Izgradnja 3D modela prostorije pomoću robotske ruke i 3D kamere

Kolačko, Deni

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

Preddiplomski sveučilišni studij Računarstvo

IZGRADNJA 3D MODELA PROSTORIJE POMOĆU ROBOTSKE RUKE I 3D KAMERE

Završni rad

Deni Kolačko

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

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

Osijek, 20.09.2021.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Deni Kolačko
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4076, 28.07.2017.
OIB studenta:	79628730699
Mentor:	Prof.dr.sc. Robert Cupec
Sumentor:	Dr. sc. Petra Pejić
Sumentor iz tvrtke:	
Naslov završnog rada:	Izgradnja 3D modela prostorije pomoću robotske ruke i 3D kamere
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Vrlo dobar (4)
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: 1 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	20.09.2021.
Datum potvrde ocjene Odbora:	23.09.2021.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 29.09.2021.

Ime i prezime studenta:	Deni Kolačko
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4076, 28.07.2017.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Izgradnja 3D modela prostorije pomoću robotske ruke i 3D kamere**

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

Tema ovog završnog rada je izgradnja 3D slike prostorije uz pomoć UR5 robotske ruke i LiDAR KAMERE L515 u Python programskom jeziku. Cilj rada je izrada sustava koji se sastoji od robota i kamere kako bi uz pomoć dubinskih slika prostorije koje smo dobili, napravili 3D model prostorije. Ubrzani razvoj računala nam omogućuje da pomoću 3D rekonstrukcije u slučaju vansjkih ili unutrašnjih nezgoda obnovimo zgrade, kuće, prostorije. U grani medicine 3D rekonstrukcija se koristi kako bi se rekonstruirale medicinske pločice koje se kasnije prilagođavaju obliku kostiju koje se treba imobilizirati, u slučaju sportskih nezgoda ili ljudske dobi.

Zadatak je bio napraviti sučelje koje će pomicati robota na zadanu poziciju i kada robot dođe u zadanu poziciju, tada će kamera uzeti dubinsku sliku prostorije i spremiti sliku na računalo. Pri izradi navedenog sustava koristilo se znanje stečeno na fakultetu iz kolegija „Programiranje 1“ i „Objektno orijentirano programiranje“. Za upravljanje datotekama i njihovom instalacijom u Linux operacijskom sustavu koristilo se znanje iz kolegija „Operacijski sustavi“, a za transformaciju slika iz koordinatnog sustava kamere u koordinatni sustav baze robota koristilo se dodatno znanje iz područja robotike.

Rad se sastoji iz sljedećih poglavlja. U drugom poglavlju rada je opisana upotreba i dizajn Robotskog operacijskog sustava (engl. Robotic Operating System, ROS). Nakon toga slijedi opis i karakteristike UR5 robotske ruke i LiDAR kamere L515. U trećem je poglavlju opisano dobivanje 3D oblaka točaka prostorije te je analiziran kod koji je pisan u programskom jeziku Python. Konačno, prikazane su dobivene 3D slike prostorije povezane pomoću alata Meshlab.

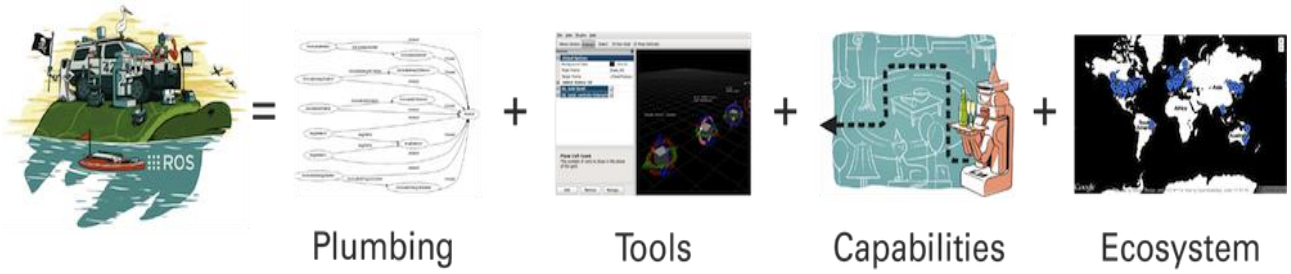
1.1. Zadatak završnog rada

Izraditi programsko rješenje u ROS (Robot Operating System) okviru za izgradnju 3D modela prostorije pomoću robotske ruke i 3D kamere, koje se sastoji od modula za odabir smjera snimanja kamere, modula za snimanje 3D slike, modula za fuziju snimljenih oblaka 3D točaka i modula za upravljanje robotskom rukom. Izrađeni sustav eksperimentalno ispitati izgradnjom 3D modela jedne ili više prostorija.

2. KOMPONENTE RAZVIJENOG SUSTAVA

2.1. ROBOTSKI OPERACIJSKI SUSTAV – ROS

Robotski operacijski sustav, ROS, nije zapravo operacijski sustav kao što je Windows ili Linux, već je skup softverskih okvira za razvoj softvera robota, implementacije često korištenih funkcionalnosti prenošenja poruka između procesa i upravljanje paketima. ROS je osmišljen kao program otvorenog tipa s ciljem da korisnici mogu izabrati skup alata i biblioteka koji su u interakciji s jezgrom ROS-a, tako da mogu napisati optimalan program koji bi odgovarao njihovim zahtjevima i potrebama. ROS podržava programske jezike C++, Python i LISP. ROS ima 4 osnovna elementa: vodovod (engl. *plumbing*), skup alata (engl. *tools*), mogućnosti (engl. *capabilities*) i ekosustav (engl. *ecosystem*) [1].



Slika 2.1. Koncept ROS-a

2.1.1 Povijest ROS-a

Početkom 2007. prvi dijelovi onoga što će na kraju postati ROS počeli su se okupljati na Stanfordu. Studenti Eric Berger i Keenan Wyrobek uočili su kako brojni studenti imaju problem s usvajanjem znanja iz područja robotike. Kako do tada nije postojao osnovni operacijski sustav za robote koji bi manipulirao kretnjama robota, njegovim senzorima itd., dvojica studenata su pokrenula stvaranje osnovnog sustava koji će pružiti raznolike mogućnosti. U početku su izgradili „Personal Robot One“ (PR1) kao hardverski prototip i počeli raditi softver za njega, uzimajući najbolje prakse iz tada najboljih robotskih programa otvorenog tipa. Prilikom traženja investitora za daljni razvoj, Eric Berger i Keenan Wyrobek su upoznali Scott Hassan, osnivača Willow Garage. Vrlo brzo su dvojica studenata počela raditi za Willow Garage gdje se ubrzo počelo raditi na PR2, drugom robotu koji pokreće ROS. U 2010. Willow Garage je ispunio jedno od svojih glavnih ciljeva, a to je poklanjanje 10 PR2 robota sveučilištima. To sve je rezultiralo proširivanjem ROS-a u svijetu robotike diljem svijeta. Prva službena distribucija ROS-a je ROS Box Turtle koja je izašla 2010. godine. Krajem veljače 2013. OSRF - Open Source Robotics Foundation je postao primarni održavatelj softvera za ROS, označavajući najavu u kolovozu iste

godine da će Willow Garage biti ugašen od svojih osnivača. U godinama koje su slijedile otkako je OSRF preuzeo razvoj ROS-a, nova verzija je bila izdana svake godine, dok je zainteresiranost za ROS u svijetu robota nastavila rasti. NASA je 2014. objavila kako će prvi robot na Međunardnoj svemirskoj stanici (engl. International Space Station) koristiti ROS. U 2017. OSRF je promijenio ime u Open Robotics [2].

