Upravljanje UR robotom pomoću robotskog operativnog sustava ROS

Pavić, Domagoj

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:061062

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Domagoj Pavić

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Doc. dr. sc. Marko Švaco, mag. ing. mech.

Student:

Domagoj Pavić

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru doc. dr. sc. Marku Švaci i dr.sc. Josipu Vidakoviću na dostupnosti, pristupačnosti, savjetima i pomoći prilikom izrade ovog završnog rada.

Također se zahvaljujem cijeloj ekipi CRTA-e na pruženoj pomoći i savjetima tijekom izrade rada.

Zahvaljujem se svojoj obitelji na podršci i strpljenju tijekom studiranja.

Domagoj Pavić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22-**	

ZAVRŠNI ZADATAK

Student: **Domagoj Pavić** JMBAG: **0035215767**

Naslov rada na hrvatskom jeziku: **Upravljanje UR robotom pomoću robotskog operativnog sustava ROS**

Naslov rada na engleskom jeziku: **Control of a UR robot using the robot operating system ROS**

Opis zadatka:

Robotski sustavi djeluju na principu pretvorbe senzorskih informacija (percepcije) u fizičko djelovanje (akcije). Prilikom razvoja ovakvih sustava potrebno je u jednu cjelinu integrirati veliki broj segmenata: obradu senzorskih informacija, kalibraciju senzorskog sustava, planiranje kretanja, kinematski model robota, sustav za detekciju kolizija, upravljački sustav robota na niskoj i visokoj razini. ROS (eng. Robot Operating System) pruža sveobuhvatnu bazu naprednih pristupa u svakom od navedenih segmenata.
U završnom radu potrebno je u ROS okruženju implementirati virtualnu UR robotsku ruku koja će putem vizijskog sustava prema boji moći klasificirati objekte u određenom radnom području, a potom ih fizičkim djelovanjem razvrstati u skupine. Upravljanje robotskom rukom kao i obrada slike pri tome trebaju biti izvedeni s jednog centralnog vanjskog računala.

U završnom radu potrebno je implementirati sljedeće:

- Algoritam za planiranje kretanja UR (UR3 ili UR5) robotske ruke u Kartezijskom radnom prostoru,
- Kalibraciju vizijskog sustava na koordinatni sustav robota („hand-eye" kalibracija),
- Algoritam za lokalizaciju objekata u jednoj ravnini,
- Segmente sustava treba testirati u ROS simulacijskom okruženju te potom verificirati na pravom robotu u Laboratoriju za autonomne sustave.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Marko Švaco

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

SAŽETAK

U ovom se radu integriraju robotski sustavi za upotrebnu preko robotskog operativnog sustava ROS. Cilj rada je osposobiti radno područje (eng. *workspace*) te uz pomoć ROS-ovih integriranih softverskih biblioteka i alata upravljati robotom. Koristeći komunikaciju između ROS-a i robota, te ugrađenu biblioteku Moveit pomičemo robota upravljajući njegovim zglobovima ili tako što mu zadajemo kartezijske koordinate. Kamera, koja također komunicira sa ROS-om, daje informacije o slici koji se onda obrađuju uz pomoć OpenCV-a na način da se izdvoji željena boja, a potom se to očitanje koristi kako bi se predmeti sortirali. Bitno je istaknuti da se robot može pomicati i uz pomoć vlastite interakcijske jedinice, ali je prednost ROS-a što može stvoriti bilo kakvo okruženje za robota sa proizvoljnom hvataljkom, te u jednu cjelinu integrirati veliki broj segmenata.

Ključne riječi: UR5e, ROS, Moveit, python, OpenCV

SUMMARY

In this work we integrate robotic systems so we can use them through the robotic operating system ROS. The goal of this work is to create a workspace and use the integrated set of software libraries and tools found inside ROS, that help us build robot applications. Using the communication between ROS and our robot and the Moveit library we can move the robot by controlling its joints or by giving it a set of cartesian coordinates. The camera, which is also communicating with ROS, is updating the information about the image so that we can process said image using OpenCV. Now that we have a processed image, we extract colour data and sort our objects based on that. It is important to note that the robot can be controlled with a teach pendant, but the advantage of using ROS is that we can create any number of different environments for the robot and its complementary gripper and to integrate a large number of different segments.

Key words: UR5e, ROS, Moveit, python, OpenCV

1. UVOD

Robotika je interdisciplinarno područje koje predstavlja sinergiju strojarstva, elektrotehnike i računarstva sa čvrstom matematičkom podlogom te kao takva pruža mogućnost za integraciju u širokom rasponu teoretskih i praktičnih problema. Roboti u industriji danas značajno utječu na održavanje ujednačene kvalitete proizvoda, efikasnog iskorištavanja materijala, sigurnosti radnika te elimininiranja ljudski grešaka, čemu je glavni razlog mnogo veća preciznost i ponovljivost robota u radu od čovjeka. Većina robota u industriji barem u nekom dijelu proizvodnog procesa treba ostvariti fizičku interakciju sa predmetom rada, zbog čega je u većini slučajeva potrebno obratiti pažnju na robotsko rukovanje. Rukovanje predstavlja veoma kompleksan zadatak kako za čovjeka, tako i za robota pa još nije definirano poopćeno rješenje problema robotskog rukovanja za različite predmete u nestrukturiranoj okolini. Jedno od rješenja je Robotski Operativni Sustav ROS koji omogućuje centralizirano upravljanje robotskim sustavom preko vanjskog računala.

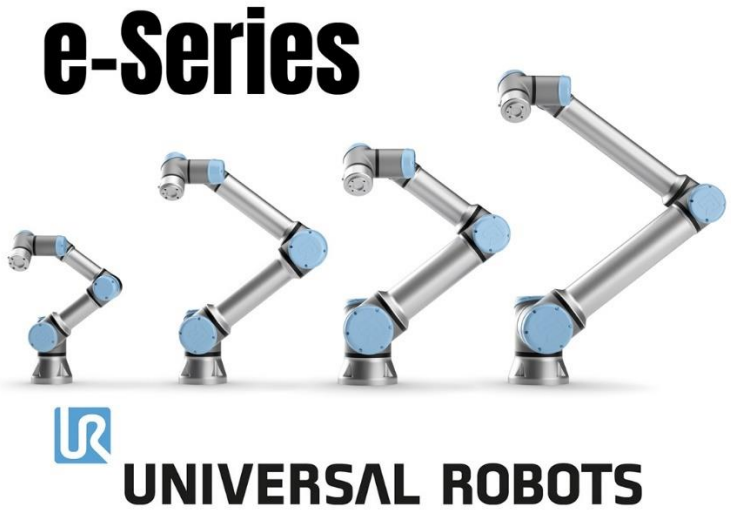
Kako bi ROS funkcionirao najbolje i najjednostavnije prvo se treba odlučiti na koji operativni sustav će se instalirati. Odabran je Linux Ubuntu koji će se pokretati preko virtualne mašine VMware. Nakon toga će se instalirati svi potrebni paketi kako bi se robot mogao pokrenuti i simulirati u ROS-ovom simulacijskom programu RVIZ-u. Za robota je odabran UR5e iz razloga što su driveri za njega dostupni iz službenih izvora [1]. Odabrana hvataljka je Robotiq 2f-85 koja je također vrlo dobro podržana [2]. Kamera Intel RealSense D435 je odabrana zbog svoje višestruke primjene, primarno obrada boje i dubinska detekcija. Glavna ideja rada je osigurati konzistentan način upravljanja robotom preko jednog centralnog vanjskog računala te pritom koristiti sve što nam ROS nudi. Iz tog je razloga odabran paket Moveit [3]. Također vrlo snažan alat, kojeg redovito održavaju stručnjaci, služi kako bi se pomicao robot preko njegovih ugrađenih upravljačkih jedinica. Implementacija ove ideje će se prikazati „pick and place“ metodom te će se uz pomoć kamere odrađivati klasifikacija na temelju boje. U danjim će se poglavljima detaljno istražiti svaki od gore navedenih pojmova te će se opisati kako se oni međusobno integriraju i nadopunjuju.

2. SUSTAV ZA ROBOTSKO RUKOVANJE

Kako bi se razumjelo s čime upravljamo moramo se prvo upoznati sa hardverom. U ovom se radu koristi kolaborativna robotska ruka UR5e, Robotiq 2f-85 hvataljka, Intel RealSense D435 kamera i vanjsko računalo koje se neće posebno obrađivati u ovom radu, zato što je kod računala na kojem se rad provodi jedino bitno da je operativni sustav Linux Ubuntu. Robot se nalazi u laboratoriju na postolju gdje je spojen na mrežu Ethernet kablom. Upute za postavljanje robota su dane u [4]. Hvataljka je pričvršćena na robota automatskim izmjenjivačem alata. Na računalo se spaja USB kablom dok se komunikacija odvija serijski UART protokolom. Industrijsko napajanje pretvara gradsku mrežu u 24V koliko je potrebno za rad hvataljke. Kamera je pričvršćena na hvataljku 3D printanim nosačem, a na računalo se spaja USB kabelom.

2.1. UR5e

Robot UR5e američke tvrtke Universal Robots je vrlo fleksibilna kolaborativna robotska ruka. To znači da ovaj robot može surađivati s čovjekom, dakle ima sigurnosne funkcije koje ga osiguravaju od sudaranja sa okolišem, samim sobom ili nanošenja ozljeda čovjeku s kojim surađuje.

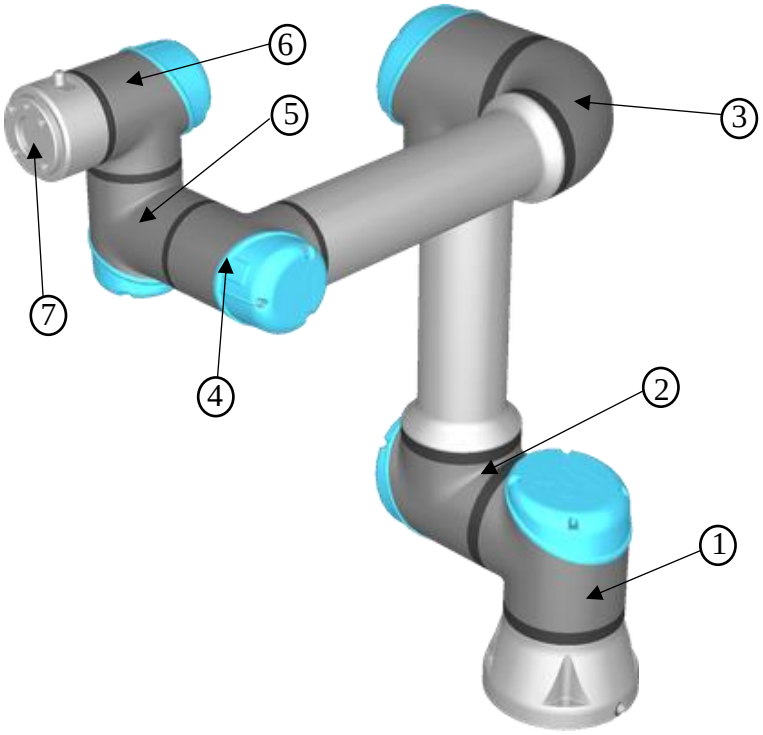


Slika 1. Universal robots e-serija [5]

Srednji član URe „obitelji“ je ujedno i odabrani robot UR5e. Maksimalno opterećenje od 5kg je idealno za automatizaciju procesa sa srednjim kilažama Radna površina mu je 850mm što ga čini idealnim omjerom snage i veličine. Šest zglobova mu omogućuju šest stupnjeva slobode gibanja. Prednost je što ga se može ručno namjestit u bilo koji položaj te dodatno namjestiti sa interaktivnom jedinicom (eng. *teach pendant*) koja dolazi uz njega. Na spomenutoj jedinici je moguće upravljati svakim od zglobova zasebno ili pomicati određeni koordinatni sustav u odnosu na odabrani sustav robota (postoje zadani koordinatni sustavi kao što su sustav krajnje jedinice, eng. *end effector*, ili sustav baze, a moguće je i definiranje vlastitih koordinatnih sustava). To također znači da se robot može programirati i sa spomenutim *teach pendantom*.

System Parameter	UR5e
Degrees of Freedom	6 rotating joints
Payload	5 kg / 11 lbs
Repeatability	±0.03 mm, with payload, per ISO 9283
Weight with cable	20.6 kg / 45.4 lbs
Reach	850 mm / 33.5 in
Motion Range	Base: ± 360° Shoulder: ± 360° Elbow: ± 360° Wrist 1: ± 360° Wrist 2: ± 360° Wrist 3: Infinite
Maximum Speed	Base: ± 180°/Sec. Shoulder: ± 180°/Sec. Elbow: ± 180°/Sec. Wrist 1: ± 360°/Sec Wrist 2: ± 360°/Sec Wrist 3: ± 360°/Sec
Power Consumption	100-240VAC, 47-440Hz
Robot Mounting	Any
Ambient Temperature	0-50°*

Slika 2. Specifikacije UR5e robota [6]



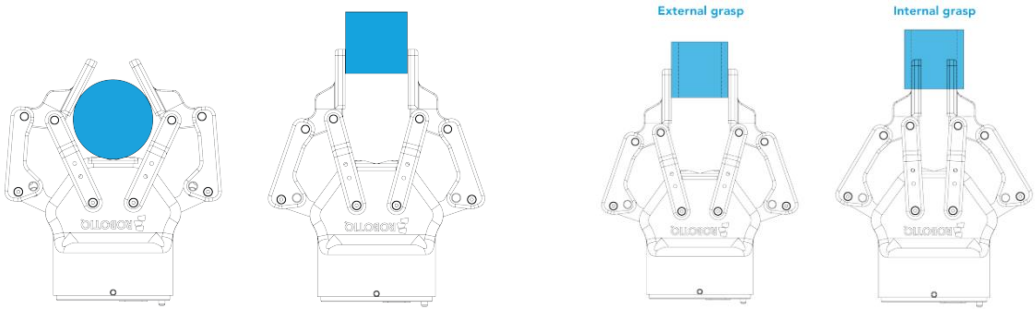
1 – baza (eng. *base*)
2 – rame (eng. *shoulder*)
3 – lakat (eng. *elbow*)
4 – zglob 1 (eng. *wrist 1*)
5 – zglob 2 (eng. *wrist 2*)
6 – zglob 3 (eng. *wrist 3*)
7 – krajnja jedinica(eng. *flange*)

Slika 3. Nomenklatura UR5e robota [7]

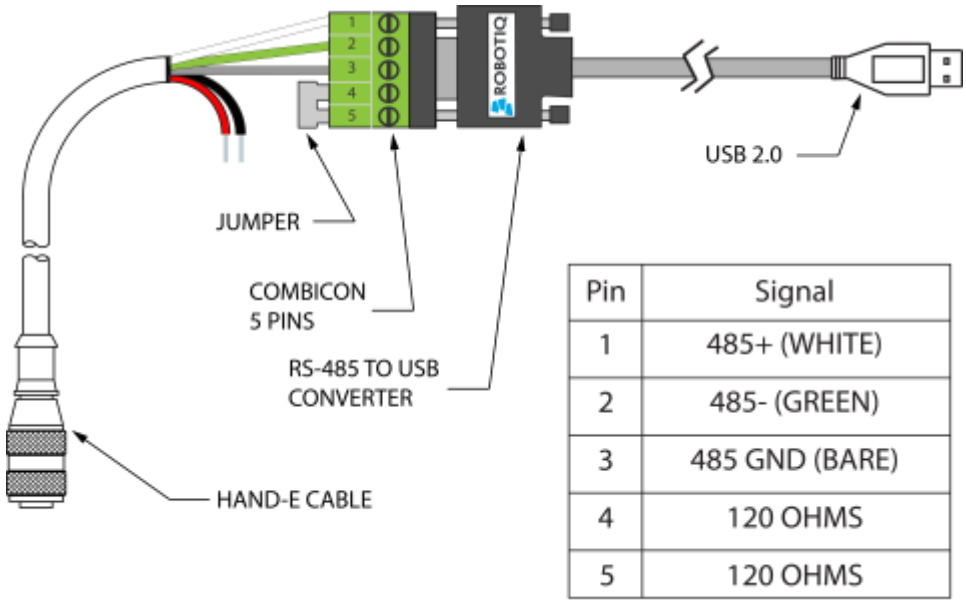
2.2. Robotiq 2f-85 hvataljka

Robotiq 2f-85 hvataljka je odabrana radi svoje raznovrsnosti prihvata, također zato što je već dostupna u laboratoriju. Hvataljka se pričvršćuje na robota automatskim sustavom zamjene alata SWA-011-000-000. Zbog svoje fleksibilnosti hvataljka ima mogućnost različitog načina primanja predmeta, zbog svoje strukture oblik predmeta nije isključujući prilikom hvatanja što se može detaljnije vidjeti na (Slika 5). Komunikaciju sa računalom ostvarujemo serijski preko USB-a, dok se kontrola odvija preko Modbus RTU protokola(Slika 6).

