Hvatanje pravokutnih predmeta robotskom rukom pomoću računalnog vida

Bošnjak, Andrej

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

Sveučilišni studij

Hvatanje pravokutnih predmeta robotskom rukom pomoću računalnog vida

Završni rad

Andrej Bošnjak

Osijek, 2022.

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

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 13.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Andrej Bošnjak
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4323, 22.07.2019.
OIB Pristupnika:	88114391471
Mentor:	Prof.dr.sc. Robert Cupec
Sumentor:	Dr. sc. Petra Pejić
Sumentor iz tvrtke:	
Naslov završnog rada:	Hvatanje pravokutnih predmeta robotskom rukom pomoću računalnog vida
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak završnog rad:	Izraditi računalni program za hvatanje pravokutnih objekata robotskom rukom UR5, koji koristi 3D kameru za određivanje položaja i dimenzija objekata te planira operaciju hvatanja za objekte različite veličine. Izrađeni program ispitati odgovarajućim pokusima. Tema rezervirana za: Andrej Bošnjak Sumentor: Petra Pejić
Prijedlog ocjene završnog 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 od strane mentora:	13.09.2022.
Datum potvrde ocjene od strane Odbora:	21.09.2022.

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 22.09.2022.

Ime i prezime studenta:	Andrej Bošnjak
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4323, 22.07.2019.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Hvatanje pravokutnih predmeta robotskom rukom pomoću računalnog vida**

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

i sumentora Dr. sc. Petra Pejić

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

Hvatanje pravokutnih predmeta robotskom rukom je posebni slučaj robotske manipulacije. Robotska manipulacija predstavlja interakciju robota s njegovom okolinom s ciljem odrađivanja ponavljajućih i opasnih poslova koji zahtijevaju veliku preciznost u kontroliranim uvjetima. Takvi su kontrolirani uvjeti većinom industrijska postrojenja, gdje roboti nemaju veliku razinu samostalnosti. U cilju povećanja robotske samostalnosti koja bi omogućila veći opseg primjene robotske manipulacije, koristi se računalni vid za dinamičko prepoznavanje okoline.

Računalni vid je znanstvena i tehnološka disciplina koja se bavi izradom sustava za prepoznavanje objekata, određivanje dimenzije i položaja objekata, praćenja objekata i sl.[1]. Računalni vid omogućava robotima, autonomnim vozilima i drugim polu-autonomnim uređajima veću samostalnost u njihovom radu, npr. robot u skladištu, robotski usisavač u kućanstvu, sustav za polu-autonomnu i potpuno autonomnu vožnju i sl. U ovom se radu računalni vid koristi za prepoznavanje predmeta i određivanje njegove veličine i položaja u odnosu na robota u svrhu prosljeđivanja naredbi robotu za hvatanje i pomicanje tog istog predmeta.

Ovaj rad se sastoji od pet poglavlja. U drugom poglavlju su opisane robotska ruka, hvataljka, kamera i Robotski operacijski sustav (ROS). U trećem poglavlju opisana je metodologija sustava, tj. struktura čvorova u robotskom operacijskom sustavu i njihova međusobna komunikacija. Također, opisana je biblioteka korištena za prepoznavanje objekata. U četvrtom su poglavlju opisani pokusi i njihovi rezultati. Zaključak ovog rada prikazan je u petom poglavlju.

1.1. Zadatak završnog rada

Izraditi računalni program za hvatanje pravokutnih objekata robotskom rukom UR5, koji koristi 3D kameru za određivanje položaja i dimenzija objekata te planira operaciju hvatanja za objekte različite veličine. Izrađeni program ispitati odgovarajućim pokusima.

2. PREGLED PODRUČJA TEME

Ljudska mogućnost hvatanja, držanja i manipuliranja objektima uključuje ruke, osjetilo dodira i povratne informacije očiju i mišića te nam omogućuje kontrolirani stisak. Autori rada [2] daju pregled napretka postignut u robotici za oponašanje ovih funkcija. Sustavi su se razvijali od jednostavnih hvataljki u načinu rada štipanja u potpuno definiranim okruženjima do robota koji mogu identificirati, odabrati i manipulirati objektima iz nasumične kolekcije. Napredak u području uzrokovan je napretkom računalnog vida, mogućnostima računalne obrade i taktilnim materijalima koji daju povratnu informaciju robotu.

Značajan je broj istraživanja iz područja detekcije i klasifikacije objekata koja se temelje na strojnom učenju i neuronskim mrežama [3-4]. Međutim, za omogućavanje robotske manipulacije, osim uspješne detekcije, potrebna je informacija o 3D položaju predmeta u odnosu na robota [5-6].

U diplomskom radu [7] opisana je detekcija pravokutnih predmeta snimljenih RGB-D kamerom na temelju algoritama robotskog vida iz biblioteke RVL, koja je korištena i u ovom radu. U tom je diplomskom radu hvatanje predmeta dvoprstnom hvataljkom montiranom na SCARA robot ostvareno na temelju vizualnog servoinga. Slična metoda je korištena i u radu [8] u kojem je predstavljen algoritam za prepoznavanje pravokutnih predmeta i metoda za hvatanje takvih predmeta, također dvoprstnom robotskom hvataljkom. Planiranje hvatanja objekta temelji se na graničnim okvirima jednostavnih objekata detektiranih na RGB-D slici, koji pružaju dovoljno informacija za pozicioniranje alata robota, orijentaciju hvataljke i širinu otvaranja alata. Ova metoda ima nekoliko ograničenja: I) pretpostavlja se da alat ima samo jedan rotacijski stupanj slobode i da se objekti mogu uhvatiti samo odozgo, II) hvatanje je moguće samo za konveksne objekte te III) stabilno hvatanje nije zajamčeno čak ni za konveksne objekte.

Unatoč tome, klasa objekata na koje se predstavljena metoda može primijeniti je izrazito široka i učinkovitost ove metode je njena očigledna prednost u usporedbi s općenitijim, ali i složenijim pristupima. Iz tog je razloga i u ovom završnom radu opisano prepoznavanje pravokutnih predmeta, a to mogu biti svakodnevni predmeti iz kućanstva, kao što su manje kutije, spremnici u obliku kvadra, knjige itd. Razlika u odnosu na spomenute metode jest u primjeni 6-osnog kolaborativnog robotskog manipulatora i troprstne hvataljke za izvršavanje hvatanja pravokutnih predmeta.

Detekcija pravokutnih objekata se koristi i u logistici, kao što je prikazano u radu [9], jer su oni idealni za automatsku manipulaciju. Iz tog su razloga njihova detekcija, identifikacija i algoritam praćenja od velike važnosti. Okluzija i malo vidno polje ograničenja su sustava koji koriste jednu RGB-D kameru te se u [9] predlaže upotreba sustava više takvih kamera. Iako se na ovaj način dobiva slika objekta iz više pogleda, ovakvi su sustavi skupi.

3. DIJELOVI SUSTAVA

Za potrebe ovog završnog rada korištena je robotska ruka UR5, hvataljka Robotiq 3 finger gripper, kamera Intel RealSense LiDAR Camera L515 i Robotski operacijski sustav. U nastavku je svaki dio sustava detaljnije opisan.

3.1. UR5

Universal Robots su osnovali Esben Østergaard, Kasper Støy i Kristian Kassow 2005. godine u Odense, Danska. Njihov je tadašnji cilj bio omogućiti malim i srednjim poduzećima dostupnost robotske tehnologije s obzirom na to da su na tadašnjem tržištu prevladavali veliki roboti industrijske namjene. UR5 je prvi proizvod tvrtke Universal Robots [10].

UR5 je prilagodljivi i lagani kolaborativni robot opće namjene dizajniran za širok spektar primjena. Prednosti su mu prilagodljivost i svestranost. Kolaborativni roboti imaju dodatne senzore za rad s ljudima u neposrednoj blizini. UR5 roboti se vrlo lako programiraju u sučelju koje je prilagođeno korisniku.

UR5 se sastoji od šest osi od kojih se svaka može okrenuti za 360 stupnjeva. 6 osi čine:

- Baza
- Rame
- Lakat
- Zglob 1
- Zglob 2
- Zglob 3

U tablici 3.1. su prikazane specifikacije UR5 robota, a na slici 3.1. je prikazan razmatrani UR5.

Tablica 3.1 Specifikacije modela UR5

Naziv modela	**UR5**
Masa	18.4 kg
Nosivost	5 kg
Domet	850 mm
Okretljivost zglobova	360°
Promjer baze	Ø149 mm

Potrošnja energije	200-570 W
Konekcija	Profinet EtherNet/IP, USB 2.0, USB 3.0
Programska podrška	Polyscope grafičko korisničko sučelje, MoveIt paket

Slika 3.1 Universal Robots UR5

3.2. Robotiq 3 finger gripper

Robotiq hvataljka s tri prsta idealna je za naprednu proizvodnju i robotsko istraživanje. Prilagođava se predmetu hvatanja, tako da se korisnik može usredotočiti na zadatak, a ne na hvatanje. Jednostavno se ugrađuje na UR robotske ruke, a programiranje same hvataljke je lako i intuitivno [11].

Hvataljka ima četiri načina hvatanja:

- Način štipanja
- Široki način rada
- Način rada sa škarama
- Osnovni način rada

Uz nezavisno upravljanje silom, položajem i brzinom za svaki prst i 4 načina hvatanja, ova hvataljka može uhvatiti predmete različitih oblika. Njene specifikacije dane su u tablici 3.2, a slika hvataljke s tri prsta je dana na slici 3.2.

