Detekcija objekta na slici snimljenoj 3D kamerom i određivanje njegovog položaja

Živčić, Dominik

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

DETEKCIJA OBJEKTA NA SLICI SNIMLJENOJ 3D KAMEROM I ODREĐIVANJE NJEGOVOG POLOŽAJA

Diplomski rad

Dominik Živčić

Osijek, 2019

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 21.09.2019.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Dominik Živčić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-922R, 27.09.2018.
OIB studenta:	32802344832
Mentor:	Prof.dr.sc. Robert Cupec
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Emmanuel-Karlo Nyarko
Član Povjerenstva:	Doc.dr.sc. Damir Filko
Naslov diplomskog rada:	Detekcija objekta na slici snimljenoj 3D kamerom i određivanje njegovog položaja
Znanstvena grana rada:	**Automatizacija i robotika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Izraditi računalni program koji na temelju snimke dobivene 3D kamerom određuje položaj objekta od interesa i njegov položaj u odnosu na kameru. Oblik objekta od interesa zadaje se pomoću mreže trokuta. Tema rezervirana za studenta Dominika Živčića.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	21.09.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 02.10.2019.

Ime i prezime studenta:	Dominik Živčić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-922R, 27.09.2018.
Ephorus podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Detekcija objekta na slici snimljenoj 3D kamerom i određivanje njegovog položaja**

izrađen pod vodstvom mentora Prof.dr.sc. Robert Cupec

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Metode prepoznavanja objekata mogu biti vrlo korisne u zadacima u kojima robot mora manipulirati s predmetima u svojoj okolini ili mora detektirati prepreku na svojoj putanji gibanja. Cilj rada je izrada računalnog programa za prepoznavanje objekata od interesa u cilju ostvarivanja planiranja putanje robotskog manipulatora po površini prepoznatog objekta. Objekt se zadaje pomoću mreže trokuta, a za prepoznavanje objekata se koristiti RGB-D kamera. Programsko rješenje je izrađeno u ROS okruženju. U drugom poglavlju je opisana teorijska podloga robotskog vida, te su opisane metode detekcije objekata koje su implementirane u programskom rješenju. Treće poglavlje opisuje izradu programskog rješenja za sustav koji se sastoji od robotskog manipulatora i kamere. Na kraju je opisan postupak testiranja sustava te su prikazani podaci koji govore o točnosti implementiranog sustava. Sustav se sastoji od industrijskog manipulatora ABB IRB 2400L i Orbbec Astra S kamere. Prikazana su trenutna ograničenja sustava te su navedene moguće preinake za poboljšanje sustava.

1.1. Zadatak rada

Izraditi računalni program koji na temelju snimke dobivene 3D kamerom određuje položaj objekta od interesa i njegov položaj u odnosu na kameru. Oblik objekta od interesa zadaje se pomoću mreže trokuta.

2. PREPOZNAVANJE OBJEKATA POMOĆU RGB-D KAMERE

U robotici prepoznavanje objekata može služiti raznim primjenama kao što su prepoznavanje objekata s kojima robot treba manipulirati, određivanje položaja objekata, detekcija prepreka i dr. Sustav robotskog vida se može opisati kao spoj percepcijskog senzora i softvera za obradu slike.

2.1. RGB-D kamera

Percepcijski senzori mogu biti RGB kamere koje daju RGB sliku, odnosno matricu vrijednosti koje predstavljaju intenzitet i boju svijetla. Stereo sustav dvije kamere omogućuje određivanje 3D pozicija točaka iz poznatog položaja jedne kamere u odnosu na drugu. Postoje i razni 3D percepcijski senzori kao što su RGB-D kamere, laserski davači udaljenosti, time-of-flight kamere i dr. U ovom radu koristi se RGB-D kamera

RGB-D kamere pripadaju skupini 3D kamera na principu strukturirane svjetlosti. Ovaj tip senzora se temelji na projiciranju svijetlosti na određenu površinu te snimanju reflektirane svijetlosti pomoću kamere. Dubina točke se izračunava metodom triangulacije poznajući međusobni odnos projektora i kamere.

Na slici 2.1 je prikazana Orbbec Astra S RGB-D kamera. Crvenom bojom je označen IR projektor, zelenom bojom RGB kamera te plavom bojom IR kamera. Ova kamera ima domet od 0.4 do 2 metra s rezolucijom od 640x480 piksela.

Slika 2.1 Orbbec Astra kamera

Zbog nesavršenosti geometrije optičkog sustava kamere može doći do izobličenja slike. Postupkom kalibracije kamere određuju se intristični parametri pinhole modela kamere te parametri izobličenja kako bi se uklonile moguće pogreške i odstupanje od pinhole modela. Intristični parametri sadrže informacije potrebne za centralnu projekciju 3D prostora na 2D ravninu. Jedan od načina kalibracije kamere je da se pred kameru postavlja kalibracijski panel sa šahovskim uzorkom te se uzimaju slike samog panela. Postupcima optimiranja se dolazi do parametara kamere. Postupak kalibracije kamere je opisan u poglavlju 2.3.

2.2. Prepoznavanje objekata

Kod većine metoda prepoznavanja objekata koristi se ista strategija. Ona se sastoji od tri glavna dijela, a to su detekcija značajki, postavljanje hipoteza te odabir najvjerojatnije hipoteze. Značajke su dijelovi snimke koji se mogu odvojiti od ostatka primjenom nekog kriterija. Kod postupka prepoznavanja objekata u oblacima točaka, najčešće se svakoj točki pridružuje vektor u smjeru normale površine objekta te se takve značajke nazivaju orijentirane točke. Postavljanje hipoteze je postupak pretpostavljanja da se neki traženi objekt nalazi na sceni. Najčešće postoji više hipoteza te je potrebno nekim postupkom odabrati najvjerojatnije hipoteze te tako pronaći objekt od interesa.

Kod problema prepoznavanja objekata zadanog oblika, rješenje postupka prepoznavanja, uz samu detekciju prisutnosti objekta na sceni, je i njegov položaj. Položaj nam govori o tome gdje se objekt nalazi u odnosu na neki referentni koordinatni sustav. Ukoliko govorimo o 3D prostoru, poziciju objekta u odnosu na neki koordinatni sustav možemo predstaviti s tri varijable a to su x, y i z koordinata ishodišta koordinatnog sustava objekta u odnosu na taj koordinatni sustav. Orijentacija nekog koordinatnog sustava u odnosu na drugi koordinatni sustav se može opisati rotacijskom matricom. Postoji rotacijska matrica za rotaciju oko x (2-1), y (2-2) i z-osi (2-3).

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}, \tag{2-1}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \tag{2-2}$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2-3}$$

Kompozicija ovih matrica je jednaka njihovom umnošku (2-4)

$$R(\phi, \theta, \psi) = \quad R_x(\phi) R_y(\theta) R_z(\psi) \tag{2-4}$$

Kombiniranjem translacije i orijentacije dobijemo transformacijsku matricu koja sadrži potpunu informaciju o položaju objekta u odnosu na neki referentni koordinatni sustav (2-5)

$$T_{AB} = \begin{bmatrix} R_{AB} & t_{AB} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \tag{2-5}$$

R_{AB} – Rotacijska matrica koja opisuje orijentaciju koordinatnog sustava A u odnosu na koordinatni sustav B

t_{AB} – Translacija ishodišta koordinatnog sustava A u odnosu na koordinatni sustav B

Osim pomoću rotacijske matrice, orijentacija se može prikazati i kvaternionima. Kvaternion predstavlja proširenje kompleksnih brojeva i ima oblik

$$a + bi + cj + dk. \tag{2-6}$$

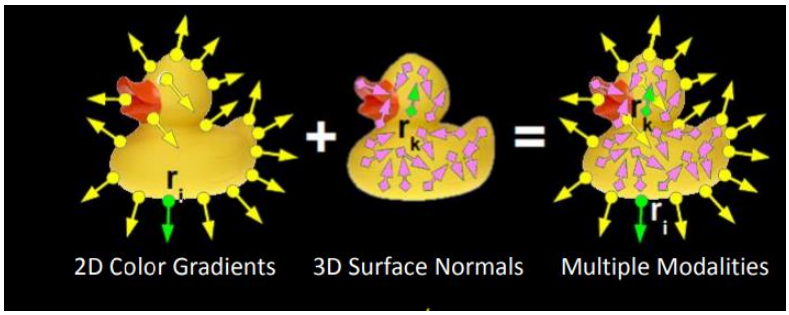
Rotacija oko osi definirane jediničnim vektorom u za kut θ može se prikazati kvaternionom

$$q = \cos\frac{\theta}{2} + sin\frac{\theta}{2}(u_x i + u_y j + u_z k). \qquad (2\text{-}7)$$

2.3. LINE-MOD metoda prepoznavanja objekata

LINE-MOD metodu je predložio Stefan Hinterstoisser [1] te je metoda implementirana u OpenCv i PCL. Metoda se koristi za prepoznavanje krutih objekata na sceni uz pomoć RGB i depth informacija.

Ova metoda se temelji na odabiru modaliteta, odnosno značajki na više različitih slika modela koji će se poslije koristiti za detekciju. Postupak odabira modaliteta se naziva treniranje modela te se ono može vršiti na RGB ili dubinskoj slici te u kombinaciji obje slike. Kod RGB slike računaju se gradijenti boje dok se na dubinskoj slici računaju normale na površinu. Na slici 2.1 se vidi primjer odabira modaliteta modela uz pomoć gradijenta boje i normala na površinu.



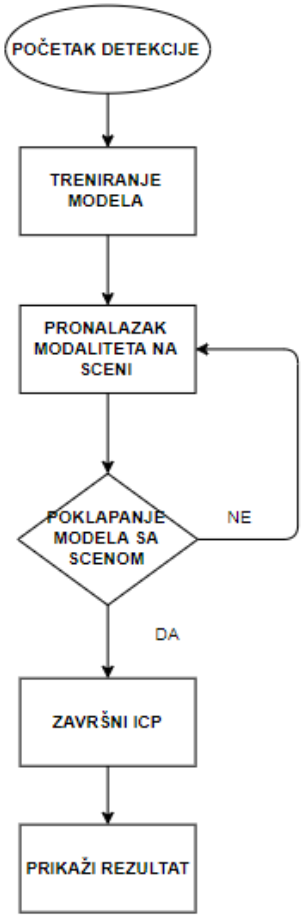
Slika 2.2 Prikaz modaliteta [2]

Jedinični vektori u smjeru normale površine objekta se nazivaju orijentirane točke. Orijentirane točke razmatranog oblaka točaka mogu se prikazati parovima (*p*,*n*) gdje je *p* vektor pozicije točke u odnosu na referentni koordinatni sustav, a *n* jedinični vektor koji predstavlja pridruženu normalu.

LINE-MOD implementacija u ORK-u (eng. Object Recognition Kitchen)[3] nudi mogućnost odabira vrste modaliteta koji će se koristiti kod treniranja i prepoznavanja objekata, odnosno može se odabrati samo izračun modaliteta na RGB slici, izračun modaliteta samo na dubinskoj slici ili kombinacija svih modaliteta. U procesu treniranja potrebno je vršiti izračun modaliteta za model iz više pogleda i različitih udaljenosti. U tu svrhu je u ORK-u implementiran generator

pogleda modela s virtualnom kamerom koji nudi mogućnost odabira količine pogleda koji će se stvoriti te odabir skale, odnosno udaljenosti s kojih će se uzimati pogled. Za što bolju detekciju objekata potrebno je odabrati veći broj pogleda na kojima će se računati modalitet.