Slika 4. Nomenklatura hvataljke [8]



Slika 5.Tipovi hvatanja [8]



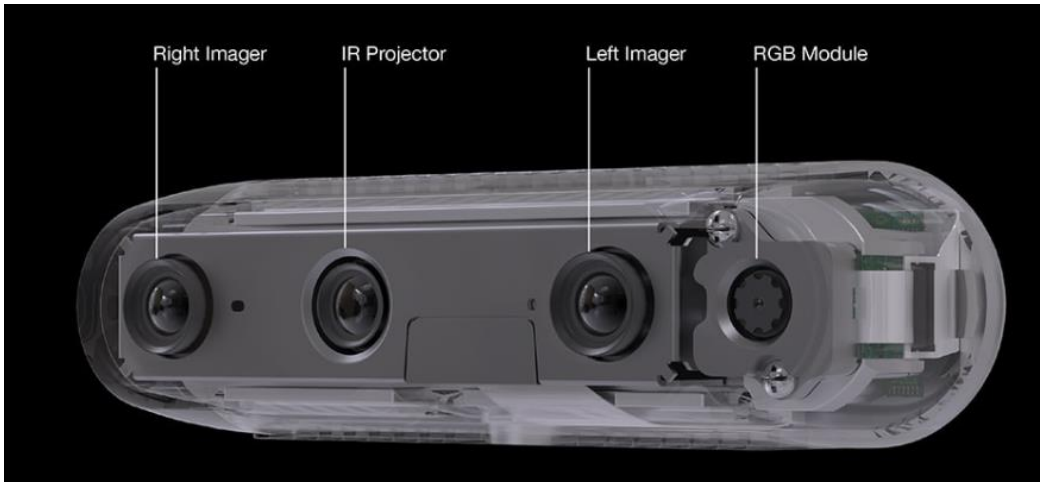
Pin	Signal
1	485+ (WHITE)
2	485- (GREEN)
3	485 GND (BARE)
4	120 OHMS
5	120 OHMS

Slika 6. Način komunikacije [8]

2.3. Intel RealSense D435 kamera

RealSense D435 je stereo kamera koja ima široku primjenu. Iako se u ovom radu koristi samo sposobnost kamere da daje preciznu sliku u pokretu, te RGB leća (Slika 8) kako bi se obradila slika za detekciju boje, ona je višenamjenska te joj ovo nije jedina funkcija. Jedna od glavnih funkcija ove kamere je detekcija dubine što se u ovom radu neće obrađivati. Također posjeduje napredne značajke kao što su stvaranje oblaka točaka (eng. *point cloud*).

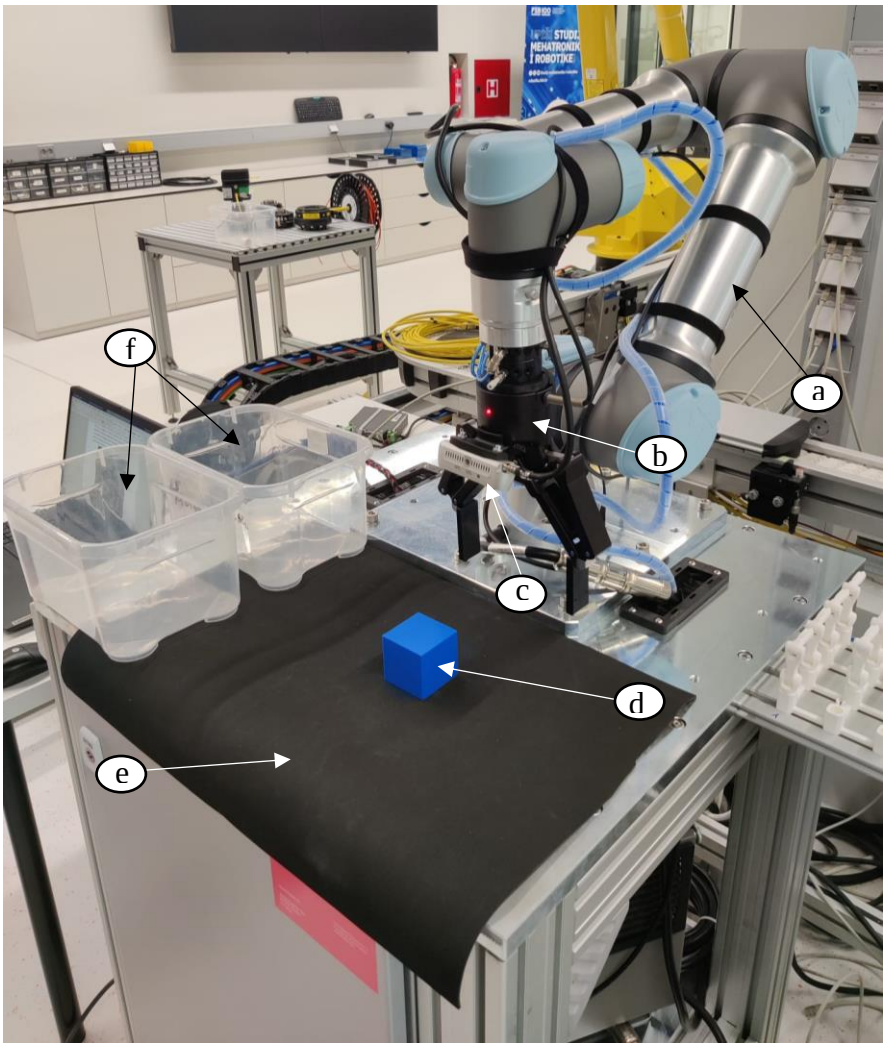
Slika 7. Intel RealSense D435 kamera [9]



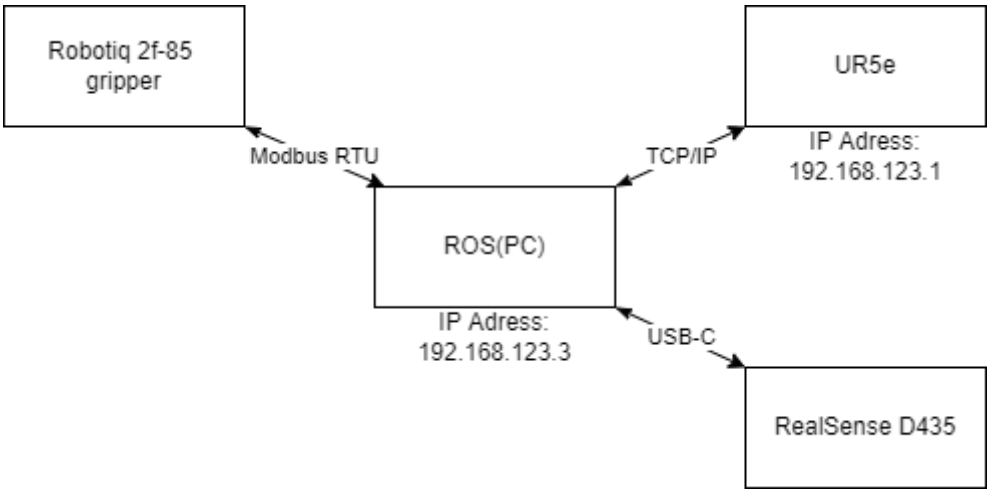
Slika 8. Leće kamere [9]

2.4. Sustav

UR5e robot je pričvršćen na postolje u laboratoriju u CRTA-i, a s računalom komunicira preko ethernet kabela. Računalo i robot moraju biti spojeni na istu mrežu (5.2) . Na krajnjoj jedinici robota (eng. *end effector)* uz primjenu izmjenjivača alata je pričvršćena hvataljka koja komunicira s računalom preko Modbus RTU protokola. Kamera se na hvataljku pričvršćuje već ranije izrađenim (3D printanim) nosačem. Komunikaciju sa računalom ostvaruje preko USB-C kabela. Sve komponente koriste ROS kako bi izmjenjivali podatke. Prikaz robotske stanice može se vidjeti na (Slika 9), a na (Slika 10) je shema komunikacije.



Slika 9. Robotska stanica : a) UR5e, b) Robotiq 2f-85 hvataljka, c) RealSense D435 kamera, d) predmet, e) podloga



Slika 10. Shema komunikacije

3. ROS ZA PLANIRANJE KRETANJA

3.1. ROS

Za optimalan način rada ROS-a vanjsko bi računalo trebalo raditi na Linux Ubuntu 18.04 (ova verzija je najpogodnija za integraciju sa ostalim programima koji se koriste u ovom radu) operativnom sustavu. Sustav ne mora nužno biti operativni sustav računala već se može koristiti virtualna mašina. U ovom radu se koristi ovaj potonji način, a za virtualnu mašinu će se koristiti VMware. Vrlo je bitno da je virtualna mašina ažurirana, u suprotnom postoji šansa da se virtualna slika neće moći upaliti.

The Robot Operating System ROS je *open-source* radno okružje koje služi za razvijanje i ponovno korištenje robotskih aplikacija. Glavna je prednost što je dostupan svima pa tako omogućuje suradnju stručnjaka, hobista i razvojnih inženjera koji svi imaju zajednički cilj: približavanje robotike svima i poboljšanje robotike kao grane. Također postoje razna rješenja poznatih problema koja su dostupna svima na web stranicama [10]. U ovom se radu koristi ROS *melodic* iz razloga što je za primjenu Moveit paketa i kod primjene na stacionarnim robotima ova verzija najstabilnija. Iz istog razloga je odabrana verzija 18.04 Ubuntu-a (najstabilnija je za ROS *melodic*). Prvi je korak u ovom radu bio naučiti kako ROS funkcionira, a za to postoje elaborirane lekcije [11]. Instalacija ROS-a na računalo je također detaljno opisana na službenoj stranici [12].

3.1.1. Način rada

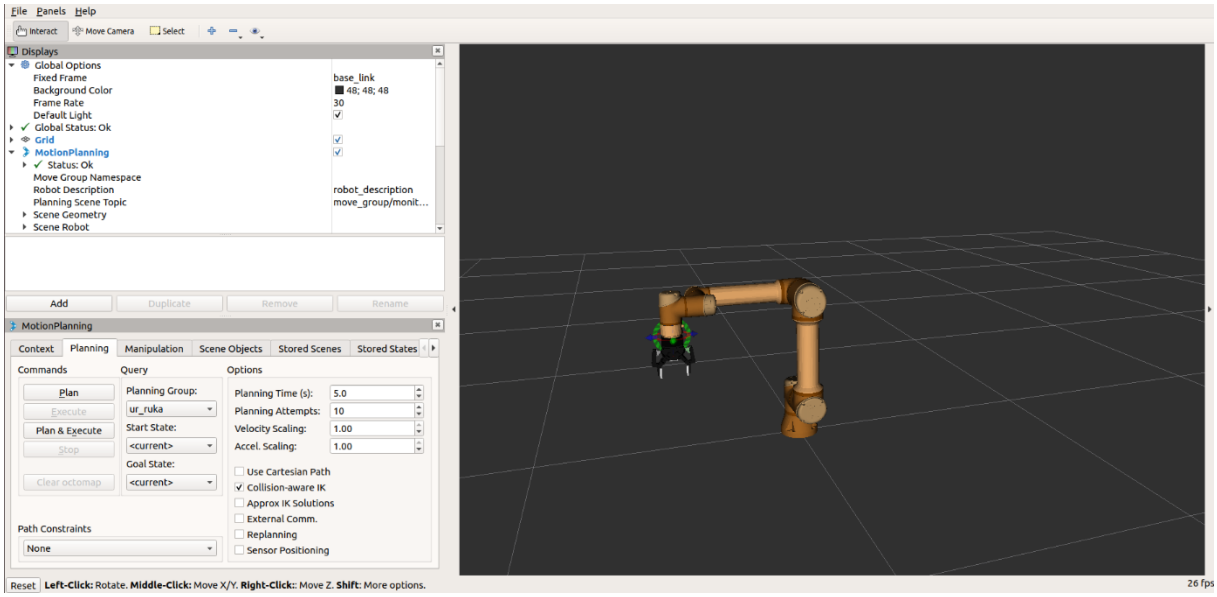
ROS komunicira unutar sebe preko teme (eng. *Topic*) koji se ponašaju kao „teme razgovora“. Ako bi se na neku temu željelo davati informacije tada bi trebao postojati „govornik“ ili u slučaju ROS-a pošiljatelj (eng. *Publisher*). Pošiljatelj je vrsta čvora (eng. *node*) (tip izvršne jedinice koja je spojena na ROS mrežu) koja neprekidno šalje podatke na temu. Kako bi se razgovor upotpunio potreban nam je slušatelj, u slučaju ROS-a pretplatnik (eng. *Subscriber*). Pretplatnik je također vrsta čvora koja prima informacije koje šalje pošiljatelj na istu temu. To je osnovni princip komunikacije unutar ROS-a. Kako bi koristili ugrađene funkcije ili već postojeće sa interneta [13] potrebno je stvoriti radno područje (*eng. workspace*) [12]. Unutar radnog područja se grade paketi. Paketi su način na koji je ROS organiziran i mogu sadržavati: čvorove, konfiguracijske datoteke, ROS-ove neovisne biblioteke, programe drugih proizvođača i slično. Cilj paketa je da pruže korisne funkcionalnosti na način koji lako obraditi tako da bi se taj program mogao lako ponovno iskoristiti.

3.2. Moveit

Moveit je ROS-ov integrirani paket te radi povrh ROS-a. Moveit objedinjuje niz funkcionalnosti potreban za upravljanje robotom kao što su: planiranje kretanja (*eng. motion planning),* manipulacija, 3D percepcija, kinematika, kontrola i navigacija. Za korištenje ovog paketa potrebno ga je instalirati [3]. Ovaj će se paket koristiti za upravljanje fizičkim i simuliranim robotom. Prednost Moveit-a proizlazi iz toga što upravlja robotom nizom kontrolera na koje možemo direktno utjecati preko računala. Također se istovremeno može upravljati s više robota ili kao što je to slučaj u ovom radu, sa simulacijom i robotskom rukom. Moveit omogućuje planiranje putanje robota što je korisno u situacijama u kojima se na pravcu između cilja i starta nalaze prepreke.