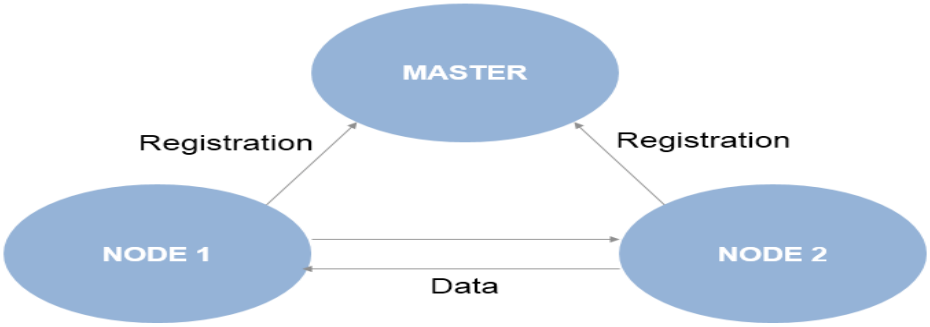
2.1.2 ROS distribucije

ROS distribucija je određena verzija ROS paketa [3]. Često se uspoređuje s Linux distribucijama. Svrha distribucija je da svojom novom verzijom dodaju atribute i određene konfiguracije kako bi se omogućila podrška dodatnim zahtjevima, poput povećanog opsega i dostupnosti.

ROS Melodic Morenia je 12. po redu objavljena ROS distribucija. Objavljena je 23. svibnja 2018. i planirani datum za gašenje Melodic distribucije je u svibnju 2023 [3]. ROS Melodic je najbolje optimiziran za Ubuntu 18.04, međutim može ga se intalirati i na druge inačice Linux sustava te na operacijskom sustavu Windows.

2.1.3 Način rada ROS-a

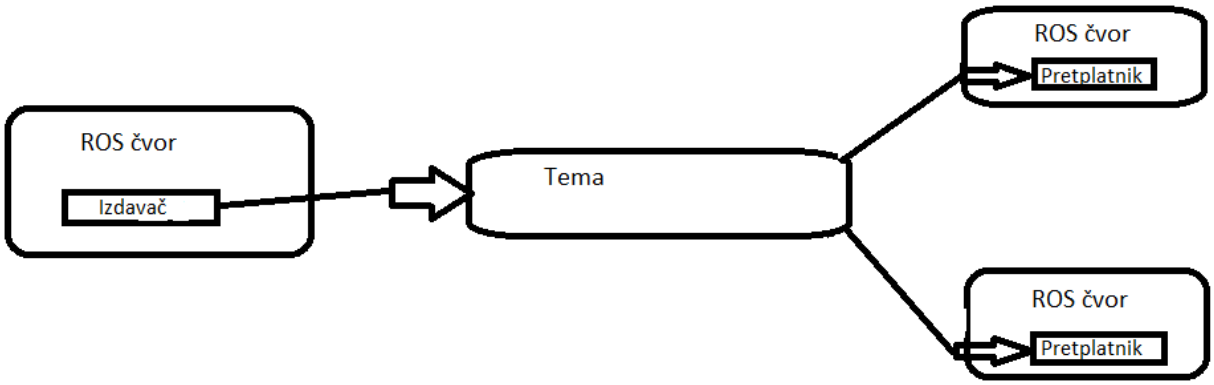
ROS procesi su prikazani kao čvorovi (engl. node) u strukturama grafova povezani bridovima zvanim teme (engl. topic). ROS čvorovi mogu prenositi poruku (engl. message) od jednog čvora do drugog koristeći teme, obaviti poziv usluga (engl. service) drugom čvoru, postaviti ili dohvatiti zajedničke podatke iz baze podataka koja se naziva parametarski poslužitelj. Proces zvan ROS Master pruža usluge imenovanja i registriranja usluga ostatku čvorova u ROS sustavu. Prati izdavače (engl. publishers) i pretplatnike (engl. subscribers) na teme kao i na usluge. Uloga Master-a je omogućiti pojedinačnim ROS čvorovima da se pronađu u procesima [4]. Na slici 2.2 prikazan je osnovni koncept rada ROS Master-a koji se brine za registraciju čvorova u sustavu.



Slika 2.2. Prikaz ROS Master-a

Čvorovi predstavljaju pokrenuti proces. Svaki čvor ima ime koje je registrirano u ROS Master-u prije nego se može koristiti. Čvorovi međusobno komuniciraju pomoću tema, poziva udaljenih usluga (engl. Remote procedure call – RPC) i poslužitelja parametara (engl. Parameter Server). Korištenje čvorova u ROS-u pruža nekoliko prednosti cjelokupnom sustavu. Postoji dodatna tolerancija kvarova jer su rušenja sustava izolirana na pojedinačne čvorove. Složenost koda je smanjena u usporedbi s monolitnim sustavima [4].

Teme su imenovane sabirnice preko kojih čvorovi razmjenjuju poruke. Teme imaju anonimnu semantiku izdavača/pretplaćivanja, koja razdvaja proizvodnju informacija od njihovog korištenja. Općenito, čvorovi ne znaju s kime komuniciraju. Umjesto toga, čvorovi koji su zainteresirani za podatke pretplaćuju se na odgovarajuću temu; čvorovi koji generiraju podatke objavljuju određenoj temi. Može postojati više izdavača i pretplatnika jedne teme. Teme su namijenjene jednosmjernoj komunikaciji. Čvorovi koji trebaju izvoditi pozive udaljenih procedura, odnosno primaju odgovor na zahtjev, umjesto tema trebaju koristiti usluge [2]. Na slici 2.3 prikazan je osnovni koncept izdavanja i pretplaćivanja ROS čvorova na jednu ili više tema.



Slika 2.3. Prikaz komunikacije izdavatelja/pretplatnika preko jedne teme i 3 čvora

Model izdavatelj/pretplatnik je vrlo fleksibilna paradigma, ali njihov jednosmjerni prijenos nije prikladan za RPC zahtjev/odgovor, koje su potrebne u distribuiranom sustavu. Usluge se često koriste za radnje koje imaju definirani početak i kraj, kao što je snimanje slike u jednom kadru ili mijenjanje pozicije robota.

2.2. UNIVERSAL ROBOTS – UR5

Universal Robots je danski proizvođač manjih fleksibilnih industrijskih kolaborativnih robotskih ruku sa sjedištem u Odenseu u Danskoj. Tvrtka je osnovana 2005. godine od ideje inženjera Esbena Østergaarda, Kaspera Støya, i Kristiana Kassowa. Proizvodi se sastoje od UR3 i UR3e,UR5 i UR5e, UR10 i UR10e i UR16e. Svi roboti su robotske ruke sa šest zglobova i vrlo male mase, uključujući kabel od 11kg, 20kg i 33kg [5].

