Određivanje trodimenzionalnog oblika i položaja transparentnih objekata u svrhu robotske manipulacije

Mihić, Vlatka

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:906497

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

Diplomski sveučilišni studij Računarstvo
Izborni blok Robotika i umjetna inteligencija

ODREĐIVANJE TRODIMENZIONALNOG OBLIKA I POLOŽAJA TRANSPARENTNIH OBJEKATA U SVRHU ROBOTSKE MANIPULACIJE

Diplomski rad

Vlatka Mihić

Osijek, 2023.

SADRŽAJ

1. UVOD

Kako se svijet robotike i umjetne inteligencije razvija, shodno tome se inženjeri suočavaju sa sve više problema i rubnih slučajeva koje je potrebno riješiti. Jedan od tih problema jest i detekcija i određivanje položaja transparentnih objekata u prostoru. Transparentni objekti, poput staklenih ili plastičnih čaša, boca, zdjela, prozorskih stakala itd. često su prisutni u kućanstvima i drugim unutrašnjim prostorima i korisno je naučiti robote da njima manipuliraju. Kako bi robot manipulirao takvim objektima, prvo je potrebno da ih ispravno detektira na slici i odredi njihov položaj u prostoru. Uobičajeno se položaj objekta u prostoru u svrhu robotske manipulacije određuje na trodimenzionalnoj slici snimljenoj dubinskom kamerom. Međutim, većina transparentnih materijala ne odbija dovoljno svjetlosti natrag prema kameri, što otežava dobivanje pouzdane povratne informacije o obliku, veličini ili položaju objekta. Osim toga, kada svjetlost prolazi kroz transparentne objekte, može doći do višestrukih refleksija unutar objekta. To može izazvati zamućenje ili izobličenje slike, što može otežati precizno određivanje oblika i položaja objekta. U nedavnim istraživanjima, predlažu se rješenja za ove probleme.

Ovaj diplomski rad se bavi rješavanjem problema detekcije i manipulacije transparentnim objektima na temelju predobrade podataka koje daje kamera. Primijenjena metoda je podijeljena na dvije veće cjeline, od kojih se prva bavi primjenom algoritama za detekciju transparentnih objekata, dok se druga bavi manipulacijom detektiranim objektima. Detekcija je rađena uz pomoć ClearGrasp [1] neuronske mreže za detekciju i segmentaciju 3D objekata na RGB i RGB-D slikama. Pomoću ulazne RGB slike, ova mreža vrši segmentaciju transparentnih objekata, rezultirajući crno-bijelom slikom koja predstavlja masku segmentacije. Kombiniranjem segmentacijske maske i ulazne RGB-D slike scene, generira se preoblikovana dubinska slika. Daljnjom primjenom globalne optimizacije na preoblikovanu dubinsku sliku, dobivaju se izlazni podaci, odnosno izlazna dubinska slika te izlazni oblak točaka. Globalna optimizacija u detekciji transparentnih objekata se fokusira na pronalazak optimalnih parametara ili postavki modela kako bi se postigla najbolja preciznost i pouzdanost u detekciji takvih objekata, bez obzira na njihove optičke karakteristike i položaj u slici. Na temelju izlaznih podataka iz ove mreže pronalazi se centar detektiranog objekta i njegove dimenzije. Uz pomoć navedenih podataka moguće je izvesti manipulaciju detektiranim objektom što je ujedno i krajnji cilj rada.

Rad je strukturiran na sljedeći način. U drugom poglavlju, najprije se pristupa pregledu dosadašnjih istraživanja koji su se posvetili rješavanju problema detekcije i manipulacije

transparentnim objektima. Shodno tome, u trećem poglavlju su definirani osnovni koncepti, principi i zakoni koje je potrebno razumjeti za uspješno rješavanje ciljnog problema. U četvrtom poglavlju, koje predstavlja metodologiju istraživanja, opisuje se metoda korištena pri detekciji transparentnih objekata. Kada je teoretski sve opisano, u petom poglavlju se opisuje eksperimentalna analiza, gdje se opisuje implementacija opisanih algoritama i eksperimentalni postav. Na temelju praktično odrađenoga dijela istraživanja provedena je evaluacija rezultata kako bi se odredio doprinos ovog diplomskog rada u području određivanja oblika i položaja transparentnih objekata u svrhu robotske manipulacije, kao i moguća unaprjeđenja, prednosti i nedostatci korištenih metoda.

2. POVEZANA ISTRAŽIVANJA

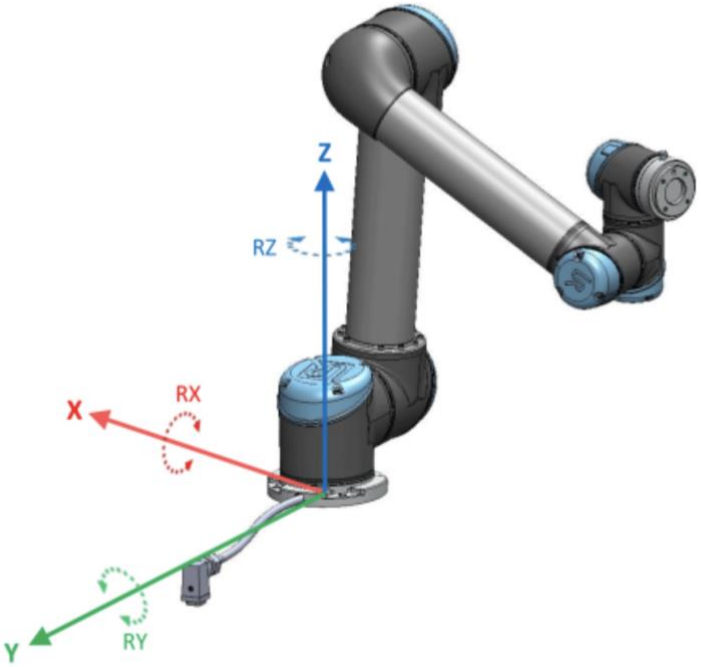
Postoji mnoštvo dosadašnjih istraživanja u okviru teme ovog rada. Jedno od njih jest opisano u radu *Reflective workpiece detection and localization for flexible robotic cells* koji se bavi problemom prepoznavanja radnih komada, odnosno obradaka korištenjem algoritma detekcije temeljenome na Viola-Jones detektoru za postizanje preciznih rezultata [2]. Viola-Jones detektor jest algoritam za detekciju objekata u slikama koji se temelji na analizi uzoraka svjetline i tame, poznatih pod nazivom Haarovi detektori, pri čemu koristi tehniku prilagodljivog pojačavanja (engl. *Adaptive Boosting*) za stvaranje klasifikatora. Prednost ove metode jest njena laka integracija s postojećim robotskim ćelijama dok se pri radu s velikim brojem vrsta radnih komada (više od 10 vrsta) ova metoda nije pokazala najefikasnijom. Istraživanje [3] je fokusirano na detekciju i uklanjanje transparentnih i spekularnih reflektirajućih utjecaja u 3D laserskim mjerenjima. Spekularna refleksija je fenomen svjetla koji se javlja kada svjetlosni snop ili zrake udare u površinu objekta i reflektiraju se pod vrlo preciznim kutom, često pod jednakim kutom u odnosu na površinu, stvarajući intenzivne i jasne svjetlosne sjajne točke ili odsjaje. Ova vrsta refleksije često se opaža na glatkim i sjajnim površinama poput stakla, vode, metala ili lakiranih površina, gdje svjetlo ima tendenciju reflektirati se na vrlo organiziran i koncentriran način, čime stvara oštre sjajne refleksije. Istraživanje[4] iz 2020. godine se bavi rješavanjem problema detekcije neujednačenih reflektivnih ili obojenih objekata. Ovaj rad kombinira projiciranje svjetlosnih uzoraka različitih boja i prilagodbu izloženosti kamere svjetlosti kako bi precizno detektirao oblike na površinama nehomogenih reflektirajućih objekata. Glavna prednost metode uključuje bolje razumijevanje oblika i reljefa, dok nedostatci mogu obuhvaćati potrebu za preciznom kontrolom svjetla i ekspozicije te osjetljivost na uvjete okoline poput osvjetljenja. Osim navedenih tehnika, u jednom od istraživanja iz 2015. godine, istraživači su se koristili TransCut metodom koja koristi linearnost svjetlosnog polja i detektor okluzije kako bi opisali transparentni objekt [5]. Linearnost svjetlosnog polja označava svojstvo svjetlosnih zraka da se mogu kombinirati kako bi se dobila ukupna slika. Ovo omogućuje preciznu segmentaciju čak i kod objekata s prozirnim ili poluprovidnim dijelovima. Ova metoda ima potencijal za precizno segmentiranje objekata s transparentnim dijelovima, ali ima izazove u pogledu svjetlosnih uvjeta te zahtjeva puno resursa. Nešto novije istraživanje iz 2021. godine *Seeing Glass: Joint Point Cloud and Depth Completion for Transparent Objects* predstavlja metodu koja rješava problem rekonstrukcije dubine (prostorne informacije) za transparentne objekte korištenjem kombinacije oblaka točaka i upotpunjavanja dubine [6]. Metoda omogućuje precizniju rekonstrukciju

geometrije transparentnih objekata, poboljšavajući kvalitetu 3D modela. Integracija oblaka točaka i upotpunjavanja dubine omogućuje više informacija za bolje razumijevanje strukture objekata. Nedostaci metode uključuju ovisnost o kvaliteti početnih oblaka točaka i dubinskih informacija te potrebu za složenom računalnom obradom i resursima. Nešto starija metoda je *ClearGrasp* iz 2020. godine [1]. Ova metoda je odabrana kao podloga za rekonstrukciju transparentnih objekata u ovome radu. *ClearGrasp* koristi duboku neuronsku mrežu za 3D detekciju i segmentaciju objekata, što može rezultirati visokom preciznošću i točnošću u detekciji te se fokusira na rješavanje izazova kao što su spekularne refleksije i transparentni objekti, što može rezultirati boljim rezultatima u takvim scenarijima. Kao i ostali radovi, ova metoda se bavi problemima detekcije i rekonstrukcije transparentnih objekata, no za razliku od njih nije računalno zahtjevan što ju čini idealnim za implementaciju.

3. TEORIJSKE OSNOVE

3.1. Osnovni koncepti u robotskoj manipulaciji

Za daljnje razumijevanje rada potrebno je najprije proći kroz osnovne koncepte u robotskoj manipulaciji. Za provedbu robotske manipulacije potreban je prvenstveno robotski manipulator, ali i senzori, sustavi upravljanja i sigurnosni mehanizmi. Tijelo robotskog manipulatora je izgrađeno od baze, članaka i alata te zglobova koji ih povezuju na način da samo prvi i zadnji članak (alat) imaju jedan susjedni članak dok ostali imaju po dva susjedna članka. Ovakva mehanička struktura tijela se naziva *kinematičkim lancem*. Postoji više tipova robotskih manipulatora koji se u osnovi dijele ovisno o vrsti zglobova od kojih su načinjeni. Glavna podjela zglobova jest na translacijske i rotacijske zglobove. Translacijski zglobovi su zglobovi koji spajaju dva članka na način da se jedan od njih može gibati linearno po jednoj osi koja je u pravilu uvijek okomita na prvi članak. Rotacijski zglobovi su slični translacijskima kod kojih je jedina razlika što se drugi članak može gibati nelinearno, točnije rotirati, duž osi prvoga članka. Jedan robotski manipulator može imati proizvoljan broj zglobova ali se u praksi najčešće koriste šestoosni manipulatori. Osim proizvoljnog broja zglobova, manipulator se može sastojati od isključivo translacijskih, isključivo rotacijskih zglobova ili od njihove kombinacije. Primjer šestoosnog



Slika 3.1. Šestoosni robotski manipulator UR5 sa šest rotacijskih zglobova

robotskog manipulatora kojemu su svi zglobovi rotacijski može se vidjeti na slici 3.1 [7]. Ovakvi roboti se danas koriste u industrijskoj automatizaciji za montažu, pakiranje, varenje i druge

zadatke. Rabe se u laboratorijskim istraživanjima za manipulaciju uzorcima te u medicinskim primjenama za precizne kirurške zahvate i dijagnostiku, dok se također koriste u logistici, prehrambenoj industriji, elektronici, edukaciji, automatiziranoj kontroli kvalitete i mnogim drugim područjima. Struktura robotskog manipulatora nosi sa sobom određene prostorne odnose među elementima manipulatora. Svaki zglob ima svoj vlastiti koordinatni sustav čije je ishodište potrebno postaviti u određenu točku koordinatnog sustava prijašnjeg zgloba kako bi se, konačno, alat, kojim se hvata i drži manipulirani objekt, postavio u određenu točku u prostoru. Razvijena su dva matematička modela uz pomoć kojih se vrši izračun za određivanje položaja ili konfiguracije robota. *Direktna kinematika* odnosi se na proces određivanja položaja ili konfiguracije alata robota na temelju poznatih vrijednosti zglobova. To znači da, ako su poznati parametri zglobova (kutovi zglobova, duljine zglobova itd.), može se izračunati položaj alata robota u prostoru. Direktna kinematika korisna je za planiranje putanje, vizualizaciju, simulaciju i druge aplikacije koje zahtijevaju poznavanje položaja alata manipulatora. S druge strane, *inverzna kinematika* je proces određivanja vrijednosti zglobova koji su potrebni da bi se postigao željeni položaj ili konfiguracija alata robota. To znači da, ako je poznat željeni položaj alata manipulatora, inverzna kinematika izračunava vrijednosti zglobova koje će dovesti do tog položaja. Inverzna kinematika korisna je za upravljanje robotom u stvarnom vremenu, praćenje ciljeva, izbjegavanje prepreka i druge aplikacije koje zahtijevaju precizno pozicioniranje alata manipulatora. Shodno tome je korištena i u manipulaciji transparentnim objektima. Osim kinematike, koja se odnosi na proučavanje geometrije i pokreta robotskih manipulatora, osnovni koncept robotske manipulacije jest i dinamika. Dinamika se odnosi na proučavanje sila, momenata i pokreta robota. Ona uključuje proučavanje zakona kretanja objekata koji se sastoje od zglobova, veza i mehanizama. Ovaj pristup omogućava razumijevanje kako se pokreti na razini zglobova prenose na vrh alata robota.

Za upravljanje robotom potrebno je dobro poznavati njegovu kinematiku i dinamiku. Većina današnjih robotskih manipulatora dolazi s gotovim sustavom za upravljanje. Ovi sustavi obično uključuju hardverski kontroler i softverski skup alata i biblioteka za upravljanje manipulatorom. Osim toga, važno je napomenuti da se neki manipulatori također mogu koristiti s drugim sustavima za upravljanje. Na primjer, ako manipulator koristi standardizirani komunikacijski protokol poput ROS-a (Robot Operating System), može se integrirati s drugim sustavima, alatima, senzorima i aktuatorima koji podržavaju taj protokol [8, 9]. To omogućuje veću fleksibilnost pri odabiru i prilagodbi sustava upravljanja manipulatorom prema potrebama projekta. Važan aspekt upravljanja robotskim manipulatorom je sigurnost, posebno kada roboti rade u blizini ljudi. Sigurnosne mjere uključuju detekciju prepreka, ograničavanje sile i brzine primijenjene na vrh

alata manipulatora te pružanje sigurnih interakcija s okolinom i ljudima. Kako bi se prepreke mogle detektirati koriste se razni vizualni senzori poput kamera, dok se pri planiranju optimalne putanje ili akcija za izvršavanje zadataka koriste različite metode i algoritmi. Izbor metode ovisi o specifičnom zadatku, okruženju i mogućnostima robota što je detaljnije opisano u narednim poglavljima.