Tablica 3.2 Specifikacije hvataljke sa 3 prsta

Naziv modela	**3 finger gripper**
Otvaranje	0-155 mm
Masa	2.3 kg
Promjer objekta za obuhvat	20-155 mm
Maksimalna nosivost (široki način rada)	10 kg
Maksimalna nosivost (način štipanja)	2.5 kg
Sila hvata (način štipanja)	30-70 N

Slika 3.2 Robotiq 3-finger gripper

3.3. Intel RealSense LiDAR Camera L515

Intel RealSense nudi višenamjenske dubinske kamere, počevši od D400 serije dizajnirane za jednostavno postavljanje i prenosivost te za unutarnju ili vanjsku upotrebu s infracrvenom stereo tehnologijom za postizanje dubinske slike. Postoji i naprednija L500 serija koja koristi solid state LiDAR tehnologiju za unutarnju upotrebu koja donosi dubinske i RGB slike u visokoj razlučivosti i točnosti. Također, Intel RealSense nudi i svoj paket za razvoj programa koji je otvorenog koda [12].

Intel RealSense LiDAR Camera L515 prvi je produkt s novom LiDAR tehnologijom. Budući da se svi izračuni dubine izvode na uređaju, on također ima niske zahtjeve za računanje u drugim dijelovima sustava, što rezultira istinskom neovisnošću od platforme. S obzirom na male dimenzije ove kamere, vrlo se lako integrira u sustav robota postavljanjem na krajnji zglob robotske ruke. Tehničke specifikacije su prikazane u tablici 3.3, a razmatrana kamera na slici 3.3.

Tablica 3.3 Specifikacije Intel RealSense LiDAR Camera L515

Naziv modela	**LiDAR Camera L515**
Domet	0.25 m – 9 m
RGB	2MP RGB Camera
Masa	100 g
Snaga	Do 3.5 W
Vidno polje	70°H × 43°V × 55°D
Rezolucija RGB slike	Do 30 FPS, 1920 × 1080 (XGA)
Rezolucija dubinske slike	Do 30 FPS, 1024 × 768 (FHD)
Dimenzije	61 mm × 25 mm
Povezivanje	USB 3.1
Programska podrška	Intel RealSense SDK

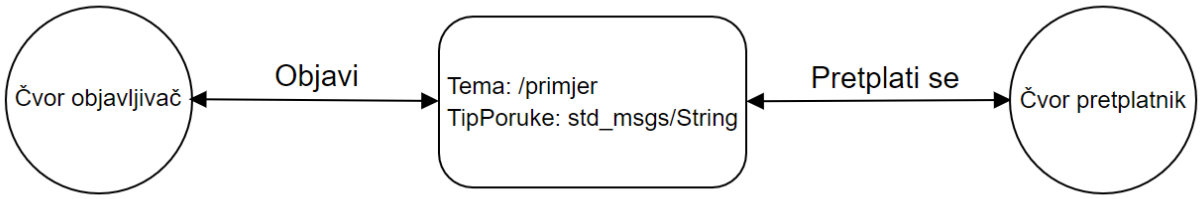
Slika 3.3 Intel RealSense LiDAR Camera L515

3.4. Robotski operacijski sustav

Robotski operacijski sustav (ROS) skup je softverskih biblioteka, alata i programskih okvira koji pomažu pri izradi robotskih aplikacija. Razvijen je 2007. godine u laboratoriju za umjetnu inteligenciju na sveučilištu Stanford. Održavan je od strane Open Source Robotics Foundation (OSRF) i u potpunosti je otvorenog koda [13].

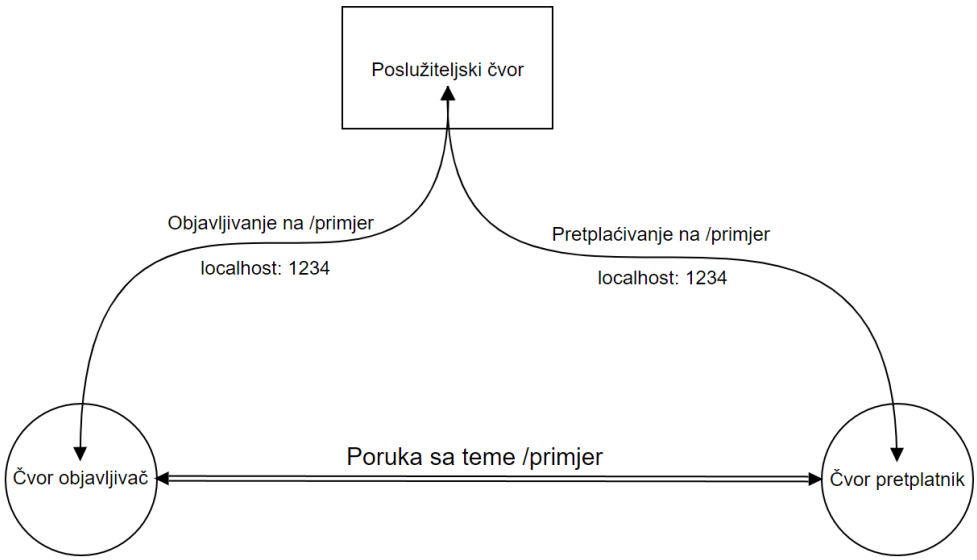
Osnovna funkcionalnost ROS-a je omogućavanje korisnicima prilagodbu konfiguracije određenih alata, biblioteka i paketa za njihovu potrebu. U ROS-u su već ugrađeni mnogi algoritmi potrebni za rad s robotom, stoga ih nije potrebno samostalno razvijati. Službeno podržani programski jezici za nadogradnju i razvijanje novih paketa i alata su Python i C++. S obzirom na to da je filozofija ROS-a otvoren kod i postoji velika zajednica, paketi i biblioteke koje su drugi korisnici sami razvili i/ili nadogradili međusobno se dijele. Osnovna funkcionalnost ROS-a proširena je alatima koji omogućuju vizualizaciju i pohranu podataka, jednostavnu navigaciju strukturom ROS paketa i stvaranje skripti za automatizaciju kompleksnih konfiguracija. Jedan od takvih alata je RViz, trodimenzionalni vizualizator koji se koristi za vizualizaciju robota i okruženja u kojem radi. Osim toga, važan alat je i catkin, sustav za izgradnju paketa u ROS-u. Temelji se na CMake, softveru otvorenog koda za izgradnju softvera i pakiranje istoga s metodom neovisnom o prevoditelju.

ROS procesi su predstavljeni kao čvorovi (engl. *nodes*) koji međusobno komuniciraju preko tema (engl. *topics*). S time je potaknuto da se programski kod razdvaja u komponente koje se odvojeno pokreću alatima unutar ROS-a kao što su rosrun za pokretanje jednog čvora unutar jednog paketa ili roslaunch za pokretanje jednog ili više čvorova iz jednog ili više paketa odjednom. Sadržaj koji se šalje preko tema je u obliku poruka (engl. *messages*). Poruke mogu biti bilo kakvog sadržaja, od običnih stringova i informacija o stanju do podataka o senzoru, naredbama za upravljanje motorom robota i sl. Da bi neki čvor mogao poslati poruku na temu, taj čvor mora postati izdavač (engl. *publisher*) na tu temu i objaviti poruku, a ako čvor želi uzeti i pročitati tu poruku, mora se pretplatiti (engl. *subscriber*) na tu temu. Model objave/pretplate je anoniman: čvorovi međusobno ne znaju tko je još pretplaćen na čvor i tko sve objavljuje na određenu temu. Pojednostavljeni prikaz čvorova i tema je vidljiv na slici 3.4.



Slika 3.4 Pojednostavljeni prikaz čvorova i tema

Kako bi čvorovi mogli međusobno komunicirati, mora se uspostaviti peer-to-peer veza između njih. To omogućuje virtualni server koji nadgleda sve čvorove zvani poslužiteljski čvor (engl. *ROS master*). Poslužiteljski čvor ne obrađuje podatke, već samo prati informacije o tome koji čvorovi su pokrenuti, koji su pretplaćeni na koje teme, koji čvorovi objavljuju na koje teme i njihove IP adrese. Pojednostavljeni prikaz koncepta poslužiteljskog čvora je prikazan na slici 3.5.



Slika 3.5 Pojednostavljeni prikaz koncepta poslužiteljskog čvora

Svi čvorovi moraju se izgraditi unutar nekog paketa u ROS-u. Za kratka programska rješenja najlakše je stvoriti novi paket, napisati kod u C++-u ili Python-u te izgraditi paket s već spomenutim softverom CMake, pri tome ispuniti potrebne datoteke CMakeLists i Packages kako bi se mogle koristiti sve vanjske biblioteke uključene u programski kod. Nakon izgradnje moguće je pokrenuti čvorove s naredbama rosrun ili roslaunch.

Jedna od mana ROS-a je nekompatibilnost njegovog softvera sa starijim verzijama robota i njihovih softvera pa tako zadatci čija rješenja nisu skalabilna često se ne mogu pokrenuti s novijom inačicom ROS-a. S obzirom da velik broj biblioteka ovisi o otvorenom kodu, glavne ROS klijentske biblioteke usmjerene su prema operacijskim sustavima temeljenim na Unix-u. Za te klijentske biblioteke Linux je naveden kao „Podržan“ dok su ostali operacijski sustavi „Eksperimentalni“ i održavani su od strane zajednice.

4. Programsko rješenje

ROS omogućuje povezivanje svih dijelova sustava da rade kao jedna cjelina. Svaki dio sustava se pokreće kao čvor koji će komunicirati s ostalim čvorovima preko tema. U prvom dijelu ovog poglavlja bit će detaljnije opisan rad robotskog vida, a u drugom detaljnije o svakom čvoru i programska rješenja za pojedine dijelove sustava.

4.1. Prepoznavanje objekata

Detekcija pravokutnih objekata ostvarena je metodom opisanom u nastavku. Metoda kao ulaznu informaciju koristi dubinsku sliku snimljenu 3D kamerom. Metoda se sastoji od detekcije ravninskih segmenata, generiranja hipoteza na temelju parova ravninskih segmenata te izračunavanja optimalnog položaja i dimenzija pravokutnog objekta na temelju najvjerojatnije hipoteze.