3.3. RVIZ

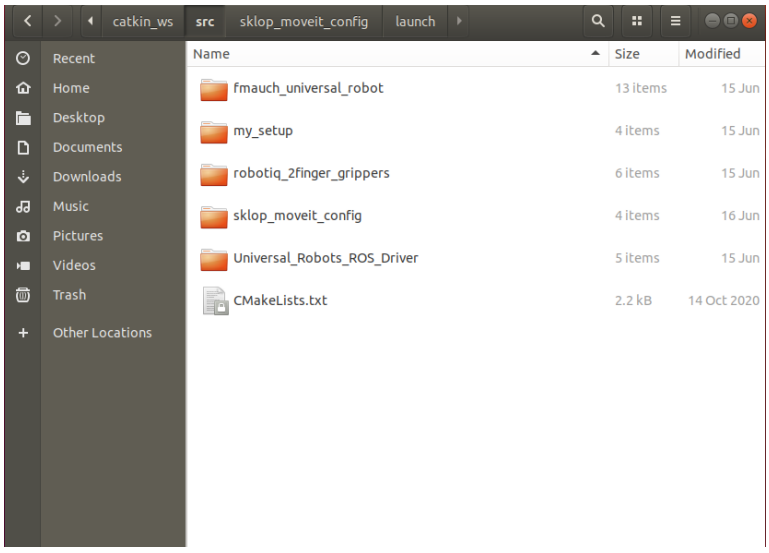
RVIZ je ROS-ovo grafičko sučelje koje se koristi za prikaz raznih informacija. Unutar RVIZ-a se moguće pretplatiti se na aktivne teme. Primarno će se koristiti za prikaz simuliranog kretnja robota i prikaz slike kamere.



Slika 11. Prikaz sklopa u RVIZ-u

3.4. Postavljanje ROS radnog područja (eng. *workspace*)

Nakon instalacije ROS *melodic-a* potrebno je prema [12] stvoriti radno područje u kojem će se nalaziti svi potrebni paketi. Kod stvaranja radnog područja (ako se pažljivo pratila prethodno navedena lekcija) automatski su se stvorila dva direktorija pod imenima *build* (sve datoteke koje generira *cmake* idu u ovaj direktorij) i *devel* (tu se nalaze kompilirane datoteke stvorene nakon što se izvrši komanda *catkin_make*). Treći i najbitniji direktorij stvara korisnik pod imenom *src*. Unutra se spremaju svi izvršivi paketi. Nakon svakog stvaranja ili dodavanja paketa (preuzimanje paketa sa [10]) potrebno je u terminalu izvršiti komandu *catkin_make*. Potrebni paketi uključuju: paket za upravljanje UR5e robotom i Robotiq 2f-85 hvataljkom. Za upravljanje UR5e robotom potrebna su nam dva paketa, *fmauch_universal_robot [1]* i *Universal_Robots_ROS_Driver* [1]. Prvi paket sadrži sve potrebne datoteke za opisivanje UR5e robota u simulaciji i izravno je povezan sa drugim paketom što znači da izvršavanje komande *catkin_make* neće uspjeti ako jedan od tih paketa nedostaje unutar *src* direktorija. Potonji paket služi za direktnu komunikaciju ROS-a i robota, a sadrži *driver* datoteke. Za upravljanje hvataljkom koristimo paket *robotiq_2finger_grippers [2]*. Taj paket sadrži sve potrebne datoteke za opis hvataljke i komunikaciju ROS-a i hvataljke. Kamera komunicira uz pomoć ugrađenog paketa u ROS-u [15]. Taj je paket potrebno dodatno instalirati kako je navedeno u [15]. Taj paket ne mora biti unutar *scr* direktorija jer komunicira direktno sa ROS-om i nisu mu potrebni paketi unutar radnog područja. Preostali paketi i njihova nomenklatura su dani na korisnikov izbor. U ovom konkretnom slučaju radi se o još dva dodatna paketa: *my_setup* i *sklop_moveit_config*. U nastavku će se detaljno objasniti korisnički stvoreni paketi.

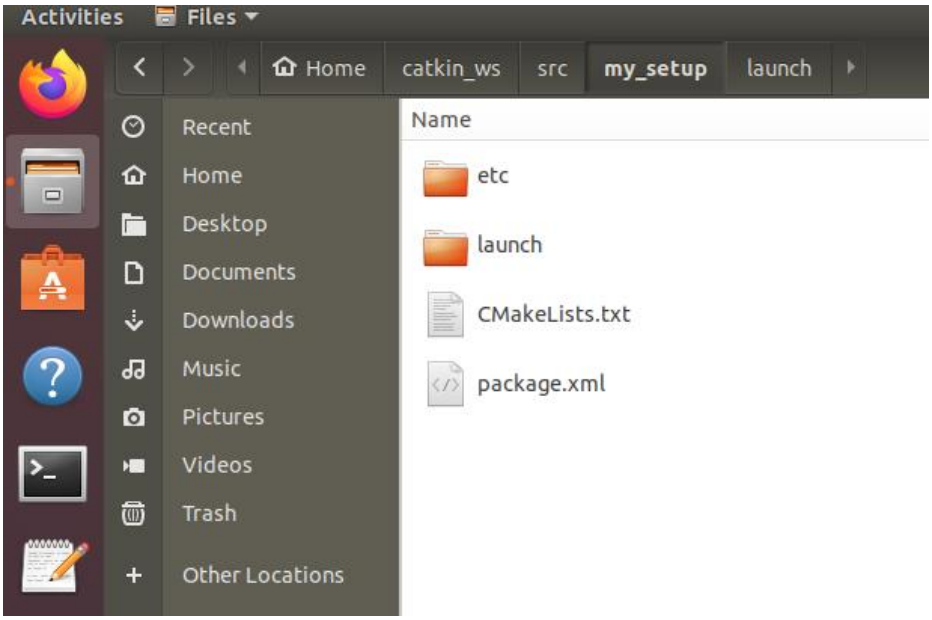


Slika 12. Prikaz radnog okruženja

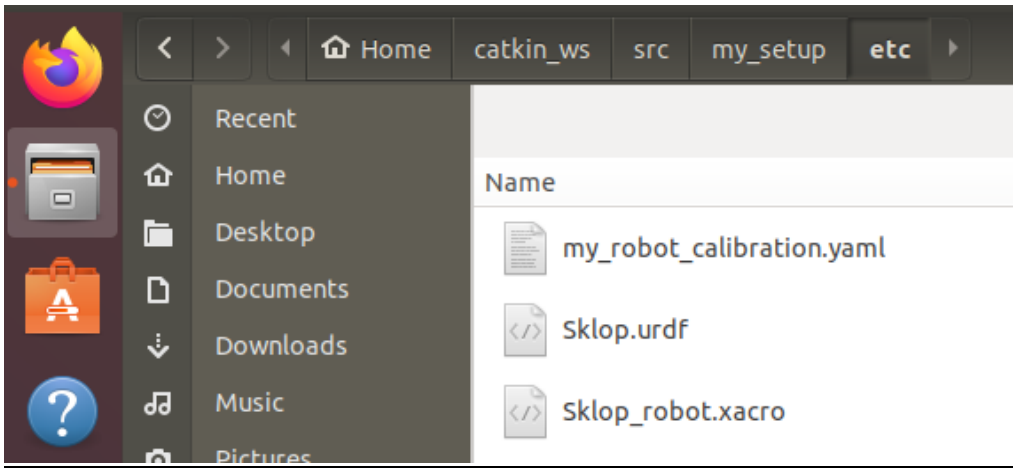
3.5. Korisnički paketi

3.5.1. Paket *my_setup*

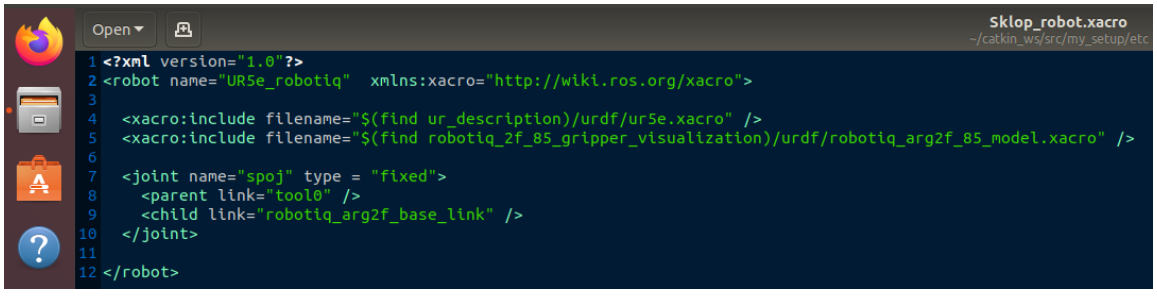
Paket je potrebno prvo stvoriti prema [14]. Unutar paketa *my_setup* nalaze se dva direktorija i dvije datoteke koje su automatski stvorene prilikom izrade paketa (Slika 13). Prvi direktorij pod nazivom *etc* sadrži *Sklop.urdf, Sklop_robot.xacro* i *my_robot_calibration* datoteke(Slika 14). Kako bi ROS znao o kojem se robotu ili sklopu radi potrebno mu je pružiti „opis“ komponenti kojima želimo upravljati i prikazati u simulaciji. *Fmauch_universal_robot* paket sadržava potreban „opis“ ili *urdf* datoteku UR5e robota (u ovom paketu se nalaze *urdf* datoteke svih Universal Robots robota, ali nama je potreban samo UR5e), dok se u *robotiq_2finger_grippers* direktoriju nalazi *urdf* hvataljke. Kako bi se stvorio robot u simulaciji koji sadrži i robotsku ruku i hvataljku kao cjelinu potrebna je *urdf* datoteka, a za to najlakši način je da se stvori *xacro* datoteka. *Xacro* je *XML* tip datoteke napisan u *macro* jeziku koji omogućuje jednostavno definiranje veza između zglobova. *Xacro* datoteka se može proširiti uz pomoć ROS-ove ugrađene funkcije: *rosrun xacro xacro.py* <*model.xacro*> > <*model.urdf*>. Na taj način uz relativno malo koda i programiranja u *macro* jeziku dobiva se elaborirani model sklopa robota i hvataljke. *Sklop_robot.xacro* (Slika 15) je *xacro* datoteka napisana za konfiguraciju UR5e robota i robotiq 2f-85 hvataljke, dok je *Sklop.urdf* generirana datoteka ranije navedenim načinom (na mjesto *model.xacro* dolazi *Sklop_robot.xacro* a na *model.urdf* proizvoljno ime *urdf* datoteke, u ovom slučaju *Sklop*). Posljednja datoteka *my_robot_calibration* će biti detaljnije objašnjena u narednim poglavljima. Također direktorij *launch* koji sadržava *my_launch_file* datoteku će biti također detaljnije objašnjen u idućem poglavlju.



Slika 13. ***My_setup direktorij***



Slika 14. ***etc*** **direktorij**



Slika 15. ***Sklop_robot.xacro datoteka***

3.5.2. Paket sklop_moveit_setup

Ovaj paket nastaje koristeći *Moveit setup assistant* (paket unutar ROS-a). Te za ovu konkretnu konfiguraciju postoje upute [16]. Pošto paketi koji se koriste u navedenim uputama nisu identični onima korištenim u ovom radu (cilj uputa je isključivo simulacija, dok je cilj ovog rada implementacija na fizičkom robotu) neke korake treba modificirati. Prvi korak koji je drugačiji je odabir konfiguracijske datoteke. Datoteka koja se odbire je ranije opisana *Sklop.urdf* datoteka. Iduće je važno obratiti pozornost na korak definiranja grupa za planiranje, u tom koraku se daje naziv grupi zglobova koji će se kasnije koristiti u kodu kako bi *Moveit* znao sa kojom skupinom zglobova se želi upravljati. Nakon toga slijedi odabir pred definiranih poza koje su proizvoljne. Kod postavljanja upravljača treba pratiti upute, ali će se kasnije nakon generiranja konfiguracijskih datoteka morati promijeniti naziv pod *controller_list* unutar stvorenog direktorija *config*, datoteke *ros-controllers.yaml* prema (Slika 16). Razlog tome je što najnovija verzija *Universal_Robots_ROS_Driver* paketa koristi *scaled_pos_joint_traj_controller* kao zadano nazivlje prilikom upravljanja robotskom rukom.

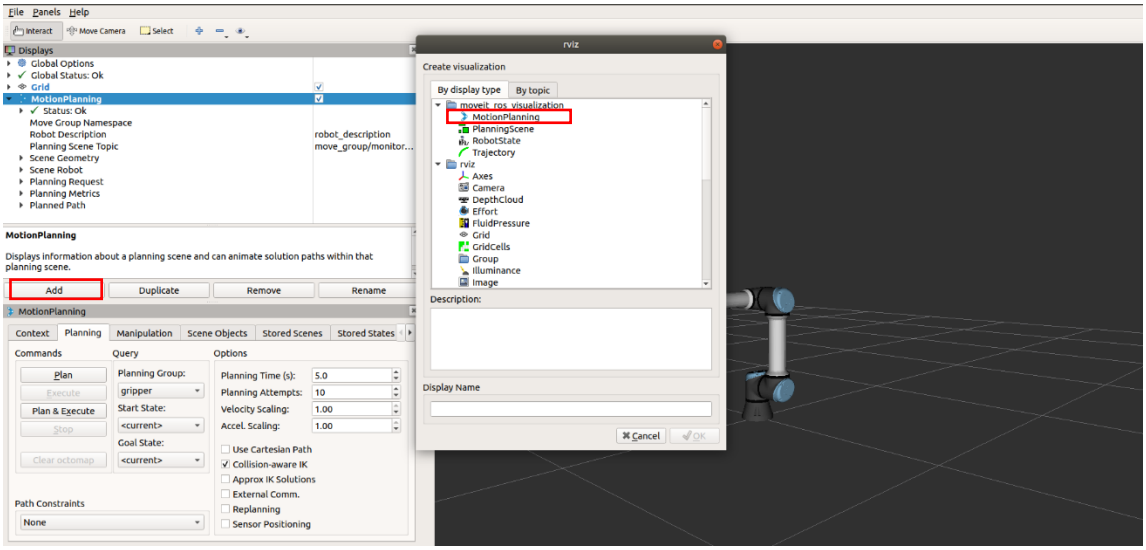
```
25 controller_list:
26   - name: "scaled_pos_joint_traj_controller"
27     action_ns: follow_joint_trajectory
28     default: True
29     type: FollowJointTrajectory
30     joints:
31       - shoulder_pan_joint
32       - shoulder_lift_joint
33       - elbow_joint
34       - wrist_1_joint
35       - wrist_2_joint
36       - wrist_3_joint
37   - name: gripper_controller
38     action_ns: follow_joint_trajectory
39     default: True
40     type: FollowJointTrajectory
41     joints:
42       - finger_joint
```

Slika 16.Datoteka *ros-controllers.yaml*

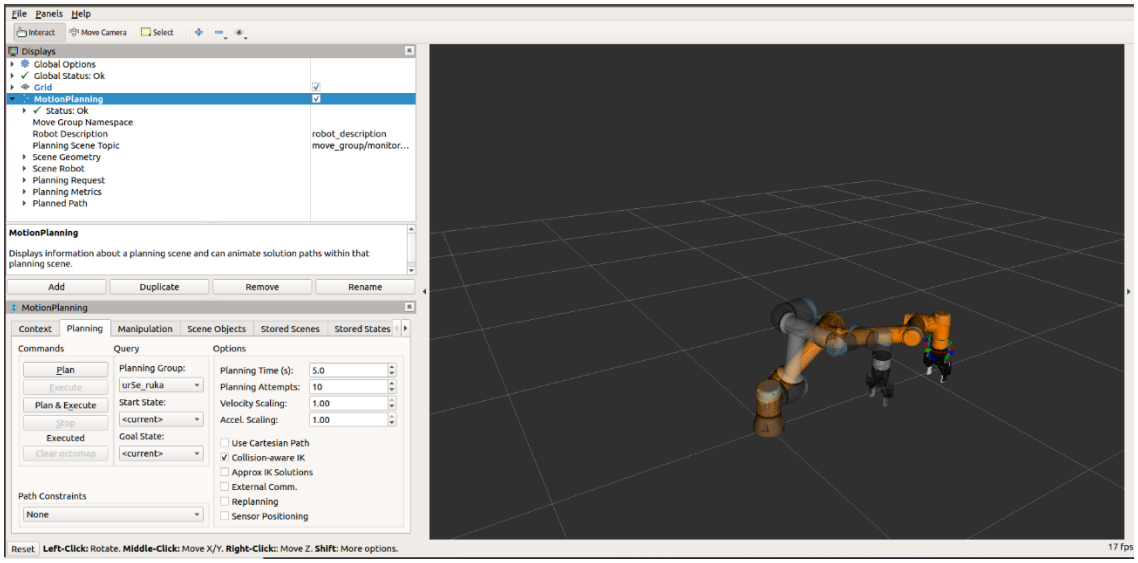
3.6. Simulacija u RVIZ-u

Najjednostavniji prikaz te ujedno i najbrži način za provjeru radi li sve je prikaz simulacije u RVIZ-u. Za tu svrhu *Moveit setup assistant* je generirao datoteku koja po imenom *demo.launch* unutar *launch* direktorija. *Demo.launch* između ostalog poziva *urdf* datoteku, koja se koristila prilikom izrade paketa, te RVIZ kako bi se unutar RVIZ sučelja dobio simulirani robot. Kako bi se robotom upravljalo unutar RVIZ-a je moguće odabrati *Moveit* vizualizacijski čvor pod imenom *Motion planning*, odabirom tipke *Add* (Slika 17). *Motion planning* nam omogućuje

upravljanje robotom izravno iz RVIZ-a (Slika 18.) Drugi način upravljanja robotom je uz pomoć *python* skripte. Potonji način će se obraditi u idućim poglavljima.