3.2. Određivanje oblika i položaja objekata

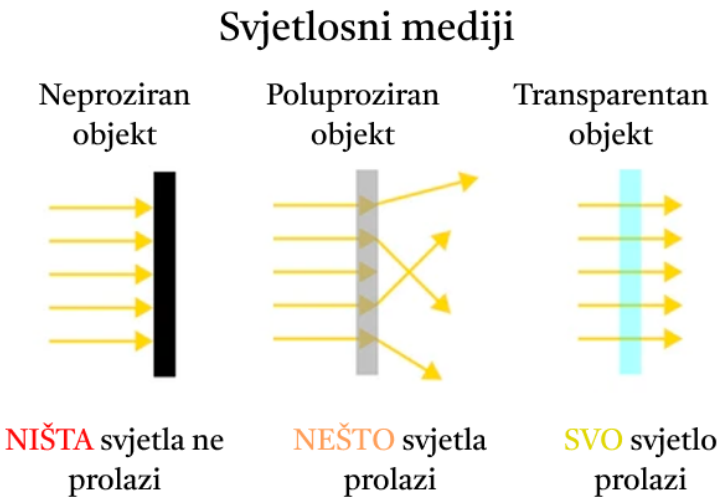
Pri određivanju oblika i položaja objekta u kontekstu robotske percepcije i manipulacije koriste se temeljni koncepti i metode. Prvi od njih jest uporaba senzora, poput kamera, laserskih skenera, 3D senzora i senzora dodira. Ovi senzori mogu mjeriti udaljenosti, detektirati rubove i konture objekata, te prikupiti informacije o geometriji i teksturi.

Kako bi se iz podataka dobivenih uporabom senzora došlo do korisnih i što točnijih informacija koriste se razne metode poput segmentacije, ekstrakcije značajki, modeliranja i prepoznavanja te je jednako bitna i kalibracija korištenih senzora. Postupak segmentacije se koristi za razdvajanje objekata od pozadine ili drugih objekata u sceni. Ovaj postupak omogućuje izdvajanje pojedinačnih objekata i stvaranje regija od interesa za daljnju analizu. Segmentacija se provodi metodama temeljenim na boji, rubovima ili regijama [10]. Metode temeljene na boji, poput K-means segmentacije, Otsuove metode i adaptivne segmentacija boje, koriste razlike u boji piksela slike kako bi odvojile određeni objekt od pozadine. Canny i Sobel metode detekcije rubova koriste gradijent slike kako bi identificirali promjene intenziteta koje signaliziraju prisutnost ruba. Canny detekcija rubova često se smatra naprednijom i preciznijom metodom od Sobel detekcije rubova, ali oba pristupa mogu biti korisna ovisno o specifičnim potrebama i zahtjevima aplikacije. Segmentacija temeljena na regijama grupira piksele sličnih svojstava u regije koje predstavljaju objekte. Nakon segmentacije, izdvojene regije se analiziraju kako bi se iz njih izdvojile relevantne značajke. To može uključivati izdvajanje rubova, linija, kutova, tekstura ili drugih karakteristika objekata koje mogu pomoći u identifikaciji i razumijevanju oblika i položaja. Histogram orijentiranih gradijenata (HOG), SIFT (Scale-Invariant Feature Transform) i SURF (Speeded-Up Robust Features) su neke od najboljih i najpoznatijih tehnika za izdvajanje značajki iz slika [11]. HOG koristi gradijente slike kako bi se dobile informacije o smjeru i intenzitetu rubova, dok SIFT i SURF pronalaze lokalne značajke koje ostaju nepromijenjene pri skaliranju, rotaciji i promjeni osvjetljenja. Ukoliko se radi sa specifičnim objektima, tehnika modeliranja i prepoznavanja može biti vrlo korisna. Ova tehnika uključuje razvoj modela koji opisuju oblike i položaje objekata od interesa. To može uključivati korištenje geometrijskih modela, statističkih modela ili modela

temeljenih na strojnom učenju. Prepoznavanje se provodi usporedbom prikupljenih značajki objekta s referentnim modelima kako bi se identificirao oblik i položaj. Osim navedenih tehnika bitnu ulogu u određivanju oblika i položaja objekta može imati i kalibracija korištenih senzora. Kalibracija se koristi za precizno određivanje položaja i orijentacije senzora u odnosu na robotsku platformu. To je važan korak kako bi se osigurala točnost mjerenja i interpretacija podataka o obliku i položaju objekata.

3.3. Transparentnost materijala i izazovi pri računalnoj percepciji transparentnih objekata

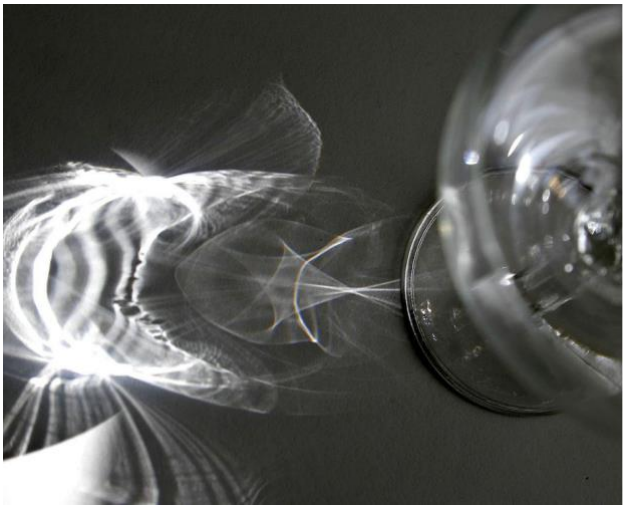
Transparentni materijali su materijali koji propuštaju svjetlost kroz sebe bez značajnijeg odbijanja ili raspršivanja. Kada je nešto transparentno, to znači da svjetlost slobodno prolazi kroz taj materijal, što omogućuje vidljivost iza promatranog objekta. Primjeri transparentnih materijala uključuju staklo, plastiku, vodu i zrak. Da bi se razumjelo kako transparentni objekti funkcioniraju, potrebno se upoznati s lomom svjetlosti. Kada svjetlost putuje kroz medij poput zraka i naiđe na granicu s drugim medijem poput stakla, dolazi do loma svjetlosti. Način prolaska i loma svjetlosti kroz svjetlosne medije je prikazan na slici 3.2. Taj fenomen je rezultat promjene brzine svjetlosti



Slika 3.2 Svjetlosni mediji i prikaz loma svjetlosti kroz njih

kada prelazi iz jednog medija u drugi. Lom svjetlosti može predstavljati izazov za senzore [12]. Na primjer, pri korištenju senzora za procjenu dubine, svjetlost koja putuje kroz transparentni objekt može biti lomljena na poluprozirnim ili neprozirnim objektima koji se nalaze iza transparentnog objekta, što rezultira pogreškama u procjeni dubine. Također, spekularne refleksije

na površini transparentnog objekta mogu ometati dobivanje točnih informacija o dubini ili uzrokovati nedostatak informacije. Osim toga, svjetlost koja prolazi kroz transparentne objekte može proizvesti artefakte poput kaustika. Kaustike su svijetle mrlje ili linije koje nastaju zbog interakcije svjetlosti s neravnim površinama ili izvorima svjetlosti što je i prikazano na slici 3.3.



Slika 3.3 Kaustika izazvana interakcijom svjetlosti s transparentnom čašom

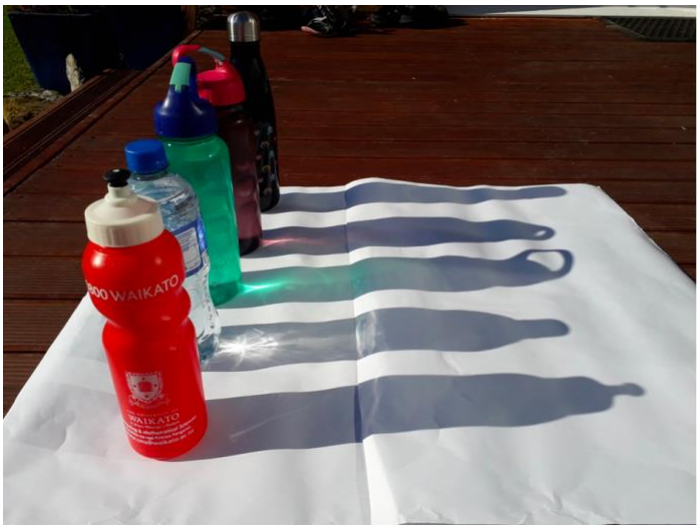
Ovi artefakti mogu dodatno otežati percepciju i obradu podataka dobivenih senzorom. Još jedan od problema je taj što pogled kroz više slojeva transparentnih materijala može rezultirati gubitkom kontrasta objekata. To može otežati identifikaciju i segmentaciju objekata u okolini. Transparentni materijali sa zakrivljenim površinama, poput zakrivljenih stakala ili leća, mogu uzrokovati pojavu znanu kao distorzija slike koja je prikazana na slici 3.4. Distorzija može utjecati na percepciju

Slika 3.4 Distorzija na površini zakrivljenog transparentnog objekta

oblika i položaja objekata. Također, veliki problem pri percepciji transparentnih objekata može se pojaviti kada transparentni objekti imaju sličan ili blizak indeks loma kao okolni medij, gdje razlikovanje tih objekata od pozadine može biti izazovno. Ovaj problem je prikazan na slici 3.5 [13].

Slika 3.5 Nevidljivost transparentnog medija (desna čaša) u tekućini kojoj je indeks loma blizak indeksu loma medija

To može rezultirati pogrešnim prepoznavanjem ili segmentacijom objekata. Isto tako, svjetlost koja prolazi kroz transparentne materijale može bitno utjecati na osvjetljenje objekata iza njih. To može uzrokovati izobličenja boje i sjene što je vidljivo na slici 3.6.

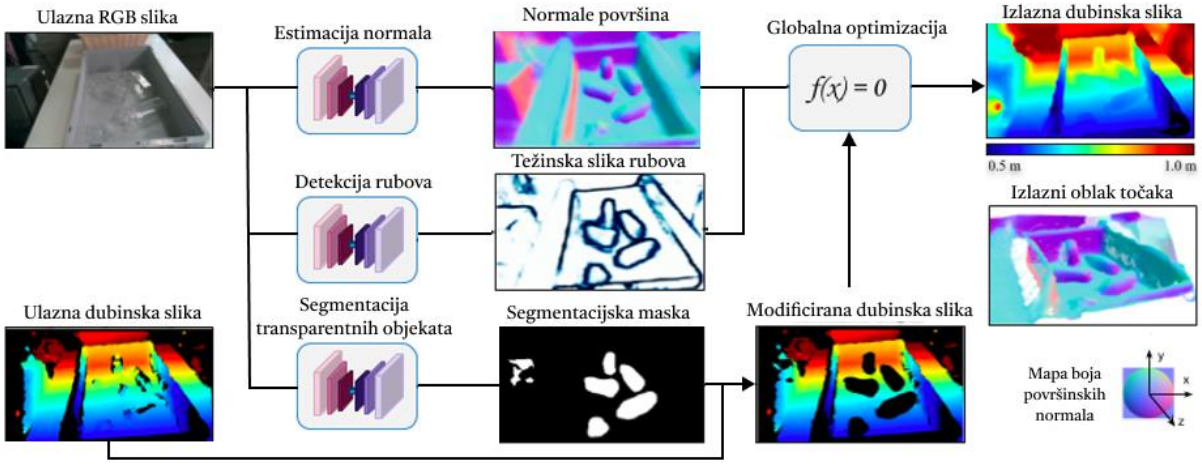


Slika 3.6 Izobličenje sjene i sjaj uzrokovan svjetlošću koja prolazi kroz transparentni materijal

4. METODOLOGIJA

4.1.Metoda ClearGrasp

Kao što je ranije spomenuto, u ovom radu je korištena metoda ClearGrasp za detekciju transparentnih objekata. ClearGrasp je model i sustav za percepciju i segmentaciju objekata temeljen na dubokom učenju [1]. Razvijen je s ciljem poboljšanja preciznosti i točnosti percepcije robotskih sustava u prepoznavanju i manipulaciji objektima.



Slika 4.1 Pregled metode

Metoda koristi tri neuronske mreže: jednu za predviđanje normala ravnina (engl. *Surface Normal Estimation*) što se može vidjeti na slici sa oznakom estimacija normala, drugu za detekciju rubova (engl. *Boundary Detection*) (uključujući i rubove kontakta) vidljivu na slici pod oznakom detekcija rubova, te treću za predviđanje maski piksela koji pripadaju transparentnim površinama označenom kao segmentacija transparentnih objekata na slici. Cilj je postići precizniju procjenu dubine transparentnih objekata, uzimajući u obzir refleksije i refrakcije svjetlosti koje se javljaju na površinama tih objekata. Ovaj sustav koristi kombinaciju RGB podataka i dubinskih informacija dobivenih pomoću 3D senzora kako bi stvorio precizne i detaljne segmentacije objekata. Primjer ovih podataka je vidljiv na slici u obliku ulazne RGB slike i ulazne dubinske slike. Metoda koristi neuronske mreže Deeplabv3+ i DRN-D-54 za predviđanje maski, normala ravnina i rubova, te se koristi globalna optimizacija, kao što je i vidljivo na slici 4.1, za rekonstrukciju 3D površina transparentnih objekata na temelju procijenjene dubine i predviđenih normala ravnina. Mreža za procjenu normala ravnina za svaki piksel RGB slike s ulaza predviđa x, y i z komponente normala ravnina, pri čemu se na taj izlaz primjenjuje L2 regularizacija kako

bi se osiguralo dobivanje normala u obliku jediničnih vektora. L2 regularizacija jest matematička tehnika skaliranja vektora ili matrica na način da imaju normu (duljinu) jednaku jedan. Izlaz mreže za detekciju rubova je mapa klasa koja označava svaki piksel slike kao jednu od tri klase: (a) ne-rub, (b) granica prekida dubine (engl. *occlusion boundary*) i (c) rubovi kontakta (kontaktne točke između objekata). Ova mapa klasa ima istu veličinu kao ulazna slika objekta i svaki piksel na mapi je označen odgovarajućom klasom. Označavanje piksela pomaže u razlikovanju različitih vrsta rubova. Osim toga, prisutnost rubova kontakta između objekata pomaže mreži u preciznijem predviđanju granica prekida dubine. Mreža za segmentaciju transparentnih objekata (engl. *Transparent Object Segmentation*) daje masku piksela koji pripadaju transparentnim površinama. Ova mreža omogućuje uklanjanje dubinskih mjerenja koji su nesigurni zbog transparentnosti objekta. Izlazi ovih mreža koriste se u postupku globalne optimizacije za rekonstrukciju 3D površina transparentnih objekata. U postupku globalne optimizacije u ClearGraspu za rekonstrukciju 3D površina transparentnih objekata koristi se algoritam koji je predložen u radu "DeepDepth" od strane Zhang i suradnika [14]. Nakon što se dobiju ulazne informacije, kao što su dubinska slika (s izostavljenim pikselima koji odgovaraju transparentnim površinama), predikcije površinskih normala te predikcije granica prekida dubine i rubova kontakta, ClearGrasp primjenjuje globalnu optimizaciju kako bi rekonstruirao dubinske vrijednosti za područja s nedostajućom dubinom. Algoritam globalne optimizacije rješava sustav jednadžbi s ciljem minimiziranja težinskog zbroja kvadratnih pogrešaka triju elemenata kako je prikazano u formuli (4-1),

