib, kuidas arvutatakse nihe.



Joonis 22: Manipulaatori nihke arvutamine

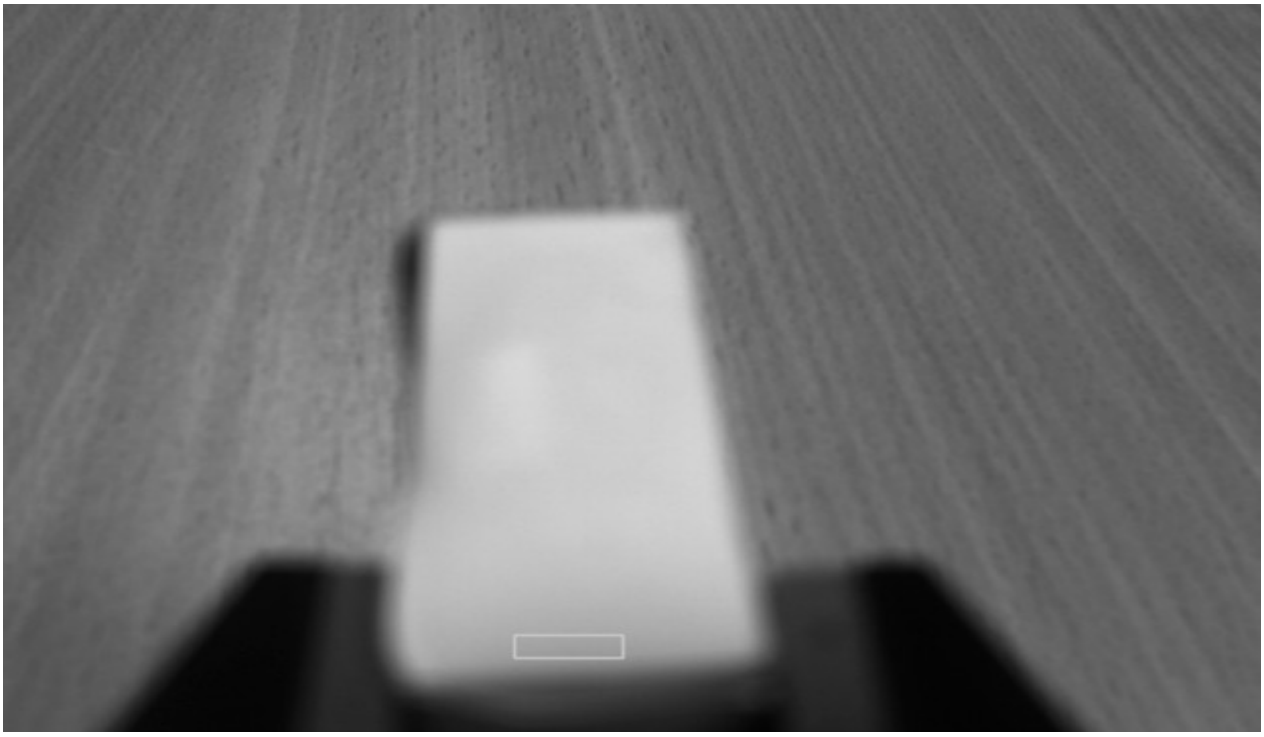
Alumine kolmnurga nurk on kaamera asukoht ning haar b poolitab vaatevälja. Haar a on pikslites kaugus vaatevälja keskelt kuni ääreni. Haar b on kaamera kaugus keskpunktist. β on pool kaamera vaatevälja ning α on nurk kaamera keskpunktist objektini. Teada on kaamera kogu vaatenurk, milleks on 25.9 kraadi ning kogu vaatevälja laius pikslites, mis on 960 px. Selle põhjal

saab väärtustada β = 12.95 kraadi ning a = 480 px, ning Δa pikslites saab kaamerast. Selleks, et leida Δa pikkus, on kõigepealt vaja leida nurk α, kasutades pikslite väärtust $\alpha = \frac{\Delta a}{a} \times \beta$ Kui α on leitud, siis $\Delta a = b \times tan\alpha$.

Kui nihe on arvutatud, siis on kogu info olemas järgmise liigutuse planeerimiseks ning olles teinud liigutuse, siis korratakse protsessi uuesti, garanteerides, et objekt jääb manipulaatori haaratsite ning kaamera keskele.

Kuna roboti liigutused ei ole alati identsed ning võib juhtuda, et mõni käik liigub planeeritud asendisse mitte kõige otsemat teed pidi. Tulemusena on lõppasend natuke paigast ära. See võib sageli juhtuda pikemate liigutustega, väiksemad liigutused on tavaliselt palju täpsemad. Kõige pikem liigutus on roboti algasendist esimesse asendisse, mis on tavaliselt 15-20 cm objektist eemal. Selle järel sooritatakse väiksemad liigutused objekti poole. Iga järgnev liigutus on 30% ligemal võrreldes eelmisega, kuni jõutakse 5 cm kaugusele, siis liigutakse z-teljel objektiga samale tasandile ning hakatakse liikuma 2 cm kaupa, kuni objekt haaratakse.

Objekti haaramine tehakse kasutades halltooni skaalat. Kaamera jälgib manipulaatori haaratsite vahel olevas ristküliku kujuga alas kindlat halltooni. Jälgitav ristkülik on asetatud kindlasse kohta, et saaks veenduda, et kui seal tuvastatakse see halltoon, siis peab seal objekt ka olema.