Serija robota UR5 i UR5e dolazi s mnoštvom prednosti za svakodnevnu upotrebu. Robotska ruka ima 6 osi i iz toga razloga je u mogućnosti doći u gotovo svaki položaj.Također svaki UR5 robot dolazi s 12' inčnim ekranom na dodir koji je kontrolna ploča za postavke i omogućava instalaciju dodatnih programskih dodataka. Grafičko okruženje Polyscope omogućuje mu brzo i relativno integriranje s okolinom [6]. Sigurnosne postavke se mogu prilagoditi za svako određeno rješenje i iz toga razloga vrlo je siguran tijekom rada. UR5 je pogodan za složena područja primjene u industriji i obrazovanju zbog svog relativno velike nosivosti od 5kg i radnim dometom od 850mm. Na slici 2.4 prikazano je grafičko okruženje Polyscope koje nam daje mogućnost mijenjanja pozicije robota. Na slici 2.5 prikazana je robotska ruka UR5 u radnom okruženju.

Svojstva	**Vrijednosti**
Masa	18.4kg
Nosivost	5kg
Domet	850mm
Domet zglobova	+/- 360°
Brzina	Svi zglobovi: 180°/s Alat: obično 1 m/s
Veličina kontrolne kutije	475 mm x 423 mm x 268 mm
I/O napajanje	24 V 2A u kontrolnoj kutiji i 12 V/24 V 600 mA u alatu
Programiranje	Grafičko korisničko sučelje Polyscope na 12 inčnom zaslonu na dodir
Glasnoća	Potpuno tih
Temperatura	Robot može raditi na temperaturama u rasponu od 0-50°C

Tablica 2.1. Tehničke karakteristike UR5

Slika 2.4. Zaslon na dodir s grafičkim okruženjem Polyscope

Slika 2.5. Universal Robots UR5

2.3. INTEL REALSENSE™ LiDAR KAMERA L515

Intel® RealSense™ LiDAR kamera L515 omogućava visoku kvalitetu dubine i malu potrošnju energije, a može generirati 23 milijuna točnih dubinskih točaka u sekundi. L515 je dubinska LiDAR kamera koja koristi MEMS tehniku zrcalnog skeniranja. To omogućuje bolju efikasnost laserske snage u usporedbi s ostalim tehnologijama trenutno.

L515 je fotoaparat visoke rezolucije s manje od 3.5 W potrošnje energije za skeniranje dubinskih točaka. L515 snima objekte koji se brzo kreću s minimalnim zamućenjem pokreta svojim unutarnjim procesorom vida. Snima objekte do 9 metara udaljenosti, predviđeno korištenja kamere je u zatvorenom prostoru, a 3D oblak točaka koje kamera snima su u rezoluciji 1024 x 768 [4]. Na slici 2.6 prikazana je montirana LiDAR kamera L515 na robotsku ruku. Na slici 2.6 prikazana je LiDAR kamera montirana na robotsku ruku UR5.

Slika 2.6. Intel RealSense™ LiDAR CAMERA L515

3. IZRADA SUČELJA IZMEĐU UR5 I LiDAR CAMERA L515

Program za dobivanje dubinskih slika prostorije je definiran s tri funkcije: „save3D“, „create3D“ i „perform circle“. Funkcija koja pokreće program je „create3D“ koja poziva funkciju „perform_circle“ s predanim nazivima dubinskih slika koje će se spremati na računalo pomoću funkcije „save3D“. Za svaki parametar funkcije „perform_circle“ poziva se funkcija „save3D“. Funkcija „perform_circle“, nakon što je obavljen jedan poziv funkcije „save3D“, rotira robotsku ruku na kojoj se kamera nalazi i ponavlja proces za ostale parametre koji su joj predani.

Za potrebu ovoga rada bilo je prvo potrebno instalirati Ubuntu 18.04. operacijski sustav na kojem je naknadno instaliran ROS Melodic. Nakon toga bilo je potrebno instalirati UR5 na operacijski sustav zajedno s *Moveit* paketom koji nam omogućuje manipulaciju UR5 robotom u ROS-u. Sljedeći korak je bio instalirati potrebne drivere za pokretanje L515 kamere. Kada smo sve potrebne programe za naš zadatak instalirali, potrebno je osmisliti program koji pokreće robota i snima dubinsku sliku. Na kraju je bilo potrebno napisati skriptu koja će transformirati 3D oblak točaka iz koordinatnog sustava kamere u koordinatni sustav baze robota kako bi dobivene slike uspješno spojili u alatu Meshlab i dobili 3D rekonstrukcije sobe.

3.1. Pokretanje UR5 robotske ruke i L515 kamere

Prilikom pokretanja robota i kamere L515, prvo moramo saznati IP adresu našeg računala koje je spojeno na istu internetsku mrežu kao i robotska ruka UR5. IP adresu našeg računala u Linux terminalu dobijemo sa sljedećom naredbom:

$ip addr show

Nakon što smo saznali IP adresu našeg računala, u Polyscope grafičkom okruženju robota upisujemo IP adresu računala kako bi uspostavili povezanost između robotske ruke UR5 i računala.

UR5 robota pokrećemo u Linux terminalu sljedećom naredbom:

$roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch

Vrlo je bitno da u Polyscope grafičkom okruženju robota upišemo ispravnu IP adresu računala koje je spojenu na mrežu. UR5 robotska ruka i računalo moraju biti spojeni na istu mrežu, u suprotnom povezanost između njih dvoje neće biti uspostavljena.

Sljedeći korak nam je upaliti L515 kameru s filterom koji omogućuje dubinsku sliku. Naredba je prikazana sljedećom linijom:

$ roslaunch realsense2_camera rs_camera.launch filters:=pontcloud

Vrlo je važno provjeriti povezanost između računala i L515 preko serijske veze USB-a koji podržava verziju 3.0. Za verzije USB-a tipa 2.0, kamera neće ispravno raditi, odnosno neće biti u mogućnosti dati dubinsku sliku prostorije.

Kada je provjereno da je sve povezano i upaljeno, sljedeći korak je preko odgovarajuće skripte u Pythonu doći do dubinskih slika.

3.2. Transformacija kordinatnih sustava kamere i alata u kordinatni sustav baze robota

Funkcija „save3D“ omogućuje transformaciju koordinatnih sustava kamere i alata robotske ruke u koordinatni sustav baze robota. Kako je svaki stvarni objekat u stvarnom svijetu opisan sa svojom orijentacijom i pozicijom, a koordinatni sustavi alata robota i kamere se razlikuju, stvarne koordinate objekata moraju biti defnirane u odnosu na jedan od zadanih koordinatnih sustava. Koordinatni sustav baze robota je konstantan u odnosu na okolinu, tako da on predstavlja najbolji izbor za transformiranje koordinata alata robota i kamere. Svaka dubinska točka slike koju snimimo će poslije transformacije koordinatnih sustava alata robota i kamere u koordinatni sustav baze robota biti smještena u koordinatnom sustavu baze robota i pozicije objekata dobivenih preko slika će biti onakve kakve vidimo u stvarnom svijetu. Robotu je pozicija definirana s x,y,z koordinatama, a orijentacija kvaternionom s x,y,z,w koordinatama. Struktura „current_pose“ predstavlja trenutne vrijednosti položaja alata robota. Njezine vrijednosti dobivamo iz biblioteke „moveit_commander“ koju smo prethodno uključili u našu skriptu.

```
17      quater1 = [current_pose.orientation.w]
18      quater2 = [current_pose.orientation.x]
19      quater3 = [current_pose.orientation.y]
20      quater4 = [current_pose.orientation.z]
21
22      translationX = [current_pose.position.x]
23      translationY = [current_pose.position.y]
24      translationZ = [current_pose.position.z]
```