Kod faze prepoznavanja objekata vrši se izračun modaliteta za cijelu scenu te se ti modaliteti uspoređuju s modalitetima dobivenim u trening fazi. Koristi se metoda hodajućeg prozora za detekciju objekta te zbog tog pristupa ova metoda spada u grupu metoda usporedbe s predloškom. Ukoliko je potrebno vršiti prepoznavanje u stvarnom vremenu treba paziti na utjecaj količine pogleda na resurse računala. Na slici 2.3 je prikaz dijagrama toka LINE-MOD detektora.



Slika 2.3 Dijagram toka LINE-MOD detektora

Određivanje orijentacije kod LINE-MOD metode nije vrlo precizno te je zato potrebno implementirati i ICP (eng. Iterative Closest Point) kako bi se minimizirala razlika između modela i scene.

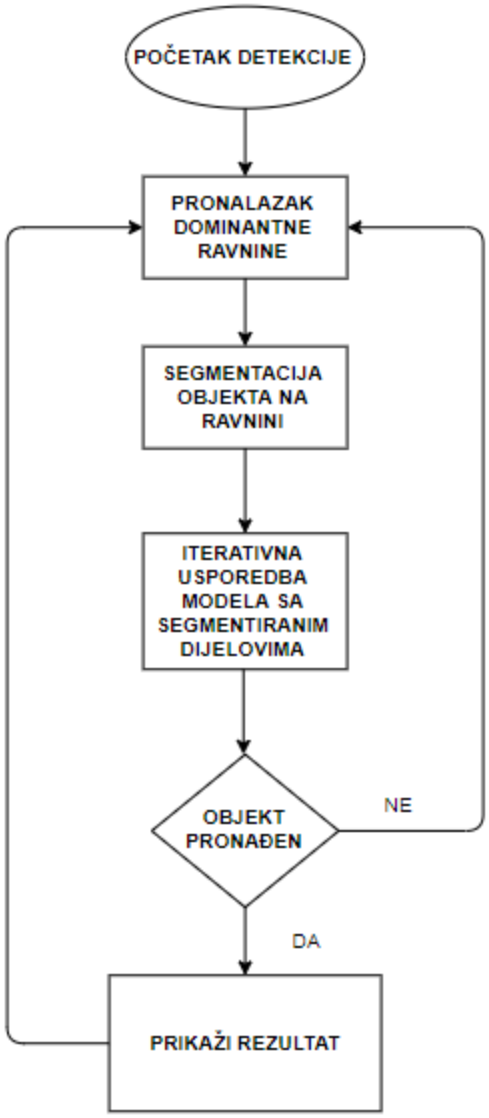
Prednosti ove metode su mogućnost rada u stvarnom vremenu te rad u zakrčenim scenama, a glavni nedostatak je slabije prepoznavanje djelomično zaklonjenih objekata.

2.4. Tabletop metoda prepoznavanja objekata

Tabletop metoda je razvijena na osnovu tabletop_object_detector ROS paketa [4]. Metoda se sastoji od dva dijela, a to su dio za prepoznavanje dominantne ravnine i dio za prepoznavanje objekata. Metoda zahtjeva 3D model objekta kojega se želi prepoznati te scenu u obliku oblaka točaka.

Prva faza je prepoznavanje dominantne ravnine na sceni. Ravnina se pronalazi RANSAC (eng. Random Sample Consensus) metodom. Dominantna ravnina se dobiva na način da se na sceni nasumično odaberu točke kroz koje se računa jednadžba ravnine. Nakon toga se provjerava koliko točaka pripada toj ravnini te se kao dominantna ravnina odabire ona koja sadržava najviše točaka. Kada je pronađena dominantna ravnina, vrši se segmentacija svih objekata koji se nalaze na toj ravnini.

Kod faze prepoznavanja objekata, vrši se iterativna usporedba modela sa segmentiranim dijelovima oblaka točaka te se tako pronalazi objekt od interesa. Kao rezultat metode dobije se translacija i orijentacija objekta u odnosu na koordinatni sustav kamere. Ova metoda je vrlo jednostavna za implementaciju te vrlo brza i pogodna za prepoznavanje objekata u stvarnom vremenu. Na slici 2.4 je prikazan dijagram toka tabletop detektora.



Slika 2.4 Dijagram toka tabletop detektora

Nedostatak metode su ograničenja koja se moraju poštovati kako bi prepoznavanje bilo uspješno. Objekt koji se prepoznaje mora biti rotacijski simetričan po z-osi te se uzima u obzir da je objekt postavljen tako da mu je z-os usmjerena prema gore, odnosno z-os objekta mora biti okomita na dominantnu ravninu. Zbog ovih ograničenja detekcija objekta ima samo dva stupnja slobode, odnosno translacija po z-osi i orijentacija su fiksirani. Ovo ograničenje smanjuje primjenu ove metode na rotacijsko simetrične predmete kao što su limenke, boce, čaše i sl.

3. PROGRAMSKO RJEŠENJE

U programskom rješenju su implementirane dvije metode prepoznavanja objekta od interesa. Implementirana je hand-eye kalibracija robotskog manipulatora i kamere te je izrađen program za demonstraciju cijelog sustava. Sve je implementirano u jednom ROS radnom okruženju.

Postoji više programskih rješenja i algoritama koji se mogu implementirati za vlastite potrebe prepoznavanje objekata od interesa. Jedno od ovih rješenja je Object Recognition Kitchen (ORK) projekt kojega je pokrenula tvrtka Willow Garage. Ovaj projekt sadrži više metoda za detekciju krutih objekata te je izgrađen na osnovi ecto programskog okvira. Ecto je c++/python programski okvir za organiziranje matematičkih izračuna u usmjerene acikličke grafove [5]. Osim samih metoda prepoznavanja ORK pruža i rješenja za 3D rekonstrukciju modela te spremanje i dohvaćanje modela iz baze podataka.

ORK se sastoji od četiri metode za prepoznavanje objekta:

- LINE-MODE
- Tabletop
- Textured Object Detection (TOD)
- Transparent Objects

3.1. Korištene tehnologije i alati

Programsko rješenje je implementirano na Ubuntu distribuciji linux operacijskog sustava. Programi su pisani u C++ i python programskom jeziku koristeći se OpenCv i PCL bibliotekama.

3.1.1. ROS

Robot Operating System (ROS) je programski okvir koji sadrži više alata i biblioteka za pisanje softvera prvenstveno u robotici [6]. ROS osigurava apstrakciju sklopovlja, kontrolu uređaja na niskoj razini, prosljeđivanje poruka između različitih procesa, upravljanje paketima i drugo. Prvenstveno je namijenjen za rad u Linux okruženju, ali podržava i rad u Windows-ima.

Svi procesi pokrenuti u ROS-u su predstavljeni kao čvorovi koji međusobno komuniciraju slanjem poruka preko određene teme. Izdavač može kreirati temu te ju oglasiti kako bi se pretplatnici mogli pretplatiti i primati informacije koje se prenose putem poruka.

Osim komunikacije pomoću tema i poruka, postoji i dvosmjerna komunikacija pomoću servisa. Za komunikaciju pomoću servisa, potreban je server i klijent. Klijent šalje upit prema serveru koji obrađuje primljenu informaciju i šalje odgovor.

ROS programi se nalaze u ROS korisničkim radnim prostorima te su organizirani u pakete. Nema ograničenja u broju radnih prostora ili broju paketa u jednom radnom prostoru. ROS podržava C++ i python programske jezike za izradu paketa.

3.1.2. Ubuntu

Ubuntu je jedna od distribucija Linux operacijskog sustava. Ubuntu je građen od mnogo dijelova koji se nazivaju softverski paketi te osim operacijskog sustava sadrže i aplikacijski softver za različite namjene. Projekt Ubuntu vodi tvrtka Canonical Ltd. koju je osnovao Mark Shuttleworth, no na projektu sudjeluju mnogi neovisni programeri i sudionici [7]

3.1.3. C++

C++ je objektno orijentirani programski jezik opće namjene i srednje razine. Nastao je kao proširenje C programskog jezika. Jedna od prednosti C++ programskog jezika je kolekcija predefiniranih klasa i tipova podataka. C++ sadrži više operatora usporedbe, aritmetike, manipulacije s bitovima te logike [8].

3.1.4. Python

Python je programski jezik opće namjene i visoke razine. Po automatskoj alokaciji memorije, Python je sličan jezicima kao što su Perl ili Ruby. U Pythonu je dopušteno objektno orijentirano, strukturno i aspektno programiranje te to čini Python sve popularnijim jezikom [9].

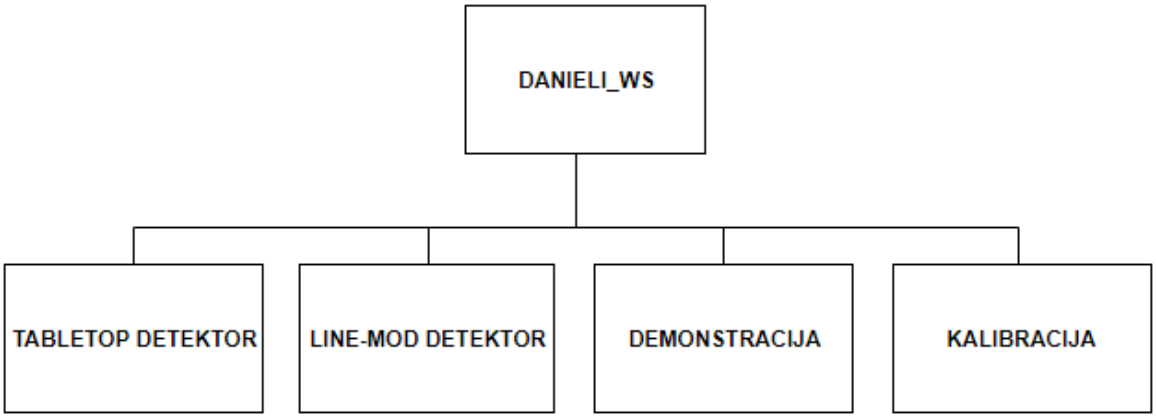
3.1.5. OpenCV i PCL

OpenCV (Open Source Computer Vision Library) je biblioteka otvorenog koda koja sadrži funkcije za izradu programa u području računalnog vida i strojnog učenja. Podržava rad na Windows, Linux, Android i Mac OS operacijskim sustavima te podržava C++, Python, Java i Matlab programske jezike [10].

PCL (Point Cloud Library) je biblioteka otvorenog koda koja sadrži algoritme za obradu oblaka točaka. Biblioteka je napisana u C++ programskom jeziku te nudi različite funkcije za manipulaciju s 3D podacima [11].