Joonis 23: Haaramisvajaduse jälgimine

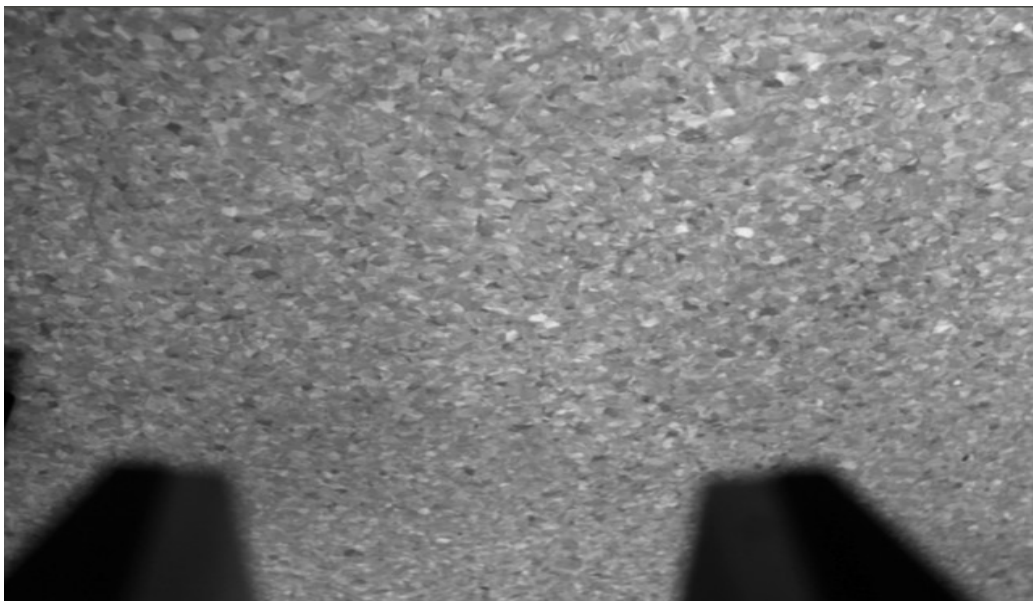
Pildil olev valge ristkülik on ala, kus jälgitakse halltooni. Kui õige halltoon on leitud, siis lõpetatakse liigutuste planeerimine 2 cm kaupa ning antakse käsk manipulaatori haaratsite kokku tõmbamiseks.

Teine võimalus on jälgida palju on objekti pealmisel pinnal alles kindla halltooniga piksleid. Kuna manipulaatori kaamera tekitab objektile varju, siis muutub manipulaatori liikumistrajektoori lõpus objekti halltoon. Seda saab kasutada, et tunda ära õige hetk haaramiseks. Kohati on see parem lahendus, sest mõnikord on manipulaatori asend objektiga võrreldes madalamal kui mõni teine kord ning kastis otsitav halltoon võib muutuda oodatust rohkem.

8. Katse

Järgnevalt illustreeritakse ühte teostatud katset kasutades pilte. Robot paikneb laua ees, 3D-kaamera on laua poole suunatud. Kasutatakse paremat manipulaatorit ning algasendis on robotkäsi suunatud maa poole.

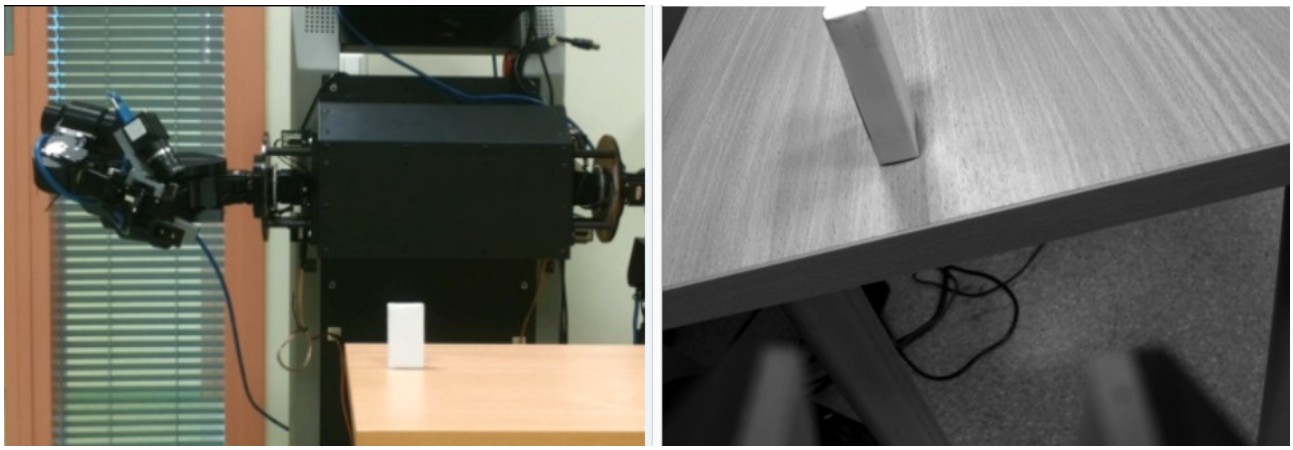
Joonis 24: 3D kaamera vaade roboti pealt



Joonis 25: Manipulaatori kaamera vaade algasendis

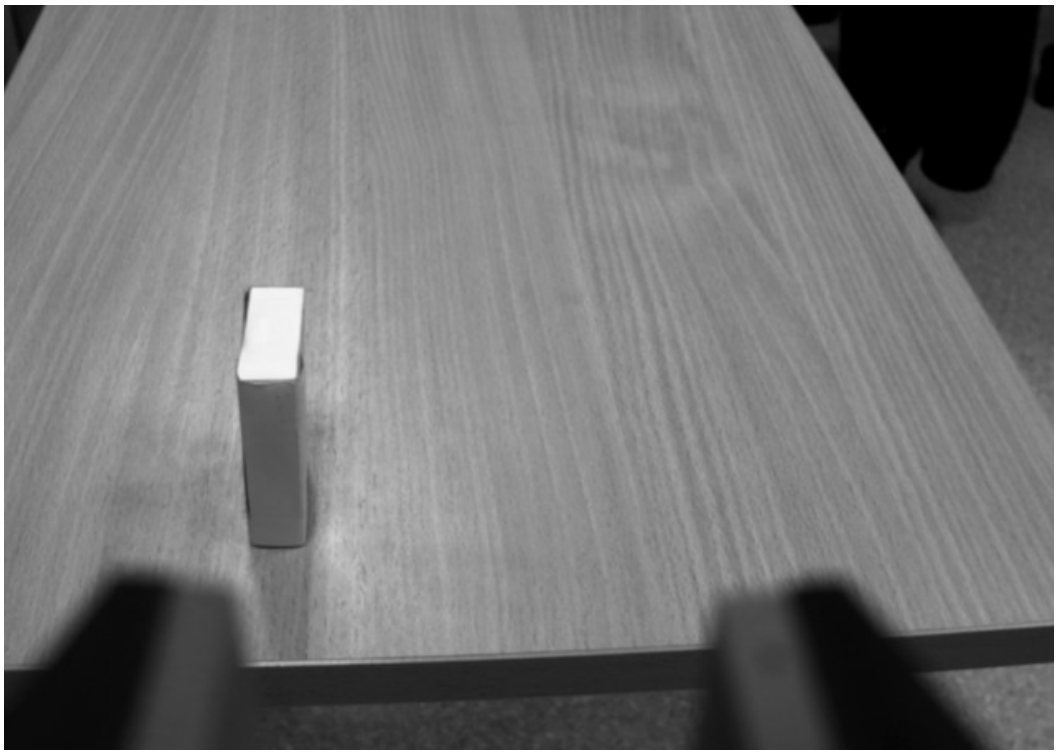
Joonis 26: Manipulaator hakkab liikuma laua poole