Ulazna dubinska slika se prvo pretvara u mrežu trokuta tako da se 3D točke koje odgovaraju susjednim pikselima dubinske slike povezuju u trokute. Nadalje, dobivena se mreža trokuta segmentira na ravninske segmente. Ravninski segmenti su približno koplanarni povezani podskupovi skupa točaka ulazne dubinske slike. Pod povezanošću skupa točaka podrazumijeva se da za svake dvije točke skupa postoji niz točaka tog skupa takav da su bilo koja dva elementa tog niza susjedne točke. Dvije točke su susjedne ako čine vrhove jednog od trokuta mreže trokuta.

Zatim se generiraju hipoteze o pravokutnim objektima na sceni na temelju parova ravninskih segmenata. Od svakog para ravninskih segmenata koji zadovoljava sljedeće uvjete generira se jedna hipoteza. Prvi je uvjet da je kut između normala ravninskih segmenata je manji od 45°. Drugi je uvjet

$$n^T\left(p'-p\right)<0 \quad \wedge \quad n'^T\left(p-p'\right)<0$$

gdje su n i n' normale razmatranih ravninskih segmenata, a p i p' njihovi centroidi.

Definira se koordinatni sustav hipotetskog objekta tako da mu je os x paralelna s normalom prvog ravninskog segmenta, a os y okomita na normale oba ravninska segmenta. Zatim se određuje najmanji kvadar čiji su bridovi paralelni s osima koordinatnog sustava hipotetskog objekta, a koji obuhvaća sve točke oba ravninska segmenta. Taj kvadar predstavlja početnu hipotezu, koja se onda popravlja pomoću ICP-algoritma (engl. *Iterative Closest Point*, ICP) koji određuje optimalnu poziciju, orijentaciju i dimenzije kvadra takve da je suma udaljenosti između točaka na površini tog kvadra i njima najbližih točaka scene minimalna.

Opisana metoda implementirana je u okviru programske biblioteke Robot Vision Library (RVL) razvijene na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek

4.2. Struktura čvorova u ROS-u

Struktura čvorova u ROS-u za potrebe ovog rada zamišljena je tako da kamera na input snimi slike i objavi ih na teme. Robotski vid se pretplaćuje na teme i uzima slike, obrađuje ih i vraća koordinatni sustav objekta u odnosu na kameru, točnije, vraća transformacijsku matricu objekta. Nadalje, taj isti program objavljuje transformacijsku matricu na teme. Program za robotsku manipulaciju se pretplaćuje na te teme i povlači transformacijsku matricu. S tim koordinatnim sustavom i ostalim poznatim koordinatnim sustavima kamere, alata i baze, računa se točan položaj objekta u odnosu na koordinatni sustav baze. Program za robotsku manipulaciju računa sve potrebne koordinate i položaje alata, ruke i hvataljke te daje naredbe robotu da uhvati predmet, podigne ga i spusti na neku drugu lokaciju.

4.2.1. Stvaranje novog radnog prostora i ROS paketa

Za lakše snalaženje po svim paketima, svi paketi koji se koriste se stavljaju u jedan radni prostor (engl. *workspace*). Radni prostor se stvara tako da se u terminalu izvede ovaj niz naredbi: [14]

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Primjer 4.1 Naredbe za stvaranje novog radnog prostora

Ako je sve prošlo u redu, trebala bi se stvoriti još dva dodatna poddirektorija *devel* i *build*. U direktorij *src* se prebacuju svi potrebni paketi za manipulaciju robotom, hvataljkom i kamerom. Paket u kojem se nalazi programsko rješenje za računalni vid i robotsku manipulaciju se stavlja unutar novog paketa. Novi paket se stvara i izgrađuje tako da se u terminalu pokrene sljedeći niz naredbi: [15]

```
cd ~/catkin_ws/src
catkin_create_pkg robot_vision std_msgs rospy roscpp
catkin_make
. ~/catkin_ws/devel/setup.bash
```

Drugi argument naredbe catkin_create_pkg predstavlja naziv paketa. Nakon što je paket stvoren, mora se urediti ovisnost tog paketa o drugima u datoteci *CMakeLists.txt*. Program za robotski vid ovisi o svim modulima u RVL-u i svim bibliotekama o kojima ovisi RVL.

4.2.2. Čvor kamere Intel RealSense LiDAR Camera L515

Zadatak čvora kamere je da konstantno objavljuje slike u boji i dubinske slike, kako bi se program za robotski vid mogao pretplatiti na te teme i snimiti sliku objekta u određenom položaju. Čvor će objavljivati slike na sljedeće teme:

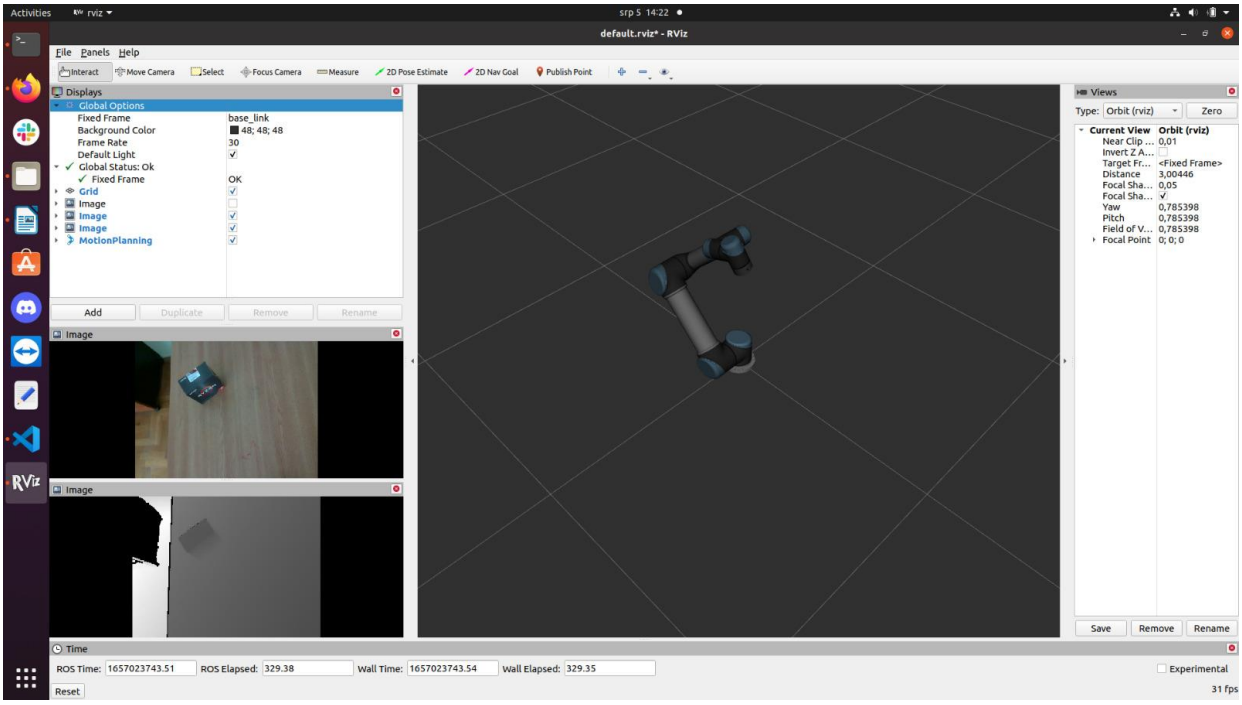
- /camera/color/image_raw
- /camera/aligned_depth_to_color/image_raw

Svi čvorovi u ROS-u se pokreću s naredbama roslaunch ili rosrun. Kamera se pokreće sljedećom naredbom:

```
$ roslaunch realsense2_camera rs_camera.launch enable_pointcloud:=true
depth_width:=320 depth_height:=240 depth_fps:=30 color_width:=640
color_height:=480 color_fps:=30 align_depth:=true
```

Primjer 4.2 Naredba za pokretanje čvora kamere

Bitan parametar pokretanja je „align_depth:=true“ koji kaže da se poravnaju dubinska slika i slika u boji i da tu sliku objavi na posebnu temu. Inače bi rezolucije dubinske slike i slike u boji bile različite te bi prepoznavanje objekata pomoću računalnog vida bilo otežano zbog smanjene rezolucije. Kako bi se moglo pratiti što kamera vidi i kako je robot pozicioniran u svom okruženju, u zasebnom terminalu se pokreće Rviz, program koji dolazi ugrađen u ROS. Pomoću njega se može prikazati robota u 3D prostoru i također prikazati što kamera objavljuje na temama

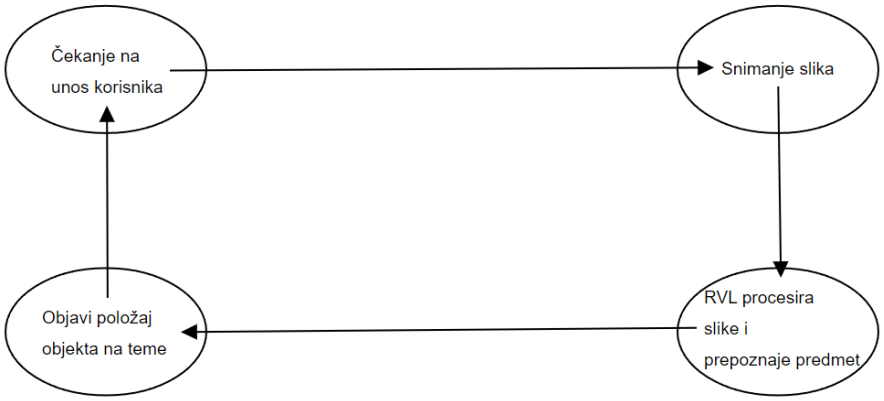


Slika 4.1 Progam RViz

color/image_raw i aligned_dept_to_color/image_raw. Kako program Rviz izgleda s aktiviranim čvorom za kameru je vidljivo na slici 4.1.

4.2.3. Programsko rješenje za prepoznavanje objekata

Kako je RVL pisan u programskom jeziku C++, tako je i ovo programsko rješenje pisano u istom programskom jeziku. Datoteka je nazvana DDDetection.cpp. Na početku programa se uključuju sva potrebna zaglavlja za funkciju iz RVL-a i za funkcije iz ROS-a. Program je podijeljen u dva dijela: funkcija za detekciju objekata i glavna funkcija koja predstavlja objavljivača i pretplatnika u ROS-u. Dijagram toka ovog programa je prikazan na slici 4.2.