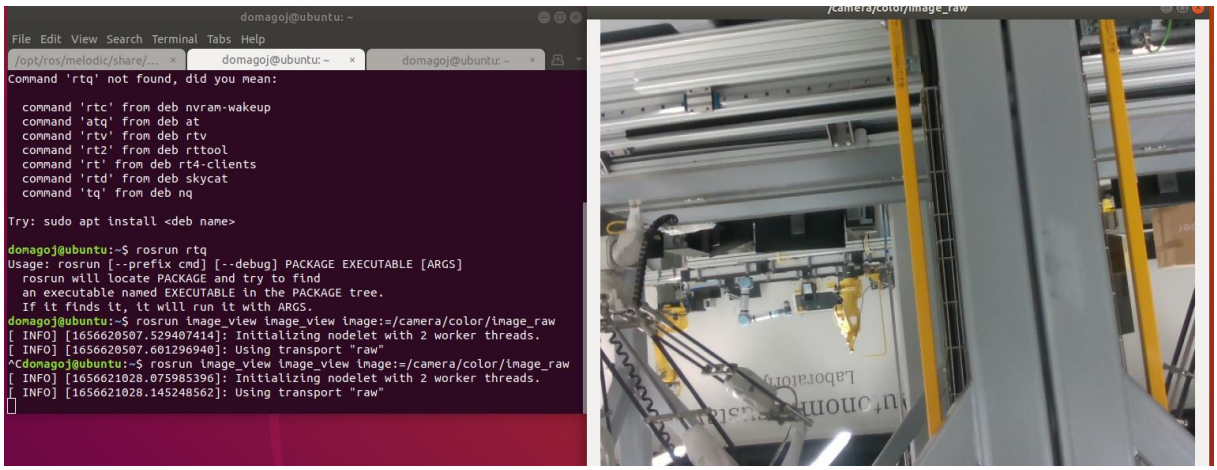
Slika 17. RVIZ sučelje



Slika 18. Prikaz kretanja u RVIZ-u

4. VIZIJSKI PROCES

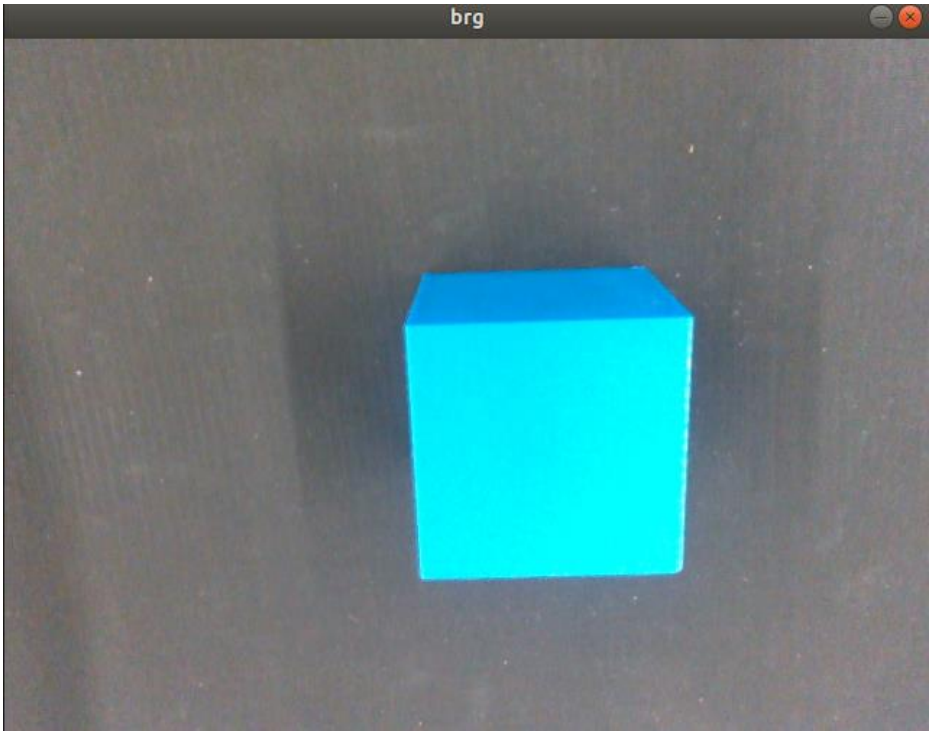
Kako bi robot sortirao po boji potreban mu je vizijski sustav. U ovom radu koristimo se Intel RealSense D435 kamerom. Nakon instalacije potrebnih paketa (kako je navedeno u poglavlju 3.4) potrebnih kako bi se kamerom upravljalo preko ROS-a preostaje još spojiti kameru na vanjsko računalo preko kojeg će se izvršavati obrada podataka. Pokretanje čvora kamere se izvršava upisivanjem *roslaunch realsense2_camera rs_rgbd.launch* komande u terminalu. Kamera na ROS šalje podatke koji su tipa *uint16*. Najjednostavniji pretplatnik koji se može napraviti je upisivanje „*rosrun image_view image_view image:=/camera/color/image_raw*“ komande u terminalu (Slika 19). Tema na koji se ovaj pokrenuti čvor spaja je *camera/color/image_raw*. Iako ROS može čitati navedeni tip podatka i pri tome dati sliku za potrebe prepoznavanja boje potreban je tip podataka koji se može obrađivati. Za obradu podataka koristi će se OpenCV. OpenCV je biblioteka otvorenog pristupa (eng. *open source*) vizijski program čije će se funkcije koristiti kako bi sliku koju kamera daje raščlanili na elemente koji će pomoći u određivanju boja sa slike. Prvo je potrebno sliku koju daje kamera prebaciti u takav tip podataka koje OpenCV može obratiti. To se radi pomoću funkcije *CvBridge* koji koristimo prema (Slika 20). Slika se u ovom trenutku prikazuje u *brg* (eng. *blue, green, red*) formatu(Slika 21). *Brg* format je analogan *rgb* formatu slike, ali sa obrnutim poretkom boja (eng. *red, green, blue*).



Slika 19. Prikaz neobrađene slike kamere

```
16      bridge = CvBridge()
17      cv2_img = bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
```

Slika 20. Funkcija *CvBridge*

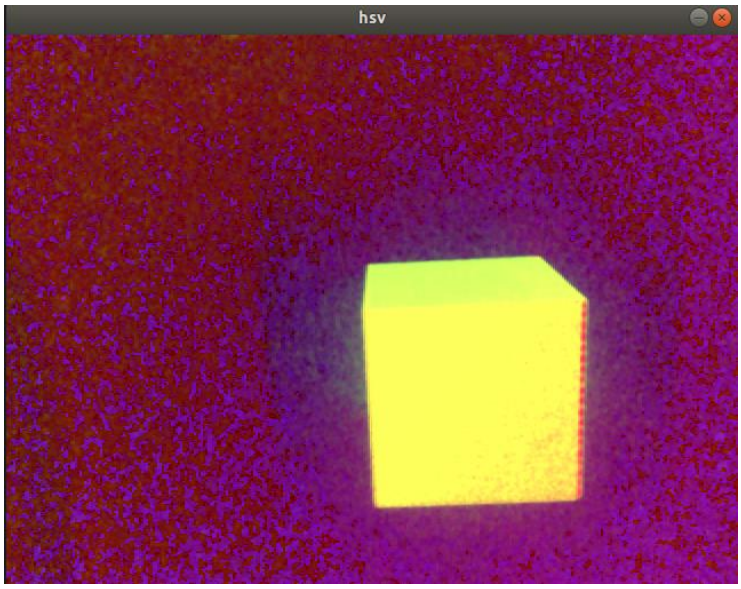


Slika 21. *Brg* format slike

S obzirom da je očitavanje boje u *brg* formatu vrlo neprecizno u kontekstu robotske primjene se koristi *hsv* format. Za pretvorbu *brg* formata u *hsv* koristi se *cvtColor* funkcija unutar OpenCv-a. Ova je funkcija višenamjenska, ali će se u ovom radu koristiti samo *COLOR_BGR2HSV* opcija(Slika 22). Uz pomoć funkcije *imshow* se dobiva prikaz slike nakon zadnje modifikacije (Slika 23).

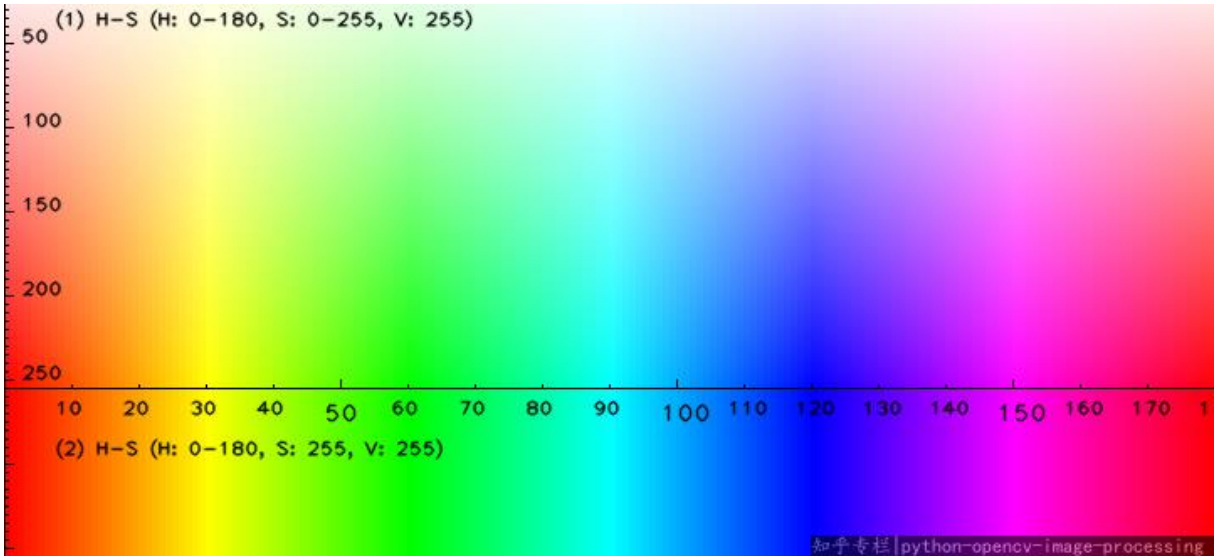
```
18        slika = cv2.cvtColor(cv2_img,cv2.COLOR_BGR2HSV)
```

Slika 22. *cvtColor* funkcija



Slika 23. Prikaz slike u *hsv* formatu

Kako bi izdvojili željenu boju potrebno je postaviti granice za tu boju unutar *hsv* polja prema (Slika 24). Za primjer je odabrana plava boja čije su granice [90, 110, 70] i [128, 255, 255] gdje prvi broj predstavlja nijansu (eng. *hue*), drugi zasićenost (eng. *saturation*) i treći vrijednost (eng. *value*). Funkcijom *inRange* se slika dijeli na područja koje se nalaze unutar navedenog raspona i na područja koja nisu (Slika 25). Na slici se nakon ovih izmjena uočavaju obrisi predmeta kojeg promatramo, ali tako da je pozadina u potpunosti tamna (Slika 26).

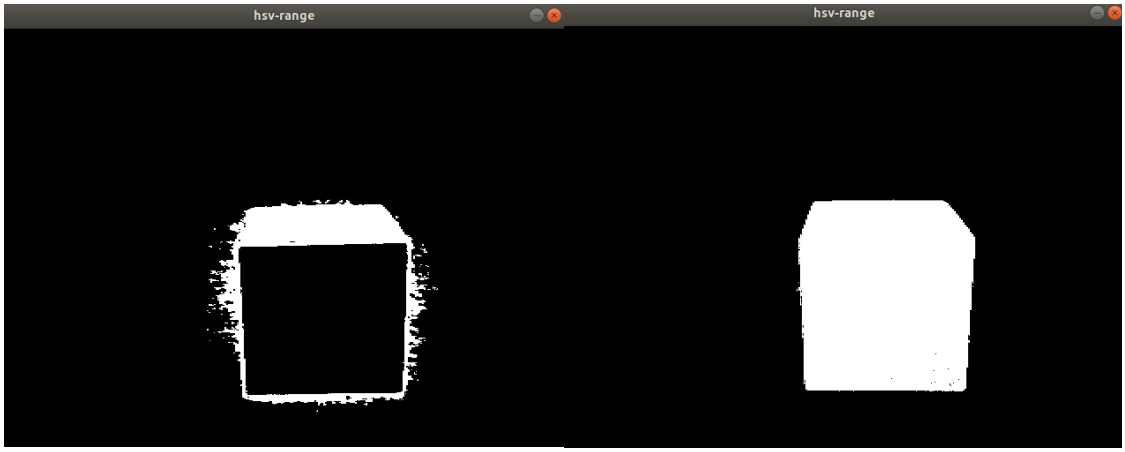


Slika 24. *Hsv* raspon boja

```
21          boundaries = [([90, 110, 70], [128, 255, 255])
22          counter = 0
23          for (lower, upper) in boundaries:
24              lower = np.array(lower, dtype = "uint8")
25              upper = np.array(upper, dtype = "uint8")
26              output = cv2.inRange(slika, lower, upper)
```

Slika 25. Prikaz funkcije *inRange*

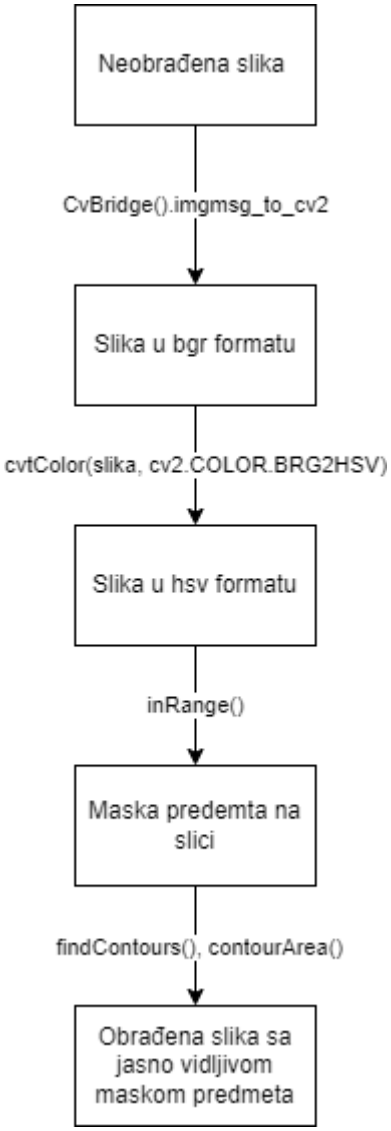
a) b)



Slika 26. a) Prikaz slike nakon uvođenja granica, b) Slika obrađena funkcijom *contourArea*

Kako je vidljivo na (Slika 26 a)) unutar granica se ne nalazi cijeli promatrani predmet. Na to utječe osvjetljenje te kut padanja svjetla. Kako bi se problem riješio koristi se funkcija *findContours* uz pomoć koje će se pikseli očitani kao boja smatrati rubovima promatranog predmeta. Kako bi cijeli predmet bio registriran kao cjelina (uključujući piksele koji nisu unutar granica radi osvjetljenja) koristi se funkcija *contourArea* (Slika 26 b)).