Kaamera hakkab jõudma laua juurde ning hetkel on kaamerast näha oa robotist.



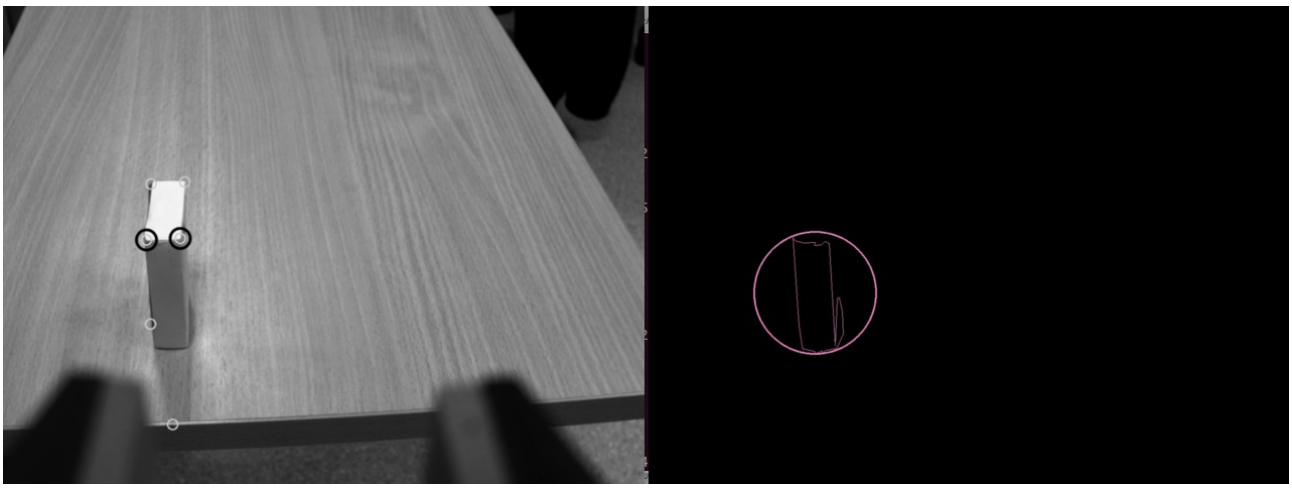
Joonis 27: Robot hakkab jõudma esimesse asendisse

Joonisel on näidatud, kus asub roboti manipulaator kaamera pildi ajal hetk enne esimesse asendisse jõudmist.



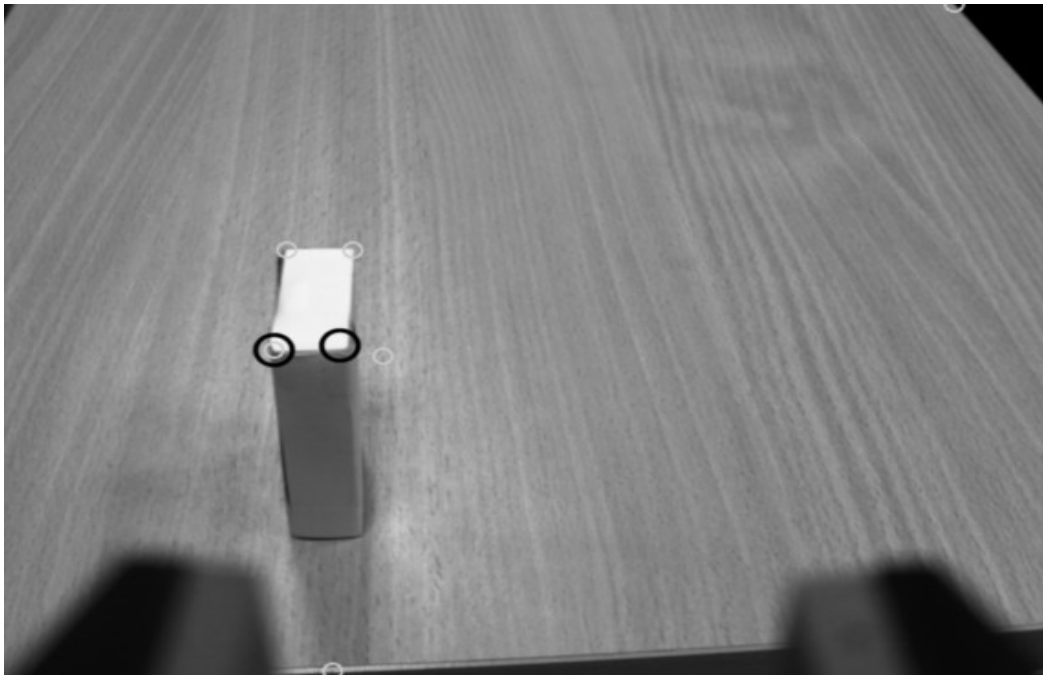
Joonis 28: Manipulaator esimeses asendis

Nüüd on manipulaator jõudnud esimesse asendisse, mis on z- ja y-teljel objektist eemal 15 cm võrra. Pildilt on näha nihe x-teljel.