Slika 4.2 Dijagram toka programa DDDetection

Da bi se mogla dohvatiti slika s kamere, čvor se mora pretplatiti na teme na koje kamera objavljuje slike. Kako bi se program mogao pretplatiti na dvije teme odjednom, moraju se međusobno sinkronizirati. U primjeru 3.3 je naveden opisani kod.

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "image_listener");
  ros::NodeHandle nh;
  cv::namedWindow("view");
  message_filters::Subscriber<sensor_msgs::Image> image_RGB(nh,
"/camera/color/image_raw",  1);
  message_filters::Subscriber<sensor_msgs::Image> image_D(nh,
"/camera/aligned_depth_to_color/image_raw", 1);
  typedef
message_filters::sync_policies::ApproximateTime<sensor_msgs::Image,
sensor_msgs::Image> MySyncPolicy;
  message_filters::Synchronizer<MySyncPolicy> img_sync(MySyncPolicy(10),
image_RGB, image_D);
```

```
  img_sync.registerCallback(boost::bind(&imageCallback, _1, _2));
  cv::destroyWindow("view");
```

Primjer 4.3 Glavna funkcija – prvi dio

Također, mora se definirati i opisati funkcija koja se poziva dokle god je program u ros::spin() ciklusu. Unutar te funkcije je kod koji sprema slike objavljene na temama od kamere. Za potrebe ovog rada, funkcija se mora pozivati u onom trenutku kada korisnik to želi te se iz tog razloga koristi funkcija ros::spinOnce().

```
void imageCallback(const sensor_msgs::ImageConstPtr& msg_RGB, const
sensor_msgs::ImageConstPtr& msg_D)
{
  ROS_INFO("Callback successful");
  cv::imwrite("/home/andrej/RVL/02_RGB.png", cv_bridge::toCvShare(msg_RGB,
"bgr8")->image);
  cv::imwrite("/home/andrej/RVL/02_D.png", cv_bridge::toCvShare(msg_D,
"16UC1")->image);
}
```

Primjer 4.4 Callback funkcija

Rad funkcije koja obrađuje slike već je prikazan u prošlom poglavlju. Nadalje, inicijaliziraju se izdavači kako bi uspostavili komunikaciju s čvorom za robota. Na ove se teme objavljuju samo dijelovi transformacijske matrice i matrica veličine predmeta, a logika spajanja matrica prepušta se čvoru pretplatniku. Također, mora se osigurati da će izdavač objaviti poruku samo onda kada zna da se netko pretplatio na tu temu. U slučaju da računalni vid ne uspije prepoznati objekt, kako bi spriječili rušenje programa, mora se osigurati da izdavač ništa ne objavi na teme. Spomenuto se postiže tako da se provjeri veličina objekta kojeg je računalni vid prepoznao kao pravokutni i ako je njegova širina, dužina ili visina manja od 0.0001 tada se može pretpostaviti da nije prepoznao nikakav objekt i u tom slučaju program ne generira hipoteze i ne objavljuje ništa na teme, nego se vraća na početak while(ros::ok()) petlje. Programski kod za opisani problem nalazi se u primjeru 4.5.

```
while(ros::ok()){
    std::cout << "Press enter to capture image" << std::endl;
    std::cin.get();
    ros::spinOnce();
    sleep(2);
    cv::Mat I = cv::imread("/home/andrej/RVL/02_RGB.png", 0);
    while(I.empty()){
      std::cout << "Failed imread(): image not found" << std::endl;
      sleep(2);
      ros::spinOnce();
      I = cv::imread("/home/andrej/RVL/02_RGB.png", 0);
    }
    std::cout << "Image found! Processing image..." << std::endl;
    RECOG::DDD::Hypothesis hyp = dddetector();
    int flag=0;
    for(int i=0;i<3;i++){
      array_msg_s.data[i]=hyp.pose.s[i];
      if(hyp.pose.s[i]<0.0001)
```

```
        flag++;
    }
    for(int i=0;i<9;i++){
      array_msg_R.data[i]=hyp.pose.R[i];
    }
    for(int i=0;i<3;i++){
      array_msg_t.data[i]=hyp.pose.t[i];
    }
    if(flag==0){
    printf("Waiting for subsribers\n");
    while(true)
    {
      if(publisher_R.getNumSubscribers()>0){
        printf("Got one subscriber on matrix R\n");
        publisher_R.publish(array_msg_R);
        break;
      }
    }
    while(true)
    {
      if(publisher_t.getNumSubscribers()>0){
        printf("Got one subscriber on matrix t\n");
        publisher_t.publish(array_msg_t);
        break;
      }
    }
    while(true)
    {
      if(publisher_s.getNumSubscribers()>0){
        printf("Got one subscriber on size\n");
        publisher_s.publish(array_msg_s);
        break;
      }
    }
    }
    else{
      printf("No hypothesis generated, not publishing to topics..");
    }
    printf("\n");
  }
```

Primjer 4.5 Glavna funkcija - drugi dio

Ovaj program kao čvor u ROS-u pokreće se sa sljedećom naredbom:

```
$ rosrun robot_vision DDDetection
```

Primjer 4.6 Naredba za pokretanje čvora DDDetection

4.2.4. Čvor robota UR5 i hvataljke Robotiq 3 finger gripper

Zadatak čvorova robota UR5 i hvataljke Robotiq 3 finger gripper je omogućiti njihovu manipulaciju kroz programska rješenja. Za pokretanje čvora robota prvo je potrebno provesti kalibraciju i stvoriti novi dokument koji će predstavljati kalibraciju za točno tog robota. To je moguće sljedećom naredbom:

```
$ roslaunch ur_calibration calibration_correction.launch
robot_ip:=192.168.22.14
target_filename:="home/user/my_robot_calibration.yaml"
```

Primjer 4.7 Naredba za kalibraciju robota

Parametar robot_ip predstavlja ip adresu na koju je spojen robot i ona se dobiva iz njegovog grafičkog sučelja. Nakon kalibracije može se pokrenuti čvor za robota sljedećom naredbom:

```
$ roslaunch ur_robot_driver ur5_bringup.launch robot_ip:=192.168.22.14
kinematics_config:=/home/andrej/my_robot_calibration.yaml
```

Primjer 4.8 Naredba za pokretanje čvora robota

U grafičkom sučelju robota se mora pokrenuti *External control*, kako bi robot znao da se manipulacija odvija kroz vanjski izvor, u ovom slučaju kroz programsko rješenje. Nakon što se to omogući u terminalu bi se trebala ispisati poruka: „*Robot ready to recieve control commands*“. S obzirom na to da se robotu daju apsolutne koordinate u koordinatnom sustavu baze robota, kako bi isplanirao najkraću putanju do te točke pokrećemo i čvor za MoveIt. MoveIt je najčešće korišteni softver za manipulaciju i korišten je na više od 150 robota. Izdat je pod uvjetima BSD licence i stoga je besplatan za industrijsku, komercijalnu i istraživačku upotrebu.[16] MoveIt se pokreće sljedećom naredbom:

```
roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch
limited:=true
```

Primjer 4.9 Naredba za pokretanje čvora MoveIt

Nadalje, u grafičkom sučelju robota, robotu se daje do znanja da postoji alat na zadnjem članku i koja je njegova masa. Hvataljku se također osposobljava iz grafičkog sučelja kao zaseban program. Po završetku programa, isti se može obrisati i u novom terminalu pokrenuti čvor za hvataljku sljedećom naredbom:

```
rosrun robotiq_3f_gripper_control Robotiq3FGripperTcpNode.py 192.168.22.11
```

Primjer 4.10 Naredba za pokretanje čvora hvataljke

Zadnji parametar naredbe je IP adresa hvataljke koja se također nalazi unutar grafičkog sučelja robota. Nakon što su se svi čvorovi pokrenuli, robot je spreman za manipulaciju.

4.2.5. Programsko rješenje za manipulaciju robotom

Programsko rješenje za manipulaciju robotom pisano je u programskom jeziku Python. Datoteka je nazvana robot_control.py. U prvom se dijelu programa uključuju sve potrebne biblioteke za program, kao što je prikazano sljedećim dijelom programa:

```
#!/usr/bin/env python
import math
import copy
from dataclasses import asdict
import moveit_commander, geometry_msgs.msg, moveit_msgs.msg, rospy, sys
from std_msgs.msg import Float64MultiArray
import numpy as np
import roslib
from time import time
from tf.transformations import euler_from_quaternion, quaternion_from_euler
from trac_ik_python.trac_ik import IK
```

```
from scipy.spatial.transform import Rotation as R

roslib.load_manifest('robotiq_3f_gripper_control')

from robotiq_3f_gripper_articulated_msgs.msg import
Robotiq3FGripperRobotOutput
```

Primjer 4.11 Uključene biblioteke za manipulaciju robotom

Program se sastoji od dva dijela, klase RobotControl i glavne funkcije. Unutar klase RobotControl su definirane funkcije za pomicanje robota, kao i za otvaranje i zatvaranje hvataljke. S obzirom na to da su funkcije kompleksnije, svaka od njih će biti zasebno objašnjena.