Slika 3.4. Preko current_pose funkcije dobivamo trenutnu poziciju i orijentaciju robota

Kvaternion u matematici služi za predstavljanje rotacije i prostorne orijentacije elementa u 3D prostoru. Konkretno, kvaternioni kodiraju informaciju o rotaciji za neki kut oko proizvoljne osi. Upotreba kvaterniona je česta u računalnoj grafici, robotici, navigaciji, dinamici leta, a koristi se i u ovome radu [7]. Vrijednosti s kojima ćemo popunjavati rotacijsku matricu dobiveni su sljedećim izrazima:

$$r_1 = 1 - \Big(2 * \big((quater1 * quater1) + (quater1 * quater1)\big)\Big) \tag{3-1}$$

$$r_2 = 2 * \big((quater2 * quater3) + (quater1 * quater4)\big) \tag{3-2}$$

$$r_3 = 2 * \big((quater2 * quater4) - (quater1 * quater3)\big) \tag{3-3}$$

$$r_4 = 2 * \big((quater2 * quater3) - (quater1 * quater4)\big) \tag{3-4}$$

$$r_5 = 1 - \Big(2 * \big((quater1 * quater1) + (quater3 * quater3)\big)\Big) \tag{3-5}$$

$$r_6 = 2 * \big((quater3 * quater4) + (quater1 * quater2)\big) \tag{3-6}$$

$$r_7 = 2 * \big((quater2 * quater4) + (quater1 * quater3)\big) \tag{3-7}$$

$$r_8 = 2 * \big((quater3 * quater4) - (quater1 * quater2)\big) \tag{3-8}$$

$$r_9 = 1 - \Big(2 * \big((quater1 * quater1) + (quater4 * quater4)\big)\Big) \tag{3-9}$$

$$\mathbf{rotationMatrix} = \begin{pmatrix} r_1 & r_2 & r_3 & translationX \\ r_4 & r_5 & r_6 & translationY \\ r_7 & r_8 & r_9 & translationZ \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3-10}$$

Sa sljedećim programskim kodom dobivena je rotacijska matrica uz pomoć kvaterniona i pozicije x,y,z koordinata.

```
26      r1 = 1 - (2 * ((quater1 * quater1) + (quater1 * quater1)))
27      r2 = 2 * ((quater2 * quater3) + (quater1 * quater4))
28      r3 = 2 * ((quater2 * quater4) - (quater1 * quater3))
29      r4 = 2 * ((quater2 * quater3) - (quater1 * quater4))
30      r5 = 1 - (2 * ((quater1 * quater1) + (quater3 * quater3)))
31      r6 = 2 * ((quater3 * quater4) + (quater1 * quater2))
32      r7 = 2 * ((quater2 * quater4) + (quater1 * quater3))
33      r8 = 2 * ((quater3 * quater4) - (quater1 * quater2))
34      r9 = 1 - (2 * ((quater1 * quater1) + (quater4 * quater4)))
35
36      rotationMatrix = np.array([[r1,r2,r3,translationX],
37                                 [r4,r5,r6,translationY],
38                                 [r7,r8,r9,translationZ],
39                                 [0,0,0,1]])
```

Slika 3.5. Računanje rotacijske matrice

Nakon što smo dobili rotacijsku matricu, izdvajamo dubinske točke u posebnu matricu te izdvajamo boje točaka slike koju smo dobili kako bismo točke na dubinskoj slici lakše prepoznali.

```
42      filename = glob.glob('{0}*.pcd'.format(name))
43      pcd = o3d.io.read_point_cloud(filename)
44      PointCloud = np.asarray(pcd.points)
45      rgb = np.asarray(pcd.colors)
```

Slika 3.6. Spremanje dubinskih točaka i boja u zasebna polja

Matricu „MatrixOfCamera“ dobivamo translacijom koordinatnog sustava kamere u koordinatni sustav alata robota. Za ovu transformaciju potrebna nam je translacija i rotacija koordinatnog sustava kamere u koordinatni sustav alata robota. Translaciju dobivamo mjerenjem, a rotaciju usporedbom koordinatnih osi ova dva koordinatna sustava. Prvi parametar matrice koji je izmjeren je 12 cm. Ta vrijednost predstavlja udaljenost leće kamere do ishodišta kordinatnog sustava alata po x-osi. Sljedeći parametar je translacija po z-osi, koja predstavlja udaljenost ishodišta vrha alata do pozicije na kojoj je kamera učvršćena na zglob robota i on iznosi 3cm. Translacija po y-osi iznosi. Sljedeći korak je da matricu u kojoj smo spremili dubinske točke prostorije pomnožimo s matricom kamere. Nakon toga dobiveni produkt množimo s rotacijskom matricom i konačni rezultat spremamo u novu matricu. Matrica kamere je popunjena sa sljedećim vrijednostima:

$$\mathbf{MatrixOfCamera} = \begin{pmatrix} 0 & 1 & 0 & -0.12 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -0.03 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3-11}$$

Matrica „MatrixOfCamera“ se sastoji od rotacijskog i translacijskog dijela. Prva tri stupca u matrici predstavljavaju rotacijski dio, a njezine vrijednosti se dobivaju na temelju odnosa koordinatnih sustava osi kamere i vrha alata robota. Zadnji stupac u matrici MatrixOfCamera predstavlja translacijski dio i on se dobiva mjerenjem udaljenosti od leće kamere do središta vrha alata robota. Ove vrijednosti su konstantne jednom kada su određene iz razloga što je kamera nepomična u odnosu na zadnji članak robota. Nakon popunjavanja matrice „MatrixOfCamera“, u sljedećem koraku ona se množi s matricom „transposedMatrixC“. Matrica „transposedMatrixC“ predstavlja vrijednosti transponiranih prostornih pozicija dobivenih točaka u koordinatnom sustavu kamere. Kako su točke u prostoru definirane s x, y, z vrijednostima, matrica „transposedMatrixC“ imat će 3 retka. Kako bi se pozicije točaka prikazale u homogenim

koordinatama i kako bi se omogućilo množenje s matricom „MatrixOfCamera“ koja ima 4 stupca i 4 retka, matrici „transposedMatrixC“ dodajemo još jedan element vrijednosti 1.

$$\mathbf{ProductOfMatrices} = \mathbf{MatrixOfCamera} * \mathbf{transposedMatrixC} \quad (3\text{-}12)$$

Nakon dobivanja produkta matrice kamere i transponirane matrice dubinskih točaka, novu matricu množimo s rotacijskom matricom da dobijemo krajnji rezultat.

$$\mathbf{Matrix} = \mathbf{rotationMatrix} * \mathbf{ProductOfMatrices} \quad (3\text{-}13)$$

Na kraju homogenoj matrici uklanjamo zadnji element koji ima vrijednost 1, te nakon toga matricu transponiramo. Dobivena matrica predstavlja oblak točaka u koordinatnom sustavu baze robota.