$$E = \lambda_D E_D + \lambda_S E_S + \lambda_N E_N B, \tag{4-1}$$

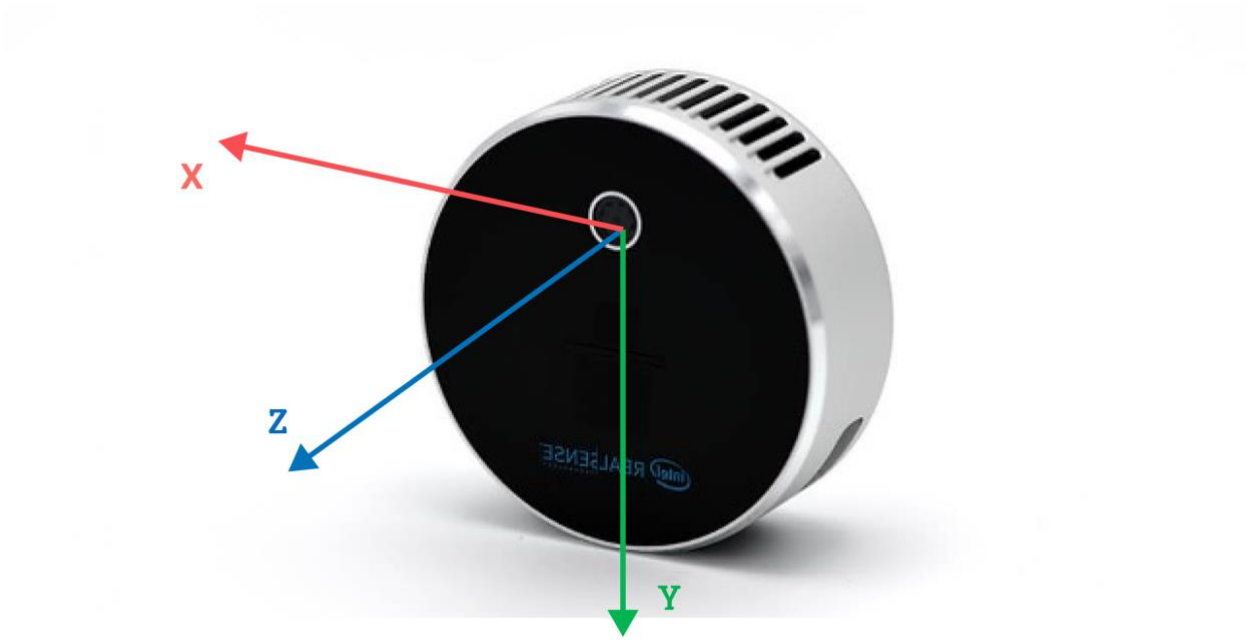
gdje su λ_D, λ_S i λ_N težinski faktori koji određuju važnost pojedinih termina, a

- E_D (Error of Depth): Mjeri udaljenost između predviđene dubine i promatrane dubine.
- E_S (Error of Smoothness): Mjeri razliku između dubine susjednih piksela kako bi se osigurala glatkoća rekonstrukcije.
- E_N (Error of Normals): Mjeri konzistentnost između predviđene dubine i predikcije površinskih normala.

Također, postoji težinski faktor B koji prigušuje elemente normala na temelju predikcije vjerojatnosti da je piksel na granici dvije površine različitih dubina. Ova vjerojatnost pomaže mreži kako ne bi došlo do krive procjene granice oko cijelog objekta, što bi otežalo globalnu optimizaciju za rješavanje dubine korištenjem predviđenih površinskih normala. Kroz eksperimente, vrijednosti težinskih faktora su podešene kako bi se postigla optimalna rekonstrukcija, a u slučaju mreže ClearGrasp, primjerene vrijednosti su λ_D = 1000, λ_S = 0.001 i λ_N = 1.0. Konačni rezultat provedbe ove metode je precizna i detaljna rekonstrukcija transparentnih objekata koja omogućuje daljnju analizu i manipulaciju.

4.2. Algoritam kalibracije kamere i robotskog manipulatora

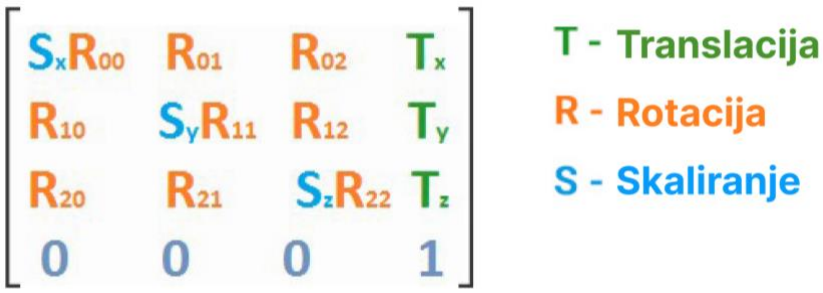
Kako su kamera i robotski manipulator postavljeni na različitim mjestima u prostoru potrebno je doći do njihovog odnosa u prostor. Koordinatni sustav kamere može se vidjeti na slici 4.2 dok se koordinatni sustav baze robotskog manipulatora može pronaći u trećem poglavlju na slici 3.1.



Slika 4.2 Koordinatni sustav kamere

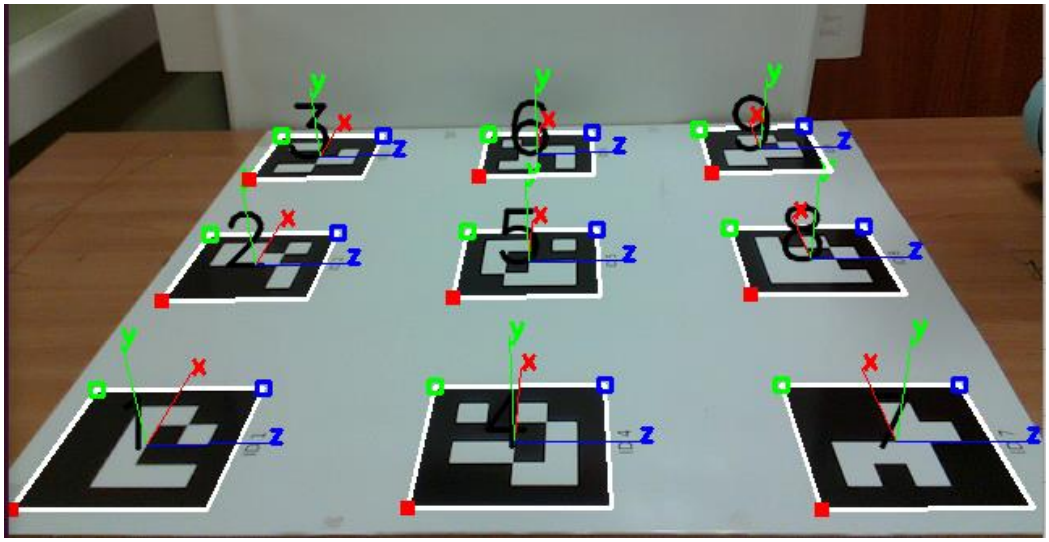
Ono što se dobije dodatnom obradom rezultata iz ClearGrasp modela jest centar objekta u koordinatnom sustavu kamere. Za manipulaciju tim objektom potrebno je doći do odgovarajuće točke u koordinatnom sustava baze robotskog manipulatora koja odgovara istoj toj točki u prostoru. Ono do čega je zapravo bilo potrebno doći jest transformacijska matrica koordinatnog

sustava kamere i koordinatnog sustava baze robotskog manipulatora. Transformacijska matrica je matematički alat koji se koristi za opisivanje transformacija u trodimenzionalnom prostoru. Ova matrica omogućuje definiranje geometrijskih transformacija, translacije (pomaka), rotacije i skaliranja, koje utječu na poziciju, orijentaciju i veličinu objekata. Transformacijska matrica izgleda kao što je prikazano na slici 4.3 i primarno se sastoji od translacijskog vektora, rotacijske



Slika 4.3 Struktura transformacijske matrice

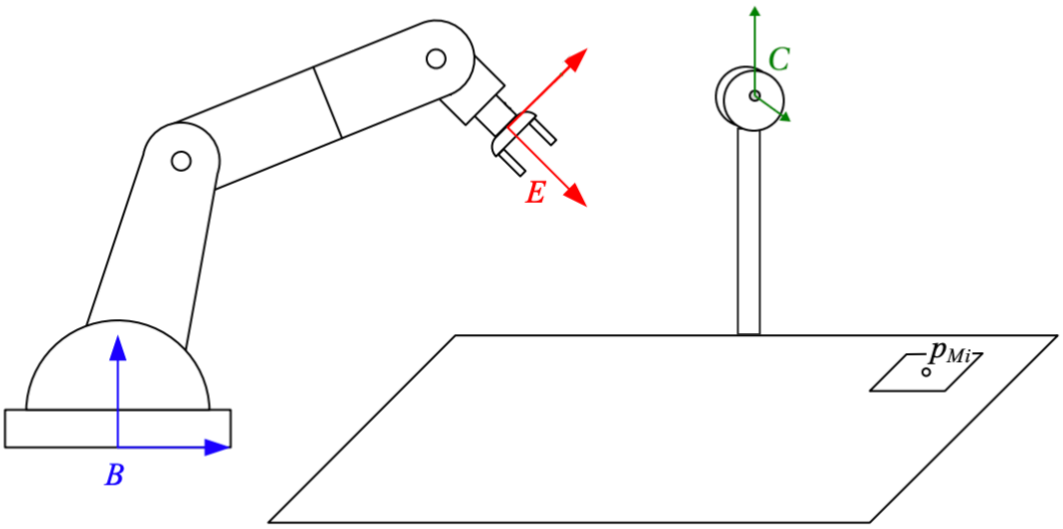
matrice i faktora skaliranja. Postoji više metoda za dobivanje transformacijske matrice između kamere i baze robotskog manipulatora kao što su kalibracija ArUco (engl. *Augmented Reality*



Slika 4.4 ArUco markeri u koordinatnom sustavu kamere

Markers) ili ručno postavljenim markerima, korištenje kalibracijskih uređaja, metode temeljene na poznatim točkama ili stereo vizija. U ovom diplomskom radu je odabrana metoda kalibracije ArUco markerima koji su prikazani na slici 4.4. ArUco je kratica za *Augmented Reality University*

of Cordoba u Španjolskoj gdje su i razvijeni. Ovi markeri su kvadratni crno-bijeli uzorci koji se koriste u računalnom vidu i proširenoj stvarnosti kako bi se omogućilo praćenje i prepoznavanje objekata u stvarnom svijetu putem kamere. Korišteni program za kalibraciju je gotovi program opisan u stručnom izvješću (engl. *Technical Report*) profesora Roberta Cupeca implementiran u njegovoj suradnji sa Lukom Loinom [15]. U izvješću se rješava nešto drugačiji problem od problema predstavljenog u ovome radu. Naime, razlika u rješavanom problemu jest pozicija kamere. U radu [15] kamera je pričvršćena na posljednji članak robotskoga manipulatora dok je u svrhu rješavanja problema ovog diplomskog rada kamera postavljena bočno od radnoga stola kako bi mogla snimiti scenu što je i prikazano na slici 4.5. Neovisno o tome, problemu se pristupa na sličan način.



Slika 4.5 Eksperimentalni postav scene pri kalibraciji kamere i robotskog manipulatora

Najprije se dodjeljuju koordinatni sustavi, kao što je prikazano na slici 4.5:

- koordinatni sustav E robotskom završetku (završni efektor),
- koordinatni sustav B njegovoj bazi i
- koordinatni sustav C kameri.

Nadalje, pretpostavlja se postojanje n markera M_i u radnom prostoru robotskog manipulatora te da je položaj vrška alata u odnosu na E definiran poznatim koordinatnim vektorom ${}^{E}p_{A}$. Neka je

položaj završnog efektora u kojem vršak alata dodiruje marker M_i označen s ${}^{B}T_{E_i}$. Tada se položaji svih markera u odnosu na B mogu izračunati kako je prikazano u formuli (4-2),

$$ {}^{B}\tilde{p}_{M_i} = {}^{B}T_{E_i}\,{}^{E}\tilde{p}_{A} \tag{4-2} $$

gdje se "~" koristi za označavanje homogenih koordinata. Zatim se pretpostavlja da će m slika biti snimljeno kamerom na kojima će biti vidljivi svi markeri, i da će položaj markera M_i u odnosu na C u j-tom prikazu biti definiran koordinatnim vektorom ${}^{C_j}\tilde{p}_{M_i}$. Onda vrijedi (4-3),

$$ {}^{B}\tilde{p}_{M_i} = {}^{B}T_{E'_j}\,{}^{E}T_{C}\,{}^{C_j}\tilde{p}_{M_i} \tag{4-3} $$

gdje ${}^{B}T_{E'_j}$ predstavlja položaj završnog efektora u odnosu na baznu poziciju robota koji odgovara j-tom prikazu. Pri implementaciji ovoga algoritma u ovom diplomskom radu broj snimljenih slika je 1. Iz formule (4-3) se dolazi do položaja markera M_i u odnosu na E u j-tom prikazu koji je definiran koordinatnim vektora ${}^{E'_j}\tilde{p}_{M_i}$ na sljedeći način:

$$ \begin{aligned} {}^{E'_j}\tilde{p}_{M_i} &= {}^{B}T^{-1}_{E'_j}\,{}^{B}\tilde{p}_{M_i} \\ {}^{E'_j}\tilde{p}_{M_i} &= {}^{B}T^{-1}_{E'_j}\,{}^{B}\tilde{p}_{M_i} \\ {}^{E'_j}\tilde{p}_{M_i} &= {}^{B}T^{-1}_{E'_j}\,{}^{B}T_{E_i}\,{}^{E}\tilde{p}_{A} \\ f\left({}^{E}T_{C}\right) &= \sum_{j=1}^{m}\sum_{i=1}^{n}\left\| {}^{E'_j}\tilde{p}_{M_i} - {}^{E}T_{C}\,{}^{C_j}\tilde{p}_{M_i}\right\|^2 \end{aligned} \tag{4-4} $$

Centroidi izmjerene točke u oba koordinatna sustava dani su redom s

$$ {}^{E}\overline{p} = \sum_{j=1}^{m}\sum_{i=1}^{n}{}^{E'_j}p_{M_i} \tag{4-5} $$

i

$$^{C}\overline{p} = \sum_{j=1}^{m}\sum_{i=1}^{n} {}^{C_j}p_{M_i} \,. \tag{4-6}$$

Svaka od izmjerenih točaka može biti predstavljena pomoću novih koordinata definiranih s formulom (4-7),

$$a_{ij} = {}^{E'_j}p_{M_i} - {}^{E}\overline{p}\,, \qquad b_{ij} = {}^{C_j}p_{M_i} - {}^{C}\overline{p} \tag{4-7}$$