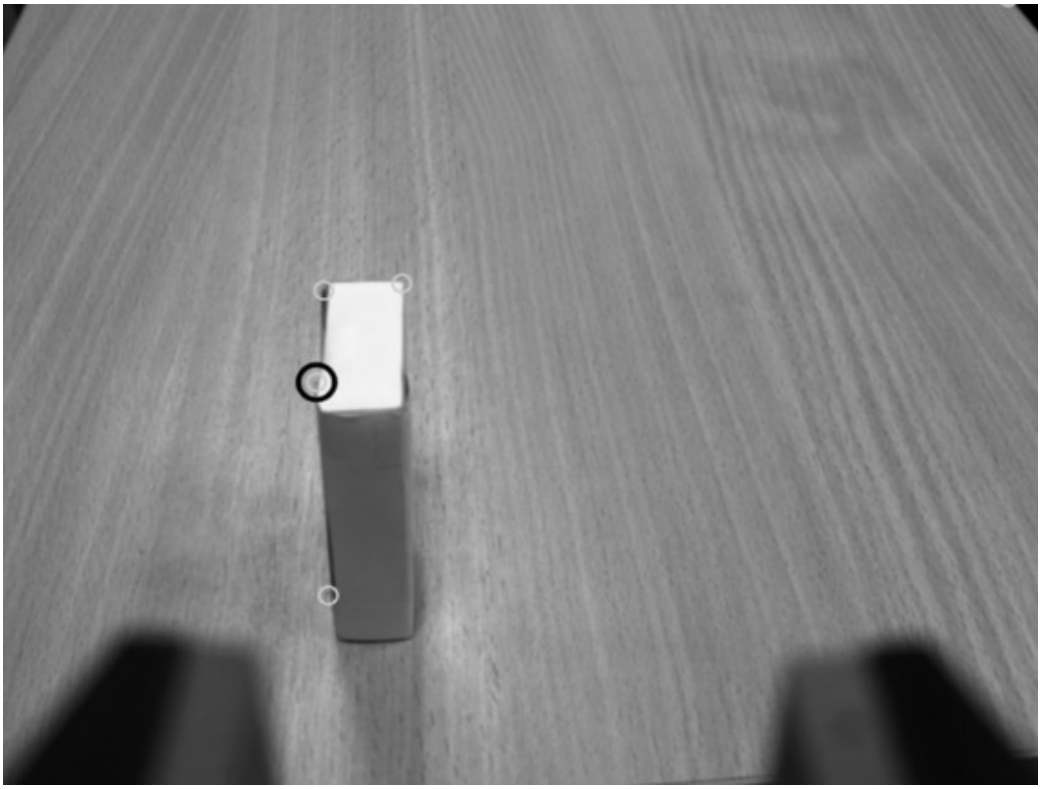
Joonis 29: Nurkade tuvastamine kontuuri põhjal

Kaamerapoolsed vasak ja parem nurk tuvastatakse kontuuri põhjal, leides kontuuri maksimaalne y-väärtus ning võrreldakse seda nurkade y-väärtusega.



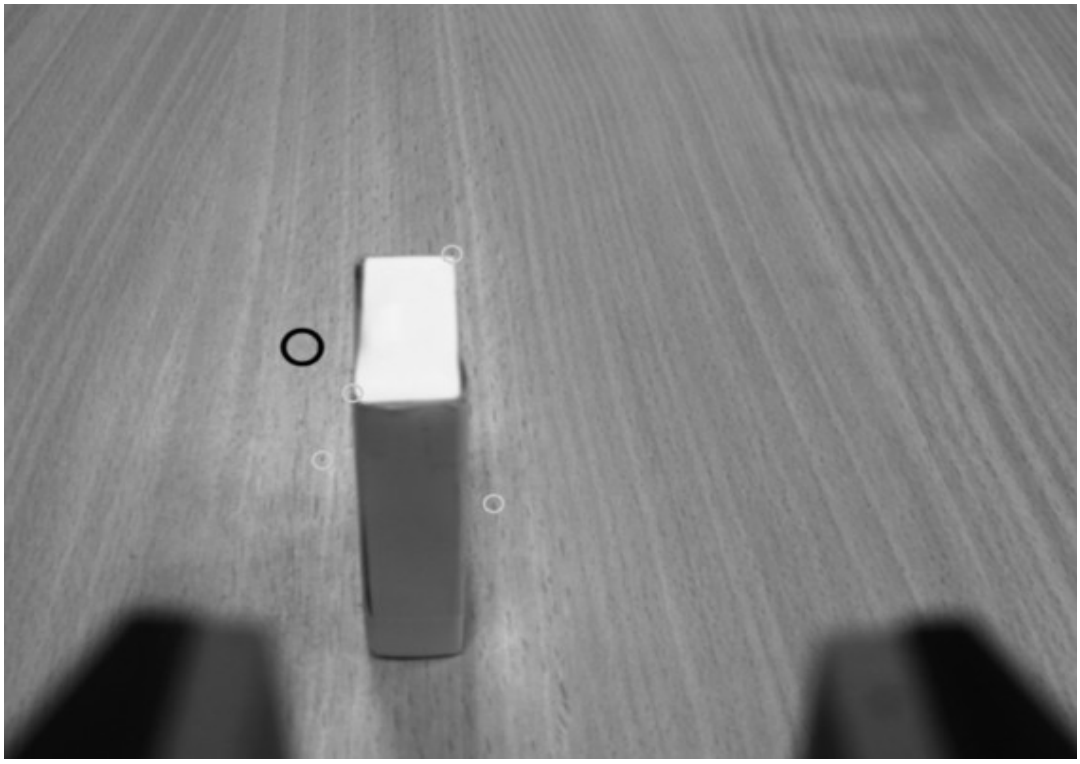
Joonis 30: Manipulaator liigub ligemale jälgides nurki

Kaamera liikudes proovitakse edasi jälgida nurki.