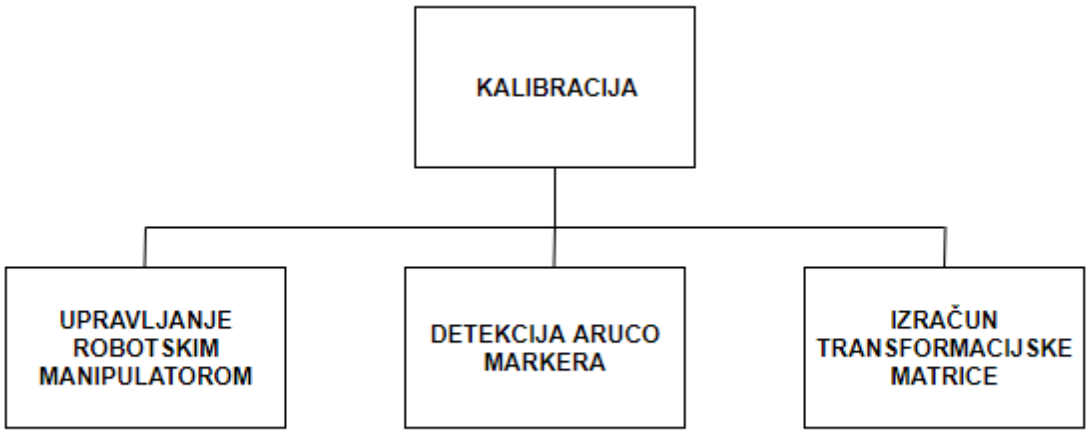
3.2. Struktura programskog rješenja

Cijelo programsko rješenje je implementirano u jedno ROS radno okruženje te se sastoji od nekoliko glavnih dijelova. Na slici 3.1 je blokovski prikaz dijelova programskog rješenja unutar jednog ROS radnog okruženja. Svi dijelovi se nalaze *u Danieli_ws* radnom okruženju. Datoteke za pokretanje demonstracije i kalibracije se nalaze u *Danieli_ws/src/moveit/doc/move_group_interface* direktoriju. Detektori se nalaze u *Danieli_ws/src* direktoriju.



Slika 3.1 Blokovski prikaz strukture programskog rješenja

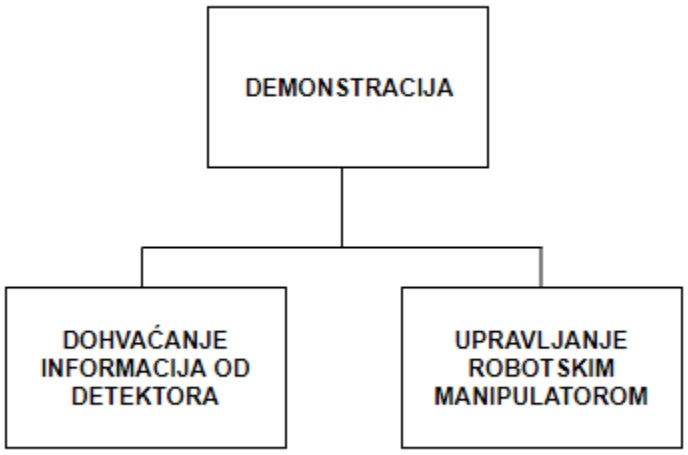
Kalibracijski dio se sastoji od tri cjeline povezane u jedno rješenje te se može pokrenuti neovisno o ostalim dijelovima. Jedan dio je zadužen za upravljanje robotskim manipulatorom, drugi za detekciju ArUco markera te treći dio za izračun transformacijske matrice koja opisuje položaj kamere i robota. Za kreiranje putanje robotskog manipulatora koristi se MoveIt ROS paket [6]. Prikaz strukture kalibracijskog dijela programa može se vidjeti na slici 3.2.



Slika 3.2 Prikaz strukture kalibracijskog dijela programa

U poglavlju 3.6 je opisan princip rada te način pokretanja ovog dijela programskog rješenja. U poglavlju 3.6.1 je opisan princip rada detekcije ArUco markera.

Demonstracijski dio se sastoji od programa za upravljanje robotskim manipulatorom koji omogućuje kretnju vrha alata robotskog manipulatora po površini prepoznatog objekta uz pomoć informacija dobivenih od tabletop detektora. Za pokretanje demonstracijskog dijela programa potreban je rad tabletop detektora, odnosno prije demonstracije je potrebno pokrenuti detektor. Na slici 3.3 je prikaz strukture demonstracijskog dijela programskog rješenja.



Slika 3.3 Prikaz strukture demonstracijskog dijela programa

Način rada i pokretanje tabletop detektora je prikazano u poglavlju 3.5. Način rada i pokretanje demonstracijskog dijela programskog rješenja je prikazano u poglavlju 3.7.

3.3. Kalibracija kamere

Prije nego li se krene u daljnju izradu programskog rješenja, potrebno je kalibrirati kameru koja će se koristiti za prepoznavanje objekata. Kao rješenje kalibracije dobivaju se intristični parametri kamere. Kako se Orbbec Astra S kamera sastoji od RGB i IR (infracrvene) kamere, potrebno je napraviti kalibraciju obje kamere.

Kako bi Orbbec Astra s radila u ROS okruženju, potrebno je instalirati ROS omot za Orbbec Astru. Postupak za instaliranje omota i svih ovisnosti je vidljiv u prilogu 1.

Na slici 3.4 se vidi način pokretanja kamere. Nakon pokretanja kamere objavljuju se teme na koje se potrebno pretplatiti kako bi dobili podatke od kamere.

```
$ roslaunch astra_launch astra.launch
```

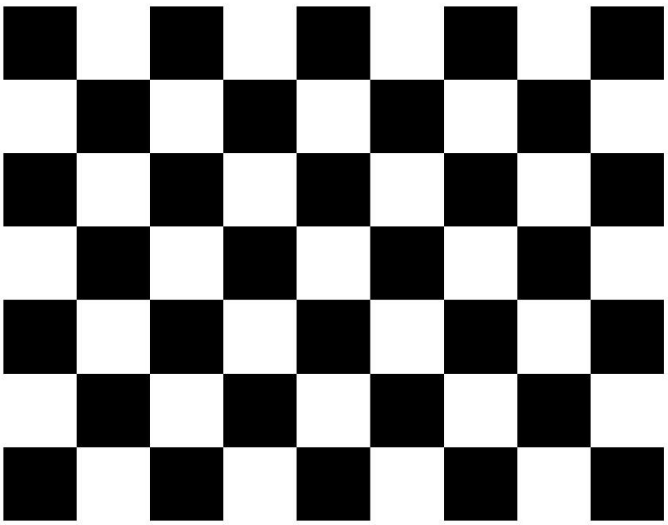
Slika 3.4 Pokretanje kamere

Kalibracija je izvedena uz pomoć *camera_calibration* ROS paketa [12]. U prilogu 2 je prikazano kako instalirati potrebne ovisnosti, a na slici 3.5 je prikaz pokretanja kalibracije.

```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108
 image:=/camera/rgb/image_raw camera:=/camera
```

Slika 3.4 Pokretanje ROS paketa za kalibraciju kamere

Uz pomoć opcije *–size* se određuje dimenzija kalibracijskog panela vidljivog na slici 3.6, dok se uz pomoć opcije *–square* određuje dimenzija kvadratića. Na kraju je još potrebno programu predati naziv teme na kojoj se objavljuje RGB slika dobivena kamerom.



Slika 3.5 Kalibracijski panel

Nakon pokretanja programa, na zaslonu računala se prikazuje slika snimljena kamerom (slika 3.6). Potrebno je postaviti kalibracijski panel ispred kamere te ga pomicati u svim osima i zakretati. Program će prikazati kada je odabran dovoljan broj slika panela te će omogućiti pritisak tipke *calibrate*. Nakon pritiska tipke prikazat će se rezultati kalibracije. Pritiskom na tipku *save*, rezultati će se spremiti na računalo u yml formatu. Pri pokretanju kamere, ovi podaci se objavljuju preko teme */camera_info*

Slika 3.6 Kalibracija kamere

Kalibracija IR kamere se izvodi na sličan način. Potrebno je pokrenuti program za kalibraciju (slika 3.8). Nakon toga slijedi isti postupak kao kod kalibracije RGB kamere.

```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108
image:=/camera/ir/image_raw camera:=/camera/ir
```

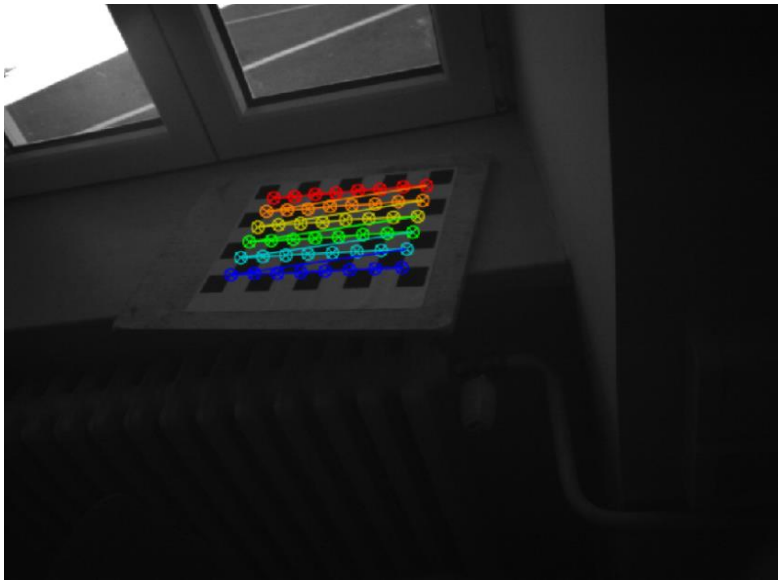
Slika 3.7 Pokretanje kalibraccije IR kamere

Kod kalibracije IR kamere pojavljuju se smetnje na slici. Kako bi se taj problem riješio, potrebno je prekriti IR projektor Orbbec Astra kamere. Projektor je označen crvenim krugom na slici 3.9.

Slika 3.8 Orbbec Astra kamera s označenim projektorom

Na slikama 3.10 i 3.11 se vidi kako prekrivanje projektora utječe na kvalitetu slike. Dok je projektor prekriven, potrebno je imati nekakav drugi izvor IR svjetla kao što je sunčevo svijetlo.

Slika 3.9 Slika dok je IR projektor otkriven



Slika 3.10 Slika dok je IR projektor prekriven

3.4. LINE-MOD metoda

Kako bi se koristila LINE-MOD metoda potrebno je instalirati ORK (Object Recognition Kitchen) projekt. Postupak instalacije je vidljiv u prilogu 3.

Za sve metode prepoznavanja objekta implementirane u ovom programskom rješenju, potrebno je imati bazu podataka koja sadrži modele predmeta u *.stl* fromatu. Koristi se Couch Db baza podataka sa svojim web sučeljem. Instalacija baze podataka i njenog web sučelja je prikazana u prilogu 4.