```
class RobotControl:
    def __init__(self):
        rospy.init_node('move_robot', anonymous=True)

        self.robot = moveit_commander.RobotCommander()
        self.scene = moveit_commander.PlanningSceneInterface()

        self.group_name = "manipulator"
        self.move_group =
moveit_commander.MoveGroupCommander(self.group_name)
        self.move_group.get_current_pose()

        self.pub = rospy.Publisher('Robotiq3FGripperRobotOutput',
Robotiq3FGripperRobotOutput, queue_size=10)
        self.ik = IK('base_link', 'tool0', solve_type="Distance")
```

Primjer 4.12 Klasa RobotControl - prvi dio

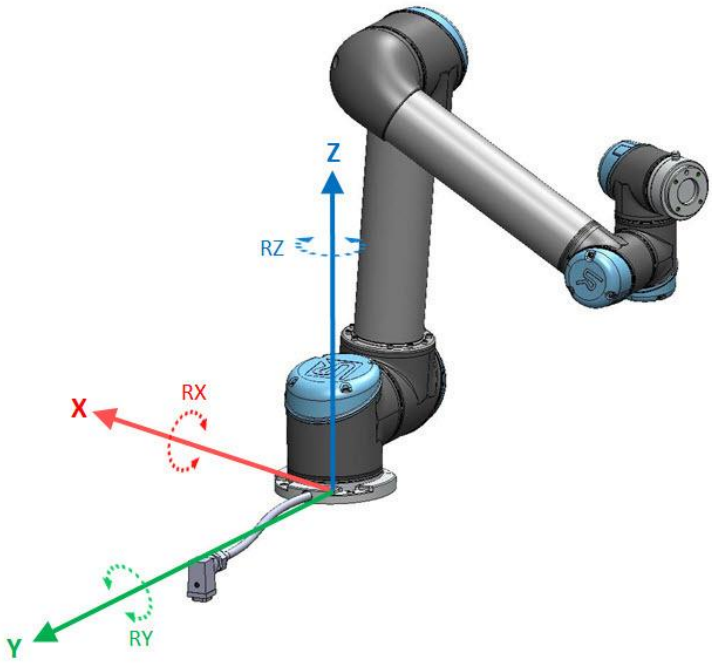
Prvo se definira instanca RobotCommander-a i PlanningSceneInterface-a kao što je prikazano u prve dvije linije koda. Zatim se stvara objekt koji pruža sučelje za planiranje pomaka za definiranu grupu u setup asistant-u. Također se definira objavljivač za output hvataljke. Planiranje putanje se obavlja pomoću inverzne kinematike. Inverzna kinematika proces je pretvaranja kartezijevih točaka u skup orijentacija zglobova za efektivno pomicanje zadnjeg članka robota u željeni položaj. [17]

```
def setPosition(self, x, y, z, Q):

        x = float(x)
        y = float(y)
        z = float(z)

        qx = Q[0,0]
        qy = Q[0,1]
        qz = Q[0,2]
        qw = Q[0,3]

        current_joints = self.move_group.get_current_joint_values()
        goal_joints = self.ik.get_ik(current_joints, x, y, z, qx, qy, qz,
qw)

        print("Inverse kin:")
        print(goal_joints)

        joint_goal = self.move_group.get_current_joint_values()
```

```
        joint_goal[0] = goal_joints[0]
        joint_goal[1] = goal_joints[1]
        joint_goal[2] = goal_joints[2]
        joint_goal[3] = goal_joints[3]
        joint_goal[4] = goal_joints[4]
        joint_goal[5] = goal_joints[5]


        self.move_group.go(joint_goal, wait=True)

        self.move_group.stop()
```

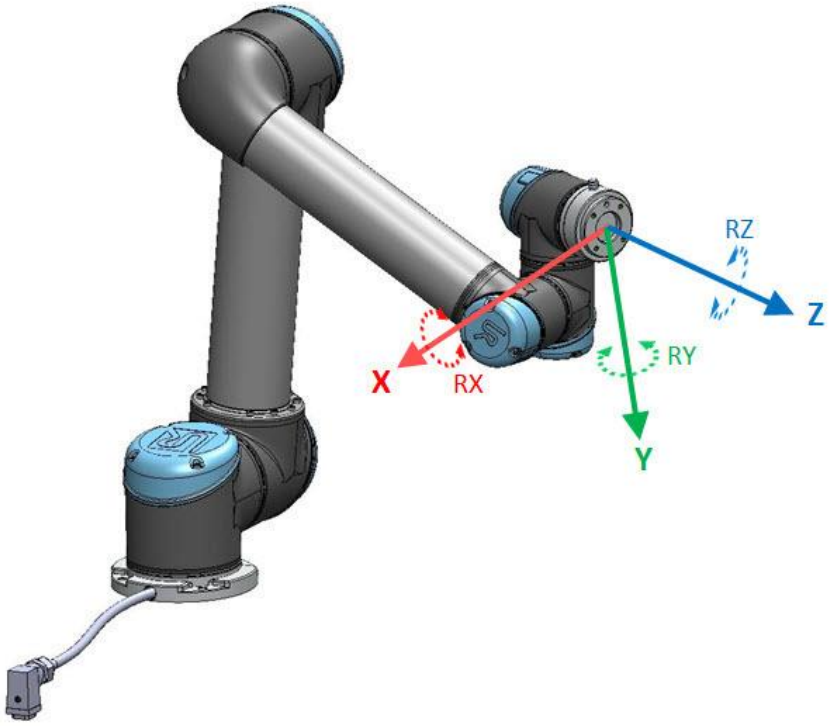
Primjer 4.13 Klasa RobotControl - drugi dio

Prva je funkcija setPosition(x, y, z, Q) koja prima tri koordinate koje predstavljaju točku na koju će se robot pomaknuti. Te točke predstavljaju točke u koordinatnom sustavu baze robota. Koordinatni sustav baze robota definiran je na slici 4.3.



Slika 4.3 Koordinatni sustav baze UR5 robota

Zadnji parametar funkcije predstavlja kvaternion matricu koja je rezultat pretvaranja iz rotacijske u kvaternion matricu. Kutovi za koji se zadnji članak treba okrenuti vrijednosti su prvog stupca te matrice. Kako bi inverzna kinematika radila, poslani parametri funkciji setPosition() i dobiveni kutovi iz kvaternion matrice moraju se proslijediti funkciji get_ik().Zatim se taj završni cilj konfiguracije svih članaka robota predaje funkciji go() i u tom se trenutku robot pomiče na zadane koordinate. Zadnji članak također ima svoj koordinatni sustav, prikazan slikom 4.4.



Slika 4.4 Koordinatni sustav zadnjeg članka UR5 robota

```
    def setPositionAndOrientation(self, position, orientation):
```

Primjer 4.14 Klasa RobotControl - treći dio

Klasa RobotControl ima još jednu vrlo sličnu funkciju kao i setPosition, koja prima objekte position i orientation iz kojih se iščitavaju vrijednosti x, y, z, qw, qx, qy i qz te se provodi isti postupak.

```
    def setPositionUpDown(self, delta_z):
```

Primjer 4.15 Klasa RobotControl - četvrti dio

Treća je funkcija setPositionUpDown koja prima jedan parametar delta_z koji predstavlja pomak po z osi zadnjeg članka robota. Koristi se nakon što se robot pomakne iznad objekta da bi se spustio do objekta kako bi ga mogao uhvatiti.

```
    def openGripper(self):

        command = Robotiq3FGripperRobotOutput()
        command.rACT = 1
        command.rGTO = 1
        command.rSPA = 255
        command.rFRA = 150
        command.rATR = 0
        command.rMOD = 1
        command.rPRA = 0

        start_time = time()

        while True:
```

```
            self.pub.publish(command)
            rospy.sleep(0.1)

            end_time = time()

            if float(end_time - start_time) >= 3.0:
                break

    def closeGripper(self):

        command = Robotiq3FGripperRobotOutput()
        command.rACT = 1
        command.rGTO = 1
        command.rSPA = 255
        command.rFRA = 150
        command.rATR = 0
        command.rMOD = 1
        command.rPRA = 100

        start_time = time()

        while True:
            self.pub.publish(command)
            rospy.sleep(0.1)

            end_time = time()

            if float(end_time - start_time) >= 3.0:
                break
```

Primjer 4.16 Klasa RobotControl - peti dio

Zatim se definiraju dvije vrlo slične funkcije, openGripper() i closeGripper() koje otvaraju i zatvaraju hvataljku. Postavljena vrijednost za zatvaranje hvataljke je command.rPRA=100, no kako hvataljka ima senzor za silu koja je potrebna da se predmet uhvati, hvataljka neće zgnječiti predmet nego će imati silu dovoljnu za podizanje. Obje funkcije imaju dio koda koji čeka onoliko vremena koliko je potrebno hvataljci da se otvori ili zatvori.

```
def currentPose_quaternionToRotation():
    Q=RC.move_group.get_current_pose().pose.orientation
    t_dots=RC.move_group.get_current_pose().pose.position
    t=np.array([[t_dots.x],
                [t_dots.y],
                [t_dots.z]])
    q0 = Q.w
    q1 = Q.x
    q2 = Q.y
    q3 = Q.z

    r00 = 2 * (q0 * q0 + q1 * q1) - 1
    r01 = 2 * (q1 * q2 - q0 * q3)
    r02 = 2 * (q1 * q3 + q0 * q2)

    r10 = 2 * (q1 * q2 + q0 * q3)
    r11 = 2 * (q0 * q0 + q2 * q2) - 1
    r12 = 2 * (q2 * q3 - q0 * q1)

    r20 = 2 * (q1 * q3 - q0 * q2)
    r21 = 2 * (q2 * q3 + q0 * q1)
```

```
    r22 = 2 * (q0 * q0 + q3 * q3) - 1

    R = np.array([[r00, r01, r02],
                  [r10, r11, r12],
                  [r20, r21, r22]])
    TAB=np.concatenate([R, t], axis=1)
    TAB=np.concatenate([TAB, [np.array([0,0,0,1])]])
    return TAB
```

Primjer 4.17 Klasa RobotControl - šesti dio

Zadnja funkcija klase RobotControl je funkcija koja ne prima nikakve parametre, nego samo vraća transformacijsku matricu trenutne pozicije robota kako bi se lakše dobile točne koordinate i pozicije zadnjeg članka robota. Glavni dio ove funkcije pretvorba je kvaterniona u rotacijsku matricu, jer funkcije za trenutnu poziciju robota vraćaju kvaternion oblik orijentacije.