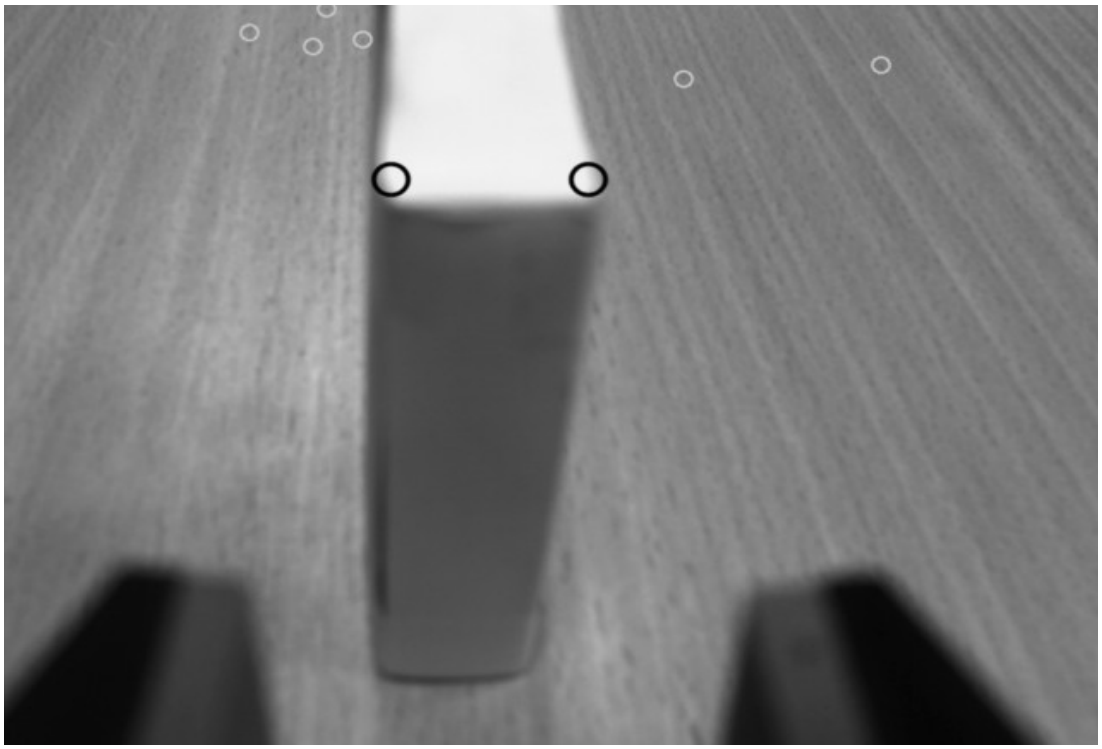
Joonis 31: Nurkade jälgimine on kaotanud ühe nurga

Nurgadetektor on kaotanud ühe nurga.



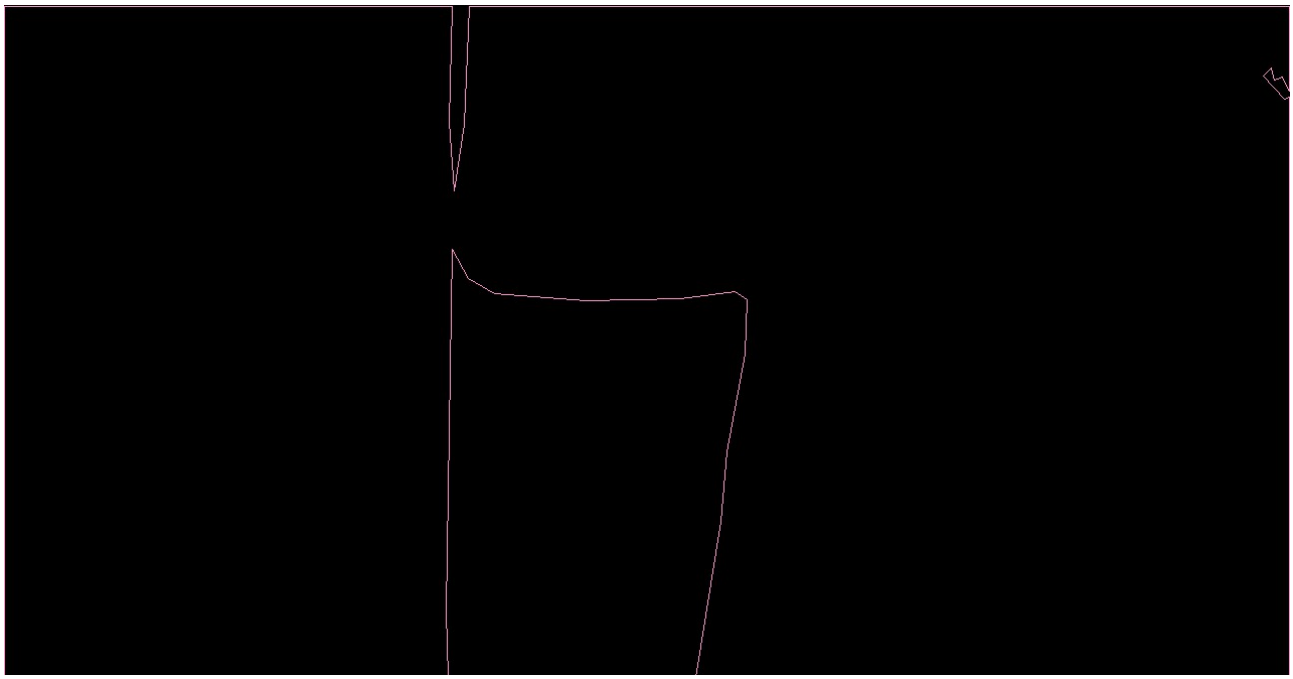
Joonis 32: Viimane asend, kui kasutatakse nurkade jälgimist

Nurgadetektor ei suuda tuvastada enam vajalikke nurki ning pildil olev nurk on vanast kaadrist.