Za određivanje transformacijske matrice između koordinatnog sustava robotskog završetka i koordinatnog sustava kamere ${}^{E}T_C$ potrebno je izračunati faktor razmjera s rotacijske matrice ${}^{E}R_C$. Faktor razmjera s može se odrediti čak i bez poznavanja same rotacijske matrice te se računa po formuli (4-8),

$$s = \sqrt{\frac{\sum_{j=1}^{m}\sum_{i=1}^{n}\left\|b_{ij}\right\|^2}{\sum_{j=1}^{m}\sum_{i=1}^{n}\left\|a_{ij}\right\|^2}} \tag{4-8}$$

gdje $\|\cdot\|^2$ označava L2 normu predanoga vektora. L2 norma, također poznata kao Euklidska norma, je matematička mjera duljine vektora u višedimenzionalnom prostoru i računa se kao kvadratni korijen sume kvadrata svih komponenti vektora. Rotacijsku matricu ${}^{E}R_C$ može se predstaviti pomoću jediničnog kvaterniona. Kvaternionom se može smatrati vektor s četiri komponente: jednom stvarnom komponentom, q_0, i tri imaginarne komponente, q_x, q_y, q_z kako je i prikazano u (4-9),

$$\dot{q} = q_0 + iq_x + jq_y + kq_z. \tag{4-9}$$

Kvaternion koji opisuje rotaciju između dva koordinatna sustava može biti određen uz pomoć matrice N:

$$N = \begin{bmatrix} \left(S_{xx}+S_{yy}+S_{zz}\right) & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & \left(S_{xx}-S_{yy}-S_{zz}\right) & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}+S_{yx} & \left(-S_{xx}+S_{yy}-S_{zz}\right) & S_{yz}+S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & \left(-S_{xx}-S_{yy}+S_{zz}\right) \end{bmatrix} \tag{4-10}$$

gdje je

$$S = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zx} & S_{zz} \end{bmatrix} = \sum_{j=1}^{m}\sum_{i=1}^{n} a_{ij} b_{ij}^{T} \, . \tag{4-11}$$

Kvaternion koji predstavlja rotaciju je svojstveni vektor koji odgovara najvećim svojstvenim vrijednostima matrice N. Rotacijska matrica, ${}^{E}R_{C}$, može se definirati korištenjem dobivenog kvaterniona ili svojstvenog vektora:

$${}^{E}R_{C} = \begin{bmatrix} 1-2q_y^2-q_z^2 & 2\left(q_x q_y - q_z q_0\right) & 2\left(q_x q_z + q_y q_0\right) \\ 2\left(q_x q_y + q_z q_0\right) & 1-2q_x^2-q_z^2 & 2\left(q_y q_z - q_x q_0\right) \\ 2\left(q_x q_z - q_y q_0\right) & 2\left(q_y q_z + q_x q_0\right) & 1-2q_x^2-q_y^2 \end{bmatrix}. \tag{4-12}$$

Tada je vektorski pomak između dva koordinatna sustava definiran sa:

$${}^{E}t_{C} = {}^{E}\overline{p} - s \cdot {}^{E}R_{C} \cdot {}^{C}\overline{p} \, . \tag{4-13}$$

Shodno tome je položaj ${}^{E}T_{C}$ potpuno definiran koristeći jednadžbe (4-12) i (4-13). Ovaj postupak, modificiran za primjenu u ovom diplomskom radu, se može zapisati i u obliku kalibracijskog

1: Za svaki marker $M_i, i = 1, \ldots, n$
2: Postavi vršak alata u centar markera
3: Snimi ${}^{B}T_{E_i}$.
4: Postavi kameru u poziciju iz koje su svi markeri sadržani u vidnom polju kamere.
5: Snimi sliku s kamerom.
6: Detektiraj markere u snimljenoj slici. Neka ${}^{C_j}p_{M_i}$ označava poziciju markera M_i u odnosu na koordinatni sustav kamere gdje je j = 1.
7: Snimi ${}^{C_j}p_{M_i}$, i = 1, …, n i ${}^{B}T_{E'_j}$.
8: Izračunaj ${}^{E'_j}\tilde{p}_{M_i}$, i = 1, …, n, j = 1 koristeći (4-4).
9: Izračunaj ${}^{E}\overline{p}$ koristeći (4-5).
10: Izračunaj ${}^{C}\overline{p}$ koristeći (4-6).
11: Izračunaj a_{ij}, b_{ij}, i = 1, …, n, j = 1 koristeći (4-7).
12: Izračunaj s koristeći (4-8).
13: Izračunaj S koristeći (4-11).
14: Izračunaj N koristeći (4-10).
15: Izračunaj ${}^{E}R_{C}$ iz kvaterniona koji predstavlja a koji predstavlja svojstveni vektor koji odgovara najvećoj svojstvenoj vrijednosti matrice N koristeći (4-12).
16: Izračunaj ${}^{E}t_{C}$ koristeći (4-13).
17: Vrati s, ${}^{E}R_{C}$ i ${}^{E}t_{C}$.

Pseudokod 4.1. Proces kalibracije kamere i robotskog manipulatora

algoritma što je i prikazano u pseudokodu 4.1.

5. IMPLEMENTACIJA I EKSPERIMENTALNA EVALUACIJA

5.1. Izbor opreme i alata

Odabir odgovarajuće opreme i alata ključan je za uspješnu provedbu eksperimenata. Za potrebe ovog diplomskog rada, koristi se LIDAR (engl. *Light Detection and Ranging*) kamera sa slike 5.1 kao senzor za prikupljanje podataka o okolini. Intel RealSense LiDAR Camera L515 je napredna

Slika 5.1 Intel RealSense LiDAR kamera L515

LIDAR kamera koja koristi laserske zrake za mjerenje udaljenosti i generiranje trodimenzionalne reprezentacije okoline [16]. S horizontalnim vidnim poljem od 70° i vertikalnim vidnim poljem od 55° snima širok spektar podataka okoline. Može snimati do 23 milijuna točaka u sekundi, pružajući visoku razlučivost i detaljne trodimenzionalne slike. Kamera također ima integrirani akcelerometar i žiroskop za stabilizaciju slike i praćenje pokreta. Podržava različite načine rada, uključujući snimanje oblaka točaka, obojenog oblaka točaka i dubinske slike. Često se koristi u robotici, autonomnoj vožnji, mapiranju i navigaciji, a primjene se nalaze i u industrijskim postavkama poput inspekcije, nadzora, planiranja prostora i virtualne stvarnosti. Što se tiče detekcije transparentnih objekata, pruža prednosti kao što je pružanje preciznih podataka o udaljenosti bez obzira na transparentnost i može detektirati promjene u svjetlosti i površinske nedostatke. Međutim, spekularni odrazi na transparentnim površinama mogu uzrokovati izobličenja u podacima o dubini, a snimanje složenih unutarnjih struktura ili finih detalja transparentnih objekata može biti neprecizno. Kamera pruža podatke o udaljenosti, dubini i trodimenzionalnoj strukturi, obično prikazane kao oblak točaka ili dubinska slika u formatima poput PLY (Polygon File Format), XYZ ili PCD (Point Cloud Data).

Za manipulaciju transparentnim objektima je korišten Universal Robots, UR5 robotski manipulator koji se može vidjeti na slici 5.2. To je kolaborativni industrijski robot namijenjen za

Slika 5.2 Universal Robots UR5 robotski manipulator

rad u blizini ljudi, s visokom preciznošću i fleksibilnošću u šest osi. Odlikuje ga visoka preciznost, ponovljivost i mogućnost suradnje s ljudima, dok su mu nedostaci ograničena nosivost (5 kg) i brzina (1 m/s) izvođenja radnji. Zahtijeva podatke o položaju objekata, dimenzijama i orijentaciji, često kroz XYZ koordinate, Eulerove kutove ili kvaternione, te senzorske podatke za detekciju objekata i orijentaciju. Uz robotski manipulator kao alat korištena je Universal Robots Robotiq 3-Finger hvataljka (engl. *gripper*) prikazana na slici 5.3 koja ima tri prsta

Slika 5.3 Universal Robots Robotiq 3-Finger hvataljka

dizajnirana za hvatanje i manipulaciju različitim objektima [17]. Prsti ove hvataljke mogu se

prilagoditi obliku i veličini objekta koji se hvataju, omogućujući hvatanje različitih oblika bez potrebe za preinakama. Jedna od iznimno korisnih značajki je precizna kontrola. Robotiq 3-Finger Gripper omogućava iznimno preciznu kontrolu nad položajem i pritiskom svakog prsta tijekom manipulacije objektima. Ova preciznost igra ključnu ulogu u sigurnom i učinkovitom rukovanju raznovrsnim objektima. Integracija ove hvataljke u postojeće robotske sustave jednostavna je i brza. Također, ugrađene sigurnosne značajke čine je sigurnom za rad u prisutnosti ljudi i okoline.

Programi za izvedbu obrade podataka i upravljanje robotskim manipulatorom su pisani pomoću programskog jezika Python u integriranom razvojnom okruženju Visual Studio Code. Python je visoko razinski, interpretirani programski jezik koji se ističe svojom jednostavnošću i čitljivošću. On je popularan izbor za različite vrste programiranja, uključujući web razvoj, znanstveno istraživanje, analizu podataka i automatizaciju zadataka. Python podržava različite paradigme programiranja poput proceduralnog, objektno orijentiranog i funkcionalnog programiranja. Integrirano razvojno okruženje (IDE) Visual Studio Code (VS Code) pruža bogat skup alata i funkcionalnosti za razvoj aplikacija. On podržava više programskih jezika, uključujući Python, i nudi korisne značajke poput sintaksnog označavanja, automatskog dovršavanja koda, debagiranja, kontrolu verzija i integraciju terminala. VS Code je popularan među programerima zbog svoje brzine, jednostavnosti korištenja i fleksibilnosti. Python i VS Code često se koriste zajedno za razvoj Python aplikacija. Programeri mogu pisati, testirati i pokretati Python kod u VS Code-u, iskoristiti njegove dodatke za proširenje funkcionalnosti i integrirati ga s drugim alatima i servisima. To čini kombinaciju Python i VS Code idealnom za produktivan i učinkovit razvoj softvera.

ROS je fleksibilan i distribuirani okvir (engl. *framework*) za razvoj robotskih aplikacija. To je kolekcija biblioteka i alata koji olakšavaju programiranje, upravljanje senzorima i aktuatorima, komunikaciju između modula i izgradnju složenih robotskih sustava. Kako je rad izvođen na računalu sa Ubuntu 20.04 operacijskim sustavom za upravljanje robotskim manipulatorom korištena je pripadajuća Noetic verzija ROS komunikacijskog protokola za kojega je bilo potrebno omogućiti korištenje funkcionalnosti u Pythonu i olakšati integraciju s drugim bibliotekama i alatima. Za omogućavanje tih funkcionalnosti, u Pythonu, postoje popularne biblioteke poput rospy, roscpp, roslaunch i rviz koje pružaju podršku za razvoj ROS aplikacija.

5.2.Proces kalibracije kamere i robotskog manipulatora

Za kalibraciju je najprije potrebno postaviti ArUco markere u prostor gdje su vidljivi kameri i ujedno u dometu alatu robotskog manipulatora. Iz ovog razloga markeri su postavljeni na stol na kojemu je smješten i robotski manipulator te je kamera iz istoga razloga postavljena bočno od stola na određenu udaljenost iz koje može uhvatiti sve markere. Kada je radno okruženje postavljeno slijedi instalacija ROS-a. Ona uključuje ažuriranje paketa i njihovu distribuciju, dodavanje ROS repozitorija u sustav, instaliranje potrebnih alata te instaliranje određene verzije ROS-a putem naredbe koja je vidljiva u kodu 5.1. Nakon instalacije, potrebno je inicijalizirati ROS,

```
        sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
1:      $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
2:      sudo apt install curl
        curl -s
        https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
3:      sudo apt-key add -
4:      sudo apt update
5:      sudo apt install ros-noetic-desktop-full
```

Kod 5.1 Instalacija ROS Noetic okruženja

a potom i pozvati odgovarajuće skripte za pravilno pokretanje ROS-a. Instalacija ArUco markera, ključnog za precizno praćenje objekata, uključuje instalaciju Python paketa putem pip alata. Proces se temelji na uputama s web stranice, uz dodatne ispravke kako bi se prevladale potencijalne prepreke. Za postavljanje kamere, koriste se dokumentirani koraci kako bi se osigurala ispravna konfiguracija. Ujedno, dodaju se Udev (engl. *Userspace device manager*) pravila kako bi se omogućilo pravilno prepoznavanje kamere. Udev pravila su skup pravila i konfiguracija koji se koriste u operacijskim sustavima temeljenim na Linuxu kako bi se omogućila interakcija i upravljanje različitim uređajima spojenim na računalo. Instalacija Moveita, ključnog alata za planiranje pokreta robota, obavlja se kroz ROS naredbe vidljive u kodu 5.2. Moveit je softverski okvir (engl. *framework*) otvorenog koda koji pruža alate i biblioteke za planiranje i

```
1:      sudo apt-get install ros-noetic-moveit -y
2:      sudo apt-get install ros-noetic-trac-ik -y
```

Kod 5.2 Instalacija MoveIt paketa

izvođenje pokreta robotskih manipulatora što programerima omogućava da efikasno planiraju i izvode pokrete robotskih manipulatora kako bi postigli željene zadatke, poput hvatanja, premještanja i manipulacije objektima u prostoru. Nadalje, provodi se instalacija SOEM (engl. *Simple Open EtherCAT Master*) paketa koji omogućava komunikaciju s robotskom hvataljkom

koja se izvodi naredbom sa koda 5.3. Za pretvorbu između oblika pozicije i kvaterniona

```
1:      sudo apt-get install ros-noetic-soem
```

Kod 5.3 Instalacija SOEM paketa

i oblika matrice potrebno je preuzeti biblioteku transforms3d pomoću pip alata. Kada je sve navedeno preuzeto i postavljeno potrebno je postaviti okruženje programa koje se provodi pozivanjem naredbi sa koda 5.4 nakon čega je još potrebno kalibrirati robotskog manipulatora što

```
        mkdir -p calibration_program/catkin_ws/src && cd
1:      calibration_program/catkin_ws/src
        git clone
2:      https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.git
        git clone -b calibration_devel
3:      https://github.com/fmauch/universal_robot.git
4:      git clone https://github.com/TAMS-Group/robotiq
5:      cd robotiq
6:      git checkout noetic-devel
7:      cd ../..
8:      sudo apt-get update && sudo apt-get upgrade -y
9:      rosdep update
10:     rosdep install –from-paths src –ignore-src -y
11:     catkin_make
```

Kod 5.4 Postavljanje okruženja projekta

je izvedeno pokretanjem programa za kalibraciju koji će stvoriti my_robot_calibration.yaml dokument koji sadrži rezultate kalibracije. Ovaj postupak izrade kalibracije robotskog manipualtora je prikazan u kodu 5.5. Nakon što je stvoren my_robot_calibration.yaml dokument

```
1:      source devel/setup.bash
2:      mkdir ~/.ros/robot
        roslaunch ur_calibration calibration_correction.launch
        robot_ip:=192.168.22.14
3:      target_filename:=”robot/my_robot_calibration.yaml”
4:      cp ~/.ros/robot/my_robot_calibration.yaml
5:      (pwd)/src/universal_robot/ur_description/urdf/my_robot_calibration.yam
```