Dodavanje modela u bazu podataka sastoji se od dva dijela, prvo se radi novi objekt u bazi te mu se onda pridružuje model u *stl* formatu. Nakon pokretanja prve naredbe, program vraća ID objekta u obliku niza znakova. Taj ID je potrebno upisati u drugu naredbu na mjesto OBJECT_ID. Ovaj postupak je vidljiv na slici 3.12.

```
$rosrun object_recognition_core object_add.py -n Naziv objekta -d "Kratki opis objekta" --commit
$rosrun object_recognition_core mesh_add.py OBJECT_ID object.stl --commit
```

Slika 3.12 Dodavanje modela u bazu podataka

Kada je model dodan u bazu podataka može se krenuti s treningom modela za LINE-MOD detekciju. Za trening je potrebno prilagoditi nekoliko parametara u ovisnosti o obliku i veličini predmeta za detekciju. Na slici 3.13 su parametri koji su postavljeni za trening kutije dimenzija 170x85x50 milimetara Ovi parametri se upisuju u *linemod_train.cpp* datoteku u *src* direktoriju *linemod* ROS paketa.

```
params.declare(&Trainer::param_n_points_, "renderer_n_points", "Renderer parameter: the number of points on the sphere.", 150);
params.declare(&Trainer::param_angle_step_, "renderer_angle_step", "Renderer parameter: the angle step sampling in degrees.", 10);
params.declare(&Trainer::param_radius_min_, "renderer_radius_min", "Renderer parameter: the minimum scale sampling.", 1.0);
params.declare(&Trainer::param_radius_max_, "renderer_radius_max", "Renderer parameter: the maximum scale sampling.", 1.8);
params.declare(&Trainer::param_radius_step_, "renderer_radius_step", "Renderer parameter: the step scale sampling.", 0.4);
params.declare(&Trainer::param_width_, "renderer_width", "Renderer parameter: the image width.", 640);
params.declare(&Trainer::param_height_, "renderer_height", "Renderer parameter: the image height.", 480);
params.declare(&Trainer::param_focal_length_x_, "renderer_focal_length_x", "Renderer parameter: the focal length x.", 525.0);
params.declare(&Trainer::param_focal_length_y_, "renderer_focal_length_y", "Renderer parameter: the focal length y.", 525.0);
params.declare(&Trainer::param_near_, "renderer_near", "Renderer parameter: near distance.", 0.1);
params.declare(&Trainer::param_far_, "renderer_far", "Renderer parameter: far distance.", 1000.0);
```

Slika 3.13 Parametri za LINE-MOD trening

S ovim parametrima se postavlja udaljenost virtualne kamere od modela, kut pomicanja kamere, visina i širina slike i dr.

Kada su obavljene sve konfiguracije, pokreće se trening dio LINE-MOD detektora. Način pokretanja treninga se vidi na slici 3.14.

```
$rosrun object_recognition_core training -c 'putanja_do_linemod_paketa'/conf/training.ork
```

Slika 3.14 Pokretanje LINE-MOD treninga

Kada se odradi trening može se krenuti u detekciju objekta od interesa. Prije pokretanja detekcije potrebno je konfigurirati teme na koje se pretplaćuje detektor. Ovi parametri se nalaze u *conf* direktoriju ROS paketa u datoteci pod nazivom *detection.ros.ork.* Na slici 3.15 je prikazano sve što je potrebno dodati u datoteku.

```
source1:
  type: RosKinect
  module: 'object_recognition_ros.io'
  parameters:
    rgb_frame_id: 'camera_rgb_optical_frame'
    rgb_image_topic: '/camera/rgb/image_raw'
    rgb_camera_info: '/camera/rgb/camera_info'
    depth_frame_id: 'camera_depth_optical_frame'
    depth_image_topic: '/camera/depth_registered/image_raw'
    depth_camera_info: '/camera/depth_registered/camera_info'
```

Slika 3.15 Konfiguracija tema kamere za detekciju objekta

Isto tako potrebno je konfigurirati parametre za samu detekciju objekta. Ovi parametri također ovise o samom predmetu koji se želi detektirati. Parametri korišteni za detekciju kutije dimenzija 170x85x50 milimetara se vide na slici 3.16. Ovdje se može postaviti granična vrijednost nakon koje će se registrirati da je predmet prepoznat, može se uključiti ili isključiti vizualizacija, postaviti ICP granična vrijednost i dr. Ovi parametri se upisuju u datoteku *linemod_detect.cpp* koja se nalazi u *src* direktoriju *linemod* ROS paketa.

```
object_recognition_core::db::bases::declare_params_impl(params, "LINEMOD");
params.declare(&Detector::threshold_, "threshold", "Matching threshold, as a percentage", 93.0f);
params.declare(&Detector::visualize_, "visualize", "If True, visualize the output.", false);
params.declare(&Detector::use_rgb_, "use_rgb", "If True, use rgb-based detector.", true); //true
params.declare(&Detector::use_depth_, "use_depth", "If True, use depth-based detector.", true);
params.declare(&Detector::th_obj_dist_, "th_obj_dist", "Threshold on minimal distance between detected objects.", 0.6f); //0.04
params.declare(&Detector::verbose_, "verbose", "If True, print.", false);
params.declare(&Detector::depth_frame_id_, "depth_frame_id", "The depth camera frame id.", "camera_depth_optical_frame");
params.declare(&Detector::icp_dist_min_, "icp_dist_min", "", 0.06f);//0.06
params.declare(&Detector::px_match_min_, "px_match_min", "", 0.4f); //0.25
```

Slika 3.16 Parametri za detekciju objekta

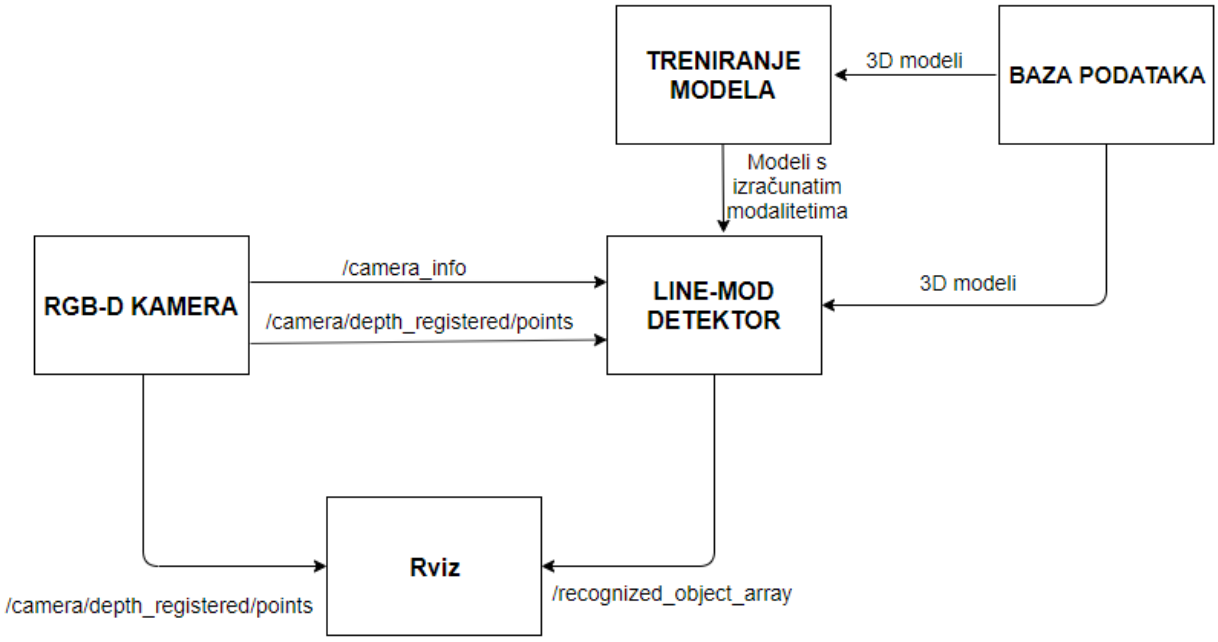
Kada je trening završen može se pokrenuti detekcija objekta. Na slici 3.17 se vidi način pokretanja detekcije objekta. Svaka naredba se pokreće u zasebnom terminalu.

```
$roslaunch astra_launch astra.launch
$rosrun rviz rviz
$rosrun rqt_reconfigure rqt_reconfigure
$rosrun object_recognition_core detection -c  'putanja_do_linemod_paketa'/conf/detection.ros.ork
```

Slika 3.17 Pokretanje detekcije objekta

U prozoru koji se otvori kod pokretanja *rqt_reconfigure* paketa, potrebno je otvoriti *camera/driver* dio te odabrati opciju *depth_registration.*

Na slici 3.18 je blokovski prikaz rada detektora s napisanim temama preko kojih komuniciraju razni dijelovi sustava. Blok RGB-D kamera predstavlja ROS omotač koji omogućuje korištenje kamere u ROS okruženju.



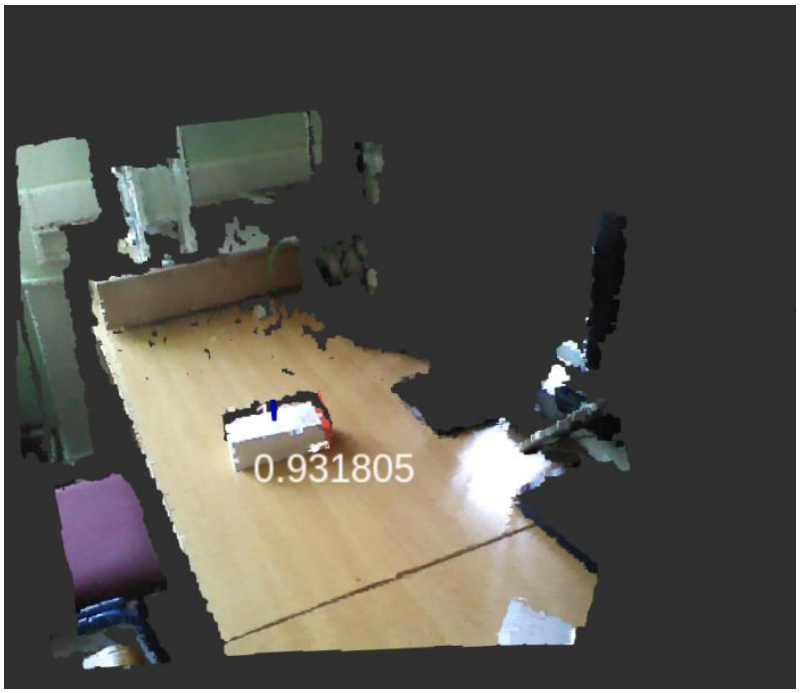
Slika 3.18 Blokovski prikaz LINE-MOD detektora

LINE-MOD detektor stvara temu *reognized_object_array* putem kojega se šalju poruke koje sadrže informaciju o detektiranom predmetu. Poruka sadrži informaciju o poziciji i orijentaciji prepoznatog predmeta, također možemo vidjeti koliko je algoritam siguran da je predmet prepoznat. Na slici 3.19 je detaljniji prikaz poruke koja se šalje spomenutom temom.

U ROS-u postoji unaprijed instaliran alat za vizualizaciju raznih tipova podataka pod nazivom rviz. Kako bi rviz mogao prikazati podatke, mora se pretplatiti na odgovarajuće teme. Za prikaz oblaka točaka, rviz se mora pretplatiti na temu *camera/depth_registered/points*, a za prikaz prepoznatog objekta na temu */recognized_object_array*. Na slici 3.20 je prikaz detekcije predmeta u rviz-u s označenim detektiranim predmetom.

```
pose:
  header:
    seq: 0
    stamp:
      secs: 1568997629
      nsecs: 892141096
    frame_id: "camera_rgb_optical_frame"
  pose:
    pose:
      position:
        x: -0.0904588252306
        y: -0.0291749536991
        z: 0.629553198814
      orientation:
        x: 0.882207632065
        y: 0.01795450598
        z: 0.033751104027
        w: 0.469306111336
```

Slika 3.19 Poruka koja se šalje putem teme recognized_object_array



Slika 3.20 Prikaz detekcije u rviz-u

3.5. Tabletop metoda

Tabletop metoda se također nalazi u ORK projektu. Također je implementirana u ROS okruženju kao zasebni ROS paket. Može se pokrenuti neovisno o LINE-MOD metodi.