Na slici 3.7 prikazano je transfomiranje u koordinatni sustav baze robota.

```
47
48      PointCloud = np.insert(PointCloud, 3, 1, axis = 1)
49      transposedMatrixC = np.transpose(PointCloud)
50
51      MatrixOfCamera = [[0 ,1 ,0 ,-0.12],
52                        [-1, 0, 0, 0],
53                        [0 ,0 ,1, -0.03],
54                        [0 ,0 ,0 ,1]]
55
56      ProductOfMatrices = MatrixOfCamera @  transposedMatrixC
57
58      Matrix = rotationMatrix @ ProductOfMatrices
59      FinalMatrix = Matrix[:-1]
60      FinalMatrix = np.transpose(FinalMatrix)
```

Slika 3.7. Transformiranje u koordinatni sustav baze robota

Na kraju dubinsku sliku spremamo na računalo uz pomoć biblioteke „Open3D“ [8] [9]. Na slici 3.8 prikazano je spremanje dubinskih slika u .ply formatu.

```
62      ply = o3d.geometry.PointCloud()
63      ply.points = o3d.utility.Vector3dVector(FinalMatrix)
64      ply.colors = o3d.utility.Vector3dVector(rgb)
65      o3d.io.write_point_cloud('FinalMatrix_{0}.ply'.format(name),ply)
```

Slika 3.8. Spremanje 3D oblaka točaka u boji u .ply formatu na računalo

3.3. Pozicioniranje kamere i snimanje dubinskih slika

Na početku funkcije „create_3D“ inicijaliziran je čvor potreban za ostvarivanje komunikacije s robotom. Nakon toga su inicijalizirane varijable move_group i joint_goal koje nam služe za zadavanje i pomicanje zglobova, te dobivanje informacija o njihovim trenutnim vrijednostima. Na kraju funkcije „create_3D“ radimo poziv funkcije „perform_circle“ s proizvoljnim nazivima slika koje će funkcija spremati na računalo.

Funkcija „perform_circle“ služi za spremanje dubinskih slika na računalo. Funkciji kao ulazne parametre predajemo nazive dubinskih slika, kako bi bilo olakšano pretraživanje slika u alatima za 3D rekonstrukciju dobivenih dubinskih slika prostorije. U for petlji „perform_circle“ pozivamo funkciju „save3d“ koja transformira snimljeni oblak točaka i sprema ga u datoteku. Nakon snimanja jedne slike, alat na kojem je montirana kamera rotiramo za $\frac{1}{3} * \pi$, što je u stupnjevima jednako 60°. Kako bi zahvatili cijelu prostoriju moramo rotirati alat robota na kojemu se kamera nalazi za puni krug koji je jednak 360°. Kamera L515 nema leću koja može snimiti cijelu prostoriju iz jedne snimljene slike, pa kako bi to omogućili podijelili smo puni krug na šest jednakih dijelova koji iznosi 60°. Nakon što je kamera snimila prvu dubinsku sliku i njezin se rezultat pohranio na računalo, zadnji članak robotske ruke na kojem se nalazi kamera rotiramo za 60° preko varijable joint_goal[]. Funkcije move_group.go() i move_group.stop() služe za postavljanje robota u položaj definiran vrijednostima varijabli zglobova sadržanim u varijabli joint_goal. Rotiranje zgloba trajati će 1 sekundu, zbog varijable rospy.sleep() u kojoj smo unutar zagrada predali proizvoljno vremensko trajanje procesa. Tada for petlja obavlja poziv za sljedeći paramatar. Ovaj se proces nastavlja sve dok kamera ne snimi svih šest dubinskih slika prostorije. Nakon što je snimljena posljedna slika, robot se vraća u početni položaj pošto je napravio puni krug.

Na slici 3.9 prikazan je programski kod za dobivanje dubinskih 3D slika poda prostorije.

```
def perform_circle(a, b, c, d, e, f):
    names = [a, b, c, d, e, f]
    move_group = moveit_commander.MoveGroupCommander('manipulator')
    joint_goal = move_group.get_current_joint_values()

    for x in names:
        save_3d(x)

        joint_goal[4] -= 1/3 * np.py
        move_group.go(joint_goal, wait = True)
        move_group.stop()
        rospy.sleep(1)

def create_3D():
    rospy.init_node('robot_control', anonymous = True)
    move_group = moveit_commander.MoveGroupCommander('manipulator')
    joint_goal = move_group.get_current_joint_values()
    perform_circle('first_position','second_position','third_position','fourth_position','fifth_position','sixth_position')
```

Slika 3.9. Programski kod za uzimanje šest dubinskih slika rotiranih za 60° oko baze robota

Sljedeći korak je pomaknuti zglobove robota kako bi se dobio 3D oblak točaka poda prostorije u kojoj se robot nalazi. Kako bi obuhvatili puni krug poda prostorije od 360°, zglobove robota moramo postaviti u novi položaj koji nam omogućuje da samo rotiranjem alata robota na kojem se kamera nalazi dobijemo 3D oblak točaka poda prostorije. Pri postavljanju robota u navedeni

položaj, alat na kojem se nalazi kamera ne smije biti na udaljenosti u manjoj od 30cm od objekta kojeg snima. Postavke s kojima je kamera korištena su bile „max_range“ kako bi se dobile i dubinske točke za objekte koji su bili udaljeniji od kamere. U jednom od svojih parametara „max_range“ ima atribut „min_distance“ kojem je osnovna vrijednost 30cm. Njezina se vrijednost ne može smanjivati za navedenu postavku kamere. Vrijednosti kutova zglobova robota prikladne za snimanje poda prostorije su sljedeće:

$$joint\_goal[0] = \frac{1}{2} * \pi \quad (3\text{-}14)$$

$$joint\_goal[1] = -\frac{1}{2} * \pi \quad (3\text{-}15)$$

$$joint\_goal[2] = \frac{1}{18} * \pi \quad (3\text{-}16)$$

$$joint\_goal[3] = \frac{1}{3} * \pi \quad (3\text{-}17)$$

$$joint\_goal[4] = \frac{13}{36} * \pi \quad (3\text{-}18)$$

$$joint\_goal[5] = \frac{1}{2} * \pi \quad (3\text{-}19)$$

Na slici 3.10 prikazan je programski kod za premještanje alata robota u položaj za snimanje poda.

```
rospy.init_node('robot_control', anonymous = True)
move_group = moveit_commander.MoveGroupCommander('manipulator')
joint_goal = move_group.get_current_joint_values()
joint_goal[0] = 1 / 2 * np.py
joint_goal[1] = - 1 / 2 * np.py
joint_goal[2] = 1 / 18 * np.py
joint_goal[3] = 1 / 3 * np.py
joint_goal[4] = 13 / 36 * np.py
joint_goal[5] = 1 / 2 * np.py
```

Slika 3.10. Programski kod za pomicanje robotske ruke u položaj optimalan za uzimanje dubinskih slika poda prostorije koji se nalazi u funkciji „create_3D“

Nakon što je robot došao u željeni položaj za uzimanje dubinskih slika poda prostorije, tada ponovno pozivamo funkciju „perform_circle“ koja će snimiti šest dubinskih slika prostorije na kojima se vidi pod prostorije. Alat koji ćemo rotirati ostaje isti zato što se na njemu nalazi kamera te da dobijemo željeni prikaz poda sobe u vidokrugu od 360°, taj alat ćemo baš kao i u prethodnom slučaju rotirati za 60° šest puta sve dok se ne vrati u svoju početnu poziciju. Kod koji je korišten za dobivanje dubinskih točaka poda prostorije je identičan kodu sa slike 3.9.