Kod 5.5 Izrada kalibracije robota

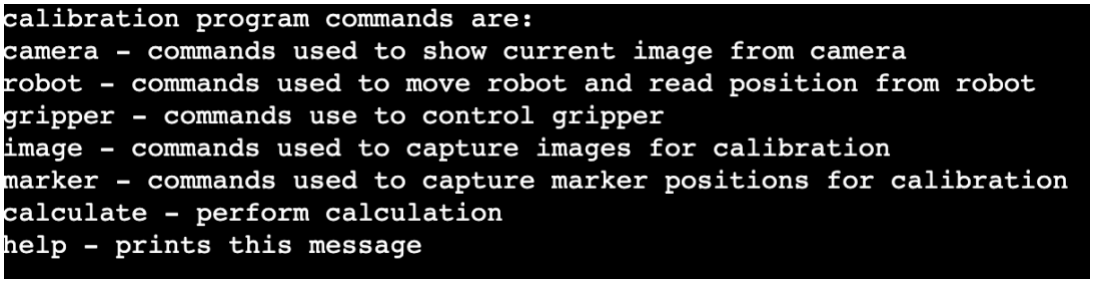
moguće je pokrenuti program za kalibraciju kako bi se došlo do transformacijske matrice. Uz pokretanje skripte za kalibraciju potrebno je u isto vrijeme pokrenuti i sve naredbe prikazane u kodu 5.6 kojima se pokreću svi ROS čvorovi potrebni za izvršavanje kalibracije. Svaka od ovih linija koda se pokreće u zasebnom tekstualnom sučelju (engl. *terminal*). Najprije se pokreće roscore koji je ključni čvor (engl. *node*) u ROS ekosustavu. To je centralna komponenta koja pokreće ROS centralni registar (engl. *master server*) i postavlja temelj za komunikaciju između

različitih ROS čvorova unutar robotskog sustava. Roscore je neophodan za pokretanje i koordinaciju ROS aplikacija i omogućava im da međusobno razmjenjuju podatke, naredbe i informacije. Nakon toga se pokreće upravljački program (engl. *driver*) kamere, što je softverska komponenta koja omogućava komunikaciju i upravljanje Intel RealSense kamerama u računalnom sustavu. Shodno tome, potrebno je pokrenuti i upravljački program za komunikaciju i upravljanje

```
1:      roscore
        roslaunch realsense2_camera rs_camera.launch enable_pointcloud:=true
        depth_width:=640 depth_height:=480 depth_fps:=15 color_width:=640
2:      color_height:=480 color_fps:=15 align_depth:=true
        source devel/setup.bash & roslaunch ur_robot_driver ur5_bringup.launch
3:      robot_ip:=192.168.22.14 kinematics_config:=”my_robot_calibration.yaml“
        source devel/setup.bash & roslaunch ur5_moveit_config
4:      ur5_moveit_planning_execution.launch limited:=true
        source devel/setup.bash & rosrun robotiq_3f_gripper_control
5:      Robotiq3FGripperTcpNode.py 192.168.22.11
6:      source devel/setup.bash & rosrun calibration_program calibration.py
```

Kod 5.6 Pokretanje programa za kalibraciju

Universal Robots robotskim manipulatorima putem ROS okvira kao i za uspostavljanje komunikacije s Robotiq 3-Finger hvataljkom preko TCP/IP (engl. *Transmission Control Protocol/Internet Protocol*) veze koristeći njegovu IP (engl. *Internet Protocol*) adresu (u ovom slučaju 192.168.22.11). Uz sve upravljačke programe potrebno je i pokrenuti ROS datoteku za pokretanje (engl. *launch file*) koja se koristi za pokretanje komponente planiranja kretanja za Universal Robots robotski manipulator u okviru MoveIt paketa. Ova datoteka igra ključnu ulogu u postavljanju i upravljanju planiranjem kretanja za Universal Robots robotskog manipulatora unutar ROS okvira. Datoteke za pokretanje (često s ekstenzijom .launch) su XML datoteke koje definiraju kako će se konfigurirati i pokrenuti različite ROS komponente, kao što su čvorovi, parametri, servisi i drugi čimbenici. Ove datoteke omogućuju programerima da organiziraju i upravljaju raznim čvorovima i komponentama svojeg robotskog sustava na koordiniran način. Kada je sve navedeno pokrenuto i ne javlja nikakve greške pri radu potrebno je otvoriti novo tekstualno sučelje iz kojega se pokreće glavni program za kalibraciju. Ovaj program ima ranije



Slika 5.4 Popis naredbi i njeni opisi dobiveni upisivanjem naredbe "help"

definiran skup naredbi koje se mogu vidjeti upisivanjem naredbe „help“ unutar tekstualnog sučelja. Popis naredbi koji se dobije upisivanjem naredbe „help“ je prikazan na slici 5.4. Da bi se uspješno provela kalibracija potrebno je spremiti poziciju markera u koordinatnom sustavu kamere i koordinatnom sustavu baze robotskog manipulatora. Ovaj postupak se provodi upisivanjem sljedećih naredbi u tekstualno sučelje sljedećim redoslijedom:

- image capture: Snima sliku i sprema je u listu.
- camera aruco: Prikazuje trenutačnu sliku s detektiranim ArUco markerima. Primjer ove slike je vidljiv na slici 4.4.
- camera coords <marker index>: Izračunava koordinate do središta markera na temelju trenutačne slike kamere.
- marker capture <id>: Snima marker u memoriju programa.
- calculate: Izvršava izračun te kao rezultat vraća transformacijsku matricu.

Naredbe na izračunavanje koordinata do središta markera i snimanje markera potrebno je pokrenuti za svaki marker posebno upisivanjem broja markera unutar šiljastih zagrada.

5.3.Eksperimentalno okruženje

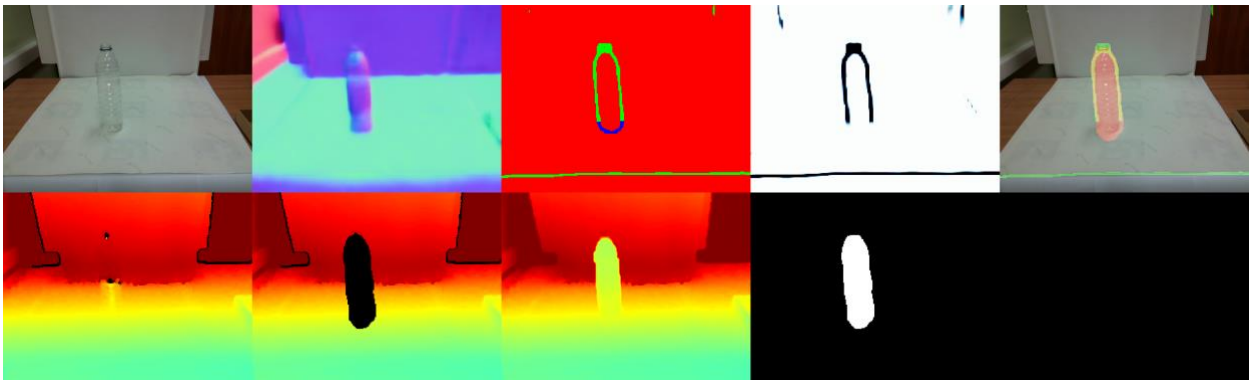
Eksperimentalno okruženje igra ključnu ulogu u osiguravanju pouzdanih i reproduktivnih

Slika 5.5 Eksperimentalno okruženje

rezultata. Kao što je navedeno, korištena oprema se sastoji od UR5 robotskog manipulatora

postavljenoga na stolu, Intel RealSense LiDAR L515 kamere koja je smještena lijevo od robotskog manipulatora odnosno u našem slučaju u negativnom smjeru x osi robotskog manipulatora te desetak centimetara u pozitivnom smjeru y osi baze robotskoga manipulatora. Ovo okruženje se može vidjeti na slici 5.5 te je ovako postavljeno kako bi se snimani prostor kamerom poklapao sa prostorom dometa alata robotskog manipulatora.

Kamera je spojena na računalo putem 3.1 generacije USB-a tipa C, dok se računalo i robotski manipulator povezuju spajanjem na postojeću Ethernet mrežu. Kako je kamera poprilično osjetljiva na svjetlost, za osvjetljenje je korišteno unutarnje umjetno svjetlo. Prilikom nekoliko pokusa pokazalo se da objekte koji nisu transparentni mreža zna detektirati kao transparentne ukoliko se nalaze na udaljenosti većoj od jednoga metra. Stoga je scenu na kojoj su smješteni objekti za detekciju bilo bitno ograditi na način da prevelike dubine i drugi objekti ne mogu utjecati na detekciju objekta. Primjer detekcije se može vidjeti na slici 5.6 gdje je prva slika u mreži ulazna, druga slika predstavlja normale ravnina, na trećoj slici su prikazani predviđeni rubovi, četvrta sadrži granične težine okluzije(engl. *occlusion boundary weights*), peta je zapravo ulazna RGB slika preko koje je ružičastom bojom postavljena maska, a žutom i zelenom rubovi, šesta slika predstavlja ulaznu dubinsku sliku, sedma predobrađenu ulaznu dubinsku RGB sliku, na osmoj slici



Slika 5.6 Rezultat detekcije transparentnog objekta korištenjem ClearGrasp metode

se nalazi izlazna dubinska RGB slika dok posljednja slika predstavlja masku detektiranoga objekta. U ovome primjeru se mogu vidjeti ulazi i izlazi iz ClearGrasp mreže kao i međurezultati kojima se ona koristi za dobivanje krajnjih rezultata. Potrebno je uočiti kako ulazna dubinska slika ne daje gotovo nikakvu informaciju o transparentnom objektu dok je primjenom ClearGrasp modela objekt u potpunosti detektiran.

5.4. Provođenje pokusa

Proces detekcije i manipulacije transparentnih objekata sastoji se od nekoliko koraka. Najprije se postavlja LiDAR kameru u prostor manipulacije kako bi se dobile dubinske informacije i stvorila percepcija prostora. Nakon toga, na zahtjev korisnika, uzima se slika scene koja uključuje transparentne objekte kojima se manipulira. Dobiveni rezultati detekcije i segmentacije transparentnih objekata uz pomoć ClearGrasp mreže se obrađuju kako bi se postavili ciljevi za robotski manipulator. To uključuje određivanje pozicije, oblika i dimenzija objekata, što omogućuje planiranje potrebnih manipulacijskih radnji. Nakon što su postavljeni ciljevi, koriste se Python skripte uz pomoć ROS-a za upravljanje robotskim manipulatorom. Ove skripte omogućuju precizno i sigurno hvatanje i manipulaciju transparentnih objekata. Robotski manipulator premješta i manipulira objektima prema zadanim ciljevima. Naposljetku se koriste programirane skripte i algoritmi koji omogućuju precizno pozicioniranje, rotiranje i postavljanje objekata na željena mjesta.

5.5. Implementacija ClearGrasp metode

Kako bi se čitav pokus mogao pokretati pomoću jedne datoteke za pokretanje bilo je potrebno prilagoditi ClearGrasp metodu na način da se pokreće u obliku ROS čvora. Ovaj proces uključuje nekoliko koraka. Prvo, ROS okolina mora biti inicijalizirana u Python skripti kako bi komunikacija s ROS-om bila omogućena. To uključuje uvođenje ROS biblioteka i inicijalizaciju čvora. Ovdje su nabrojene i opisane biblioteke i paketi potrebni za prilagodbu izvornoga koda:

- rospy: Python biblioteka koja omogućuje Python skriptama da komuniciraju s ROS-om, objavljuju ROS poruke i primaju ROS poruke.
- message_filters: Message Filters je biblioteka koja pomaže pri obradi i filtriranju ROS poruka kako bi se osigurala njihova sinkronizacija i koherentnost.
- Image, CameraInfo: Omogućuju pristup ROS porukama tipa "Image" i "CameraInfo". "Image" poruke obično sadrže slikovne podatke, dok "CameraInfo" poruke sadrže informacije o kameri kao što su kalibracija, matrice projekcije i slično.
- CvBridge: Biblioteka za konverziju slika iz ROS formata u OpenCV format kako bi se omogućila obrada slika u Pythonu pomoću OpenCV-a.

Bitno je napomenuti da je neke od ovih biblioteka potrebno preuzeti dok su neki preuzeti prethodnom instalacijom ROS-a. Nakon podešavanja ROS okoline dolazi do mijenjanja samog koda skripte. U glavnoj funkciji programa ostaje samo inicijalizacija objekta novodefinirane klase što je prikazano u kodu 5.7. U mnogim ROS programima, čvorovi se često definiraju u obliku klasa, iako korištenje klasa nije obavezno. Čvorovi su izvršne jedinice u ROS-u koje obavljaju određene zadatke. Svaki čvor je samostalna jedinica koja može primati i slati poruke. Čvorovi obično obavljaju specifične funkcionalnosti ili zadatke u robotskom sustavu. Pri prilagodbi koda je izvršni program implementacije ClearGrasp metode predstavljen klasom *ClearGraspNode*, odnosno ROS čvorom. U konstruktoru ove klase se inicijalizira ROS čvor i pretplate na teme (engl.

```
1:      source /opt/ros/noetic/setup.bash
```

Kod 5.8 Inicijalizacija radnog okruženja

topic) *'/camera/color/image_raw'*, *'/camera/aligned_depth_to_color/image_raw'* i *'/camera/color/camera_info'*, a u metodi callback se definira logika za obradu dolaznih poruka.

```
1:      mkdir -p ~/live_demo_ws/src
2:      cd ~/live_demo_ws/
3:      catkin_make
```

Kod 5.9 Stvaranje ROS radnog okruženja

Tema *'/camera/color/image_raw'* sadrži sliku u boji snimljenu s kamerom, *'/camera/aligned_depth_to_color/image_raw'* tema sadrži dubinsku sliku koja je poravnata sa slikom u boji, dok *'/camera/color/camera_info'* tema sadrži kalibracijske informacije kamere, poput intrinzičnih parametara kamere. ROS pretplaćivanje (engl. *subscription*) omogućuje čvoru da se pretplati na određenu temu kako bi primao podatke koji se objavljuju na toj temi. Tema je naziv pod kojim se objavljuju određene vrste podataka, kao što su senzorski podaci, slike, laserska

```
1:      if __name__ == '__main__':
2:          cgn = ClearGraspNode()
```

Kod 5.7 Glavna funkcija programa live_demo.py

očitavanja itd. Kako je čvor pretplaćen na navedene teme, *callback* metoda se automatski poziva kako bi se obradile te poruke. Obrada poruka se odvija u novodefiniranoj metodi *main* koja sadrži istu logiku koja je već postojala u prvobitnom kodu. Potpuni prilagođeni kod je vidljiv u prilogu ovog rada. Osim mijenjanja koda programa potrebno je i postaviti radno okruženje (engl. *workspace*). Ono se postavlja prvenstveno inicijalizacijom pomoću naredbe prikazane u kodu 5.8. Nakon toga se pomoću niza naredbi vidljivih u kodu 5.9 kreira vlastito radno okruženje čime će se stvoriti struktura direktorija za ROS. Unutar radnog okruženja je potrebno stvoriti ROS paket u kojemu će se nalaziti prethodno mijenjani program. Paket je organizacijska jedinica koja sadrži

ROS čvorove, biblioteke, konfiguracijske datoteke i druge resurse potrebne za izvođenje određene funkcionalnosti. On se kreira pozivanjem 'catkin_create_pkg ' naredbe iz direktorija (engl. *directory*) '/src' unutar radnog okruženja gdje se pohranjuju izvorni kodovi ROS paketa. Ova naredba zahtijeva nekoliko argumenata, uključujući ime paketa i popis ovisnosti koje će paket koristiti što je vidljivo u kodu 5.10. U stvoreni ROS paket, točnije u '/src' direktorij tog paketa, sprema se izmjenjeni kod koji je moguće pokrenuti korištenjem 'rosrun' naredbe kao što je i prikazano u kodu 5.11.