Ova metoda prvo pronalazi dominantne ravnine, odnosno nalazi nekakvu ravnu podlogu na kojoj se može nalaziti objekt od interesa. Isto kao i u prethodnoj metodi, potrebno je dodati modele u bazu podataka. Postupak je opisan u poglavlju 3.4.

Prije pokretanja *tabletop* detektora, potrebno je konfigurirati teme koje će detektor koristiti za dobivanje podataka od kamere. Na slici 3.21 se vidi dio koji je potrebno dodati u *detection.object.ros.ork* datoteku koja se nalazi u *conf* direktoriju *tabletop* paketa.

```
parameters:
  rgb_frame_id: 'camera_rgb_optical_frame'
  rgb_image_topic: '/camera/rgb/image_raw'
  rgb_camera_info: '/camera/rgb/camera_info'
  depth_frame_id: 'camera_depth_optical_frame'
  depth_image_topic: '/camera/depth_registered/image_raw'
  depth_camera_info: '/camera/depth_registered/camera_info'
```

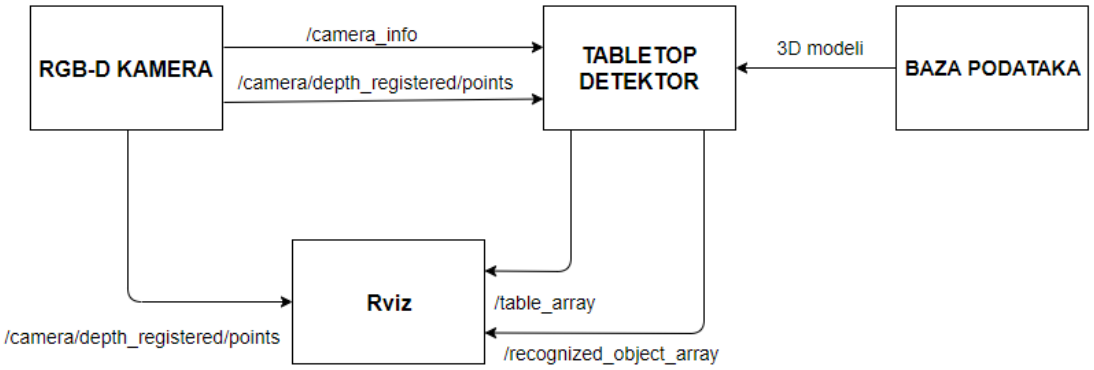
Slika 3.21 Konfiguracija tema za dobivanje podataka iz kamere

Na slici 3.22 je prikazan cijeli postupak pokretanja kamere i *tabeltop* detektora. Svaka naredba mora biti pokrenuta u zasebnom terminalu.

```
$roslaunch astra_launch astra.launch
$rosrun rviz rviz
$rosrun rqt_reconfigure rqt_reconfigure
$rosrun object_recognition_core detection -c 'putanja_do_tabletop_paketa'/conf/detection.table.ros.ork
$rosrun object_recognition_core detection -c 'putanja_do_tabletop_paketa'/conf/detection.object.ros.ork
```

Slika 3.22 Pokretanje tabletop detektora

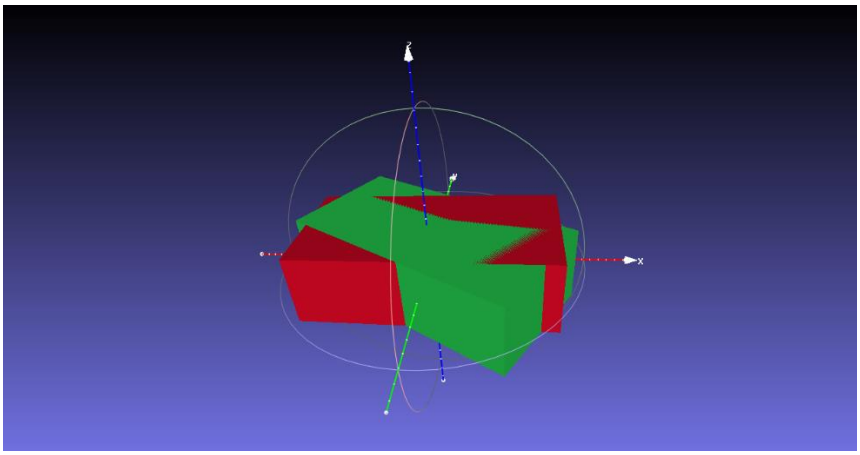
Na slici 3.23 je blokovski prikaz rada detektora s napisanim temama preko kojih komuniciraju razni dijelovi sustava.



Slika 3.23 Blokovski prikaz dijelova detektora

Kao što je objašnjeno u poglavlju 2.2, Tabletop metoda zahtjeva da objekt od interesa bude simetričan po z-osi. Ova metoda uvijek daje jednaku poziciju i orijentaciju kada je predmet prepoznat. Jedino u čemu se razlikuju prepoznati predmeti je vrijednost koja govori koliko je dobro predmet prepoznat.

Predmet kojega želimo prepoznavati ne zadovoljava uvjet simetričnosti te je bilo potrebno smisliti način kako bi se problem riješio. Ova metoda omogućava postavljanje više modela u bazu podataka te se tijekom detekcije prikazuje model koji je prepoznat s najvećom sigurnošću. To se iskoristilo kako bi se riješio problem simetričnosti po z-osi. U bazu podataka se postavilo 90 modela istog predmeta gdje je svaki model zakrenut za 2 stupnja po z-osi. Na slici 3.24 je prikaz modela koji je poravnat po osima (crveni model), te modela koji je zakrenut za 30 stupnjeva po z-osi (zeleni model).



Slika 3.24 Prikaz modela za detekciju

Svaki model u bazi podataka ima svoj jedinstveni ID uz pomoć kojega možemo znati koliko je model zakrenut oko svoje z-osi. Kada je predmet prepoznat moramo izračunati njegovu orijentaciju u odnosu na koordinatni sustav kamere, odnosno moramo ga zakrenuti za određeni broj stupnjeva po z-osi. Funkcija za računanje orijentacije se nalazi u *Demonstration.cpp* datoteci u *Danieli_ws/src/moveit/doc/move_group_interface* direktoriju te je dio demonstracijskog dijela programskog rješenja. Formula 3-1 prikazuje izračun vrijednosti za koju moramo rotirati predmet oko njegove z-osi.

$$rot = (i_{id} * 2 - 90) * \frac{\pi}{180} \tag{3-1}$$

Gdje je:

rot – vrijednost za koju moramo rotirati predmet oko z-osi u radijanima

i_{id} – redni broj (ID) objekta u bazi podataka

Kada izračunamo vrijednost stupnjeva za rotaciju, potrebno je stvoriti transformacijsku matricu pomoću koje ćemo napraviti transformaciju objekta. Kako objekt moramo samo rotirati, translacijski dio postavljamo na vrijednost nula.

$$T_1 = \begin{bmatrix} \cos(rot) & -\sin(\text{rot}) & 0 & 0 \\ \sin(\text{rot}) & \cos(\text{rot}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3-2}$$

Tabletop detektor preko teme *recognized_object_array* šalje informacije o prepoznatom objektu. Translacijski dio šalje u obliku x, y, z koordinata, dok rotacijski dio šalje u obliku kvanterniona. Iz tih podataka je potrebno stvoriti transformacijsku matricu, a postupak stvaranja se vidi na slici 3.25. Ulaz u funkciju su x, y, z koordinate i kvaternioni, a funkcija vraća transformacijsku matricu.

```
MatrixXd TransfromationMatrixCalc(float x,float y,float z,float q_w,float q_x,float q_y,float q_z){
 MatrixXd T_Matrix(4,4);
 T_Matrix(0,0)=1 - 2 * q_y*q_y - 2 * q_z*q_z;
 T_Matrix(0,1)=2 * q_x*q_y - 2 * q_z*q_w;
 T_Matrix(0,2)=2 * q_x*q_z + 2 * q_y*q_w;
 T_Matrix(0,3)=x;
 T_Matrix(1,0)=2 * q_x*q_y + 2 * q_z*q_w;
 T_Matrix(1,1)=1 - 2 * q_x*q_x - 2 * q_z*q_z;
 T_Matrix(1,2)=2 * q_y*q_z - 2 * q_x*q_w;
 T_Matrix(1,3)=y;
 T_Matrix(2,0)=2 * q_x*q_z - 2 * q_y*q_w;
 T_Matrix(2,1)=2 * q_y*q_z + 2 * q_x*q_w;
 T_Matrix(2,2)=1 - 2 * q_x*q_x - 2 * q_y*q_y;
 T_Matrix(2,3)=z;
 T_Matrix(3,0)=0;
 T_Matrix(3,1)=0;
 T_Matrix(3,2)=0;
 T_Matrix(3,3)=1;
 return T_Matrix;
}
```

Slika 3.25 Funkcija za izračun transformacijske matrice

Kako bismo dobili stvarni položaj objekta u koordinatnom sustavu kamere potrebno je pomnožiti transformacijsku matricu dobivenu od tabletop detektora s transformacijskom matricom T_1 (3-3).

$$T_{OK} = T_T * T_1 \tag{3-3}$$

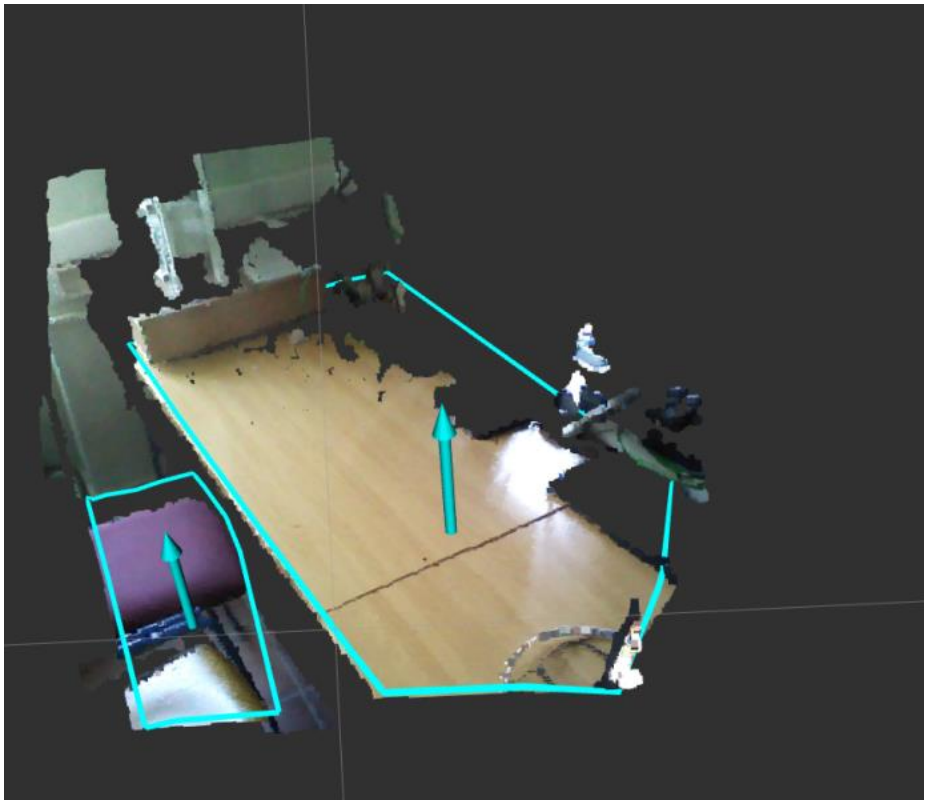
Gdje je:

T_{OK} – Transformacijska matrica koja nam govori gdje se objekt nalazi u odnosu na koordinatni sustav kamere

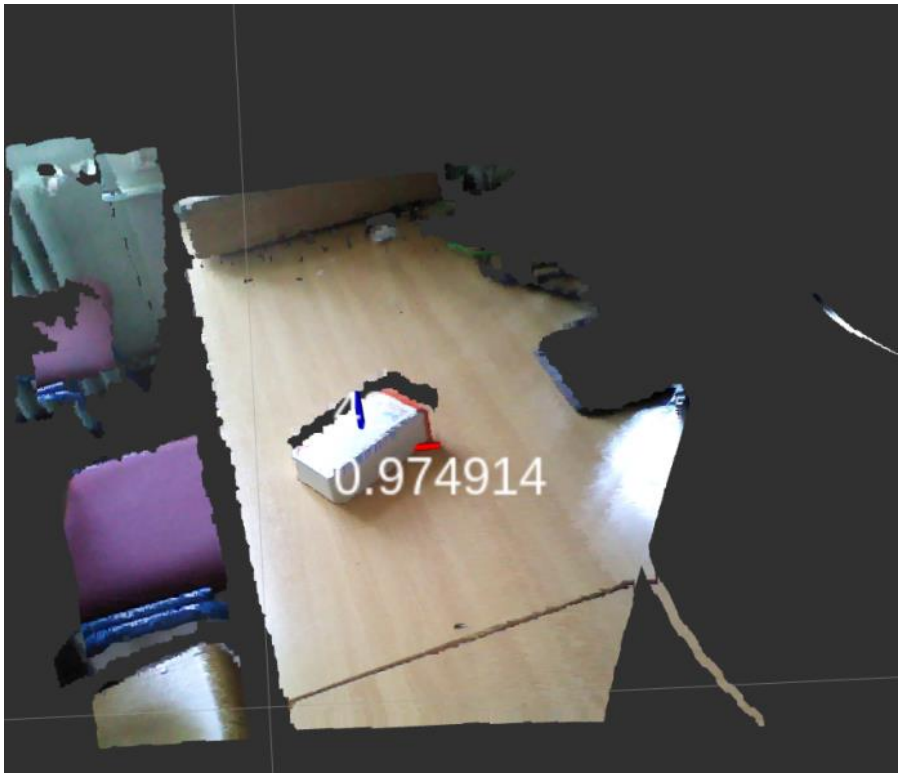
T_T – Transformacijska matrica dobivena uz pomoć informacija od tabletop detektora

T_1 – Transformacijska matrica za rotaciju predmeta oko z-osi

Nakon pokretanja tabletop detektora, u rviz-u se mogu vidjeti rezultati prepoznavanja ravnina i objekata. Kako bi se vidjele prepoznate ravnine mora se dodati *OrkTable* objekt u rviz-u (slika 3.26). Za vizualizaciju prepoznatog objekta u rviz se mora dodati *OrkObject* objekt (slika 3.27). Oba objekta se moraju pretplatiti na svoje teme, a to su *table_array* za prikaz ravnina te *recognized_object_array* za prikaz prepoznatih predmeta.



Slika 3.26 Prepoznate ravnine

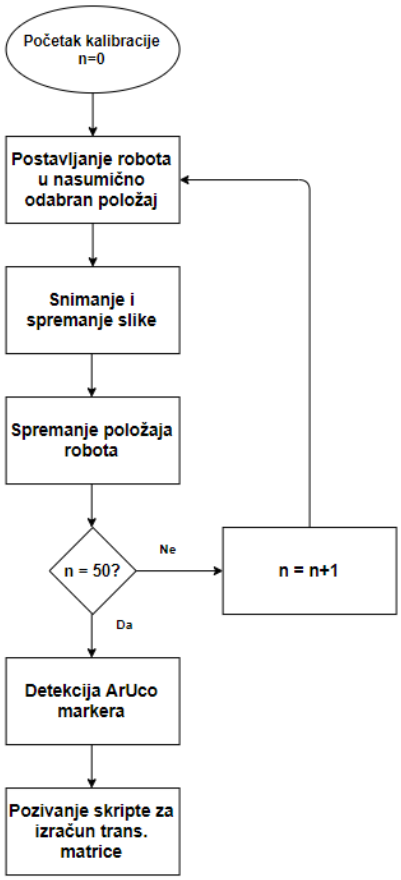


Slika 3.11 Prepoznati objekt

3.6. Hand-eye kalibracija

Kako bi robot mogao manipulirati s objektom koji je prepoznat uz pomoć kamere potrebno je znati gdje se objekt nalazi u odnosu na koordinatni sustav robota. Kod postupka prepoznavanja objekta položaj objekta je opisan u odnosu na referentni koordinatni sustav kamere. Kako bismo znali gdje je objekt u odnosu na koordinatni sustav robota, moramo znati gdje se nalazi kamera u odnosu na robota. Transformacijsku matricu koja opisuje odnos položaja robota i kamere dobijemo primjenom hand-eye kalibracije.

Programsko rješenje za hand-eye kalibraciju se sastoji od programa za detekciju ArUco markera, programa za pozicioniranje robotskog manipulatora te programa za izračun transformacijske matrice koja opisuje odnos između koordinatnog sustava šestog zgloba robota i kamere. Cijelo rješenje je napisano u ROS okviru te se pokreće kao ROS pokretačka datoteka. U poglavlju 3.6.1 je opisan postupak detekcije panela s ArUco markerima te spremanje dobivenih rezultata. Na slici 3.28 je dijagram toka kalibracijskog dijela programskog rješenja.



Slika 3.28 Dijagran toka kalibracijskog dijela programa

Na slici 3.29 je postupak pokretanja kalibracijskog dijela programa. Prva naredba pokreće kameru, dok se s drugom naredbom ostvaruje komunikacija robota i računala. Treću naredbu je potrebno pokrenuti u zasebnom terminalu u direktoriju */Danieli_ws/marker_python/Aruco_tracker/Slike.* Ova naredba omogućava spremanje slika za vrijeme kalibracije. S posljednjom naredbom se pokreće hand-eye kalibracija robota i kamere. Pokretačka datoteka *Calibration.launch* poziva skriptu za detekciju ArUco markera, skriptu za računanje transformacijske matrice te funkcije za pozicioniranje robotskog manipulatora.

```
$roslaunch astra_launch astra.launch

$roslaunch abb_irb2400_moveit_config moveit_planning_execution.launch sim:=false
 robot_ip:=172.16.106.185.

$rosrun image_view image_saver image:=camera/rgb/image_raw _save_all_image:=false
 _filename_format:=%d.%s __name:=image_saver

$roslaunch moveit Calibration.launch
```

Slika 3.29 Pokretanje programa za hand-eye kalibraciju

Algoritam za izračun transformacijske matrice je napisan u Matlab programskom jeziku te se pokreće uz pomoć GNU Octave programa otvorenog koda []. Kao ulazne parametre zahtijeva x, y i z koordinate svakog markera na kalibracijskom panelu te pripadajući položaj šestog zgloba robotskog manipulatora. Izlaz iz programa je transformacijska matrica koja opisuje odnos između koordinatnog sustava šestog zgloba robota i koordinatnog sustava kamere.

Kada se pokrene programsko rješenje za kalibraciju, robot se postavlja u položaj u kojemu je na kameri vidljiv cijeli kalibracijski panel s markerima. Na slici 3.30 je vidljiv jedan od položaja robota iznad kalibracijskog panela.

Slika 3.30 Jedan od položaja u postupku kalibracije

Nakon toga slijedi snimanje slike uz pomoć kamere te spremanje slike u zadani direktorij. Kada je slika spremljena robot ide u idući položaj te se ponavlja postupak spremanja slike. Svaki položaj

robota se sprema u istom obliku kao i položaji ArUco markera. U programu se može odrediti koliko puta će se ponoviti cijeli postupak. Kada je spremljena slika u zadnjem položaju, poziva se skripta za detekciju ArUco markera nakon koje se poziva skripta za izračun transformacijske matrice. Dobivena transformacijska matrica se sprema u tekstualnu datoteku kako bi se mogla koristiti u drugim programskim rješenjima.

Na slici 3.31 je prikaz dobivene transformacijske matrice. Iz nje se može vidjeti točan odnos između kamere i šestog zgloba robotskog manipulatora.

```
TCG =

   -0.9993    0.0058    0.0374    0.0058
   -0.0041   -0.9990    0.0454    0.0762
    0.0376    0.0452    0.9983    0.0651
         0         0         0    1.0000
```

Slika 3.31 Transformacijska matrica koja opisuje odnos robota i kamere

3.6.1. Detekcija ArUco markera

Za kalibraciju je bilo potrebno napraviti program za detekciju ArUco markera te određivanje njihovih pozicija (x, y, z koordinate). Ovo programsko rješenje je napisano u python programskom jeziku koristeći se OpenCv bibliotekom.

Potrebno je prepoznati više ArUco markera na jednom panelu te za svaki izračunati njegovu poziciju. Na slici 3.32 je prikaz panela s markerima.

Slika 3.32 Panel s ArUco markerima

Napisani program za detekciju markera zahtijeva da sve slike panela budu u istom direktoriju u *.jpg* formatu. Program prolazi kroz sve slike u direktoriju te za svaku sliku radi detekciju markera. Rezultat detekcije će koristiti program napisan u Matlab programskom jeziku pa su rezultati napisani u takvom obliku da se mogu spremiti u .m datoteku i učitati u Matlab program.