Jedina razlika nalazi se u posljednjoj liniji funkcije „create3D“, u kojoj su promijenjeni nazivi parametara s kojima ćemo pozivati funkciju „perform_circle“. Svaki parametar nam predstavlja jednu dubinsku sliku koja će se nakon transformacije spremati na računalo, te kako bi znali koje smo slike dobili, mijenjamo njihova imena s kojima će se spremati na računalo. U suprotnom, ako ostavimo iste nazive parametara u pozivu funkcije „perform_circle“ koji su korišteni u prethodnom spremanju 3D točaka prostorije, prethodno će se slike obrisati, a nove će se spremiti na računalo s istim nazivima. Na slici 3.11 prikazana je promjena u programskom kodu u odnosu na sliku 3.9, odnosno korišteni nazivi za spremanje dubinskih slika poda prostorije na računalo.

```
def create_3D():
    rospy.init_node('robot_control', anonymous = True)
    move_group = moveit_commander.MoveGroupCommander('manipulator')
    joint_goal = move_group.get_current_joint_values()
    perform_circle('seventh_position','eighth_position','ninth_position','tenth_position','eleventh_position','twelfth_position')
```

Slika 3.11. Promjena programskog koda potrebna za snimanje šest dubinskih slika poda prostorije

I na kraju je potrebno dobiti dubinsku sliku stropa prostorije kako bi bila obuhvaćena cijela prostorija u rekonstrukciji 3D modela. U slučaju kada su se radile 3D oblaci točaka stropa prostorije, nije se trebalo voditi računa o tome na kojoj će udaljenosti biti robotska ruka te i sama kamera od stropa pošto je strop bio dovoljno visok, pa samim time i dovoljno udaljen od leće kamere. Zadane položaje robota u kojima robot snima 3D oblak točaka se nalaze na sljedećoj slici kao i kod korišten za dobivanje dubinskih slika stropa sobe. Vrijednosti u kojem je robot postavljen za optimalno uzimanje 3D oblaka točaka stropa prostorije su sljedeće:

$$joint\_goal[0] = -\frac{31}{2} * \pi \quad (3\text{-}20)$$

$$joint\_goal[1] = -\frac{1}{2} * \pi \quad (3\text{-}21)$$

$$joint\_goal[2] = 0 \quad (3\text{-}22)$$

$$joint\_goal[3] = \frac{13}{36} * \pi \quad (3\text{-}23)$$

$$joint\_goal[4] = \frac{17}{36} * \pi \quad (3\text{-}24)$$

$$joint\_goal[5] = \frac{1}{2} * \pi \quad (3\text{-}25)$$

Na slici 3.12 prikazan je programski kod za premještanje alata robota u optimalan položaj za dobivanje slike stropa.

```
rospy.init_node('robot_control', anonymous = True)
move_group = moveit_commander.MoveGroupCommander('manipulator')
joint_goal = move_group.get_current_joint_values()
joint_goal[0] = -31 / 2 * np.py
joint_goal[1] = - 1 / 2 * np.py
joint_goal[2] = 0
joint_goal[3] = 13  / 36 * np.py
joint_goal[4] = 17 / 36 * np.py
joint_goal[5] = 1 / 2 * np.py
```

Slika 3.12. Programski kod za pomicanje robotske ruke u položaj optimalan za uzimanje dubinskih slika stropa prostorije

Kao i u prethodnom korištenju programskog koda za dobivanje dubinskih slika poda prostorije, u slučaju stropa prostorije programski kod je identičan, a jedina razlika je opet u nazivima parametara poziva funkcije „perform_circle“. Kako smo ih prethodno proizvoljno nazvali po rednim brojevima na engleskom jeziku, taj niz smo nastavili i sa ostalim dobivenim slikama stropa prostorije. Na slici 3.13 prikazana je promjena u programskom kodu u odnosu na sliku 3.9, odnosno korišteni nazivi za spremanje dubinskih slika stropa prostorije na računalo.

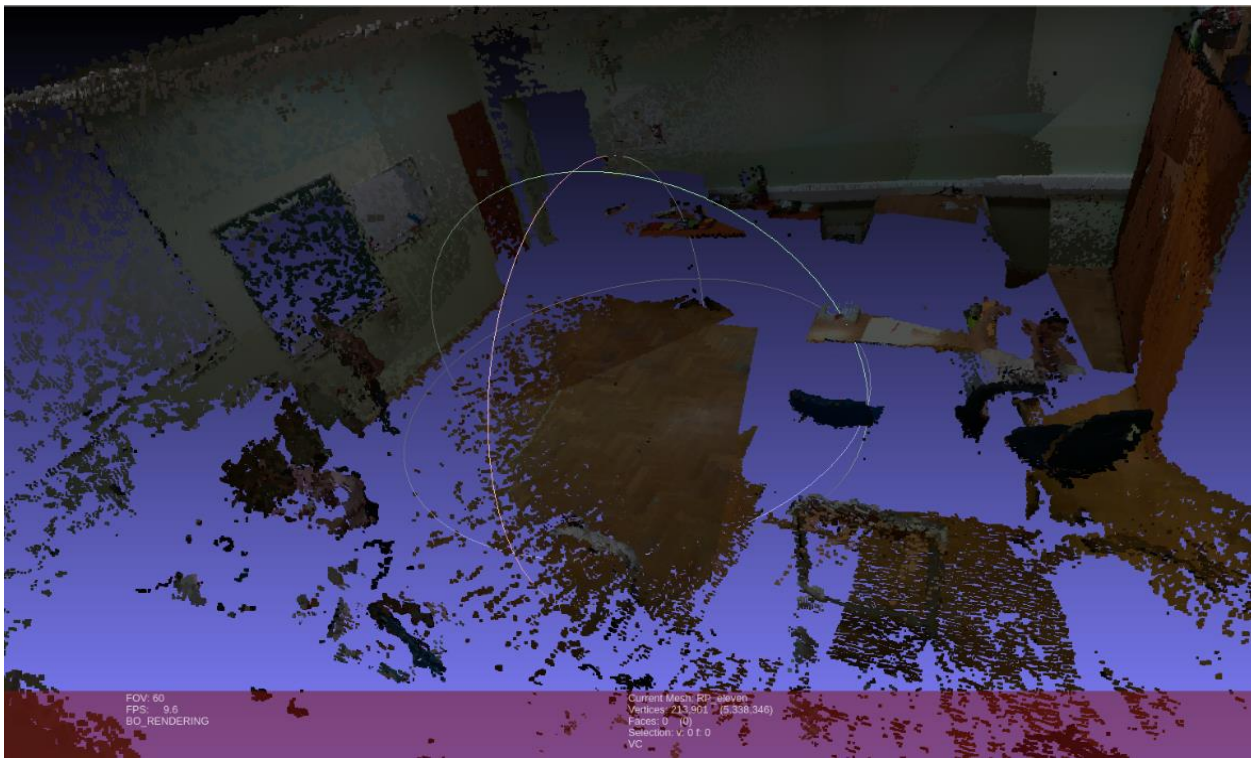
```
def create_3D():
    rospy.init_node('robot_control', anonymous = True)
    move_group = moveit_commander.MoveGroupCommander('manipulator')
    joint_goal = move_group.get_current_joint_values()
    perform_circle('thirteenth_position','fourteenth_position','fifteenth_position','sixteenth_position','seventeenth_positi
    ,'eighteenth_position')
```

Slika 3.13. Promjena programskog koda potrebna za snimanje šest dubinskih slika stropa prostorije