Zadnji problem koji se javlja kod očitavanja slike su smetnje ili „šumovi“. Radi nesavršenosti kamere određeni pikseli povremeno ulaze unutar postavljenih granica. Kao rješenje ovog problema u ovom radu uvodi se *if* grananje unutar *python* skripte uz pomoć koje određujemo najmanju veličinu konture koju promatramo. Primer toka obrade slike se može vidjeti na (Slika 27).



Slika 27. Dijagram obrade slike

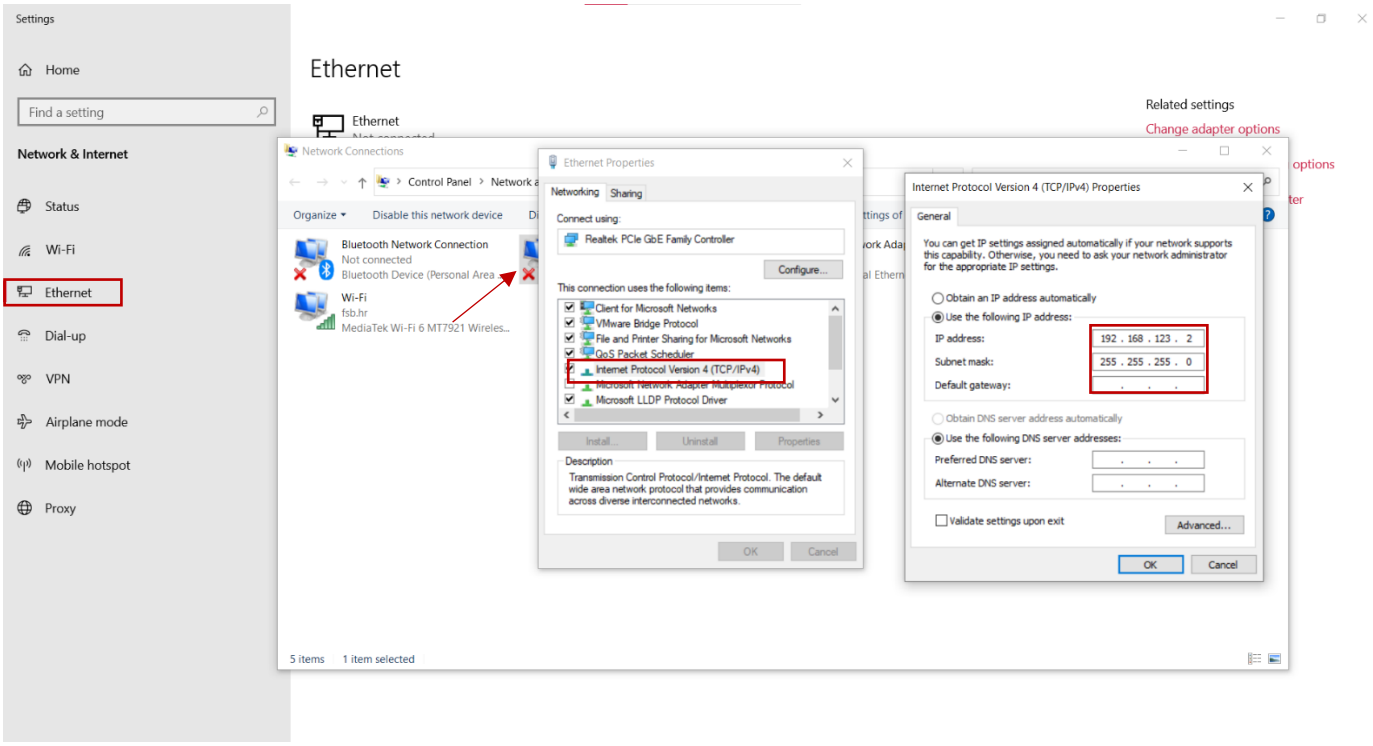
5. EKPERIMENTALNA VALIDACIJA

5.1. Uvod

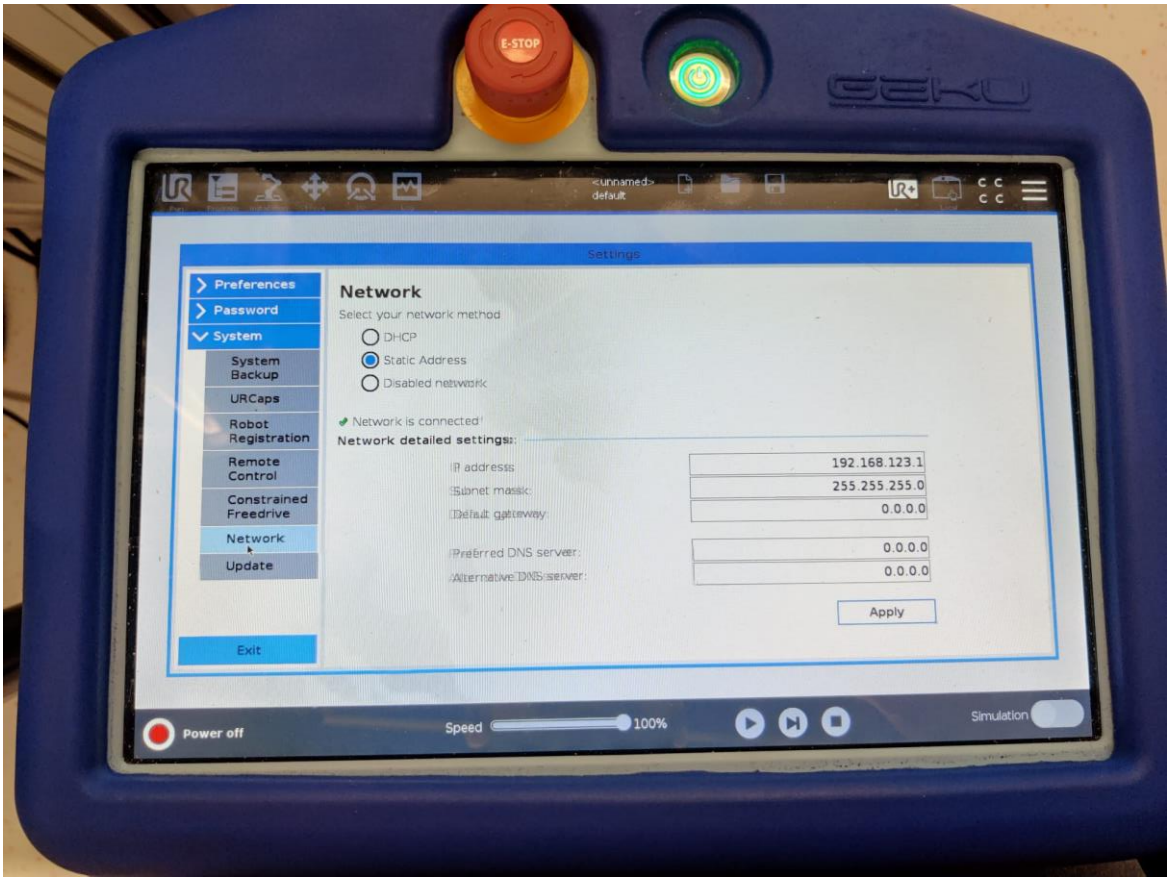
Kao pokazni primjer rada robota, hvataljke i kamere preko ROS-a odabran je „Pick and Place“ proces. Kako bi robotom upravljali preko ROS-a potrebno je pokrenuti komunikacijsku čvor koja se u ovom radu pokreće uz pomoć *my_launch_file* datoteke. Kada je komunikacija s ROS-a i robota spremna pokreće se Moveit upravljački čvor pod nazivom *UR5e_robotiq_moveit_planning_execution.launch.* Ovaj čvor provjerava popis zglobova robota te pokreće Moveit planiranje i izvršavanje. Nakon što se inicijalizacija odradi u potpunosti može se pokrenuti RVIZ kao vizualizacijski alat. Za upravljanje hvataljkom i kamerom potrebno je pokrenuti još dva zasebna čvora, *robotiq_action_server.launch* i *realsense2_camera rs_rgbd.launch*. Ovi čvorovi se mogu pokrenuti neovisno od prijašnje dvije.

5.2. UR5e i računalo

Ranije je spomenuto da se konekcija između robota i vanjskog računala ostvaruje ethernet kabelom. Kako bi konekcija ispravno radila potrebno je prilagoditi računalo tako da su robot i računalo na istoj mreži (Slika 28). IP adresa na robotu se namješta preko *teach pendant-a* (Slika 29). Dogovorom je određeno da robot ima IP adresu 192.168.123.1, računalo s operacijskim sustavom Windows 192.168.123.2, a virtualna mašina Linux Ubuntu 192.168.123.3. Bitno je provjeriti je li konekcija ostvarena, u Windows operativnom sustavu putem *cmd* prozora i komandom *ping 192.168.123.1* (adresa robota), analogno tome u Linux Ubuntu unutar terminala. Oba prozora moraju vratiti gubitak od 0%.



Slika 28. Upute za postavljanje konekcije



Slika 29. Konfiguracija na *teach pendant-u*

Prije no što je moguće upravljati robotskom rukom potrebno je napisati program koji će nam omogućiti vanjsko upravljanje robotom. Program se piše prema uputama [17]. Idući korak je izvući kalibraciju robota prema [1]. Prethodni korak nije nužan, ali se preporučuje. Prema *ur5e_bringup.launch* izrađuje se korisnička datoteka, konkretno u ovom radu datoteka je nazvana *my_launch_file*, koja se nalazi u prethodno spomenutom *my_setup* paketu unutar *launch* direktorija. Izmjene u odnosu na „original“ uključuju dodavanje spomenute kalibracijske datoteke pod nazivom *my_robot_calibration* i dodavanje IP adrese robota izravno u korisničku datoteku (Slika 30). U ovom se radu prilikom detaljnog praćenja uputa javljao problem gubitaka podataka između robota i računala (Slika 31). Kako bi se problem riješio napravljene su dvije izmijene. Prva izmjena je dodavanje linije koda unutar *hardware_interface.cpp* datoteke koja se nalazi u *Universal_Robots_ROS_Driver* paketu (Slika 32). Razlog je što se ROS nalazi na virtualnoj mašini, pa je komunikacije sporija nego kada bi se ROS nalazio na operativnom sustavu računala. Druga izmjena se radi na *teach pendantu* .

Naime radi se o dodavanju ugrađene funkcije čekanja prema (Slika 33). Sada je moguća komunikacija ROS-a i UR5e robota, pokretanjem ranije navedene *my_launch_file* datoteke nakon postavljanja robota u radno stanje pritiskom na *ON* i namještanjem odgovarajuće težine za *Active payload* prema [8] (Slika 34). U slučaju da je komunikacija uspješna na terminalu se ispisuje iduća poruka (Slika 35). Nakon pritiska tipke *play* (Slika 34) u terminalu se ispisuje iduća poruka (Slika 36).

```
4   <arg name="robot_ip" default="192.168.123.1"/>

14  <arg name="kinematics_config" default="$(find my_setup)/etc/my_robot_calibration.yaml"
```

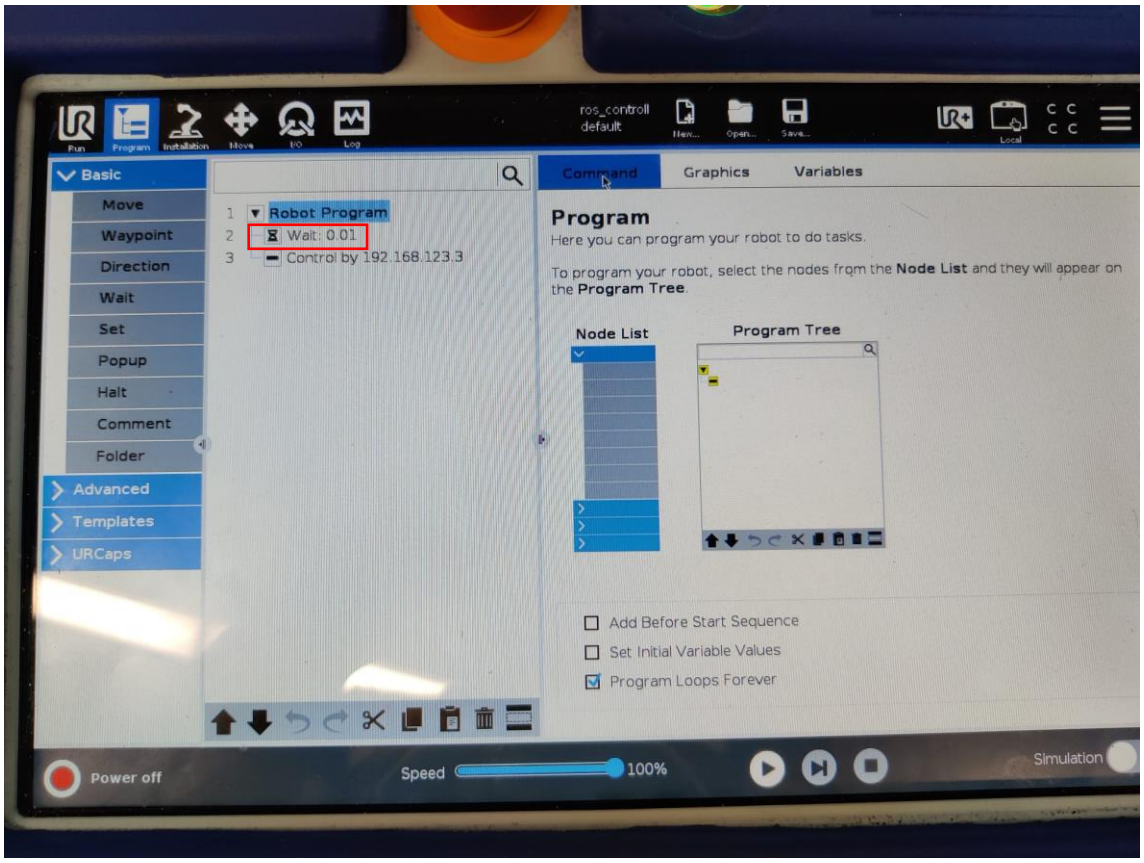
Slika 30. Dodatci *my_launch_file*-u

```
[ INFO] [1643621526.737159877]: Connection to reverse interface dropped.
[ERROR] [1643621527.077103309]: Can't accept new action goals. Controller is not running.
[ERROR] [1643621527.369324909]: Can't accept new action goals. Controller is not running.
[ERROR] [1643621527.680257409]: Can't accept new action goals. Controller is not running.
[ERROR] [1643621527.9004096229]: Can't accept new action goals. Controller is not running.
[ERROR] [1643621533.401119490]: Can't accept new action goals. Controller is not running.
```