```
RC = RobotControl()

print("Waiting for messages on topics /matrix_R and /matrix_t")
Matrix_R=Float64MultiArray()
Matrix_t=Float64MultiArray()
Matrix_R=rospy.wait_for_message("/matrix_R", Float64MultiArray)
print("Got matrix R")
Matrix_t=rospy.wait_for_message("/matrix_t", Float64MultiArray)
print("Got matrix t")
objectSize=rospy.wait_for_message("/size", Float64MultiArray)
print("Got object size")

objectSize=np.reshape(objectSize.data, (3,1))
```

Primjer 4.18 Glavna funkcija - prvi dio

U glavnoj funkciji prvo se instancira objekt klase RobotControl, zatim se pretplaćuje na teme /matrix_R, /matrix_t i /size i čeka da se pojavi poruka na njima, tj. čeka da se program DDDetection izvede. Pri dohvatu poruka, podatci se preoblikuju u matrice pogodne za rad u Pythonu pomoću funkcija iz biblioteke numpy [18].

```
Matrix_R=np.reshape(Matrix_R.data, (3,3))
Matrix_t=np.reshape(Matrix_t.data, (3,1))
TOC=np.concatenate([Matrix_R, Matrix_t], axis=1)
TOC=np.concatenate([TOC, [np.array([0,0,0,1])]])
print("Matrix TOC")
print(TOC)

TCA=np.array([[-0.9988957, 0, -0.0469827, 0],[0, -1, 0, 0.190],[-0.0469827,
0, 0.9988957, -0.015],[0, 0, 0, 1]])

print("Matrix TCA")
print(TCA)

TAB=currentPose_quaternionToRotation()
print("Matrix TAB")
print(TAB)

TOB =TAB@TCA@TOC

print("Matrix TOB")
print(TOB)
```

```
tOB = TOB[0:3, 3]

print("Point tOB")
print(tOB)
```

Primjer 4.19 Glavna funkcija - drugi dio

Točke koje se predaju funkcijama za manipulaciju robota moraju biti točke unutar koordinatnog sustava baze robota. S obzirom na to da program DDDetection prepoznaje objekt i računa odnos koordinatnog sustava objekta u odnosu na koordinatni sustav kamere, treba se provesti niz množenja matrica kako bi se u konačnici dobio odnos između koordinatnog sustava baze robota i koordinatnog sustava objekta. To se dobiva sljedećim izrazom:

$$^{B}T_{O} = {}^{B}T_{A} * {}^{A}T_{C} * {}^{C}T_{O} \tag{4-1}$$

Gdje je:

$^{B}T_{A}$ – Transformacijska matrica iz alata u bazu. Ona se dobiva iz funkcije RobotControl.current_pose_quaternionToRotation().

$^{A}T_{C}$ – Transformacijska matrica iz kamere u alat, dana je jednadžbom (4-2)

$$\begin{bmatrix} -0.9988957 & 0 & -0.0469827 & 0 \\ 0 & -1 & 0 & 0.19 \\ -0.0469827 & 0 & 0.9988957 & -0.015 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-2}$$

$^{C}T_{O}$ – Transformacijska matrica iz objekta u kameru. Logika preoblikovanja matrica dobivenih sa tema /matrix_R i /matrix_t u oblik transformacijske matrice je takva da se formira matrica čija su prva tri stupca rotacijska matrica (4-3), a četvrti stupac translacijski vektor (4-4). Zatim se dodaje se još jedan red s vrijednostima [0,0,0,1], gdje zadnji element predstavlja skaliranje transformacije (1 označava omjer 1:1).

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} \tag{4-3}$$

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{4-4}$$

U kodu se još uzima i translacijski dio dobivene matrice $^{B}T_{O}$, jer upravo taj dio predstavlja koordinate točke centra objekta u odnosu na koordinatni sustav baze robota.

```
#offset above object in m
offset=0.15
```

```
#aligning gripper with object for grabbing
TAG = np.array([[0.7071068, -0.7071068,  0.0000000,
0],[0.7071068,  0.7071068,  0.0000000, 0],[
0.0000000,  0.0000000,  1.0000000 , -0.18],[0, 0, 0, 1]])
if(objectSize[0]>objectSize[1]):
    TGO = np.array([[0, 1, 0, 0], [1, 0, 0, 0], [0, 0, -1, offset], [0, 0,
0, 1]])
else:
    TGO = np.array([[1, 0, 0, 0], [0, -1, 0, 0], [0, 0, -1, offset], [0, 0,
0, 1]])
TAB=TOB@TGO@TAG
tOB = TAB[0:3, 3]

#Quaternion matrix from rotation
r = R.from_matrix([TAB[0:3, 0:3]])
Q= r.as_quat()
print("Quaternion values for rotation (Matrix Q):")
print(Q)

#save current position for later use
original_orientation=RC.move_group.get_current_pose().pose.orientation
original_position=RC.move_group.get_current_pose().pose.position
input("Press enter to continue")

RC.setPosition(tOB[0], tOB[1], tOB[2], Q)
print("Finished ", offset ,"m above object")
input("Succesful? Press enter to continue")
```

Primjer 4.20 Glavna funkcija - treći dio

Za postizanje ispravne orijentacije hvataljke u odnosu na objekt, provodi se niz množenja matrica kako bi se hvataljka postavila u željeni položaj. Koordinatni sustav hvataljke je definiran kao desni koordinatni sustav čija z-os gleda prema dolje, a ishodište je u središtu trokuta koji čine vrhovi prstiju hvataljke. Izraz koji opisuje poravnavanje hvataljke dan je sljedećom jednadžbom:

$$ {}^{B}T_{A} = {}^{B}T_{O} * {}^{O}T_{G} * {}^{G}T_{A} \tag{4-5} $$

Gdje je:

${}^{B}T_{A}$ – Transformacijska matrica iz alata u bazu. Iz nje saznajemo kut za koji se treba hvataljka okrenuti kako bi mogla uhvatiti predmet

${}^{B}T_{O}$ – Transformacijska matrica iz objekta u bazu. Izračunata je jednadžbom (4-1).

${}^{O}T_{G}$ – Transformacijska matrica iz hvataljke u objekt. Ova matrica može poprimiti sljedeće oblike:

$$ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & offset \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4-6} $$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & offset \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4\text{-}7)$$

Matrica ${}^{O}T_{G}$ će poprimiti oblik (4-6) ako se hvataljka poravnava po y-osi predmeta, a oblik (4-7) ako se poravnava po x-osi predmeta. To ovisi o tome koja je stranica predmeta duža, tj. kako je točno predmet orijentiran na stolu. Također, unutar matrice definirana je varijabla *offset* koja predstavlja koliko metara iznad predmeta će se hvataljka postaviti, kako ne bi udarila u predmet prilikom pomicanja.

${}^{G}T_{A}$ – Transformacijska matrica iz alata u hvataljku. Definirana je sljedećom matricom:

$$\begin{bmatrix} 0.7071068 & -0.7071068 & 0 & 0 \\ 0.7071068 & 0.7071068 & 0 & 0 \\ 0 & 0 & 1 & -0.18 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4\text{-}7)$$

Kada se dobije matrica ${}^{B}T_{A}$, ona se treba pretvoriti u kvaternion oblik kako bi odgovarala parametru funkcije setPosition(). Prije prvog pomicanja robota, sprema se njegova trenutna orijentacija i pozicija, kako bi se nakon podizanja predmeta robot mogao vratiti u isti položaj.

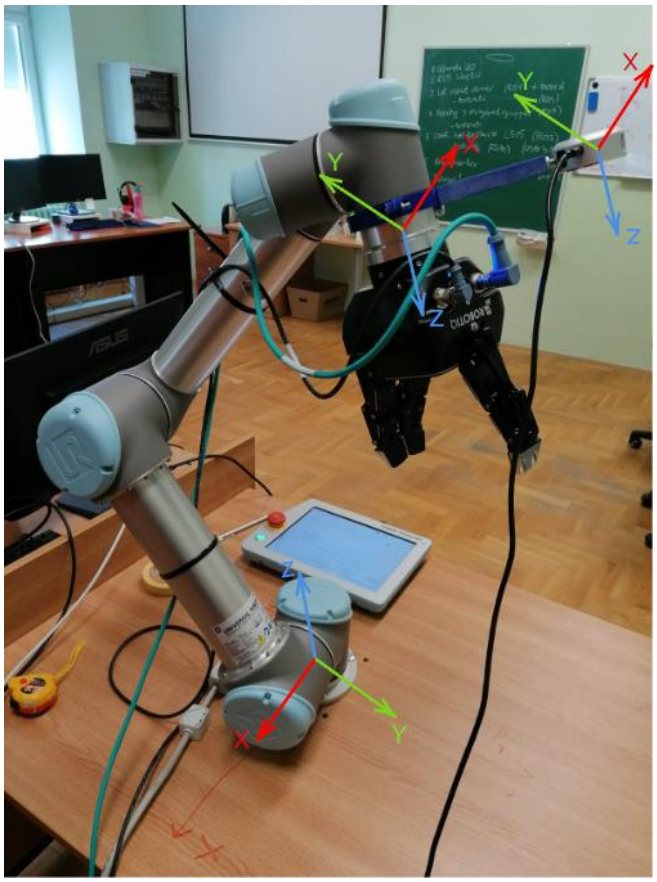
```
RC.openGripper()

#offset from tip of fingers to object center
new_offset=offset-objectSize[2]/2

RC.setPositionUpDown(-new_offset)
print("Finished coming down to object")
input("Is robot ready to grip the object? Press enter to continue")

RC.closeGripper()
print("Caught the object")