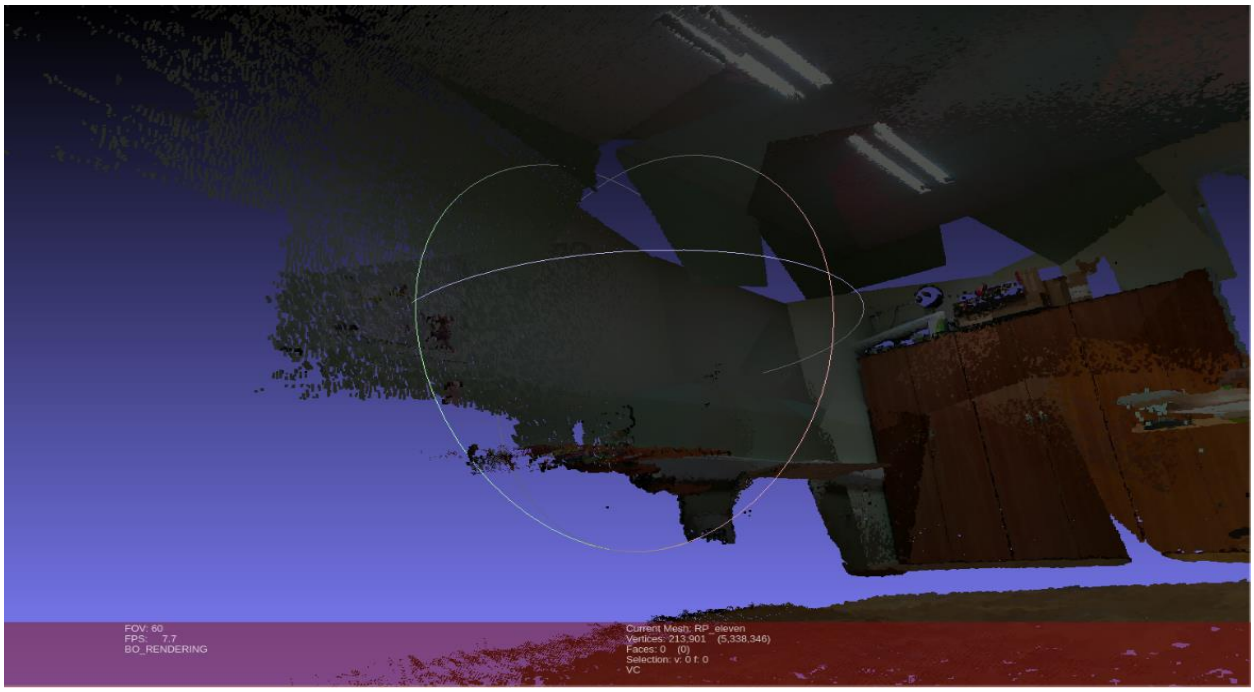
4. 3D MODEL PROSTORIJE

Pri snimanju 3D oblaka točaka prostorije potrebno je bilo obratiti pozornost na položaj robotske ruke na koju je kamera montirana. Problem je predstavljala udaljenost i položaj kamere u pojedinim položajima robotske ruke u kojim kamera snima oblak točaka. Slika bi ponekad bila nejasna ili bi pojedine stvari, kao što su prozor ili stol u prostoriji, smanjila ili uvećala u odnosu na druge objekte u prostoriji zbog spajanja rubnih točaka između dva dobivena oblaka točaka i nesavršenosti pozicioniranja alata robota. Testiranjem je utvrđeno da kamera pruža bolje slike s unutarnjim osvjetljenjem. Nadalje, što su objekti udaljeniji od kamere to će oni biti prikazani s manje točaka.

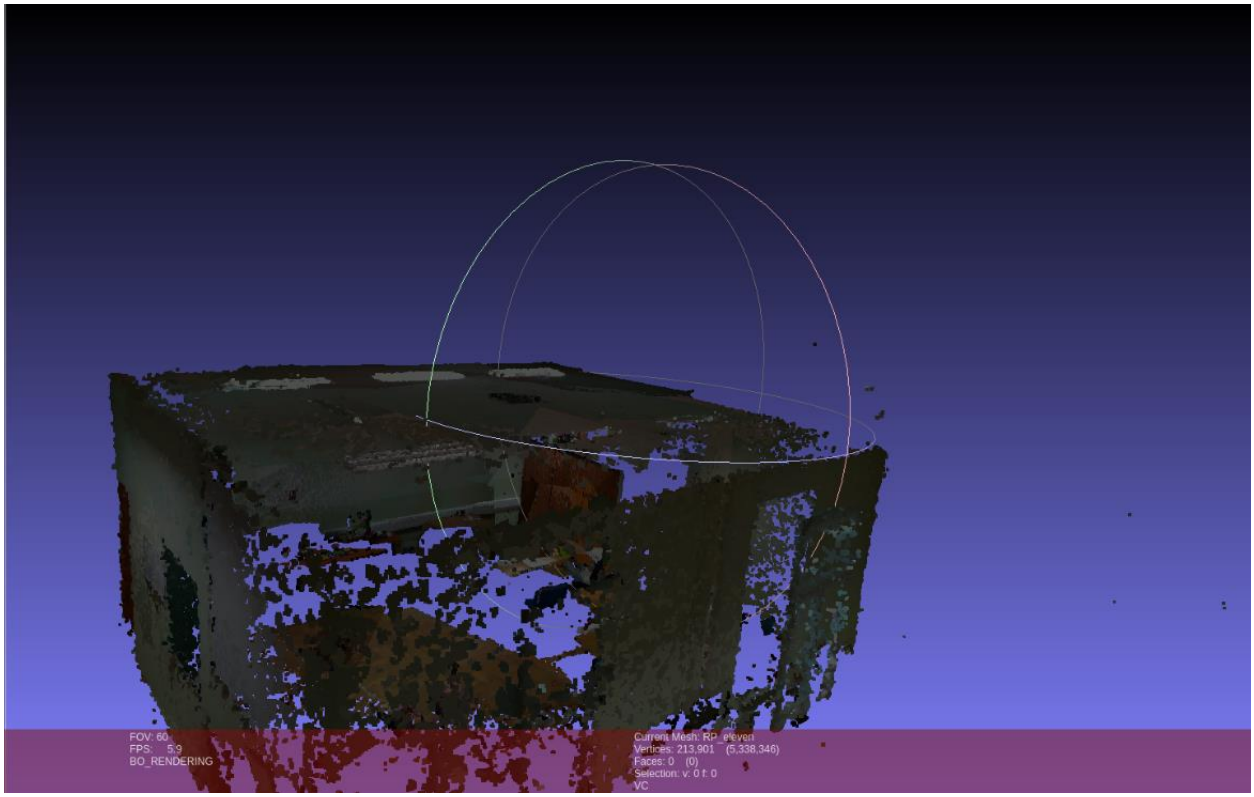
Pohranjeni 3D oblak točaka koji smo dobili kao rezultat izrađenog programa su povezane u programskom alatu Meshlab. Na slikama 4.1, 4.2, 4.3 i 4.4 prikazani su dobiveni 3D modeli prostorije u programskom alatu Meshlab.