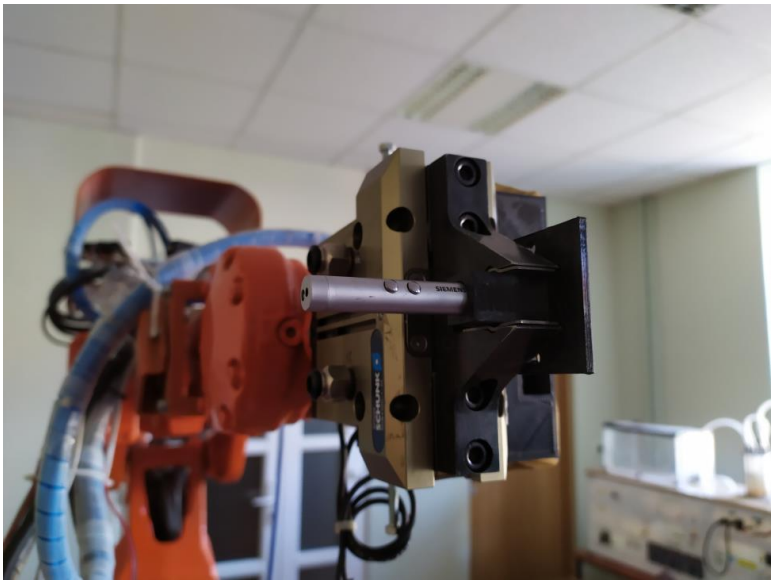
Na slici 3.33 je prikaz rezultata za jednu sliku panela. Prvi stupac označava redni broj markera na panelu, dok ostali stupci predstavljaju x, y i z koordinate markera

```
xyz(:,:,1) =   [1 -0.0402206576561 -0.0921413372111 0.454751908596;
                2 0.0262144268486 -0.0739214192527 0.452674261292;
                3 0.0916623536673 -0.0561323993795 0.454377442099;
                4 -0.0573647607545 -0.021666016328 0.448943208475;
                5 0.00904863848798 -0.00458012704588 0.445758641456;
                6 0.0736660073632 0.012854620729 0.444678874555;
                7 -0.0737670495198 0.0469793581473 0.437876100021;
                8 -0.00788114582684 0.0645467817931 0.444639864424;
                9 0.0573720282287 0.0819812195488 0.445947920504;
                ];
```

Slika 3.33 Rezultat detekcije ArUco markera

3.7. Demonstracija sustava kamere i robota

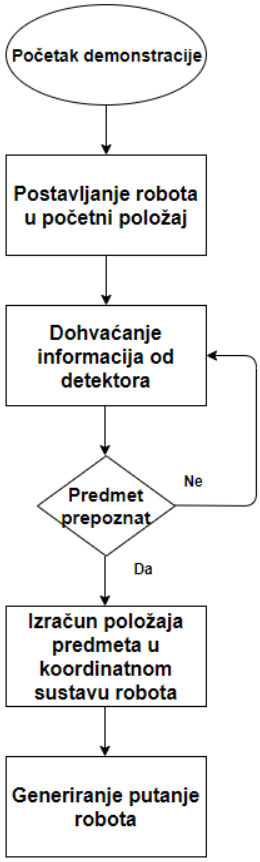
Demonstracija sustava ostvaruje gibanje vrha alata robotskog manipulatora po rubovima gornjeg dijela predmeta koji se nalazi ispred robota. Na hvataljku robotskog manipulatora je postavljen laser koji predstavlja vrh alata (slika 3.34).



Slika 3.34 Hvataljka s postavljenim laserom

Za rad demonstracijskog dijela programa potreban je tabletop detektor koji se ne poziva pokretanjem demonstracijske pokretačke datoteke. Tabletop je potrebno pokrenuti kao zasebni čvor. U demonstracijskom dijelu se stvara pretplatnik koji se pretplaćuje na temu *recognized_object_array* te tako prima informacije od detektora. Način pokretanja demonstracije je prikazan na slici 3.37, a način dobivanja informacija o prepoznatom objektu je prikazan na slici 3.36.

Na slici 3.35 je dijagram toka demonstracijskog dijela programskog rješenja.



Slika 3.35 Dijagram toka demonstracijskog dijela programa

Informacijama koje se šalju u obliku poruke preko teme u ROS-u, može se vrlo lako pristupiti u python i C++ programskom jeziku. Na slici 3.36 se može vidjeti primjer funkcije koja dohvaća podatke preko teme *recognized_object_array* te uzima u obzir samo najbolje prepoznati objekt. Ova funkcija se poziva svaki put kada dođe nova poruka s *recognized_object_array* teme.

```
void objectCallback(const object_recognition_msgs::RecognizedObjectArray objects_msg) {

  if((int)objects_msg.objects.size() > 0){
    int j=0;
    double conf=0;

    // pronađi najbolji predmet
    for (int i = 0; i < objects_msg.objects.size(); ++i) {
        if(objects_msg.objects[i].confidence > conf){
            conf= objects_msg.objects[i].confidence;
            j=i;
        }
    }
      object_qx = objects_msg.objects[j].pose.pose.pose.orientation.x;
      object_qy = objects_msg.objects[j].pose.pose.pose.orientation.y;
      object_qz = objects_msg.objects[j].pose.pose.pose.orientation.z;
      object_qw = objects_msg.objects[j].pose.pose.pose.orientation.w;
      object_x=objects_msg.objects[j].pose.pose.pose.position.x;
      object_y=objects_msg.objects[j].pose.pose.pose.position.y;
      object_z=objects_msg.objects[j].pose.pose.pose.position.z;
      id = objects_msg.objects[j].type.key.c_str();
  }
}
```

Slika 3.36 Dohvaćanje informacija koje se šalju putem teme recognized_object_array

Izraz (3-4) prikazuje način izračuna položaja objekta u odnosu na bazni koordinatni sustav robota.

$$T_{OB} = T_{6B} * T_{K6} * T_{OK} \tag{3-4}$$

Gdje je:

T_{OB} – Transformacijska matrica koja opisuje položaj objekta u odnosu na bazni koordinatni sustav robota

T_{6B} – Transformacijska matrica koja opisuje položaj 6. zgloba robota u odnosu na bazni koordinatni sustav. Ova matrica se računa uz pomoć informacija o trenutnom položaju robota. Pozicija robota je zadana s x, y i z koordinatama, a orijentacija je zadana pomoću kvaterniona. Transformacijska matrica se onda računa postupkom opisanim na slici 3.25.

T_{K6} – Transformacijska matrica koja opisuje položaj kamere u odnosu na 6. zglob robota. Ova matrica je rezultat hand-eye kalibracije.

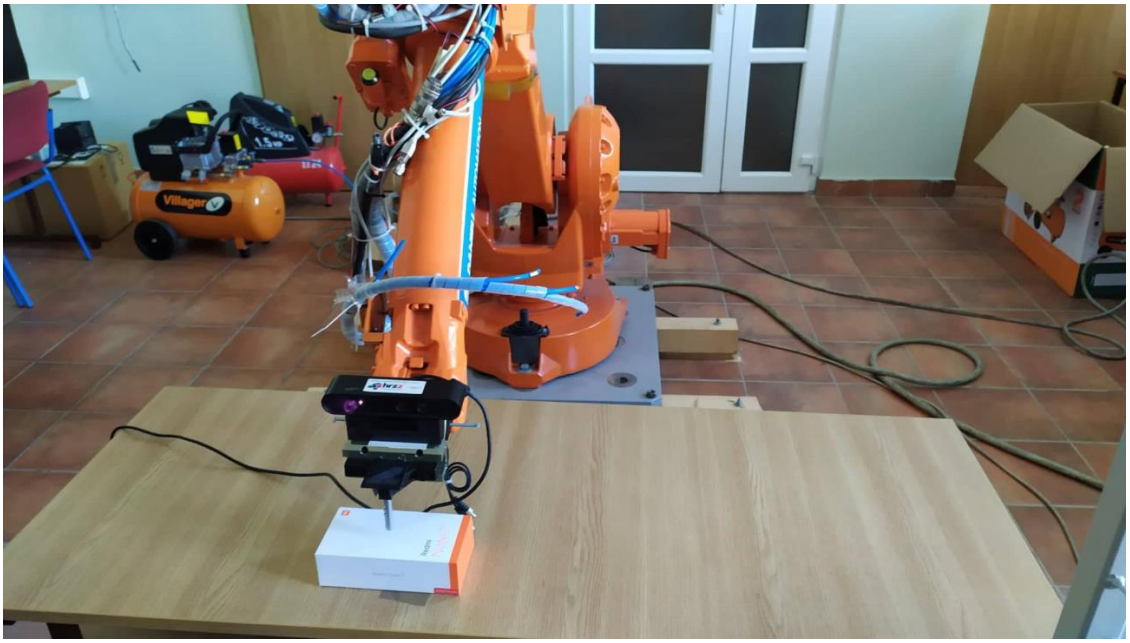
T_{OK} – Transformacijska matrica dobivena kao rezultat prepoznavanja objekta tabletop metodom

Na slici 3.37 je postupak pokretanja demonstracijskog dijela programa. Prva naredba pokreće kameru dok druga naredba pokreće tabletop detektor. Nakon toga je potrebno uspostaviti komunikaciju robota i računala. Zadnjom naredbom se pokreće demonstracija u kojoj se dohvaća pozicija predmeta te se vrh alata robota giba po rubovima gornje strane predmeta.

```
$roslaunch astra_launch astra.launch

$rosrun object_recognition_core detection -c src/tabletop/conf/detection.object.ros.ork

$roslaunch abb_irb2400_moveit_config moveit_planning_execution.launch sim:=false
 robot_ip:=172.16.106.185.

$roslaunch moveit Demonstration.launch
```

Slika 3.37 Postupak pokretanja demonstracijskog dijela programa

Na slici 3.38 je prikaz rada sustava za vrijeme demonstracije.



Slika 3.12 Prikaz gibanja vrha alata robota po površini predmeta

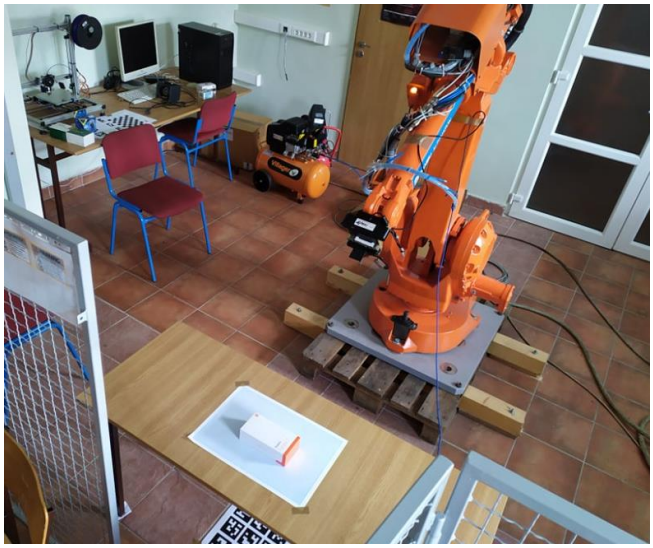
4. TESTIRANJE SUSTAVA

Sustav za testiranje se sastoji od industrijskog robota ABB IRB 2400L na kojega je postavljena Orbbec Astra S RGB-D kamera. Na slici 4.1 se vidi kamera postavljena na manipulatoru.

Slika 4.1 Robotski manipulator s postavljenom kamerom

Prije testiranja sustava sa softverom za prepoznavanje objekata, potrebno je napraviti hand-eye kalibraciju pomoću koje određujemo odnos koordinatnog sustava kamere i šestog zgloba robota. Način rada i pokretanje hand-eye kalibracije je opisano u poglavlju 3.6.

Pri pokretanju softvera za testiranje, robotski manipulator se postavlja u položaj u kojemu kamera ima pogled na stol (slika 4.2).