5.6. Implementacija pokusa

Pokusi su provedeni u ranije opisanom eksperimentalnom okruženju kako bi se smanjile vanjske smetnje i testirala učinkovitost ClearGrasp mreže. Kako se kao rezultat detekcije ClearGrasp mrežom dobije oblak točaka čitave scene, najprije je potrebno obraditi dobiveni rezultat kako bismo izdvojili samo objekt od interesa. Osim oblaka točaka mreža daje i masku transparentnog detektiranoga objekta pomoću koje se dobije oblak točaka objekta. Način izdvajanja oblaka točaka objekta iz scene se može vidjeti u priloženome kodu 5.12. Kod ovog postupka najprije je potrebno

```
1:      kernel = np.ones((5, 5), np.uint8)
2:      img_erosion = cv2.erode(self.mask_predicted, kernel, iterations=5)
3:      output_depth_object = img_erosion * self.output_depth
```

Kod 5.12 Izdvajanje oblaka točaka objekta iz oblaka točaka scene

suziti masku transparentnog objekta kako bi se objekt prikazao sa što manje odstupanja ili stršećih (engl. *outlier)* vrijednosti. Ovo je izvedeno erozijom maske transparentnog objekta. Erozija slike

```
1:      catkin_create_pkg cleargrasp rospy std_msgs
```

Kod 5.10 Stvaranje ROS paketa

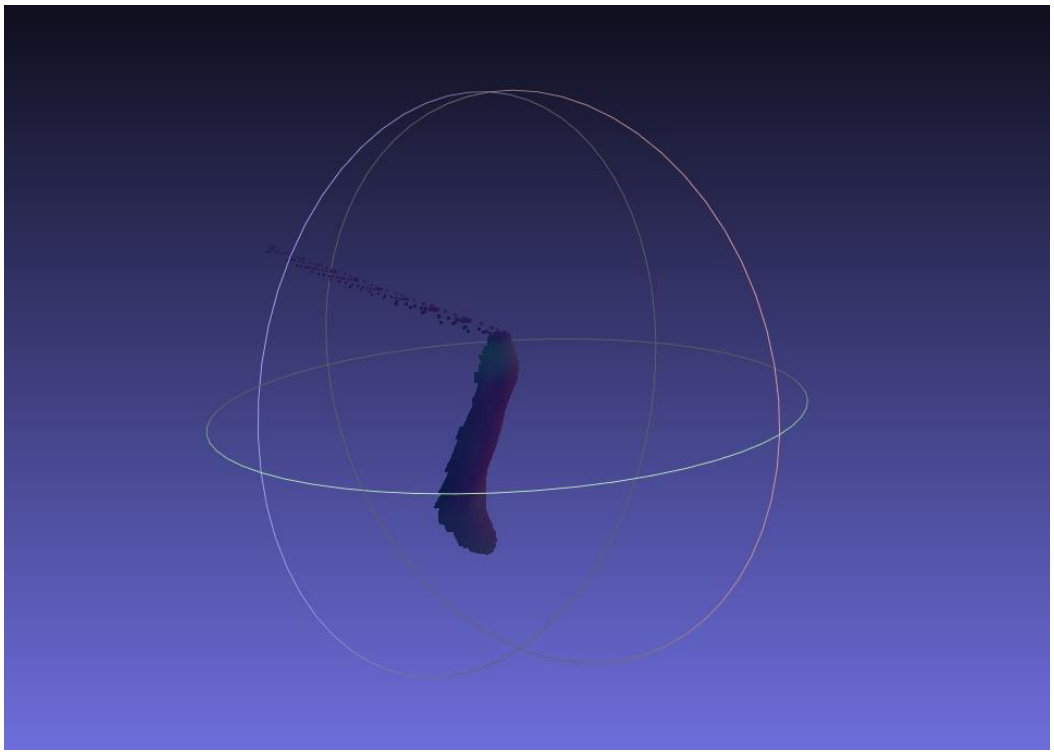
je osnovna operacija u obradi slika koja se koristi za smanjenje objekata na slici ili maski. Ova operacija temelji se na konceptu pomicanja strukturnog elementa (obično kvadratnog ili pravokutnog oblika) preko slike i provjere kako se taj element uklapa s pikselima na slici.

```
1:      rosrun cleargrasp live_demo.py
```

Kod 5.11 Naredba za pokretanje ROS skripti

Kvadratni oblik koji se koristi u eroziji slike (kao i u dilataciji i drugim morfološkim operacijama) naziva se strukturom elementa ili jezgrom (engl. *kernel*). Strukturni elementa je matrica ili kvadrat koji se koristi za oblikovanje i transformaciju piksela na slici tijekom morfoloških operacija. Najprije se inicijalizira matrica veličine 5 koja predstavlja jezgru, nakon čega se pomoću funkcije

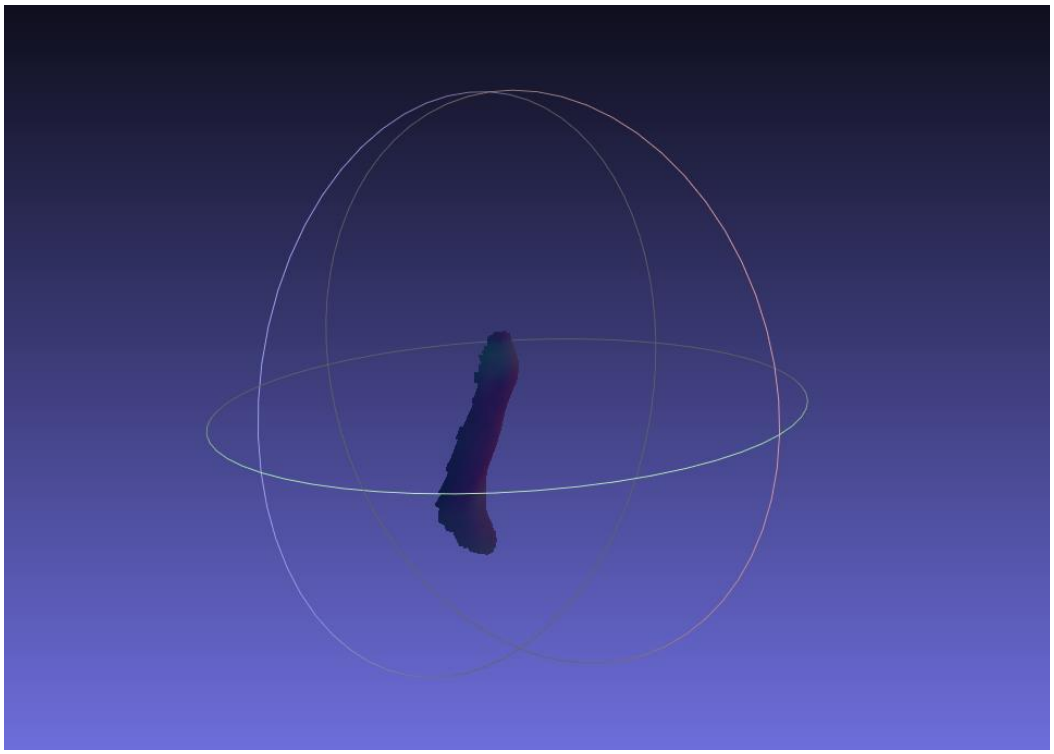
za eroziju iz biblioteke OpenCV (engl. *Open Source Computer Vision Library*) erodira početna



Slika 5.7 Primjer oblaka točaka transparentnog objekta

maska transparentnog objekta s ranije definiranom jezgrom [18]. Rezultat ove funkcije je nova slika (*img_erosion*) u kojoj su objekti smanjeni ili suženi. Kako bi se dobio oblak točaka izdvojenoga objekta potrebno je iz dubinske slike maknuti sve ostale točke osim onih koje su dio samoga objekta. Ovo se postiže množenjem izlazne dubinske slike sa erodiranom maskom. Pošto je maska zapravo matrica popunjena vrijednostima 0 ili 1 sve što je crno na maski objekta će biti crno i u novoj dubinskoj slici (*output_depth_object*). Odnosno, svim pikselima konačne dubinske slike koji nisu dio objekta će dubina biti postavljena na 0. Za sve točke dubinske slike s dubinom 0 smatra se da su u beskonačnoj udaljenosti od kamere. U kontekstu generiranja oblaka točaka, to znači da te točke neće biti uključene u rezultirajući oblak točaka. Izlazni oblak točaka s kojim se izvodi daljnji pokus može se vidjeti na slici 5.7. Ovi oblaci točaka znaju sadržavati odstupanja od stvarnoga objekta u obliku točaka koje predstavljaju objekt, ali zapravo nisu dio objekta. Kako bi se uklonila ova odstupanja ili stršeće vrijednosti korištena je Python biblioteka Open3D, odnosno njena metoda *remove_statistical_outlier()* koja iz predanoga oblaka točaka (*point_cloud_og*) miče odstupanja koja ne smatra dijelom toga oblaka točaka [19]. Nakon provedbe ove metode potrebno je izabrati pogodne točke koje ne uključuju stršeće vrijednosti što se postiže metodom biblioteka Open3D *select_by_index()*. Primjer izlaznog oblaka točaka sa stršećim vrijednostima prikazan je

na slici 5.7. Kada su sva odstupanja uklonjena dobije se oblak točaka kao na slici 5.8(*point_cloud*).



Slika 5.8 Primjer oblaka točaka transparentnog objekta bez „stršećih“ vrijednosti

Ovakav oblak točaka je prikladan za određivanje centra objekta. Iz njega je potrebno doći do graničnoga okvira (engl. *bounding box*) objekta iz kojega se računa njegov centar. Kako bi se došlo do graničnog okvira objekta također se koristi biblioteka Open3D, odnosno njena metoda *get_oriented_bounding_box()*. Centar objekta se naposljetku dobije metodom *get_center()*. Ovime je uspješno je odrađena detekcija i obrada njenih rezultata. Kod koji je bilo potrebno implementirati je vidljiv u kodu 5.13. Manipulacija objektom se sastoji od hvatanja objekta i

```
1:      point_cloud_og = o3d.io.read_point_cloud(path + "output-point-
2:      cloud/output-point-cloud-object.ply")
3:      _, ind = point_cloud_og.remove_statistical_outlier(30, 7.0)
4:      point_cloud = point_cloud_og.select_by_index(ind)
5:      bounding_box = point_cloud.get_oriented_bounding_box()
6:      center = bounding_box.get_center()
```

Kod 5.13 Postupak dobivanja centra objekta iz oblaka točaka objekta

njegovog premještanja na ranije definiranu lokaciju. Kod korišten za manipulaciju objektom je napisan unutar *move_robot.py* Python skripte koja je konstruirana u obliku ROS čvora po uzoru na prethodno implementiranu ClearGrasp metodu. Za razliku od implementacije ClearGrasp metode ovdje je manipulacijski čvor implementiran da objavljuje (engl. *publish*) podatke na temu umjesto pretplaćivanja na istu. Klasa *RobotControl* kojom je definiran ROS čvor za manipulaciju

i njen konstruktor su prikazani u kodu 5.14. Za upravljanje

```
1:      class RobotControl:
2:          def __init__(self):
3:              rospy.init_node('move_robot', anonymous=True)
4:              self.robot = moveit_commander.RobotCommander()
5:              self.scene = moveit_commander.PlanningSceneInterface()
6:              self.group_name = "manipulator"
                self.move_group =
7:      moveit_commander.MoveGroupCommander(self.group_name)
                self.pub = rospy.Publisher('Robotiq3FGripperRobotOutput',R
8:      obotiq3FGripperRobotOutput, queue_size=10)
9:              self.ik = IK('base_link', 'tool0', solve_type="Distance")
```

Kod 5.14 Konstruktor RobotControl klase

robotskim manipulatorom korištena je Python biblioteka *moveit_commander*. S *moveit_commander* mogu se izvoditi zadatci kao što je postavljanje ciljanih pozicija i orijentacija za manipulator, planiranje putanja, izvođenje tih putanja te praćenje i upravljanje robotskim manipulatorom u stvarnom vremenu. Kako bi se koristile sve mogućnosti ove biblioteke potrebo je stvoriti instancu klase *RobotCommander* iz *moveit_commander* biblioteke koja pruža sučelje za upravljanje robotskim manipulatorom, omogućavajući dohvaćanje informacija o robotu, kao što su dostupni zglobi i veze. Uz instancu klase *RobotCommander* potrebno je stvoriti i instancu klase *PlanningSceneInterface* koja omogućava integraciju s planiranim scenama, uključujući dodavanje i uklanjanje objekata iz scene te postavljanje okoliša (engl. *environment*) u kojem će se planirati pokreti robota. Četvrtom linijom koda se postavlja naziv grupe (engl. *group name*) manipulatora koji će se kontrolirati. Naziv grupe se obično odnosi na skup zglobova i veza koje čine manipulator. Nakon postavljanja naziva grupe stvara se instanca *MoveGroupCommander* klase koja pruža sučelje za planiranje pokreta za određenu grupu manipulatora, u ovom slučaju, za grupu nazvanu "manipulator". To omogućava postavljanje ciljane pozicije i orijentacije te planiranje pokreta za manipulator. Slijedeća linija stvara ROS izdavača (engl. *publisher*) koji će se koristiti za slanje komandi hvataljci. U posljednjoj liniji se stvara instanca klase IK koja se koristi za izračun inverznih kinematika robota. Ova klasa je dio biblioteke trac_ik_python koja pruža implementaciju rješavanja inverzne kinematike (IK) za robotske manipulatore i kinematičke lance [20]. Parametri 'base_link' i 'tool0' definiraju referentne okvire (engl. *reference frames*) koji se koriste za izračun inverznih kinematika, a 'solve_type' označava tip rješavanja inverznih kinematika, u ovom slučaju, koristeći udaljenost. Inverzna kinematika je metoda koja se koristi za određivanje položaja zglobova robota na temelju željenog položaja alata ili krajnjeg efektora s obzirom na referentni okvir. Ova metoda ima za cilj minimizirati razliku između stvarnog i željenog položaja alata na temelju udaljenosti ili greške u položaju. Udaljenost se često mjeri u

trodimenzionalnom prostoru kao euklidska udaljenost između stvarnog i ciljanog položaja alata. Postavljanje pozicije robota je definirano u metodi *setPosition* čiji je kod prikazan u kodu 5.15. Ovoj metodi se zadaju željena pozicija i orijentacija alata na temelju kojih se uz

```
1:      def setPosition(self, x, y, z, qx, qy, qz, qw):
2:           current_joints = self.move_group.get_current_joint_values()
             goal_joints = self.ik.get_ik(current_joints, x, y, z, qx, qy,
3:      qz, qw)
4:           self.move_group.go(goal_joints, wait=True)
5:           self.move_group.stop()
```

Kod 5.15 Metoda setPosition()

pomoć *get_ik()* metode dobiju željene vrijednosti zglobova. Ova metoda obavlja inverznu kinematiku i računa vrijednosti zglobova koje će postaviti alat na željenu poziciju. Trenutne pozicije svih zglobova se dobivaju pomoću *get_current_joint_values()* metode. Poziva se metoda *go()* kojoj se predaje željena pozicija kako bi se robot pomaknuo prema željenim pozicijama zglobova. Ova funkcija poziva planiranje kretanja i izvršava ga. Na kraju, poziva se metoda *stop()* kako bi se zaustavilo kretanje robota. Za otvaranje i zatvaranje hvataljke robota su definirane metode *openGripper()* i *closeGripper()* čiji je kod vidljiv u kodu 5.16. U obe metode je potrebno stvoriti instancu *Robotiq3FGripperRobotOutput* klase koja se koristi za kontrolu ili postavljanje parametara za hvataljku korištenu u ovome diplomskom radu. Za instancu ove klase postavljaju se sljedeći parametri:

- rACT (engl. *Action Request*) - Ovaj parametar označava je li akcija hvatanja aktivna ili ne. Ako je postavljen na 1, tada je akcija hvatanja aktivna, a ako je postavljen na 0, tada nije.
- rGTO (engl. *Go To*) - Ovaj parametar označava kreće li se hvataljka prema ciljanim pozicijama ili ne. Ako je postavljen na 1, tada se hvataljka kreće prema cilju, a ako je postavljen na 0 se ne kreće.
- rSPA (engl. *Speed (finger A)*) - Ovaj parametar određuje brzinu otvaranja ili zatvaranja hvataljke. Vrijednost može biti postavljena u rasponu od 0 do 255, gdje veća vrijednost označava veću brzinu.
- rFRA (engl. *Force (finger A)*) - Ovaj parametar kontrolira silu kojom hvataljka drži predmet. Vrijednost se može postaviti u rasponu od 0 do 255, gdje veća vrijednost označava veću silu.