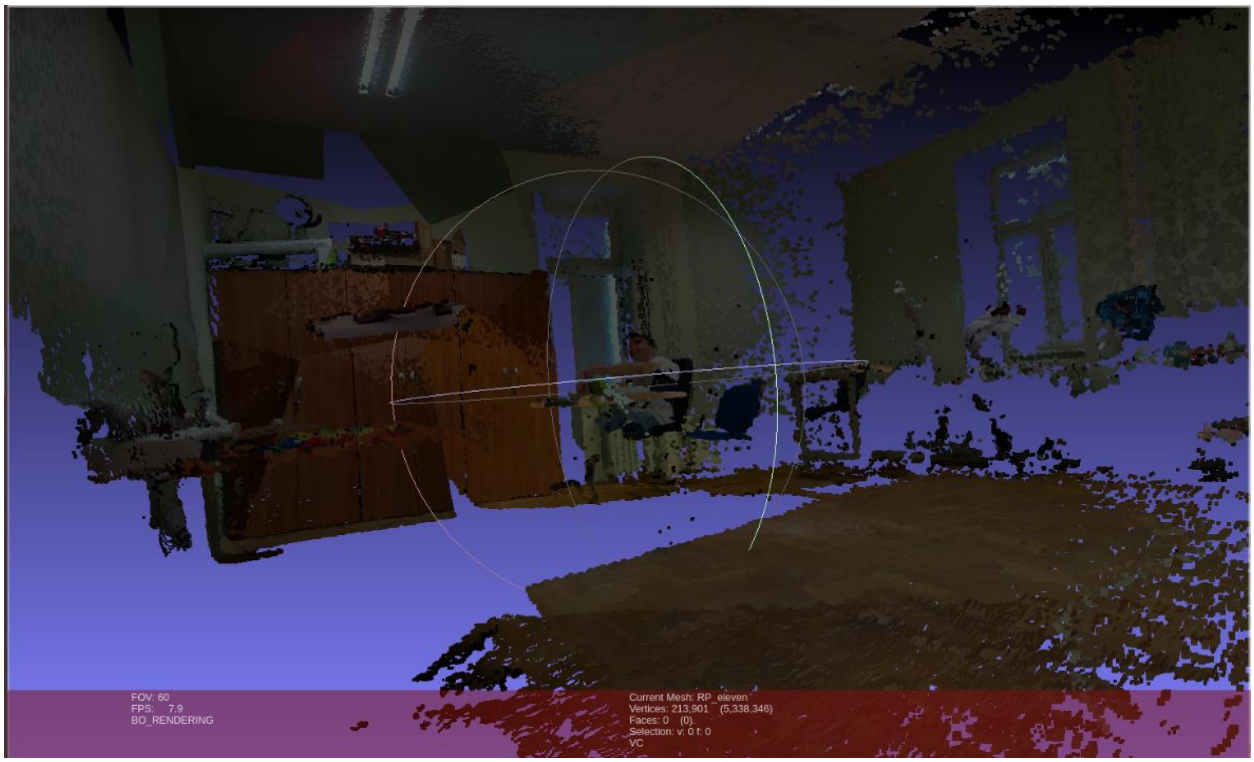
Slika 4.1. Dobiveni 3D model prostorije s objektima koji su udaljeni od kamere



Slika 4.2. Dobiveni 3D model prostorije u okolini unutrašnjeg svjetla i kamere



Slika 4.3. Prikaz dobivenog 3D modela prostorije



Slika 4.4. Prikaz dobivenog 3D modela prostorije iz drugog kuta

5. ZAKLJUČAK

Dobiveni 3D model prostorije je vrlo realističan i mogu se prepoznati objekti i njihov smještaj u prostoriji. Sučelje napisano za interakciju između robota i kamere ne zahtjeva velike resurse računala, poput memorije potrebne za pohranu slika ili brzine procesora za pokretanje i uspješno izvođenje programa.

Intel RealSense™ LiDAR KAMERA L515 prikladna je za rekonstrukcija 3D objekata koji su na udaljenosti do nekoliko metara, u unutrašnjim prostorijama. Montiranjem na robotsku ruku UR5 postignuta je stabilnost kamere te je poznata transformacija dobivenih 3D oblaka točaka u koordinatni sustav baze robota.

U daljnjem istraživanju se umjesto UR5 robotske ruke mogu koristiti dronovi ili pak drugi tip robota koji nam daju veću mogućnost i fleksibilnost. Osim samog snimanja prostorije u kojima je montirana robotska ruka bilo bi korisno dobiti i cijelu rekonstrukcije zgrade u obliku 3D modela gdje onda daljnjim postupcima možemo napraviti i virtualne šetnje i interakcije s objektima u prostoriji.

LITERATURA

[1] https://www.ros.org/about-ros/, posjećeno 19.6.2021.

[2] http://wiki.ros.org/ROS/Concepts, posjećeno 19.6.2021.

[3] http://wiki.ros.org/Distributions, posjećeno 20.6.2021.

[4] https://en.wikipedia.org/wiki/Robot_Operating_System, posjećeno 21.6.2021.

[5] https://hr.mouser.com/new/intel/intel-l515-realsense-lidar-camera/, posjećeno 24.6.2021.

[6] https://en.wikipedia.org/wiki/Universal_Robots, posjećeno 27.6.2021.

[7] https://www.mybotshop.de/Universal-Robots-UR5-UR5e_1, posjećeno 27.6.2021.

[8] https://en.wikipedia.org/wiki/Quaternion, posjećeno 29.6.2021.

[8] http://www.open3d.org/docs/0.9.0/tutorial/Basic/file_io.html, posjećeno 30.6.2021.

[10] http://www.open3d.org/docs/0.12.0/python_api/open3d.utility.html, posjećeno 30.6.2021.

SAŽETAK

Cilj ovoga rada je automatska izgradnja 3D modela prostorije. Kako bi dobili 3D model prostorije, prvo je bilo potrebno napisati programski kod koji će omogućiti interakciju između robotske ruke UR5 i Intel RealSense™ LiDAR KAMERE L515 u programskom jeziku Python koji će na osnovu predanih joj parametara za zglobove robota, isplanirati i postaviti robota u zadani položaj i onda uzeti dubinsku sliku prostorije, pretvoriti je u 3D oblak točaka te ju spremiti na računalo u .ply formatu. U radu je opisana povijest i uloga ROS-a u današnjem svijetu, opis glavnih obilježja UR5 robotske ruke te karakteristike i svojstva LiDAR kamere L515. Nadalje, rad sadrži opis programskog koda u razvojnom okruženju Visual Studio Code. Na kraju je prikazana 3D rekonstrukcija prostorije pomoću programskog alata Meshlab.

Ključne riječi: robotska ruka UR5, izgradnja 3D modela prostorije, ROS, LiDAR CAMERA L515

ABSTRACT

The goal of this thesis was to develop a robotic system for automated reconstruction of indoor environments in the form of 3D models. In order to obtain a 3D model of a room, it was necessary to write a programming code which will enable interaction between a UR5 robotic arm and an Intel RealSense ™ LiDAR CAMERA L515 in the Python programming language. The developed program plans robot motion and places the robot in an appropriate position. Then the camera acquires a point cloud and saves it in the .ply format on a computer. The thesis describes the history and role of ROS in today's world, the description of the main features of the UR5 robotic arm, as well as the characteristics and properties of the LiDAR CAMERA L515. Furthermore, it contains a description of the program code written in the Visual Studio Code environment. Finally, a 3D room reconstruction is presented using the Meshlab tool.

Keywords: robotic arm UR5, 3D room reconstruction, ROS, LiDAR CAMERA L515

PRILOZI