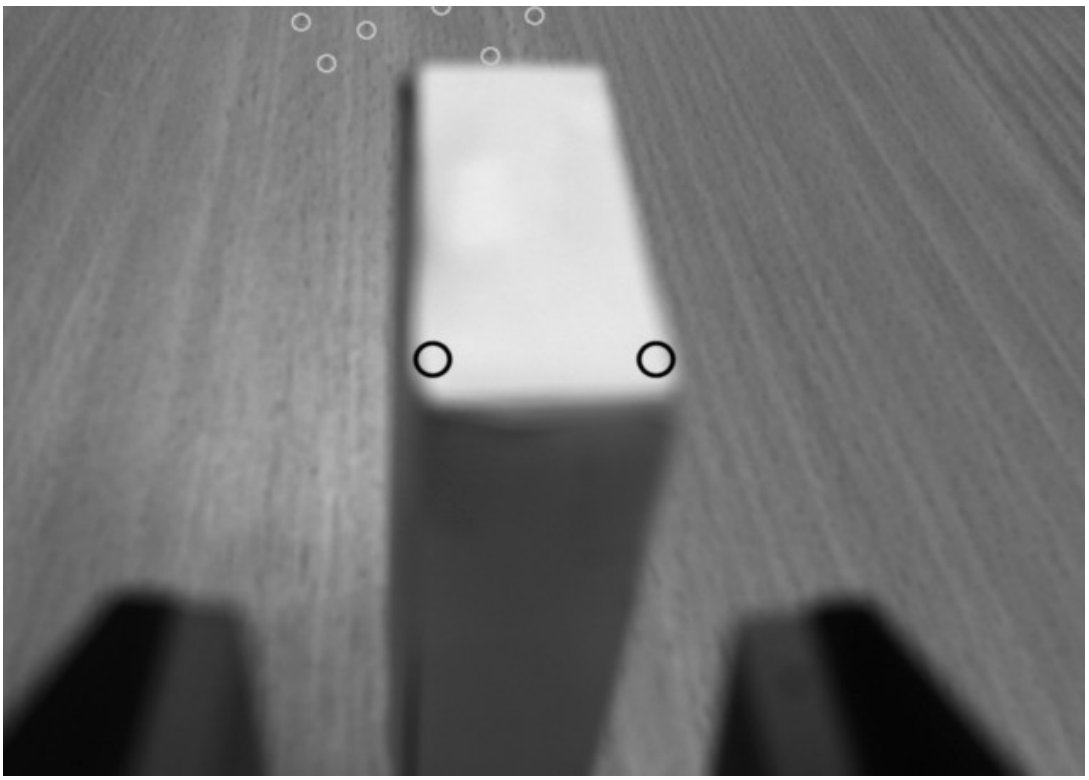
Joonis 33: Võetakse kasutusele värvi jälgimine

Siin lülitutakse üle halltooni jälgimine, mis suudab edukalt leida üles vajalikut nurgad. Objekti ümber on näha nurgadetektori poolt otsitavaid nurki, mis ei suuda pildilt leida vajalikke nurki, näidates, et nurgadetektorile ei saa ligidal olles loota.

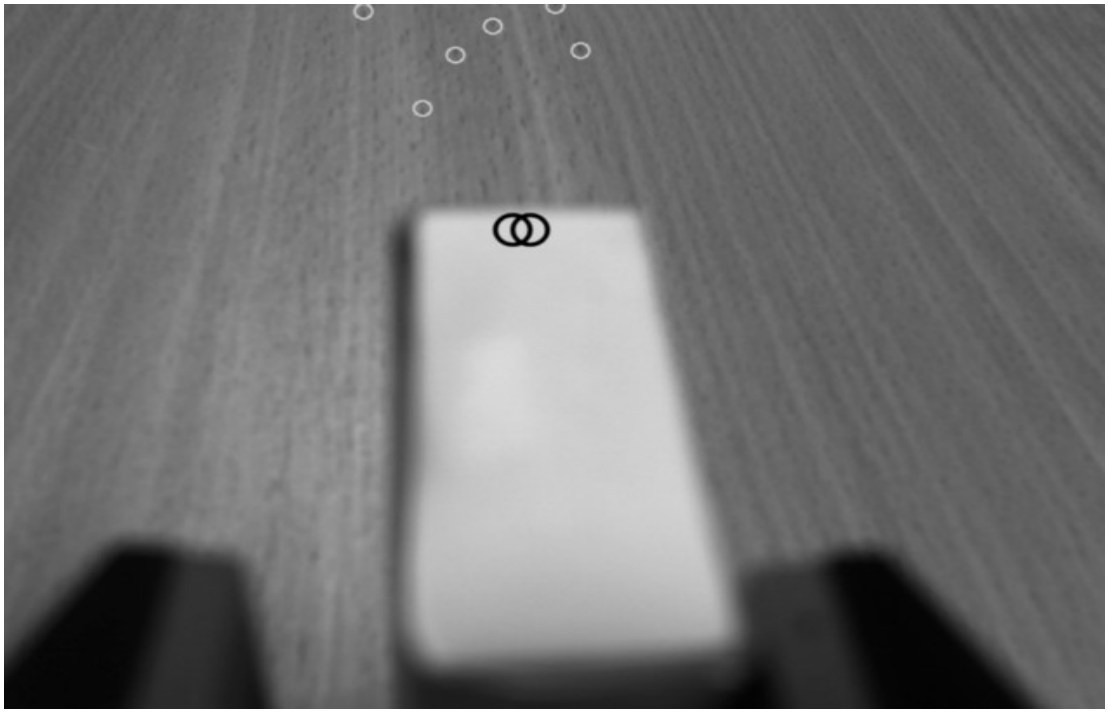


Joonis 34: Kontuuride jälgimine ligidalt

Kui peaks juhtuma, et manipulaatori kaamera satub niivõrd eemale objektist, et ei ole näha otsitavat halltooni, siis kontuuri jälgimine suudab kaamerapoolset tumedat poolt siiski jälgida. Antud katses oli valge värv nähtav ning kontuuridetektorit ligidal ei kasutatud, kuid on näha, et see töötaks.