- rATR (engl. *Automatic Release*) - Ovaj parametar označava automatsko otpuštanje predmeta. Ako je postavljen na 1, tada će hvataljka automatski otpustiti predmet nakon hvatanja, a ako je postavljen na 0, tada neće automatski otpustiti.
- rMOD (engl. *Gripper Mode*) - Ovaj parametar određuje način rada hvataljke. Vrijednost može biti postavljena u rasponu od 0 do 3, ovisno o željenom načinu rada.

```
1:      def openGripper(self):
2:          command = Robotiq3FGripperRobotOutput()
3:          command.rACT = 1
4:          command.rGTO = 1
5:          command.rSPA = 255
6:          command.rFRA = 150
7:          command.rATR = 0
8:          command.rMOD = 1
9:          command.rPRA = 0
10:         start_time = time()
11:         while True:
12:             self.pub.publish(command)
13:             rospy.sleep(0.1)
14:             end_time = time()
15:             if float(end_time - start_time) >= 3.0:
16:                 Break
17:     def closeGripper(self):
18:         command = Robotiq3FGripperRobotOutput()
19:         command.rACT = 1
20:         command.rGTO = 1
21:         command.rSPA = 255
22:         command.rFRA = 50
23:         command.rATR = 0
24:         command.rMOD = 1
25:         command.rPRA = 100
26:         start_time = time()
27:         while True:
28:             self.pub.publish(command)
29:             rospy.sleep(0.1)
30:             end_time = time()
31:             if float(end_time - start_time) >= 3.0:
32:                 Break
```

Kod 5.16 Metode openGripper() i closeGripper()

- rPRA (engl. *Position Request (Finger A)*) - Ovaj parametar definira željenu poziciju prsta A hvataljke. Vrijednost ovog parametra može varirati od 0 do 255, gdje svaka vrijednost predstavlja određenu poziciju za prst A. Pozicija prsta A može se kontrolirati ovim

parametrom kako bi se precizno postavila otvorenost ili zatvorenost hvataljke. Postavljanjem ovog parametra na nižu vrijednost može se zatvoriti prst A i stisnuti predmet koji se drži hvataljkom, dok postavljanje na višu vrijednost otvara prst A i oslobađa predmet [21].

U svakoj se metodi nakon postavljanja parametara koristi petlja za kontinuirano objavljivanje naredbe za zatvaranje odnosno otvaranje, s čekanjem od 3 sekunde kako bi se osiguralo da se hvataljka zatvori odnosno otvori tijekom tog vremena. Kako se dobiveni centar objekta nalazi u koordinatnom sustavu kamere najprije je bilo potrebno transformirati tu točku u koordinatni sustav baze robotskog manipulatora pomoću transformacijske matrice. Ova transformacijska matrica je dobivena kalibracijom kamere koja je izvedena prije provedbe pokusa. Transformacija točke iz jednog koordinatnog sustava u drugi se može vidjeti u kodu 5.17. Ulazna točka koju želimo transformirati je predstavljena kao trodimenzionalna točka sa svojim koordinatama x, y i z. Ona je proširena dodavanjem četvrte koordinate, 1, kako bi se omogućila homogena transformacija. Kako bi se izračunala nova koordinata točke koristi se operator @, što je operator matričnog množenja. Za rezultat matričnog množenja dobije se matrica s jednim retkom čiji su elementi x, y i z koordinate transformirane točke. Svaki dio pokusa je opisan u zasebnoj skripti. Sveukupno postoje tri Python skripte kojima se provodi detekcija, obrađuje rezultat detekcije te provodi manipulacija transparentnim objektom. Detekcija se provodi pomoću modificirane skripte *live_demo.py*, obrada dobivenog oblaka točaka i računanje centra objekta je smještena u skriptu *get_center_and_dimensions.py* dok je manipulacija objektom smještena u skriptu *move_robot.py*.

```
1:      def transform(point):
2:          transformation_matrix = np.matrix([[ 0.011134993096219537, -
3:          0.4265458347569073,  0.8800191123863065, -0.9008058369236389  ],
4:          [-0.9770448284683437,    0.03288445435089571,
5:          0.028301788109787576,  0.6169276026386865  ],
6:          [-0.04193316678852926,  -0.8794749791202019,  -
7:          0.4257515073484868,    0.5162731031956954  ],
8:          [ 0.0,      0.0,        0.0,         1.      ]])
9:          point = np.append(point, 1)
10:         transformed = transformation_matrix@point
11:         xT = transformed[0, 0]
12:         yT = transformed[0, 1]
13:         zT = transformed[0, 2]
14:         return xT, yT, zT
```

Kod 5.17 Transformacija centra objekta iz koordinatnog sustava kamere u koordinatni sustav baze robota

Svaki od rezultata jedne skripte je bio korišten od strane druge zbog čega je bilo potrebno te

rezultate pohraniti kako bi ih druga skripta mogla učitati i koristiti. Primjer čitanja ovih rezultata može se vidjeti u kodu 5.18 gdje se može vidjeti spremanje koordinata centra objekta u tekstualni

```
1:      center = get_object_center(path)
2:      result_folder_path = path + "result-center-and-dimensions/"
3:      os.makedirs(result_folder_path)
4:
5:      center_file = open(result_folder_path + "center.txt", "w")
6:      center_file.write(str(center))
```

Kod 5.18 Pohrana koordinata centra objekta u tekstualni dokument center.txt

dokument *center.txt* kao i u kodu 5.13 gdje se može vidjeti na koji način je učitan oblak točaka

```
1:      int32 my_phase
2:      bool isFinished
3:      ---
4:      bool isMyTurn
```

Kod 5.19 CheckCurrentPhase.srv

koji je rezultat detekcije. Da bi pokus tekao sekvencijalno bilo je potrebno implementirati servis (engl. *service*) koji će baratati redoslijedom izvedbe ovih programa. ROS servis je komunikacijski mehanizam koji omogućuje ROS čvorovima da šalju zahtjeve i primaju odgovore od drugih

```
1:  #!/usr/bin/env python3
2:  import rospy
3:  from cleargrasp.srv import CheckCurrentPhase, CheckCurrentPhaseResponse
4:  current_phase = 0
5:  def handle_current_phase(req):
6:      global current_phase
7:      isMyTurn = False
8:      if req.my_phase == current_phase and req.isFinished is not True:
9:          isMyTurn = True
10:         print("current phase"+str(current_phase))
11:     elif req.my_phase == current_phase and req.isFinished == True:
12:         isMyTurn = False
13:         current_phase += 1
14:     return CheckCurrentPhaseResponse(isMyTurn)
15: def check_current_phase_server():
16:     rospy.init_node('check_current_phase_server')
17:     s = rospy.Service('check_current_phase', CheckCurrentPhase,
18:         handle_current_phase)
19:     rospy.spin()
20: if __name__ == "__main__":
21:     check_current_phase_server()
```

Kod 5.20 handle_experiment.py

čvorova unutar ROS ekosustava. Na ovaj način je osigurano „čekanje“ na obradu svakog

pojedinog programa. Ovo je implementirano uvođenjem zastavica *my_phase*, *isFinished* i *isMyTurn* koje predstavljaju trenutnu fazu, kraj programa te govore je li trenutni program na redu za rad. Za implementaciju servisa bilo je potrebno stvoriti poruku koja se šalje servisu kako bi znao reći koji je program na redu. Izgled ove poruke se može vidjeti u kodu 5.19 dok se u kodu 5.20 može vidjeti implementacija samoga servisa odnosno kako on barata poslanim porukama. Kao što se može vidjeti iz priloženoga koda servis kreće od faze 0 nakon koje čeka njen završetak kako bi povećao vrijednost brojača *current_phase* koji označava trenutnu fazu. U svakoj skripti je implementiran dio čekanja na izvedbu kao što se može vidjeti u kodu 5.21. Na ovaj način je osigurano nesmetano izvođenje pokusa pokretanjem samo jedne datoteke za

```
1:   def callback(self, rgb_msg, depth_msg, camera_info_msg):
2:       global isFinished
3:       global isMyTurn
4:       rospy.wait_for_service('check_current_phase')
5:       current_phase = rospy.ServiceProxy('check_current_phase',
6:                       CheckCurrentPhase)
7:       isMyTurn = (current_phase(0, isFinished)).isMyTurn
8:       if(isMyTurn == True and isFinished == False):
9:       isMyTurn = False
10:          print("Usao u main")
11:          self.main(rgb_msg, depth_msg, camera_info_msg)
```

Kod 5.21 Primjer implementacije čekanja programa na njihov red unutar skripte live_demo.py

pokretanje pod nazivom *detect.launch*. Pomoću ove datoteke pokreću se sve tri Python skripte u obliku ROS čvorova kao i potrebni čvorovi za pokretanje kamere te robotskoga manipulatora. Sadržaj ove datoteke se može vidjeti u prilogu P.5.2. Za pokretanje "launch" datoteka u ROS-u koristi se *roslaunch* naredba u terminalu. Ova naredba omogućava ROS-u da interpretira i izvrši definicije koje su navedene u "launch"

```
1:   roslaunch cleargrasp detect.launch
```

Kod 5.22 Primjer pokretanja eksperimenta pomoću naredbe roslaunch

datoteci. Primjer korištenja ove naredbe može se vidjeti u kodu 5.22. Ovaj primjer predstavlja način pokretanja svih programa pomoću kojih su provedeni eksperimenti.

5.7. Evaluacija rezultata

Ukupno je provedeno petnaest pokusa u kojima su za objekte detekcije korištene transparentne plastične boce i čaše koje se mogu vidjeti na slici 5.9. Trajanje svakoga pokusa je prosječno iznosilo 8 minuta. Na trajanje pokusa je najviše utjecalo potrebno vrijeme obrade slike od strane ClearGrasp mreže, odstranjivanje stršećih vrijednosti te brzina kretanja robotskog manipulatora koja je bila postavljena na 20% iz sigurnosnih razloga. U obzir su se uzela dva položaja za svaki

Tablica 5.1 Rezultati provedenih pokusa

Pokus	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*Ukupan broj pokusa*
Uspješno	Ne	Ne	Ne	Da	Da	Da	Da	Da	Da	Da	Da	Da	Da	Ne	Ne	10
Korišteni objekt	a	a	a	a	a	a	a	a	a	b	b	b	b	c	a	
Polegnuti objekt	Ne	Ne	Ne	Ne	Ne	Ne	Da	Da	Da	Ne	Ne	Da	Da	Ne	Ne	

objekt, kada je objekt uspravan kao što se može vidjeti na slici 5.9 i kada je objekt polegnut što

a) b) c)

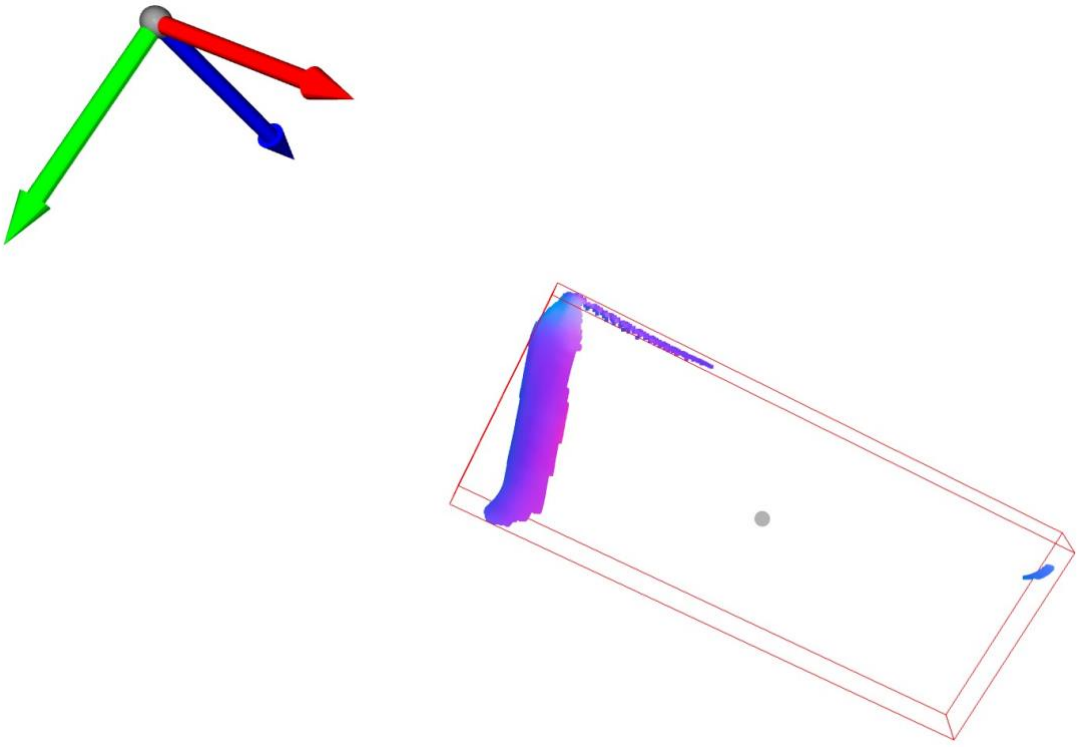
Slika 5.9 Objekti korišteni pri izvedbi pokusa

se može vidjeti na slici 5.10. Kao što se može vidjeti u tablici 5.1, od petnaest pokusa deset ih je bilo uspješnih. Pri izvedbi ostalih pokusa najveći problem je stvarala post-obrada oblaka točaka o kojoj je govoreno u prijašnjim poglavljima. Naime, ukoliko se obradom oblaka točaka objekta nisu uklonile sve „stršeće“ vrijednosti centar objekta nije mogao biti točno određen jer je orijentirani okvir objekta obuhvaćao i te „stršeće“ vrijednosti. Orijentirani okvir objekta je u tim slučajevima izgledao kao na slici 5.11 pri čemu je rezultat pokusa bilo pogrešno pozicioniranje robotske ruke

pri hvatanju objekta što se može vidjeti na slici 5.12. Do grešaka ovoga tipa došlo je u četiri pokusa.