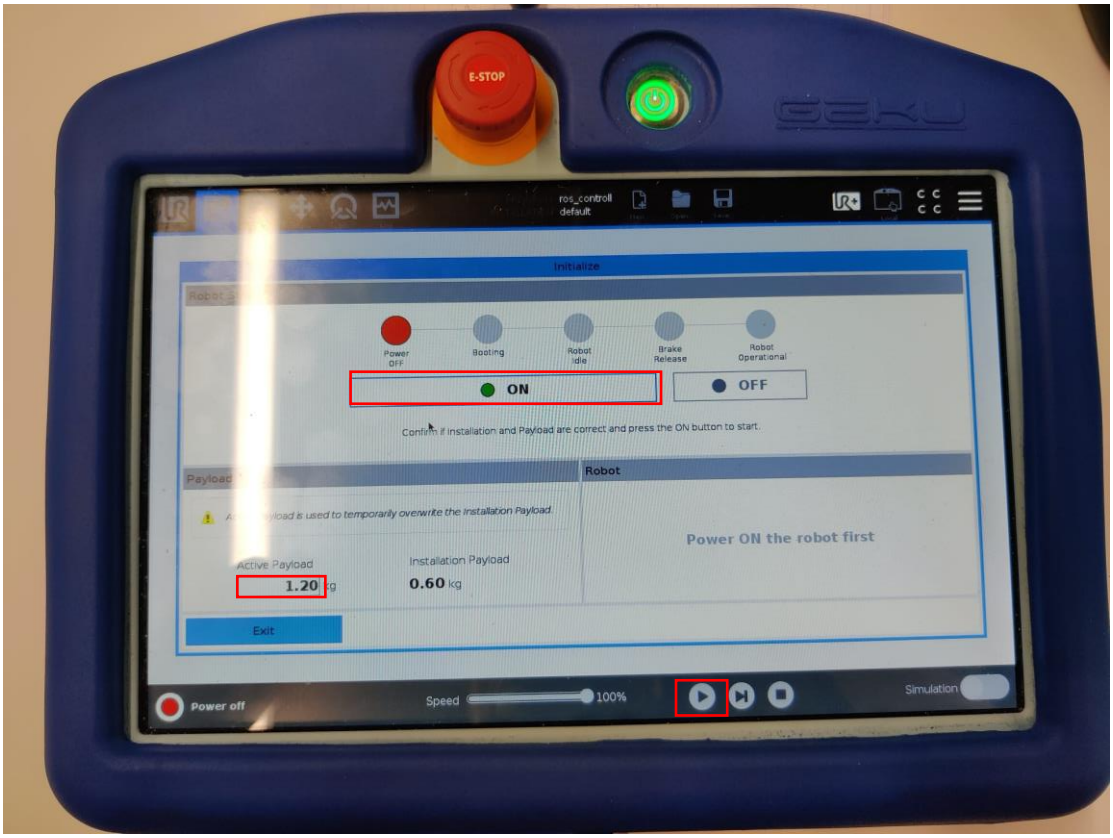
Slika 31. Primjer greške

```
311       std::bind(&HardwareInterface::passthroughTrajectoryDoneCb, this, std::placeholders::_1));
312       ur_driver_->setKeepaliveCount(5);
```

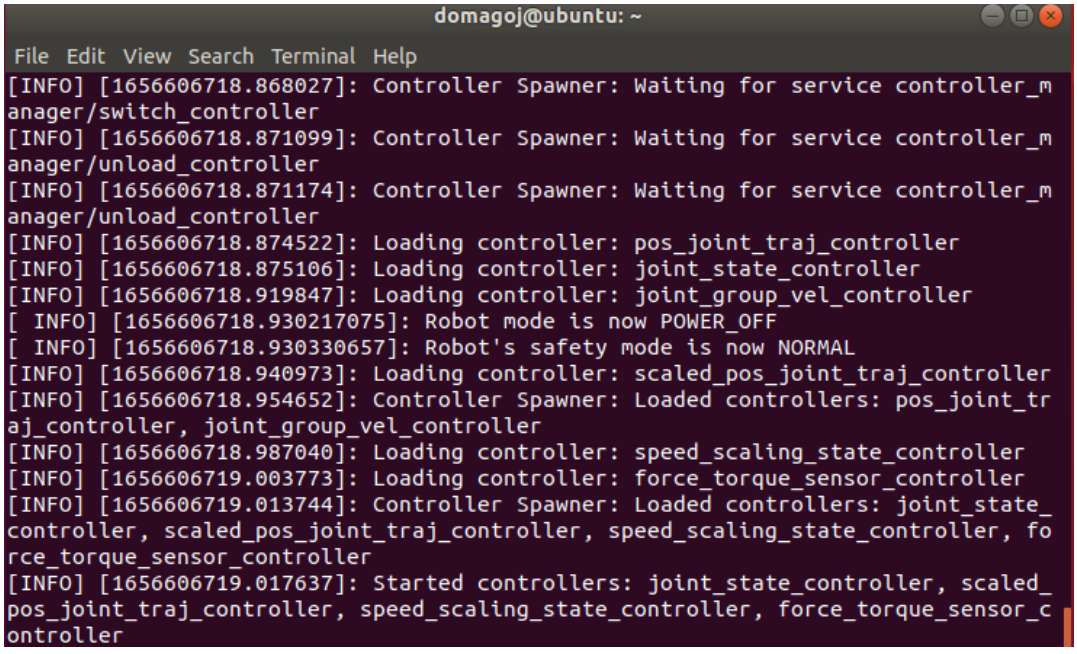
Slika 32. *Hardware_interface* datoteka



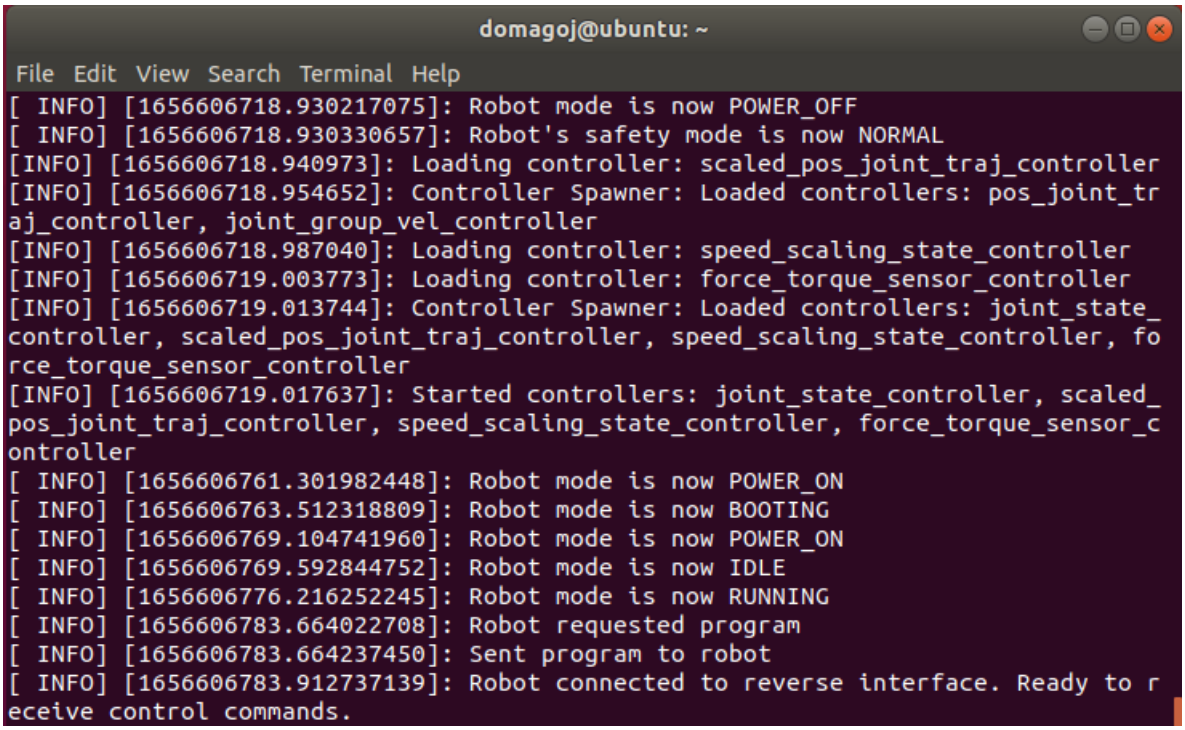
Slika 33. Program na *teach pendantu*



Slika 34. Priprema za pokretanje robota

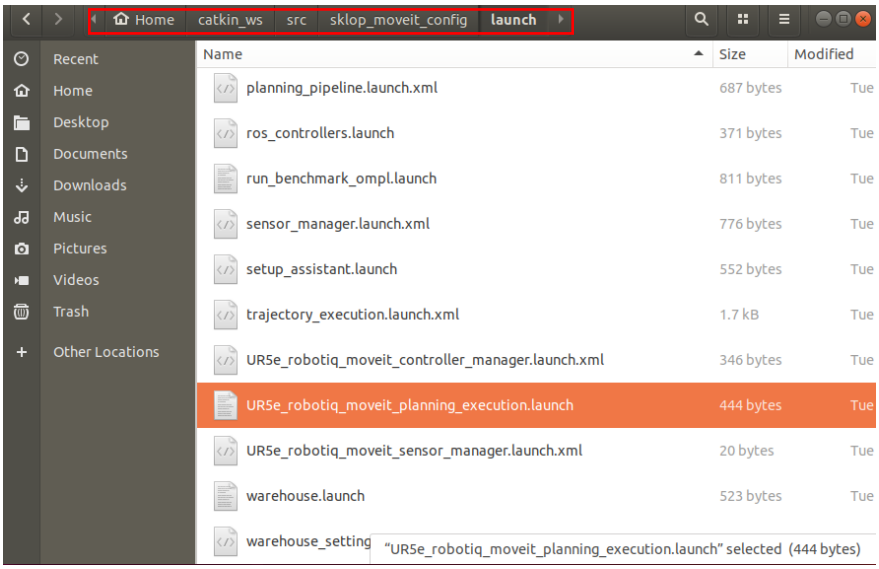


Slika 35. Uspješna komunikacija robota i ROS-a

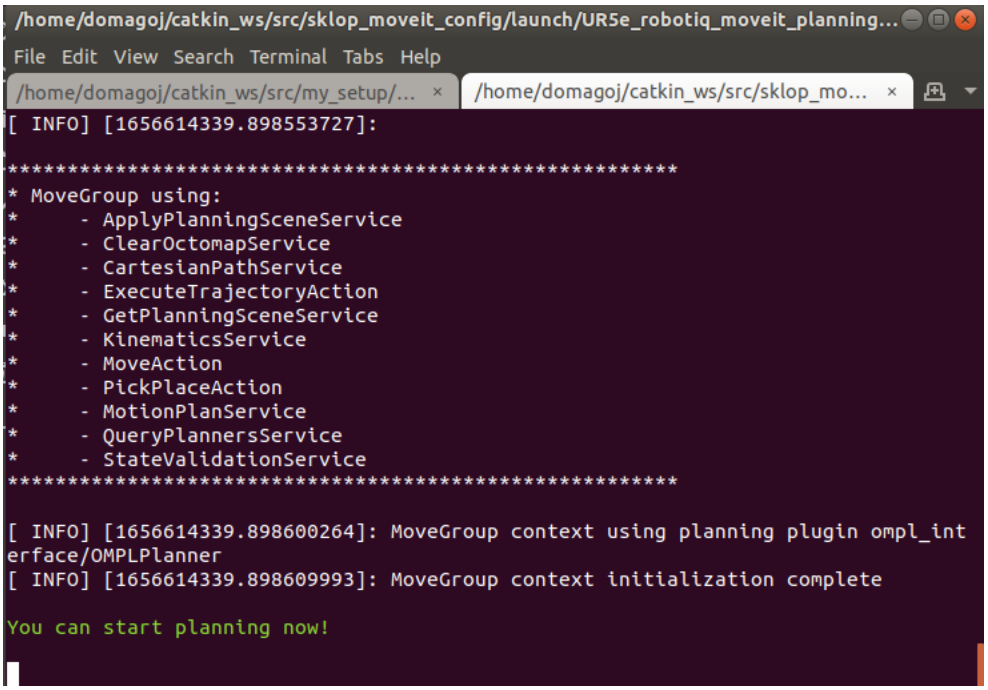


Slika 36. Prikaz stabilne komunikacije

Kako bi upravljanje robotom te željenom grupom zglobova preko Moveit-a bilo moguće potrebno je inicirati izvršni čvor. Po uzoru na [1] stvorit će se datoteka pod nazivom *UR5e_robotiq_moveit_planning_execution.launch*, te će se smjestiti u *launch* direktorij unutar *sklop_moveit_config* paketa (Slika 37). Nakon što se sve svi procesi prilikom pokretanja završe i na terminalu se ispiše (Slika 38), pokretanjem pravilno strukturirane *python* skripte ROS će preko Moveit-a biti u stanju izvršiti zadane naredbe.



Slika 37. Korisnička datoteka *UR5e_robotiq_moveit_planning_execution.launch*

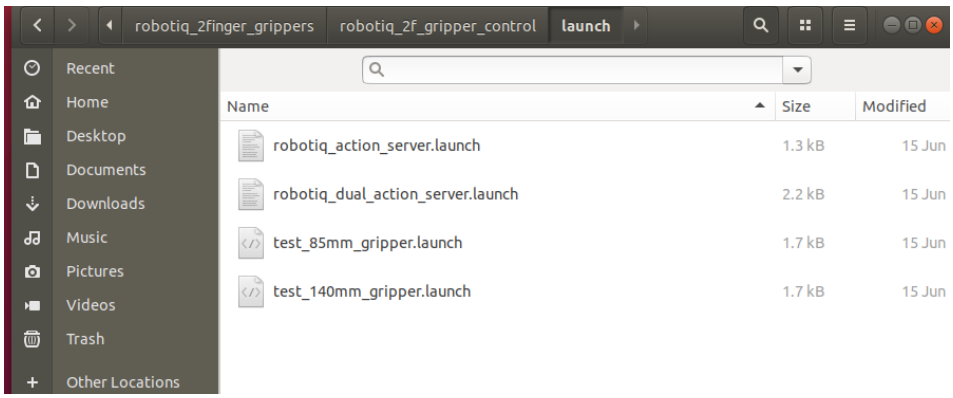


Slika 38. Prikaz uspješno pokrenutog izvršnog čvora

5.3. Robotiq 2f-85 hvataljka i računalo

LED lampica hvataljke će zasvijetliti crvenom bojom (Slika 39) čim joj se spoji napajanje. Kako bi se omogućila komunikacija s računalom moramo osigurati USB konekciju. Za razliku od robota hvataljkom ne upravlja Moveit direktno već je za hvataljku potrebno pokrenuti dodatni čvor. Ovaj čvor se pokreće iz datoteke *robotiq_2finger_grippers [2]* paketa, unutar direktorija *robotiq_2f_gripper_control, robotiq_action_server.launch* (Slika 40). Čvor stvara server preko kojeg hvataljka komunicira sa ROS-om. Također server omogućava korištenje *python* i klasa za kontrolu hvataljke. Prilikom pokretanja čvora za upravljanje hvataljkom, hvataljka će promijeniti boju LED lampice u plavu (Slika 41) te će izvršiti inicijalno zatvaranje i otvaranje prstiju.