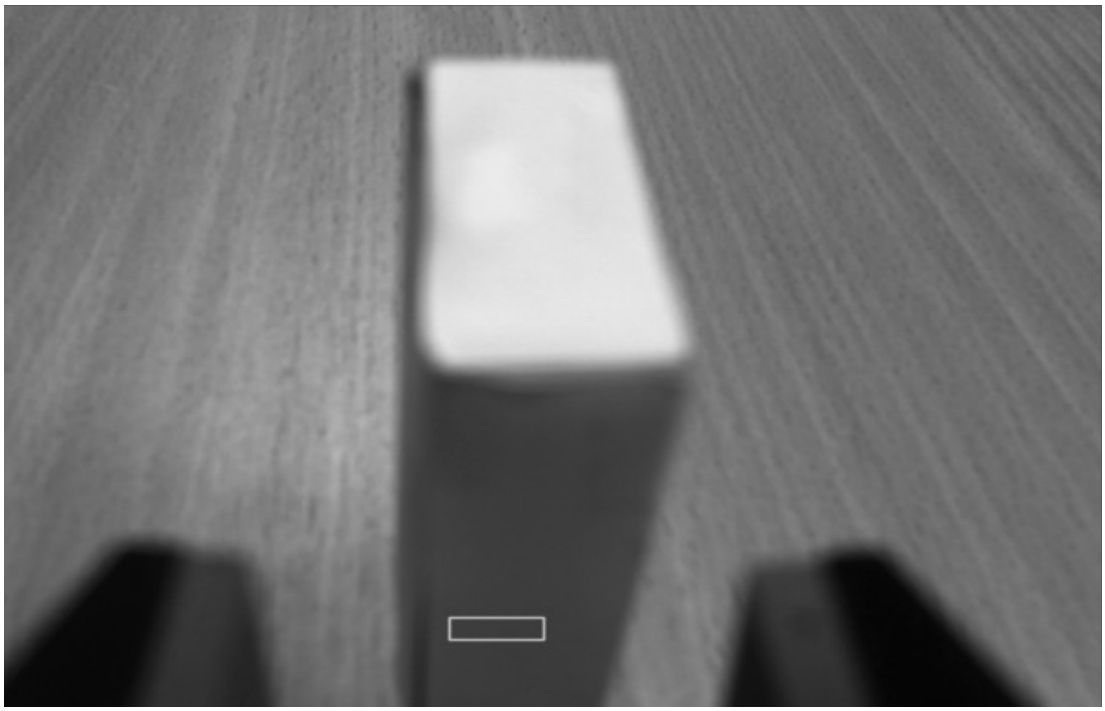


Joonis 35: Värvi jälgimine järgmises asendis



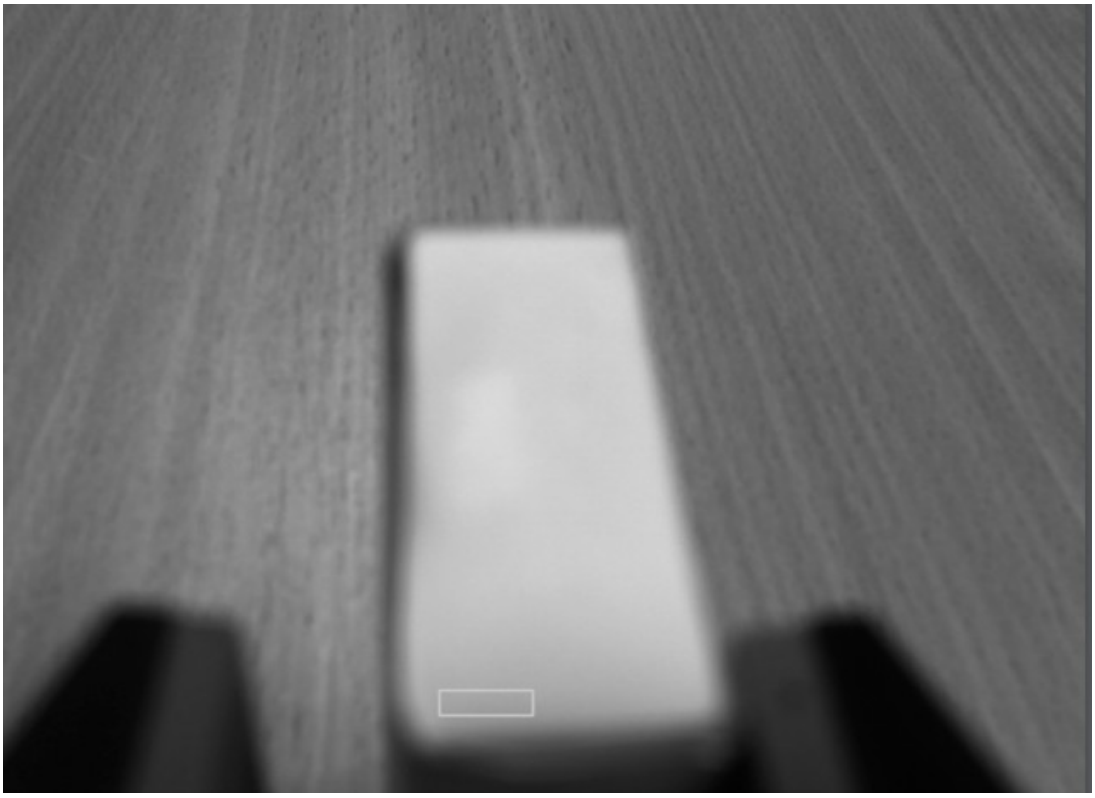
Joonis 36: Värvi jälgimine lõppasendis

Kuna manipulaator tekitab varju objekti kohale, siis väheneb jälgitava hallvärvi suurus ning märgatakse vajalikku hallvärvi ainult objekti lõpus. Kuna oluline on, et kahe punkti erinevus oleks objekti keskel, siis pole see oluline.