RC.setPositionUpDown(new_offset)
print("Lifted the object")

input("Can the robot go down and let go of object?")
RC.setPositionUpDown(-new_offset+0.01)
RC.openGripper()

print("Let go of object")

input("Can the robot go back to original position? Press enter to continue")
#Go back to starting position
RC.setPositionAndOrientation(original_position,original_orientation)

print("Sequence done!")
```

Primjer 4.21 Glavna funkcija - četvrti dio

Prije samog spuštanja i hvatanja objekta, hvataljka se prvo otvara. Varijabla *new_offset* predstavlja visinu za koju se robot mora spustiti da bi mogao uhvatiti objekt. Dobiva se tako da se od *offset*-a oduzme pola visine objekta. Zatim se poziva funkcija setPositionUpDown() s upravo izračunatom varijablom *new_offset*. Nakon što se robot spusti do objekta, hvataljka se zatvara. Kao što je navedeno, hvataljka ima senzore za silu potrebnu da se predmet uhvati. Zatim se robot podiže nazad na visinu na kojoj je bio s funkcijom setPositionUpDown() i time završava proces hvatanja objekta. Po završetku, robot se vraća u poziciju u kojoj se nalazio prije pokretanja programa DDDetection.

5. Eksperimentalni postav

Eksperimentalni postav se sastoji od sustava UR5 robotske ruke, koja na zadnjem članku ima pričvršćenu hvataljku Robotiq 3 finger gripper te montiranu kameru Intel RealSense LiDAR Camera L515, kao što je prikazano na slici 5.1. Početni je položaj robota takav da je objekt koji pokušavamo podići u vidnom polju kamere i da su mu vidljive najmanje dvije stranice. Na ovaj način počinjemo testirati razvijeni sustav. Podizanje svakog objekta testirat ćemo 10 puta u raznim položajima na stolu i raznim orijentacijama objekta, takvim da mu duža os bude u x ili y smjeru. U svim je slučajevima objekt polegnut i na slici snimljenoj kamerom su vidljive dvije stranice.



Slika 5.1 Eksperimentalni postav s označenim koordinatnim sustavima

5.1. Rezultati

Prvi predmet s kojim je sustav testiran je kutija za zaštitne maske. Dimenzije predmeta su:

- Širina: 18.5 cm
- Duljina 9.8 cm

- Visina 8 cm

Rezultati eksperimenta s kutijom za zaštitne maske dani su sljedećom tablicom:

Redni broj eksperimenta	Položaj predmeta: (x, y) [cm]	Prepoznavanje predmeta	Pomicanje robota i orijentacija hvataljke	Podizanje i spuštanje predmeta	Vraćanje u početni položaj
1	(2, 63)	✓	✓	✓	
2	(-18.2, 52.5)	✓	✓	✓	✓
3	(15, 56)	✓	✓	✓	✓
4	(15, 60)	✓	✓	✓	
5	(0, 63,4)	✓	✓	✓	
6	(10.5, 65.2)	✓			
7	(0, 52.6)	✓	✓	✓	✓
8	(23.1, 57.8)	✓			
9	(23.1, 57.8)	✓	✓	✓	✓
10	(-28.2, 58.5)	✓	✓	✓	✓

5.1 Rezultati eksperimenata s kutijom za zaštitne maske

Uspješnost pokusa evaluirana je po koracima izvedbe programa, a to su:

- Prepoznavanje predmeta
- Pomicanje robota i orijentacija hvataljke
- Podizanje i spuštanje predmeta
- Vraćanje u početni položaj

U pokusima u kojima je predmet u istom položaju (npr. 8. i 9. pokus) promijenjena je konfiguracija početnog položaja zglobova kako bi inverzna kinematika pronašla bolju putanju do predmeta.

Drugi predmet s kojim je sustav testiran je tetrapak. Njegove dimenzije su:

- Širina: 18.7 cm
- Duljina: 7 cm
- Visina: 7 cm

Rezultati eksperimenata s tetrapakom dani su sljedećom tablicom:

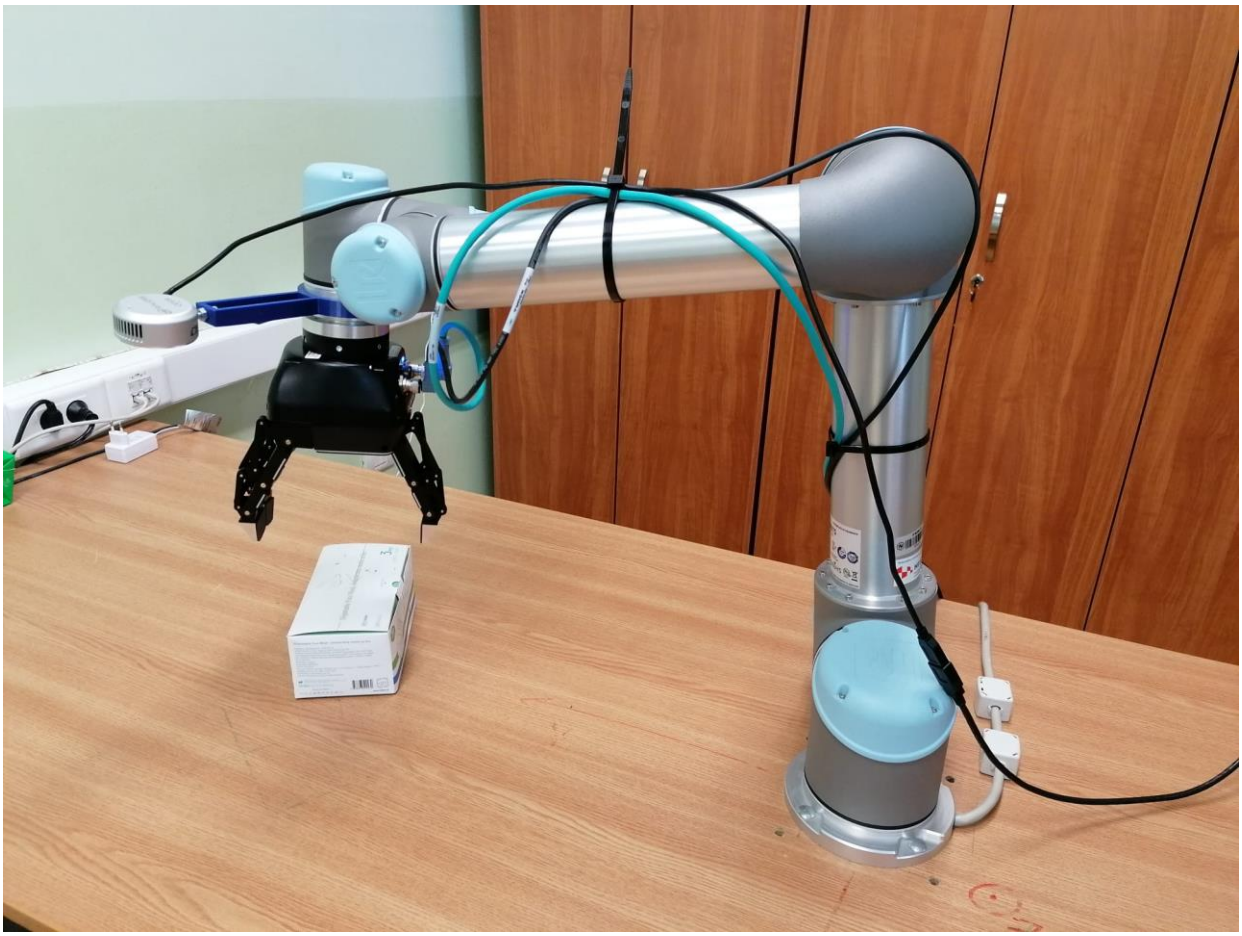
Redni broj eksperimenta	Položaj predmeta: (x, y) [cm]	Prepoznavanje predmeta	Pomicanje robota i orijentacija hvataljke	Podizanje i spuštanje predmeta	Vraćanje u početni položaj
1	(0, 65)	✓			
2	(0, 65)	✓	✓	✓	✓
3	(-15.8, 53.2)	✓	✓	✓	✓
4	(14, 52)	✓	✓	✓	✓
5	(16.3, 52.8)	✓	✓	✓	✓
6	(-26.3, 63.7)	✓			
7	(-18, 41.1)	✓	✓	✓	✓
8	(0, 69.2)	✓	✓	✓	✓
9	(-9.3, 50.1)				
10	(11.5, 53.3)	✓	✓	✓	✓

5.2 Rezultati eksperimenata s tetrapakom

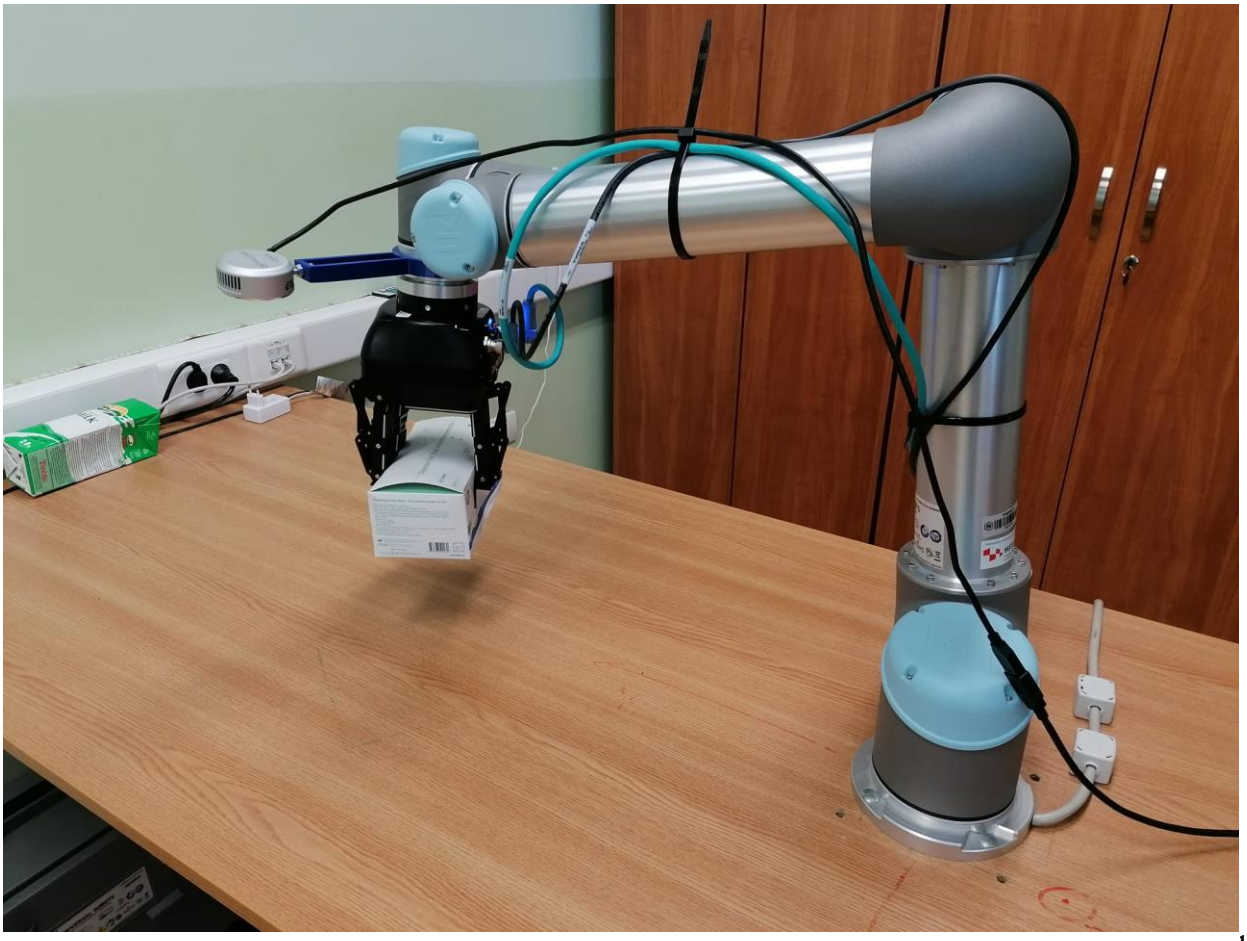
U pokusu 6 nije pronađena putanja jer je predmet predaleko postavljen od baze robota. U pokusu 9 predmet nije prepoznat zbog deformacije tetrapaka, tj. stranice gdje hvataljka hvata predmet su postale konkavne od pritiska.

Početni položaj robota, s predmetom kojim je sustav testiran i koraci po kojima su mjereni rezultati eksperimenata, prikazani su na sljedećim slikama:

Slika 5.2 Početni položaj robota



Slika 5.3 Položaj nakon pomicanja robota i orijentacije hvataljke



Slika 5.4 Položaj robota s uhvaćenim predmetom

Slika 5.5 Položaj nakon ispuštanja predmeta

6. Zaključak

U ovom je radu opisana teorijska podloga i programsko rješenje za prepoznavanje pravokutnih predmeta pomoću kamere Intel RealSense LiDAR Camera L515 i računalnog vida, te podizanje i spuštanje istih pomoću robotske ruke UR5 i hvataljke Robotiq 3 finger gripper. Programsko rješenje računalnog vida napisano je u programskom jeziku C++ i implementirano u ROS-u, a sastoji se od čvorova za pomicanje robota i hvataljke te čvorova računalnog vida koji međusobno razmjenjuju podatke.

Razvijeni sustav podrazumijeva fiksirani početni položaj robota i fiksiranu kameru zbog preciznosti određivanja točnog položaja predmeta i orijentacije hvataljke, koji se dobivaju s precizno izmjerenim parametrima transformacijskih matrica. Također, cijeli predmet se mora nalaziti unutar vidnog polja kamere te barem dvije stranice moraju biti vidljive kako bi program za prepoznavanje objekata mogao prepoznati predmet kao pravokutan.

Provedeni pokusi pokazuju da sustav uglavnom radi onako kako je zamišljen, a uzrok većine pogrešaka je krivo planiranje putanje robota. Sustav se može poboljšati definiranjem stola na kojem je robotska ruka fiksirana kako bi robot planirao putanje koje ne prolaze kroz stol. Stol je moguće dodati kao element unutar Moveit-a i označiti ga kao objekt sudara (engl. *collision object*)

LITERATURA

[1] M. Hrga, Računalni vid, *Zbornik radova Veleučilišta u Šibeniku*, sv., br. 1-2/2018, str. 207-216, 2018., dostupno na: https://hrcak.srce.hr/198597 [14. rujna 2022.].