Slika 39. Hvataljka spojena na napajanje



Slika 40. Lokacija *robotiq_action_server* datoteke

Slika 41. Hvataljka uspješno spojena s računalom

5.4. Intel RealSense D435 kamera i računalo

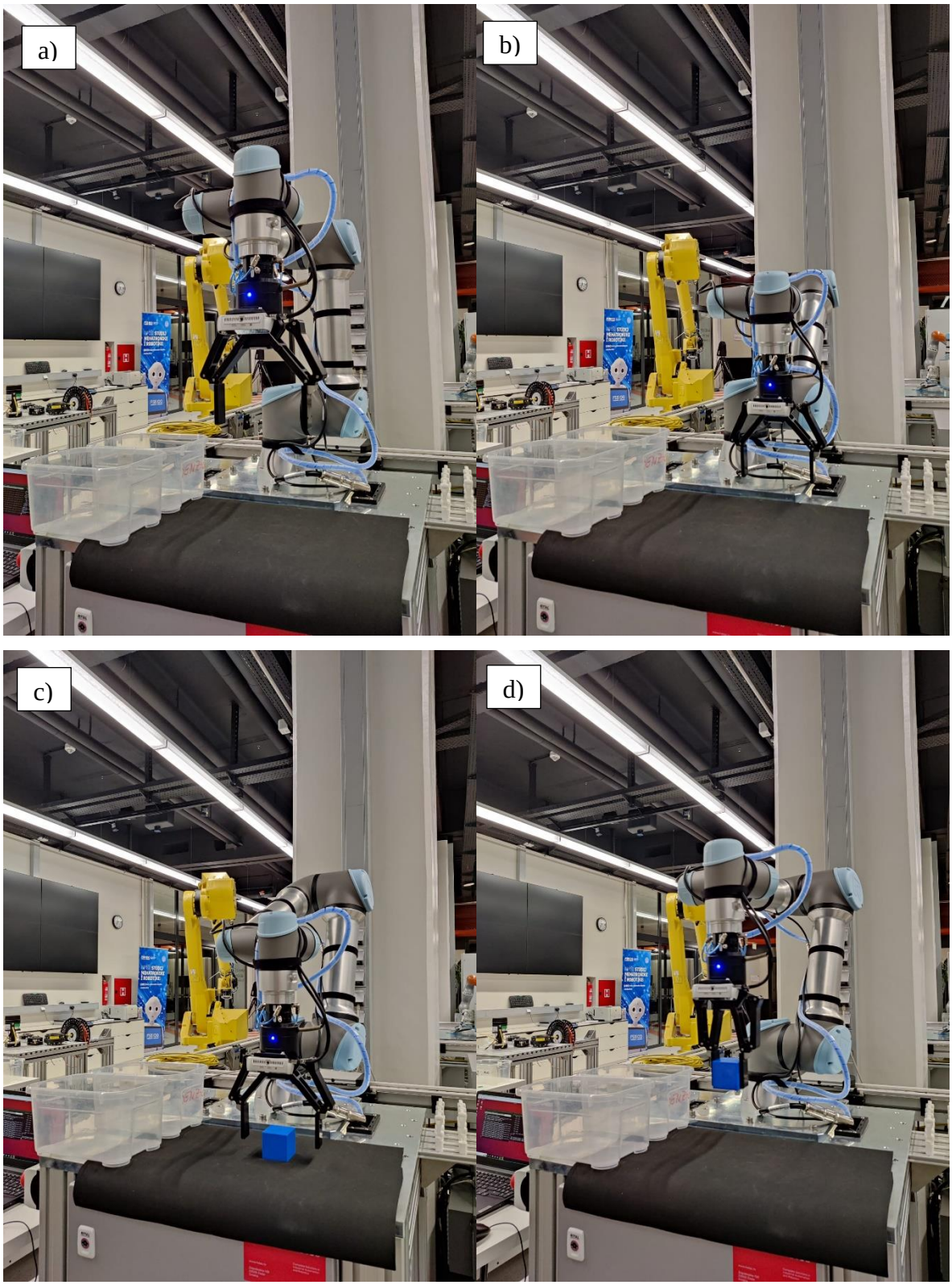
Kamera se s računalom spaja USB tipa A na USB tipa C (tip C je na strani kamere).

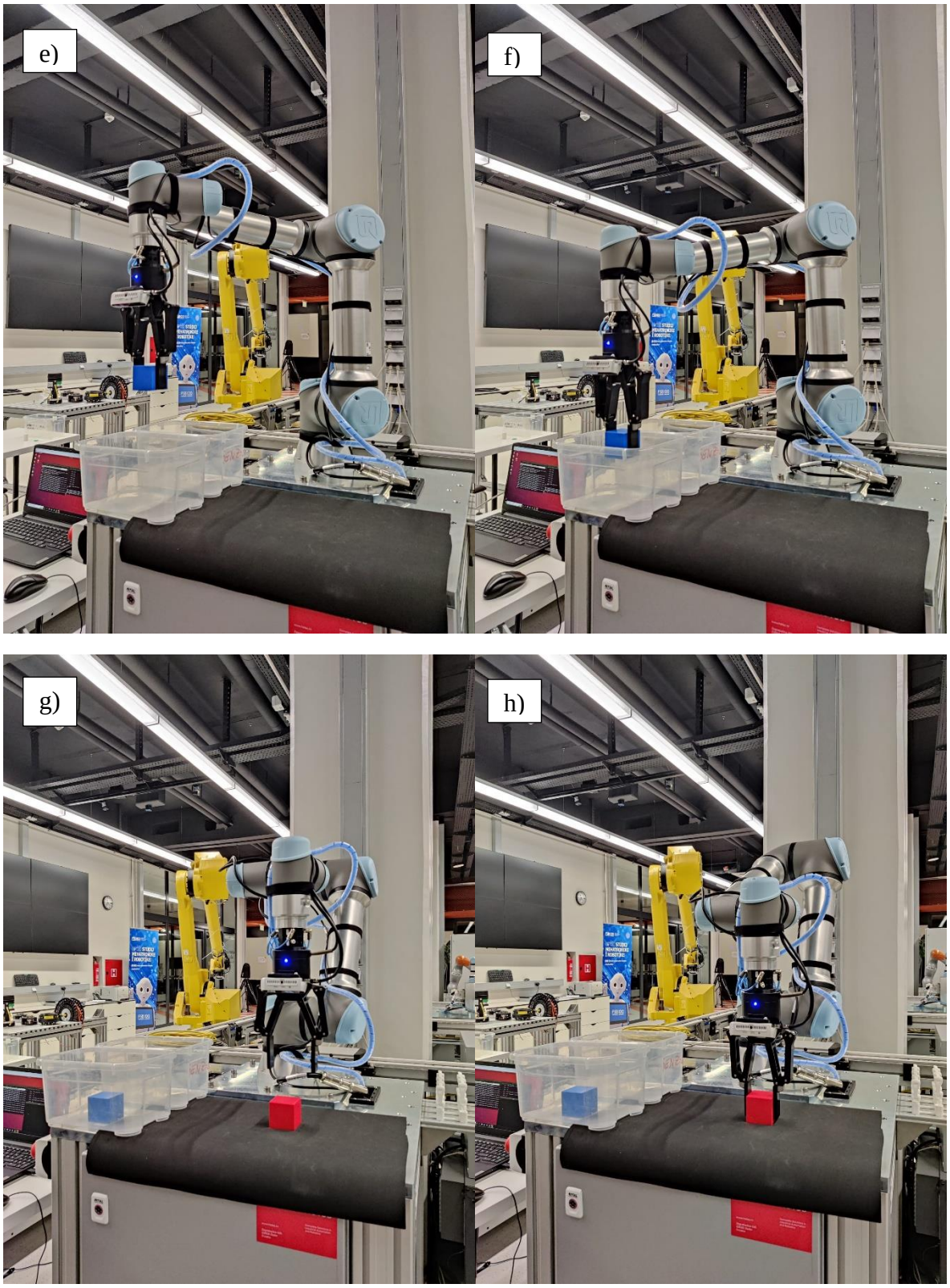
Pokretanje čvora kamere se izvršava upisivanjem *roslaunch realsense2_camera rs_rgbd.launch* komande u terminalu. Paket koji se ovom komandom poziva je instaliran je u prijašnjim koracima ovog rada te se on vodi kao integrirani paket unutar ROS-a. Kako bi se uvjerili da kamera radi može se pokrenuiti „*rosrun image_view image_view image:=/camera/color/image_raw"* komanda u terminalu. Na taj se način pretpati na temu slike koju kamera šalje.

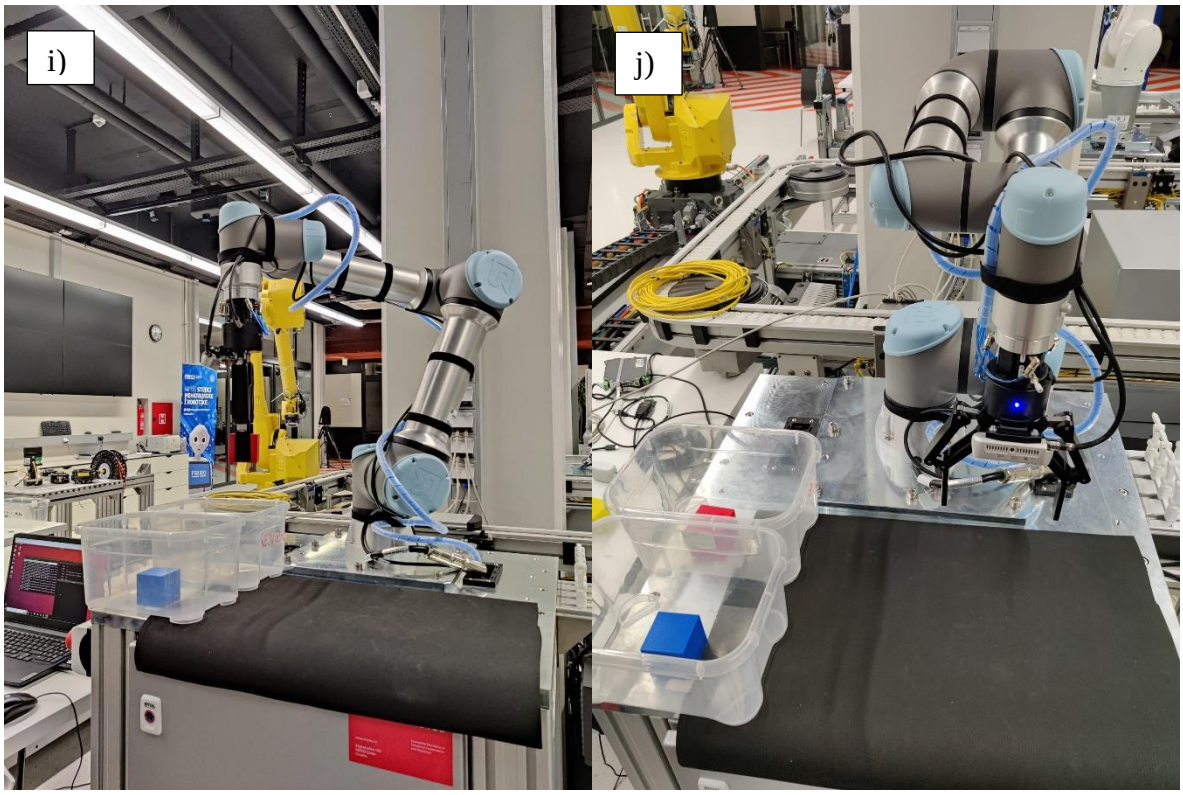
5.5. Primjer rada sustava

U ovom se radi izvodi „pick and place" proces koristeći navedeni hardver. Proces se odvija preko Moveit komadni unutar *python*-ove skripte. Kod je napisan tako da se nakon učitavanja

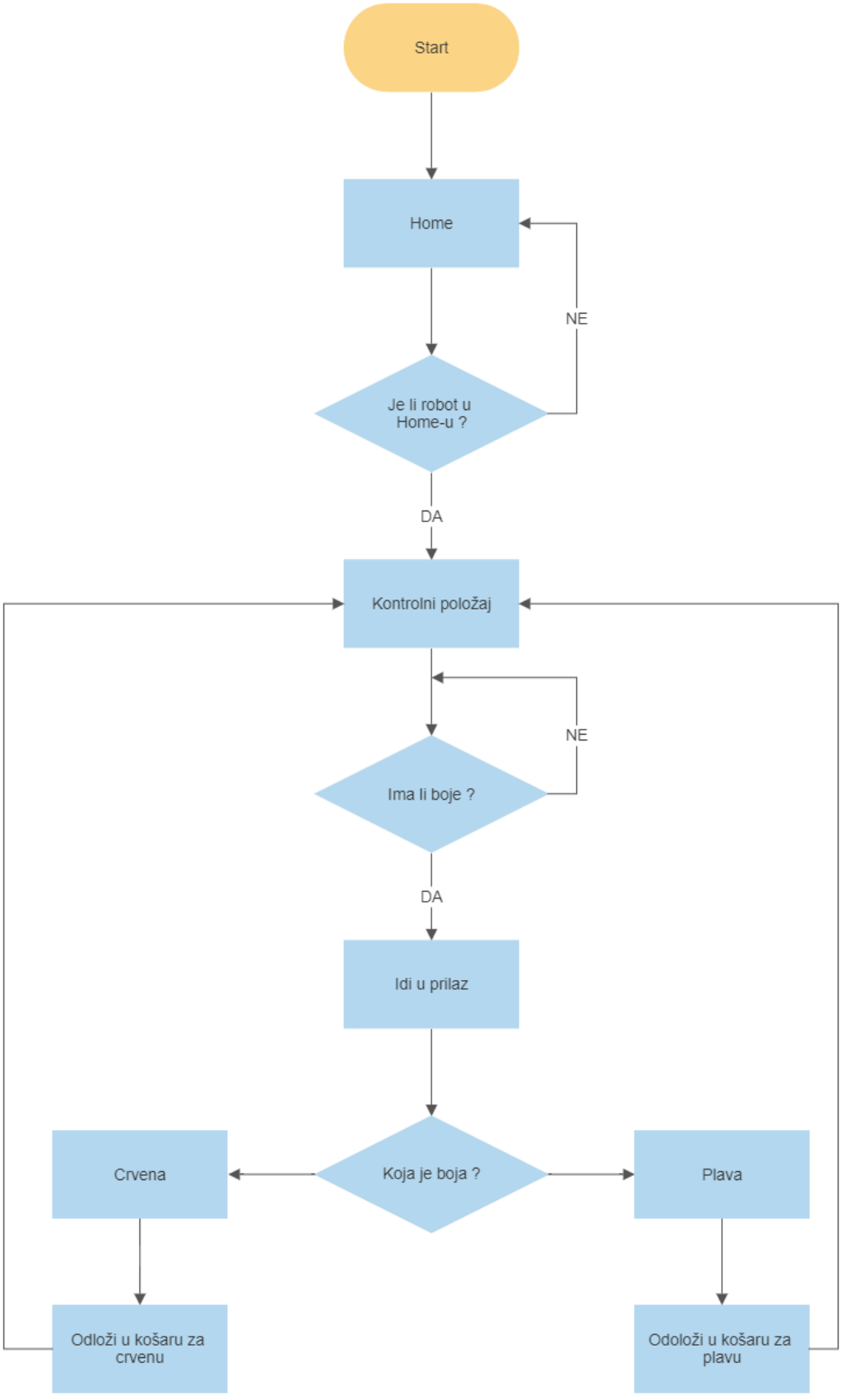
potrebnih operatora inicijaliziraju svi potrebni objekti. Također se funkcije Moveit-a pridodaju jednom objektu kojeg se kroz kod koristi za pozivanje funkcija. Rješenje zadatka je osmišljeno tako da se predmeti kreću po pokretnoj traci te dolaze na fiksirano mjesto izuzimanja. Robot kreće iz početnog *Home* položaja koji se u pravilu postavlja preko Moveit-ove integrirane funkcije *get_current_joint_values*. Funkcija dobavlja trenutne položaje zglobova robota te ih postavlja u željene vrijednosti. Nakon odlaska u početni položaj robot dolazi u kontrolnu točku funkcijom *set_pose_target* iz koje uz pomoć kamere određuje boju predmeta i na osnovi toga sprema predmete u odgovarajuće kutije. Sve kretnje osim odlaska u početni položaj se odvijaju u kartezijskom koordinatnom sustavu. Kod radi tako da se nakon dolaska robota u kontrolnu točku koristi funkcija *wait_for_message*. To je ROS-ova ugrađena funkcija i razlikuje se od pretplatničke funkcije po tome što čeka jednu poruku sa teme i nakon toga prekida komunikaciju. Pretplatnička funkcija nakon što primi informaciju odlazi u funkciju koju je korisnik odredio. Problem s time je što čak i ako je red čekanja informacija minimalan (minimalan red čekanja je jedna informacija) on će uvijek imati novu informaciju na čekanju. To znači da dok robot izvršava ostatak koda (sprema predmet određene boje na određeno mjesto) u redu čekanja mu je već nova boja, a ta nova boja je zapravo ista ona boja predmeta koji se u tom trenutku sortira. U trenutnoj konfiguraciji nakon sortiranja predmeta robot se vraća u kontrolnu točku i čeka idući predmet. Sve fiksne točke su napisane u odvojenoj tekstualnoj datoteci radi lakše izmjene ili modifikacije. Obrada boje se odrađuje uz pomoć *OpenCV* funkcija. Prvo se podatci moraju prebaciti iz formata koji ROS daje u takve koje će *OpenCV* funkcije moći razumjeti. *OpenCV* boje sa slike razmatra u *bgr* formatu, a za ovaj zadatak je potreban *hsv* format. Koristeći funkciju koju nudi *OpenCV* pod nazivom *COLOR_BGR2HSV* prilagođavamo sliku potrebama zadatka. Idući korak je odrediti granice unutar *hsv*-a za boje koje očekujemo kod predmeta. Nakon toga funkcijom *inRange* izdvajamo slike na kojima se nalaze boje koje očekujemo od onih na kojima tih boja nema. Ako je boja nađena i ima više od određenog boja piksela (u ovom slučaju više od 5000 piksela) stvara se kontura unutar koje se proglašava jednoznačna boja. Taj proces je tu kako bi se eliminirali šumovi.