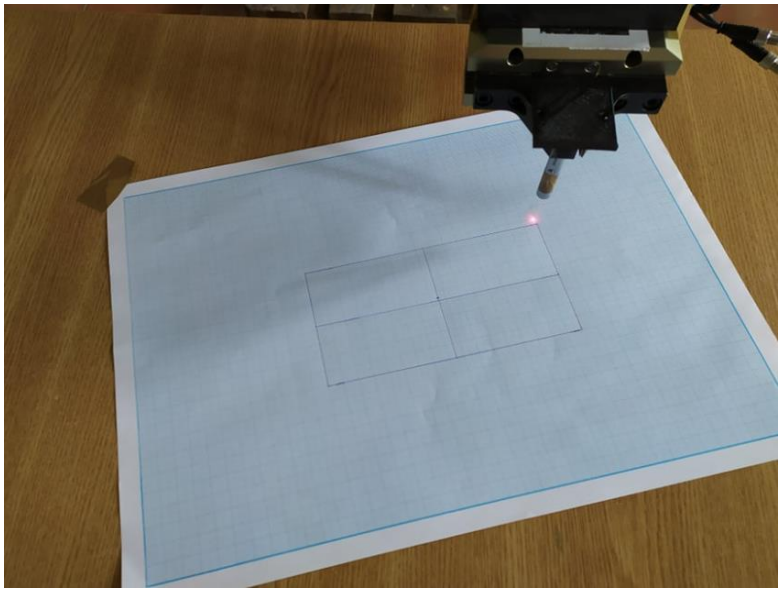


Slika 4.2 Početni položaj manipulatora

Kako bi se precizno izmjerila razlika između položaja stvarnog objekta i prepoznatog, objekt se iscrtava na milimetarskom papiru te se postavlja na stol ispred robota (slika 4.3) Kada se objekt prepozna, uklanja se s milimetarskog papira.

Slika 4.3 Objekt na milimetarskom papiru

Na alat robotskog manipulatora je postavljen laser kako bi se provjerilo koliko je dobro objekt prepoznat bez fizičkog kontakta robota i stola. Robot se dovodi na četiri položaja iznad objekta tako da laser osvjetljava milimetarski papir. Jedan od tih položaja je vidljiv na slici 4.4.



Slika 4.4 Jedan od položaja robota iznad stola

Na svakom položaju se na milimetarskom papiru označi mjesto osvjetljavanja lasera. Postupak se ponavlja deset puta te se rezultati testiranja iznose u poglavlju 4.2.

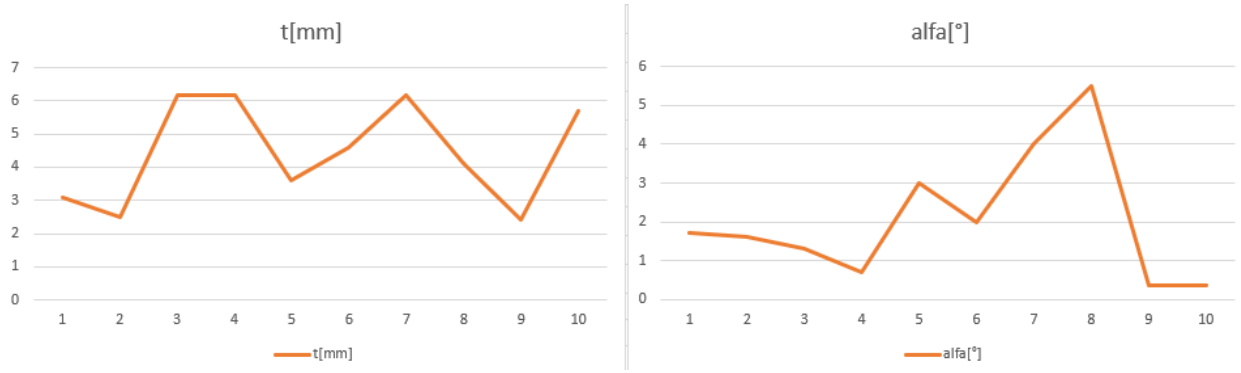
4.1. Rezultat testiranja sustava

U tablici 4.1 su prikazani rezultati testiranja sustava nakon deset mjerenja. U prvom stupcu je označen redni broj mjerenja, u drugom stupcu je greška u translaciji, dok je u trećem stupcu greška u orijentaciji predmeta. Prosječna greška u translaciji iznosi 4.3 milimetra, dok je prosječna greška u orijentaciji 2 stupnja.

Tablica 4.1 Rezultat mjerenja pogreške u translaciji i rotaciji

Mjerenje	t[mm]	alfa[°]
1.	3.1	1.7
2.	2.5	1.6
3.	6.15	1.3
4.	6.15	0.7
5.	3.6	3
6.	4.6	2
7.	6.15	4
8.	4.1	5.5
9.	2.4	0.35
10.	5.7	0.35

Na slici 4.1 je grafički prikaz grešaka u određivanju orijentacije i translacije.



Za gibanje alata po površini predmeta greška mora biti manja od dobivene. Ovakav sustav je primjenjiv u zadacima gdje robot manipulira s objektima uz pomoć hvataljke, odnosno u zadacima dohvaćanja i premještanja objekata.

5. ZAKLJUČAK

U ovom radu su prikazane metode prepoznavanja objekta od interesa u svrhu stvaranja putanje gibanja robota po površini prepoznatog predmeta. Implementirane su dvije metode prepoznavanja od kojih je tabletop metoda pokazala puno bolje rezultate. Prikazano je rješenje za korištenje tabletop metode kod predmeta koji nisu rotacijsko simetrični oko z-osi. Metode prepoznavanja su se povezale u jedno zajedničko rješenje s programom za pozicioniranje robotskog manipulatora. Na kraju je cijeli sustav testiran kako bi se izmjerila prosječna greška cijelog sustava u određivanju translacije i orijentacije predmeta. Uzimajući u obzir dobivenu grešku, sustav se ne može koristiti za precizno gibanje vrha alata po površini predmeta. Sustav se može koristiti kod zadaća gdje robot mora manipulirati s predmetima uz pomoć hvataljke. Greška u određivanju položaja predmeta od interesa se može smanjiti na više načina. Što se tiče poboljšanja opreme, može se koristiti preciznija RGB-D kamera, a što se tiče poboljšanja softvera, može se postaviti veći broj modela predmeta u bazu podataka kako bi se povećala rezolucija detekcije predmeta. Čak i s navedenim poboljšanjima, metoda i dalje zahtjeva da objekt bude položen na ravnu podlogu, odnosno zahtjeva da z-os objekta bude okomita na podlogu. Ovo ograničenje dodatno smanjuje mogućnost primjene sustava.

6. LITERATURA

[1] - Stefan Holzer, Stefan Hinterstoißer, Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes, http://www.stefan-hinterstoisser.com/papers/hinterstoisser2012accv.pdf

[2] - Stefan Holzer,Stefan Hinterstoißer, Object Detection – LINEMOD, http://www.pointclouds.org/assets/cvpr2012/PCL_Object_Detection_LINEMOD.pdf

[3] - Object Recognition Kitchen (ORK), https://wg-perception.github.io/object_recognition_core/

[4] - Tabletop metoda, http://wiki.ros.org/tabletop_object_detector

[5] - Ecto programski okvir, http://plasmodic.github.io/ecto/

[6] - ROS, https://www.ros.org/

[7] - The story of Ubuntu, https://ubuntu.com/about

[8] - C++ Programming Language, https://www.techopedia.com/definition/26184/c-programming-language

[9] - Why Python?, https://www.linuxjournal.com/article/3882

[10] - About OpenCv, https://opencv.org/about/

[11] - Point Cloud Library, http://pointclouds.org/

[12] - camera_caibration paket, http://wiki.ros.org/camera_calibration

[13] - GNU Octave, https://www.gnu.org/software/octave/

7. SAŽETAK

U ovom radu istražene su i implementirane dvije metode prepoznavanja objekta od interesa u svrhu određivanja putanje robotskog manipulatora po površini objekta. Implementirane su LINE-MOD i tabletop metode prepoznavanja objekta. Objekt od interesa je zadan u obliku mreže trokuta. Metode za prepoznavanje objekata su implementirane u ROS okruženje te koriste informacije dobivene od RGB-D kamere za prepoznavanje. Kako bi se mogla planirati putanja robota po površini objekta, potrebno je znati gdje se nalazi objekt u odnosu na koordinatni sustav robota. U tu svrhu je u programskom rješenju implementirana hand-eye kalibracija. Na kraju je provedeno testiranje sustava koji se sastoji od industrijskog robota ABB IRB 2400L i Orbbec Astra S kamere kako bi se odredila točnost cijelog sustava.

Ključne riječi: ROS, detekcija objekta, 3D kamera, hand-eye kalibracija, c++, python

8. ABSTRACT

In this thesis, two methods for object recognition are examined and implemented for the purpose of determining the path of a robotic manipulator end effector along the object surface. LINE-MOD and tabletop methods are implemented. The developed program requires the object of interest to be provided in the form of a triangular mesh. Methods for object recognition are implemented in ROS framework and they use information obtained from an RGB-D camera. In order to be able to plan the path of a robot end effector along the surface of an object, it is necessary to know where the object is located relative to the coordinate system of the robot. For that purpose, hand-eye calibration was implemented in the software solution. Finally, testing of the ABB IRB 2400L robot and Orbbec Astra S camera system was performed to determine the accuracy of the system.

Keywords: ROS, object recognition, 3D camera, hand-eye calibration, c++, python

ŽIVOTOPIS

Dominik Živčić rođen je 17. Svibnja 1995. godine u Vinkovcima. Osnovnoškolsko obrazovanje je završio u Mirkovcima 2010. godine te je iste godine upisao tehničku školu „Ruđera Boškovića“ u Vinkovcima. Maturirao je 2014. godine te upisao preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. Preddiplomski studij završava 2017. godine te upisuje diplomski studij računarstva, izborni blok procesno računarstvo.

Prilog 1 – Instalacija ROS omota za Orbbec Astru

```
$ sudo apt install ros-kinetic-rgbd-launch ros-kinetic-libuvc
$ sudo apt install ros-kinetic-libuvc-camera ros-kinetic-libuvc-ros
$ sudo apt-get install ros-kinetic-astra-camera
$ sudo apt-get install ros-kinetic-astra-launch
```

Prilog 2 – Instalacija ROS paketa za kalibraciju kamere

```
$ rosdep install camera_calibration
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108 image:=/camera/image_raw camera:=/camera
```

Prilog 3 – Instalacija ORK (Object Recognition Kitchen)

```
$sudo apt-get install ros-kinetic-object-recognition-*
$sudo apt-get install libopenni-dev ros-kinetic-catkin ros-kinetic--ecto*
$sudo apt-get install ros-kinetic--opencv-candidate ros-kinetic--moveit-msgs
$source /opt/ros/kinetic/setup.sh
$git clone http://github.com/wg-perception/object_recognition_core
$git clone http://github.com/wg-perception/linemod
$git clone http://github.com/wg-perception/tabletop
$git clone http://github.com/wg-perception/ork_renderer
$git clone http://github.com/wg-perception/object_recognition_msgs
$git clone http://github.com/wg-perception/object_recognition_ros
$git clone http://github.com/wg-perception/object_recognition_ros_visualization
$catkin_make
```

Prilog 4 – Instalacija baze podataka

```
$sudo apt-get install couchdb
$sudo pip install -U couchapp
```