[2] A. Billard, D. Kragić, Trends and challenges in robot manipulation, *Science*, sv. 364, br. 6446, lip. 2019., dostupno na: https://www.science.org/doi/10.1126/science.aat8414 [14. rujna 2022.].

[3] A. Khan, L. Sun, G. Aragon-Camarasa, J. P. Siebert, Interactive perception based on gaussian process classification for house-hold objects recognition & sorting, *IEEE International Conference on Robotics and Biomimetics*, str. 1087–1092., pro. 2016., dostupno na: https://doi.org/10.1109/ROBIO.2016.7866470 [14. rujna 2022.]

[4] W. Wohlkinger, A. Aldoma, R. B. Rusu, M. Vincze, "3DNet: Largescale object class recognition from cad models." *2012 IEEE International Conference on Robotics and Automation*, str. 5384–5391, svi. 2012., dostupno na: https://doi.org/10.1109/ICRA.2012.6225116 [14. rujna 2022.]

[5] J. Chen, B. Lei, Q. Song, H. Ying, D. Z. Chen, J. Wu, A Hierarchical Graph Network for 3D Object Detection on Point Clouds, *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, str. 389–398., Seattle, WA, USA lip. 2020., dostupno na: https://doi.org/10.1109/CVPR42600.2020.00047. [14. rujna 2022.]

[6] P. Đurović, M. Filipović, R. Cupec, Alignment of Similar Shapes Based on their Convex Hulls for 3D Object Classification, *2018 IEEE International Conference on Robotics and Biomimetics*, str. 1568-1593, Kuala Lumpur, Malezija, pro. 2018., dostupno na: https://doi.org/10.1109/ROBIO.2018.8665154 [14. rujna 2022.]

[7] M. Mihaljević, Robotska manipulacija primjenom računalnog vida, Diplomski rad, Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Osijek, 2018. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:200:019942 [14. rujna 2022.].

[8] P. Đurović, R. Grbić, R. Cupec, Visual servoing for low-cost SCARA robots using an RGB-D camera as the only sensor, *Automatika*, sv. 58 2017., br 4. str. 495-505, lip. 2018., dostupno na: https://www.tandfonline.com/doi/full/10.1080/00051144.2018.1461771 [14. rujna 2022.]

[9] H. Hu, F. Immel, J. Janosovits, M. Lauer, C. Stiller, A Cuboid Detection and Tracking System using A Multi RGBD Camera Setup for Intelligent Manipulation and Logistics, *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 1097-1103, kol. 2021., dostupno na: https://doi.org/10.1109/CASE49439.2021.9551449 [14. rujna 2022.]

[10] Universal Robots, dostupno na: https://www.universal-robots.com/ [10. lipnja 2022.].

[11] Robotiq, dostupno na: https://robotiq.com/ [10. lipnja 2022.].

[12] Intel RealSense, dostupno na: https://www.intelrealsense.com/ [11.lipnja 2022.].

[13] Robot Operating System, dostupno na: https://www.ros.org/ [11. lipnja 2022.].

[14] Creating a workspace for catkin, dostupno na: http://wiki.ros.org/catkin/Tutorials/create_a_workspace [26. lipnja 2022.].

[15] Creating a ROS package, dostupno na: http://wiki.ros.org/ROS/Tutorials/CreatingPackage [26. lipnja 2022.].

[16] MoveIt, dostupno na: https://moveit.ros.org [29. lipnja 2022.].

[17] J. Sereno, Inverse kinematics, *Undergraduate Journal of Mathematical Modeling: One + Two*, sv. 3., br. 1., čl. 6., 2010., dostupno na: https://digitalcommons.usf.edu/ujmm/vol3/iss1/18 [14. rujna 2022.].

[18] NumPy, dostupno na: https://numpy.org/ [27. srpnja 2022.].

SAŽETAK

Opisan je sustav za detekciju pravokutnih predmeta te njihovo hvatanje i pomicanje robotskom rukom. Sustav uključuje robotsku ruku UR5, hvataljku Robotiq 3 finger gripper i RGB-D kameru Intel RealSense LiDAR Camera L515 te algoritam za prepoznavanje pravokutnih objekata razvijen u C++ biblioteci Robot Vision Library i programsko rješenje za manipulaciju robota napisano u programskom jeziku Python. Svi dijelovi sustava su povezani unutar ROS okvira. Na slici snimljenoj kamerom prepoznaje se pravokutni predmet i određuje se njegovo središte u koordinatnom sustavu kamere. Kako bi robot uhvatio predmet, pomiče se u točku dobivenu transformacijom točke središta u koordinatni sustav baze robota te se hvataljka zakreće dok se prsti hvataljke ne poravnaju sa stranicama predmeta. Tada hvataljka hvata predmet i podiže ga. Nakon pomicanja robot se vraća u početni položaj. Rezultati testiranja su prikazani tablicama, gdje je opisana uspješnost svakog koraka u izvedbi programa.

Ključne riječi: prepoznavanje objekata, računalni vid, RGB-D kamera, robotika, ROS

ABSTRACT

Title: Grasping rectangular objects with a robotic arm using computer vision

A system for detection of rectangular objects, their grasping and movement with a robotic arm is described. The system includes a UR5 robotic arm, a Robotiq 3 finger gripper, an Intel RealSense LiDAR Camera L515 RGB-D camera, as well as an algorithm for detection of rectangular objects developed in the C++ library Robot Vision Library and a programming solution for robot manipulation written in the Python programming language. All parts of the system are connected within the ROS framework. In the image captured by the camera, a rectangular object is detected and its center is determined in the camera's coordinate system. In order for the robot to grasp the object, it moves to the point obtained from transforming the center point into the base coordinate system, and the gripper is rotated until its fingers are aligned with the sides of the object. Then the gripper grasps the object and lifts it. After moving, the robot comes back to its starting position. The testing results are shown in tables, where the success of each step in the program's execution is described.

Keywords: computer vision, object detection, RGB-D camera, robotics, ROS

PRILOG

Prilog 1: Cijeli kod programskih rješenja DDDetection.cpp i robot_control.py nalazi se na linku: https://github.com/AndrejBosnjak/ZavrsniRad_Prilog