a)
b)
c)
d)


e)
f)
g)
h)



Slika 42. a) Početna pozicija, b) Kontrolna točka, c)-f) sortiranje plavog objekta, g) Kontrolna točka, h)-j) sortiranje crvenog objekta



Slika 43. Dijagram toka

6. ZAKLJUČAK

Iako je ROS vrlo moćan i koristan alat krivulja učenja je vrlo strma i potrebno je značajno uloženo vrijeme kako bi se steklo tek osnovno znanje. S velikim brojem funkcija koje sadržava nudi nam mnoge opcije pristupu problemu. Velika prednost ROS-a je ta što je dostupan svakome tko ima pristup računalu te se njime mogu baviti ljudi iz cijelog svijeta. To uvelike povećava protok informacija i projekata koji se dijele sa svim zainteresiranim sadašnjim i budućim korisnicima ROS-a. ROS nam također nudi visoku razinu modularnosti. To znači da je izmjena komponenti sustava moguća uz minimalnu izmjenu programa napisanog u programskim jezicima koje ROS podržava (*cpp, python*).

ROS je još uvijek u razvoju te mi je iznimno drago što sam dio te zajednice koja pomiče granice robotike i automatizacije [18].

LITERATURA

[1] https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

[2] https://github.com/Danfoa/robotiq_2finger_grippers

[3] https://moveit.ros.org/install/

[4] Universal Robots: User Manual UR5e 2020.

[5] https://www.businesswire.com/news/home/20220523005472/en/Universal-Robots-Conquers-New-Automation-Frontiers-at-Automate-2022

[6] https://store.clearpathrobotics.com/products/ur5e

[7] https://robodk.com/robot/Universal-Robots/UR5e

[8] https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_Instruction_Manual_e-Series_PDF_20190206.pdf

[9] https://www.intelrealsense.com/depth-camera-d435/

[10] https://github.com/

[11] http://wiki.ros.org/ROS/Tutorials

[12] http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

[13] https://github.com/ros

[14] http://wiki.ros.org/ROS/Tutorials/CreatingPackage

[15] https://github.com/IntelRealSense/realsense-ros

[16] https://roboticscasual.com/ros-tutorial-how-to-create-a-moveit-config-for-the-ur5-and-a-gripper/

[17] https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/blob/master/ur_robot_driver/doc/install_urcap_e_series.md

[18] https://github.com/StoneKnight22/ros_ur5e

PRILOG 1: TEKSTUALNA DATOTEKA FIKSNIH TOČAKA

I. 0.134264597371 -0.202172371216 0.316402921142 -0.691119122783
0.111528949086 0.699398778267 0.144072900291 kontrolna tocka

II. 0.105 -0.340 0.301 -0.691119122783 0.111528949086 0.699398778267
0.144072900291 izuzimanje

III. -0.244584369691 -0.447681987305 0.335907006643 -0.697036214179
0.140619697313 0.687266174864 0.148431202039 odlaganje crvena

IV. -0.232679416325 -0.240881668889 0.332772763365 -0.401003250599
0.589852403684 0.381450858285 0.588018518068 odlaganje plava

PRILOG 2: „PICK AND PLACE“ KOD

```
#!/usr/bin/env python
import time
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
from std_msgs.msg import String
from moveit_commander.conversions import pose_to_list, list_to_pose
import actionlib
from robotiq_2f_gripper_msgs.msg import CommandRobotiqGripperFeedback,
CommandRobotiqGripperResult, CommandRobotiqGripperAction,
CommandRobotiqGripperGoal
from robotiq_2f_gripper_control.robotiq_2f_gripper_driver import
Robotiq2FingerGripperDriver as Robotiq
import cv2
from cv_bridge import CvBridge
from sensor_msgs.msg import Image
import numpy as np
def zapisivanje_tocaka(lista):
      tocka = geometry_msgs.msg.Pose()
      tocka.position.x = float(lista[0])
      tocka.position.y = float(lista[1])
      tocka.position.z = float(lista[2])
      tocka.orientation.x = float(lista[3])
      tocka.orientation.y = float(lista[4])
      tocka.orientation.z = float(lista[5])
      tocka.orientation.w = float(lista[6])
      return tocka
def kopiranje_tocke(zeljena_tocka):
      lista = pose_to_list(zeljena_tocka)
      kopija = geometry_msgs.msg.Pose()
      kopija.position.x = float(lista[0])
      kopija.position.y = float(lista[1])
      kopija.position.z = float(lista[2])
      kopija.orientation.x = float(lista[3])
      kopija.orientation.y = float(lista[4])
      kopija.orientation.z = float(lista[5])
      kopija.orientation.w = float(lista[6])
      return kopija
class Inicijalizacija():
      def __init__(self, hvataljka):
            moveit_commander.roscpp_initialize(sys.argv)
            robot = moveit_commander.RobotCommander()
            scene = moveit_commander.PlanningSceneInterface()
            group_name = "ur5e_ruka"
            move_group = moveit_commander.MoveGroupCommander(group_name)
            display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',moveit_msgs.msg.Disp
layTrajectory,queue_size=20)
            eef_link = move_group.get_end_effector_link()
            self.kontrolna_tocka = []
            self.image_topic = "/camera/color/image_raw"
```

```
            self.image_callback,queue_size=1)
            print robot.get_current_state()
            self.robot = robot
            self.scene = scene
            self.group_name = group_name
            self.move_group = move_group
            self.display_trajectory_publisher =
display_trajectory_publisher
            self.pozicija = geometry_msgs.msg.Pose()
            self.eef_link = eef_link
            tocke = open('tocke.txt',"r")
            lista_tocaka = tocke.readlines()
            self.kontrolna_tocka =
zapisivanje_tocaka(lista_tocaka[0].split('\t')) #prilaz za ocitanje
boje
            self.izuzimanje =
zapisivanje_tocaka(lista_tocaka[1].split('\t')) #prilaz za skupljanje
kocke(potreban pomak po z)
            self.prilaz = kopiranje_tocke(self.izuzimanje)
            self.prilaz.position.z += 0.1
            self.crvena =
zapisivanje_tocaka(lista_tocaka[2].split('\t')) #odlaganje crvene
kocke(potreban pomak po z)
            self.plava = zapisivanje_tocaka(lista_tocaka[3].split('\t'))
            offset = 0.230
            self.prilaz_crvena = kopiranje_tocke(self.crvena)
            self.prilaz_crvena.position.z += offset
            self.prilaz_plava = kopiranje_tocke(self.plava)
            self.prilaz_plava.position.z += offset
            self.hvataljka = hvataljka
            self.broj_boje = 0
            self.stream_slike = 0
            return
      def image_callback(self,msg):
            print "usao u callback"
            bridge = CvBridge()
            cv2_img = bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
            slika = cv2.cvtColor(cv2_img,cv2.COLOR_BGR2HSV)
            cv2.waitKey(1)
            boundaries = [([90, 110, 70], [128, 255, 255]),([0, 50, 50],
[10, 255, 255])]
            counter = 0
            for (lower, upper) in boundaries:
                  lower = np.array(lower, dtype = "uint8")
                  upper = np.array(upper, dtype = "uint8")
                  output = cv2.inRange(slika, lower, upper)
                  rubovi = cv2.findContours(output, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[1]
                  print rubovi
                  cv2.imshow("maska", output)
                  for boja in rubovi:
                        if 5000 < cv2.contourArea(boja) :
                              if output.any():
                                    if counter == 0:
                                          print "vidim plavu"
                                          self.broj_boje = counter + 1
                                    elif counter == 1:
                                          self.broj_boje = counter + 1
```

```
                                    print "vidim crvenu"
                        break
            counter += 1
    def Home(self):
        move_group = self.move_group
        home = move_group.get_current_joint_values()
        home[0] = -pi/2
        home[1] = -pi/2
        home[2] =  pi/2
        home[3] = -pi/2
        home[4] =  3*pi/2
        home[5] = -67*pi/180
        move_group.go(home, wait=True)
        move_group.stop()
        print "doso sam kuci"
        time.sleep(1)
        return
    def micanje(self,b):
        rospy.sleep(2)
        print b
        kosare = [self.crvena,self.plava]
        tocke_prilaza = [self.prilaz_crvena,self.prilaz_plava]
        self.move_group.set_pose_target(self.prilaz)
        self.move_group.go(wait=True)
        self.move_group.set_pose_target(self.izuzimanje)
        self.move_group.go(wait=True)
        self.move_group.stop()
        self.hvataljka.polozaj_hvataljke(0)
        self.move_group.set_pose_target(self.prilaz)
        self.move_group.go(wait=True)
        self.move_group.set_pose_target(tocke_prilaza[b-1])
        self.move_group.go(wait=True)
        self.move_group.set_pose_target(kosare[b-1])
        self.move_group.go(wait=True)
        self.move_group.stop()
        self.hvataljka.polozaj_hvataljke(0.05)
        self.move_group.set_pose_target(tocke_prilaza[b-1])
        self.move_group.go(wait=True)
        self.move_group.set_pose_target(self.kontrolna_tocka)
        self.move_group.go(wait=True)
        self.move_group.stop()
        return
    def kontrolna_tocka(self):
        move_group = self.move_group
        tocka = geometry_msgs.msg.Pose()
        tocka.position.x = 0.134264597371
        tocka.position.y = -0.202172371216
        tocka.position.z = 0.316402921142
        tocka.orientation.x = -0.691119122783
        tocka.orientation.y = 0.111528949086
        tocka.orientation.z = 0.699398778267
        tocka.orientation.w = 0.144072900291
        move_group.set_pose_target(tocka)
        move_group.go(wait=True)
        move_group.stop()
        print "provjeravam boju"
        return
    def prilaz_izuzimanje(self):
```

```
            move_group = self.move_group
            tocka = geometry_msgs.msg.Pose()
            tocka.position.x = 0.105
            tocka.position.y = -0.340
            tocka.position.z = 0.316
            tocka.orientation.x = -0.691119122783
            tocka.orientation.y = 0.111528949086
            tocka.orientation.z = 0.699398778267
            tocka.orientation.w = 0.144072900291
            move_group.set_pose_target(tocka)
            move_group.go(wait=True)
            move_group.stop()
            print "provjeravam boju"
            return
      def izuzimanje(self):
            move_group = self.move_group
            tocka = geometry_msgs.msg.Pose()
            tocka.position.x = 0.105
            tocka.position.y = -0.340
            tocka.position.z = 0.291
            tocka.orientation.x = -0.691119122783
            tocka.orientation.y = 0.111528949086
            tocka.orientation.z = 0.699398778267
            tocka.orientation.w = 0.144072900291
            move_group.set_pose_target(tocka)
            move_group.go(wait=True)
            move_group.stop()
            print "provjeravam boju"
            return
      def uzmi_ostavi(self):
            self.Home()
            self.move_group.set_pose_target(self.kontrolna_tocka)
            self.move_group.go(wait=True)
            self.move_group.stop()
            rospy.sleep(2)
            print "kontroliram"
            while True:
                  self.stream_slike =
rospy.wait_for_message(self.image_topic,Image)
                  self.image_callback(self.stream_slike)
                  if self.broj_boje:
                        self.micanje(self.broj_boje)
                        self.broj_boje = 0
class Hvataljka():
      def __init__(self):
            action_name = rospy.get_param('~action_name',
'command_robotiq_action')
            self.robotiq_client =
actionlib.SimpleActionClient(action_name, CommandRobotiqGripperAction)
            self.robotiq_client.wait_for_server()
            rospy.logwarn("Client test: Starting sending goals")
            self.goal = CommandRobotiqGripperGoal()
            self.goal.emergency_release = False
            self.goal.stop = False
            self.goal.position = 0.1
            self.goal.speed = 0.1
            self.goal.force = 0.0
            self.robotiq_client.send_goal(self.goal)
```

```
            self.robotiq_client.wait_for_result()
            return
      def polozaj_hvataljke(self,raspon,speed=0.1,force=0):
            self.goal.position = raspon
            self.goal.speed = speed
            self.goal.force = force
            self.robotiq_client.send_goal(self.goal)
            self.robotiq_client.wait_for_result()
            rospy.sleep(0.2)
            return

if __name__ == '__main__':
      rospy.init_node('pickandplace', anonymous=True)
      hvataljka = Hvataljka()
      poziv_klase = Inicijalizacija(hvataljka)
      print "ulazim u f"
      poziv_klase.uzmi_ostavi(
      rospy.spin()
```