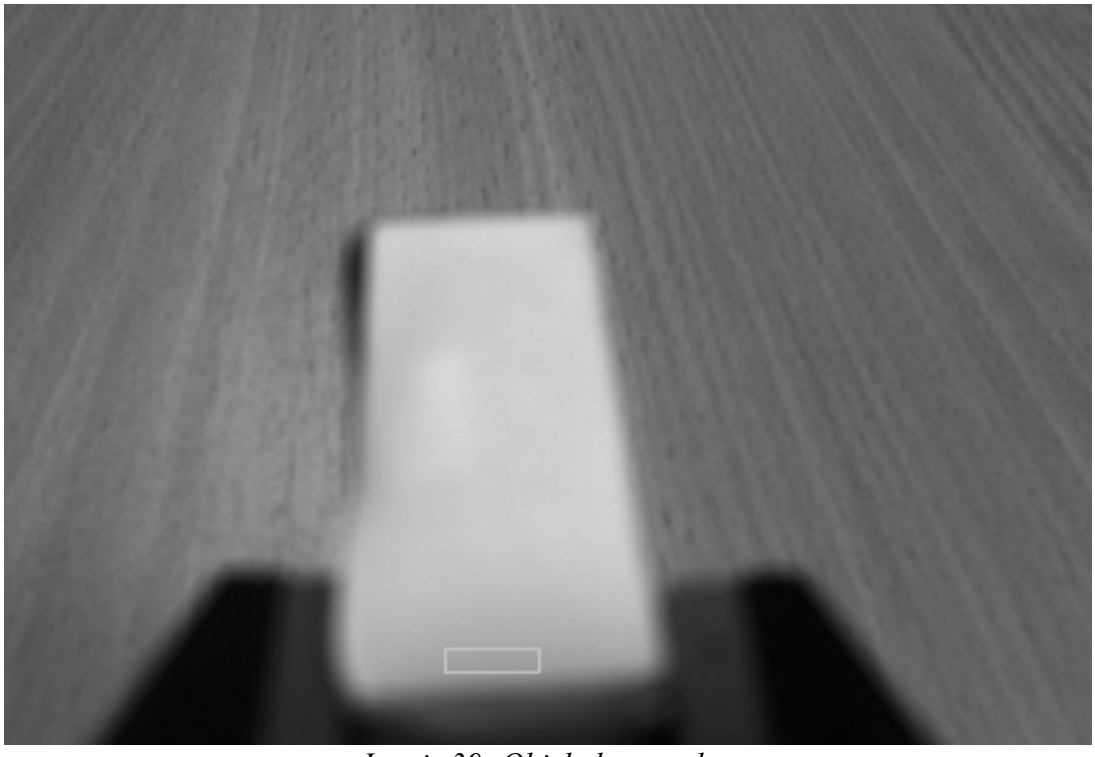
Joonis 37: Haaramisvajaduse jälgimine

Kui manipulaator on piisavalt ligidal, hakatakse jälgima halltooni



Joonis 38: Tuvastatakse, et on õige aeg haarata

Otsingu kasti alla satub õige halltoon ning antakse käsk objekt haarata.



Joonis 39: Objekt haaratakse

Objekt haaratakse ning katse on edukas.

Kokkuvõte

Lõputöö eesmärgiks oli programmeerida robotile tarkvara, mis suudaks visuaalse informatsiooni põhjal tuvastada objekt, jälgida seda ning see haarata. Lõputöö jooksul see ka loodi.

Kasutati mitmeid erinevaid visuaalse info töötlusalgoritme, millest kasutati iga manipulaatori positsiooni jaoks parimat. Kontuuri jälgimisel osutus probleemseks see, et laual olevad jooned ning manipulaatori haaratsid segasid jälgimist, kuigi kindlatel kaugustel töötas hästi. Nurgadetektor töötas kaugelt üpriski hästi - otsitavast neljast nurgast leiti peaaegu alati üles kas kolm või neli nurka, kuid objekti liikumisel ei suudetud nurki alati tuvastada ning nurki leiti ka objekti all olevalt laualt. Samuti mida ligemale liikus manipulaator, seda halvemaks muutus tuvastus, millele aitas kaasa kaamera fookuse vähenemine. Töös prooviti kasutada ka laigu jälgimist, kuid laua jooned ühineisd laiguga, muutes algoritmi kasutuks. Järeldusena võib tõdeda, et antud pildi parim kahemõõtmelise töötluse algoritm on värvi jälgimine, mis suudab igas kauguses jälgida objekti pinda. Kadu esineb ainult siis, kui manipulaator on objektile väga ligidal hetk enne haaramist. See aga ei oma suurt tähtsust, kuna sellest faasis on objekt juba haaratsite vahele juhitud, mis ongi antud töös kahemõõtmelise tuvastuse eesmärk.

Kuigi nurgadetektor ei suutnud manipulaatori laskumisel jälgida kogu perioodi ajal vajalikke nurki, siis kombineeritud lahendusega (kontuurituvastus, nurgadetektor ja halltooni jälgimine) suudeti ikkagi objekt edukalt haarata. Põhiline põhjus, miks see oli võimalik on tänu värvi tuvastusele lõpus, mis suutis manipulaatori juhtida tagasi keskele. Selle tõttu on parem kasutada värvi tuvastust kogu manipulaatori laskumise ajal.

Meelde peab jätma, et antud katses kasutati üleni valget objekti. Mõne teise kujuga või värviga objekti haaramiseks võib olla parem teine algoritm, kuid värvi tuvastus oleks kindlasti tugev konkurent iga objekti puhul.

Mitme katse käigus avastati, et 3D-kaamera keskmine viga kaamera ja objekti vahel oleva kauguse leidmisel on 1.9 cm. See viga on liiga suur, et oleks võimalik haarata objekt ainult 3D-kaamera sisendi põhjal ning manipulaatori kaamerata oleks see võimatu.

Kasutatud kirjandus

1. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A. (2009). ROS: an open-source Robot Operating System
2. Aldoma, A., Marton, Z.C., Tombari, F., Wohlkinger, W., Potthast, C., Zeisl, B., Rusu, R.B., Suat, G., Vincze, M. (2012). Point Cloud Library - Three-Dimensional Object Recognition and 6 DoF Pose Estimation
3. Raguram, R., Frahm, J.M., Pollefeys, M. (2008). A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus
4. http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html
5. http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html
6. http://opencv.org/
7. http://pointclouds.org/
8. http://www.irisa.fr/lagadic/visp/visp.html
9. http://www.ros.org/
10. Shi, J., Tomasi, C,. (1994). Good features to track
11. Canny, J. (1986). A Computational Approach to Edge Detection
12. http://www.asus.com/us/Multimedia/Xtion_PRO_LIVE/
13. http://www.ptgrey.com/flea3-usb3-vision-cameras
14. http://www.robai.com/robots/robot/cyton-gamma-1500/

Lisa 1

Loodud kood:

https://github.com/Jyrks/visual_servoing