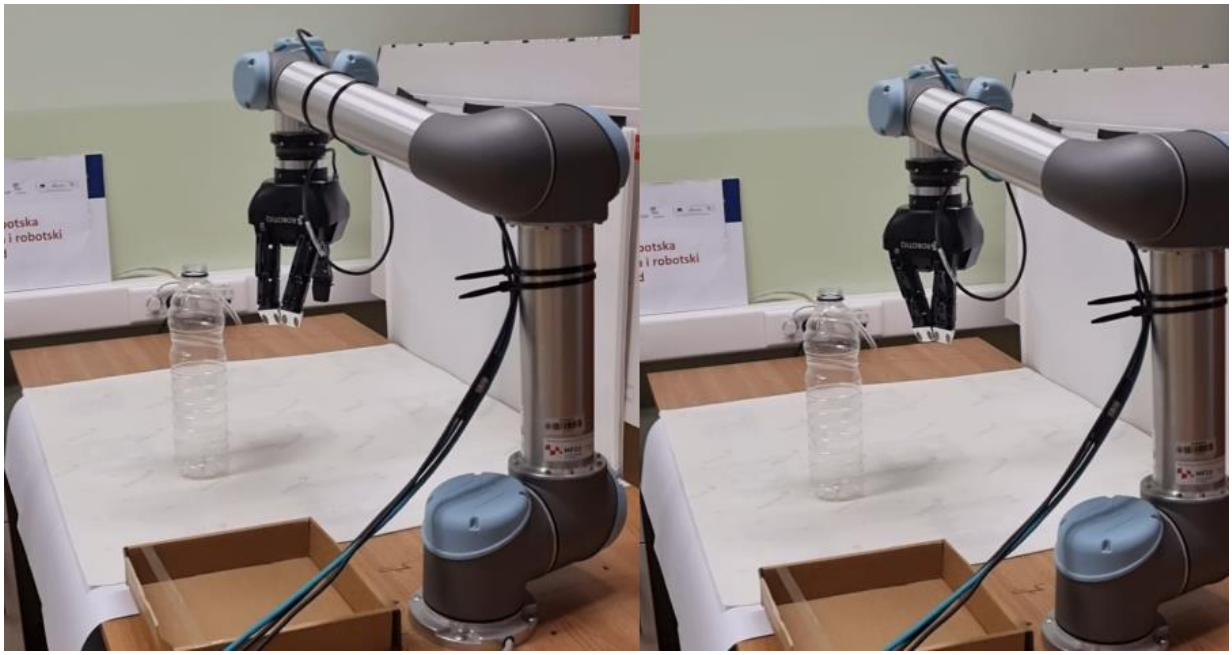
Slika 5.10 Primjer polegnutog objekta

Primijećeno je da će do ovih grešaka sigurno doći ukoliko predmet nije u potpunosti transparentan ili sadrži nešto neprozirno na sebi poput čepa u boji. Mreža će takve predmete iako se nalaze na



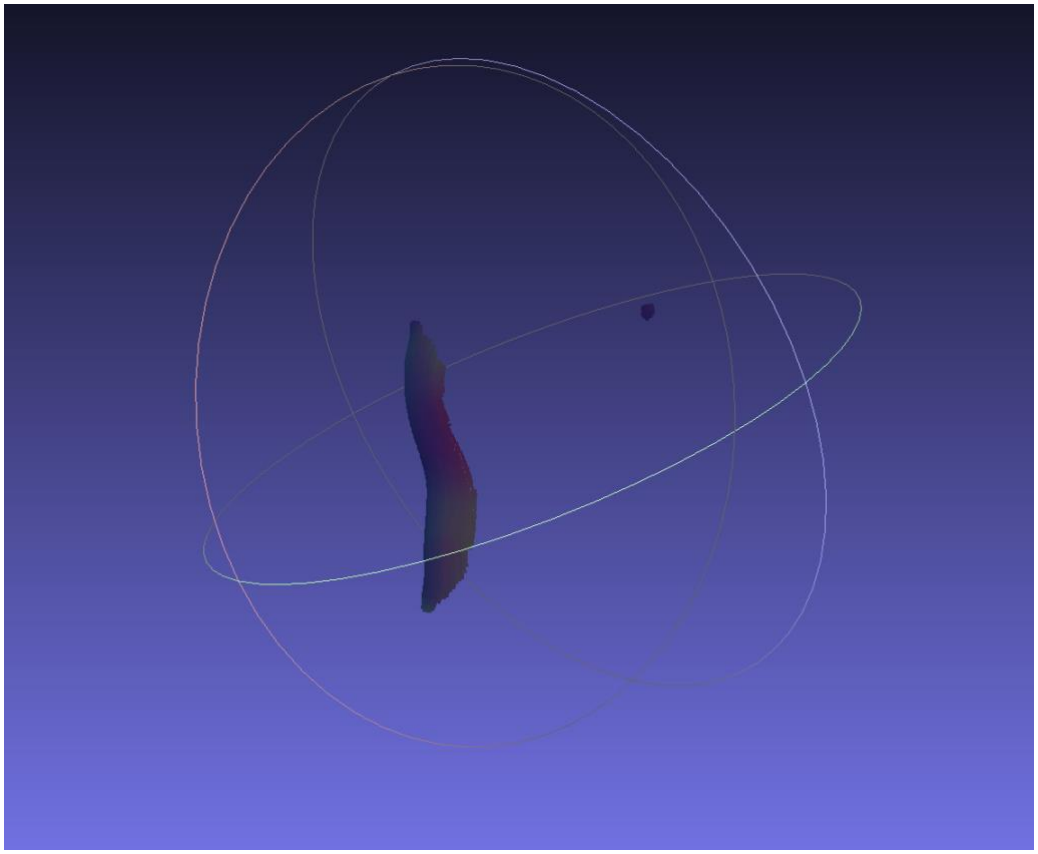
Slika 5.11 Prikaz orijentiranog okvira objekta pri lošoj obradi stršećih vrijednosti

istoj udaljenosti kao i transparentan predmet prepoznati kao da se nalazi nekoliko desetaka



Slika 5.12 Primjer pogrešnog pozicioniranja robotske ruke pri hvatanju objekta

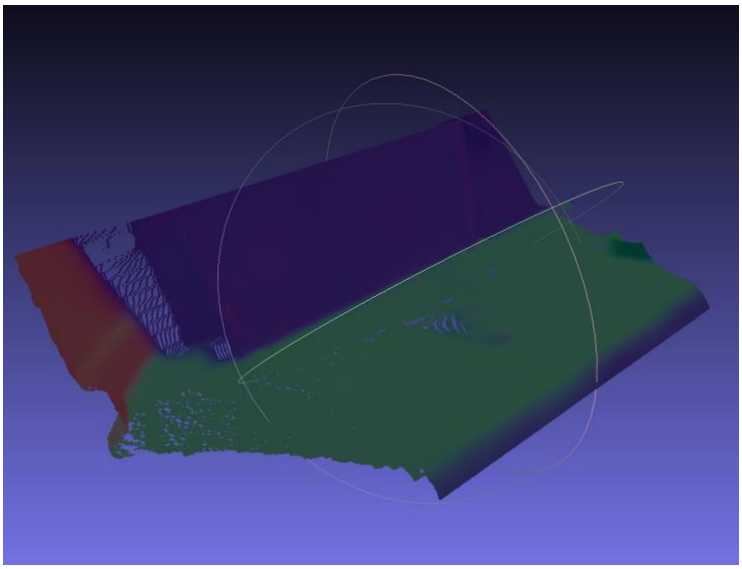
centimetara iza objekta što se može i vidjeti na slici 5.13. Kako je utvrđeno da mreža ne radi dobro



Slika 5.13 Primjer pogrešne rekonstrukcije dijela transparentnog objekta

sa neprozirnim ili obojenim poluprozirnim objektima tako nije mogla dobro prepoznati čašu sa

slike 5.9. Oblak točaka koji se dobije pri detekciji čaše se može vidjeti na slici 5.14. Ovaj oblak točaka predstavlja scenu u čijem se središtu nalazila čaša koju ClearGrasp metoda nije uspjela rekonstruirati jer se pokazalo da je čaša zapravo poluprozirna a ne transparentna.



Slika 5.14 Oblak točaka dobiven detekcijom obojene čaše

6. ZAKLJUČAK

U ovom radu je istraženo određivanje trodimenzionalnog oblika i položaja transparentnih objekata u svrhu robotske manipulacije. Primjenom metode temeljene na neuronskim mrežama, transparentni objekti su uspješno detektirani te manipulirani pomoću robotskog manipulatora.

Postignuti rezultati su pokazali kako korištena mreža uspješno prepoznaje transparentne objekte te se njeni rezultati mogu obraditi kako bi se došlo do oblika i položaja objekta čime je omogućena i njegova manipulacija. Međutim, kako bi se postigli precizniji rezultati, bilo je potrebno dodatno obraditi izlazne podatke mreže korištenjem Open3D biblioteke da bi se odredio položaj centra objekta.

U budućim istraživanjima, daljnje poboljšanje rješenja korištenjem algoritama pomoću kojih se može doći do svih dimenzija objekta, kao što je radijus baze i vrha boce, duljina ručke šalica ili drugih dimenzija transparentnih objekata, može pomoći pri manipulaciji geometrijski nepravilnim ili složenijim transparentnim objektima, što bi pridonijelo daljem napretku u području robotske manipulacije.

SAŽETAK

Ovaj rad istražuje problem određivanja trodimenzionalnog oblika i položaja transparentnih objekata za potrebe robotske manipulacije. Predstavljena je metoda ClearGrasp temeljena na neuronskim mrežama koja se koristi za detekciju transparentnih objekata. Metoda uključuje estimaciju površinskih normi, rubova i maski transparentnih objekata. U radu se korištenjem rekonstruiranog oblaka točaka dobivenog primjenom ClearGrasp metode uz pomoć metoda Open3D biblioteke dolazi do centra i oblika objekta koji daju informaciju o položaju i veličini objekta. Izračunati položaj i veličina objekta su korišteni pri manipulaciji objektom kako bi robotski manipulator znao gdje se objekt nalazi i na koji način ga uhvatiti. Eksperimentalni postav je opisan, uključujući korištene hardverske komponente i programski okvir. Nakon toga su prikazani rezultati eksperimenta, koji uključuju evaluaciju uspješnosti detekcije i manipulacije transparentnih objekata.

Ključne riječi: Robotika, Transparentni objekti, Detekcija, Robotska manipulacija, ClearGrasp

ABSTRACT

This paper explores the problem of determining the three-dimensional shape and pose of transparent objects for robotic manipulation purposes. A neural network-based method ClearGrasp is presented for the detection of transparent objects, involving the estimation of surface normals, edges, and masks. In this paper, the object's center and shape were determined by employing the reconstructed point cloud, which was generated through the application of the ClearGrasp method with the assistance of Open3D library methods. The calculated position and size of the object were then utilized in the manipulation of the object, allowing the robotic manipulator to be aware of the object's location and how to grasp it. The experimental setup is described, including the hardware components and the software framework utilized. Subsequently, the experimental results are presented, which include an evaluation of the success in detecting and manipulating transparent objects.

Keywords: Robotics, Transparent objects, Detection, Robotic manipulation, ClearGrasp

LITERATURA

[1] S. S. Sajjan, . Moore, M. Pan, G. Nagaraja, J. Lee, A. Zeng, S. Song, Synthesis.ai, Google i C. University, ClearGrasp: 3D Shape Estimation of Transparent Objects for Manipulation, IEEE, 2020.

[2] S. Astanin, D. Antonelli, P. Chiabert i C. Allett, Reflective workpiece detection and localization for flexible robotic cells, Turin, Italy: Elsevier Ltd, 2016.

[3] R. Kocha, S. Maya i A. Nuchter, DETECTION AND PURGING OF SPECULAR REFLECTIVE AND TRANSPARENT OBJECT INFLUENCES IN 3D RANGE MEASUREMENTS, ISPRS International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLII-2/W3, 2017, pp.377-384, 2017.

[4] J. Wang, Y. Zhou i Y. Yang, Three-Dimensional Shape Detection for Non Uniform Reflective Objects: Combination of Color Light Projection and Camera's Exposure Adjustment, IEEE, 2020.

[5] Y. Xu, H. Nagahara i A. Shimada, TransCut: Transparent Object Segmentation from a Light-Field Image, arXiv, 2015.

[6] H. Xu, Y. R. Wang, S. Eppel, A. Aspuru-Guzik, F. Shkurti i A. Garg, Seeing Glass: Joint Point Cloud and Depth Completion for Transparent Objects, arXiv, 2021.

[7] Universal Robots A/S., »Universal Robots,« [Mrežno]. Available: https://myurhelpresources.blob.core.windows.net/resources/PDF/SW_5_13/UR5e_User_Manual_en_Global.pdf.

[8] R. I. Maldonado-Valencia, C. H. Rodriguez-Garavito, C. A. Cruz-Perez, J. S. Hernandez-Navas i D. I. Zabala-Benavides, Planning and visual-servoing for robotic manipulators in ROS, International Journal of Intelligent Robotics and Applications volume 6, pages602–614, 2022.

[9] Open Robotics, »ROS,« [Mrežno]. Available: http://wiki.ros.org/noetic.

[10] M. W. Khan, A Survey: Image Segmentation Techniques, International Journal of Future Computer and Communication, 2014.

[11] K. H. Dr D SRINIVAS, Analysis of Various Image Feature Extraction Methods against Noisy Image: SIFT, SURF and HOG, Journal of Engineering Sciences , 2019.

[12] J. Jiang, G. Cao, J. Deng, T.-T. Do i S. Luo, Robotic Perception of Transparent Objects: A Review, IEEE, 2023.

[13] F. Dajun, G. Long i L. Zhifeng, Review of Refractive Index-Matching Techniques of Polymethyl Methacrylate in Flow Field Visualization Experiments, International Journal of Energy Research, 2023.

[14] Y. Zhang i T. Funkhouser, »Deep Depth Completion of a Single RGB-D Image,« 2018.

[15] R. Cupec, »EYE-IN-HAND CAMERA CALIBRATION,« Osijek, 2022.

[16] Intel Corporation, »Intel RealSense,« [Mrežno]. Available: https://www.intelrealsense.com/lidar-camera-l515/.

[17] Universal Robots A/S, »Robotiq 3-Finger Adaptive Robot Gripper,« [Mrežno]. Available: https://www.universal-robots.com/plus/products/robotiq/robotiq-3-finger-adaptive-robot-gripper/.

[18] P. S. Foundation, »PyPI,« [Mrežno]. Available: https://pypi.org/project/opencv-python/.

[19] Intelligent Systems Lab at the University of Washingto, »Open3D,« [Mrežno]. Available: http://www.open3d.org/docs/release/index.html.

[20] Open Robotics, »ROS.org,« [Mrežno]. Available: http://wiki.ros.org/trac_ik_python.

[21] Robotiq, »3-Finger Adaptive Robot Gripper Instruction Manual,« [Mrežno]. Available: https://assets.robotiq.com/website-assets/support_documents/document/3-Finger_PDF_20190221.pdf.

ŽIVOTOPIS

Vlatka Mihić rođena je 25. ožujka 2000. godine u Osijeku. Živi u Osijeku gdje je završila Osnovnu školu Ivana Filipovića. Poslije osnovne škole upisuje srednju školu, III. Gimnaziju u Osijeku.

Nakon završetka srednje škole upisuje sveučilišni studij Računarstva na fakultetu Elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. 2021. godine završava preddiplomski sveučilišni studij računarstva te upisuje diplomski studij računarstva, izborni blok Robotika i Umjetna Inteligencija. Za vrijeme studiranja sudjeluje u izvannastavnim aktivnostima kao što su IEEExtreme natjecanje, kolegij Napredni programeri, na trećoj godini zauzima poziciju potpredsjednice IEEE studentskog ogranka u Osijeku te od prve godine diplomskog studija postaje član studentskog zbora fakulteta i studentska pravobraniteljica.

Nakon gotovo dvije godine rada od siječnja 2021. godine do listopada 2022. godine kao tester osiguranja kvalitete u digitalnoj agenciji COBE u Osijeku daje otkaz kako bi pronašla posao u domeni izabranog izbornog bloka. U studenom 2022. godine počinje raditi kao inženjer razvoja softverskih i hardverskih rješenja u tehnološkoj tvrtki ORQA u Osijeku.

_____

Potpis autora

PRILOZI

live_demo.py

```
import os
import glob
import os
import shutil
import sys

import cv2
import numpy as np
import termcolor
import yaml
from attrdict import AttrDict
from PIL import Image
from api import utils, depth_completion_api
import rospy
import message_filters
from sensor_msgs.msg import Image, CameraInfo
from cv_bridge import CvBridge

bridge = CvBridge()

class ClearGraspNode:
    def __init__(self):
        # Load Config File
        # CONFIG_FILE_PATH = args.configFile
        CONFIG_FILE_PATH = '/home/robot/cleargrasp/live_demo/config/config.yaml'
        with open(CONFIG_FILE_PATH) as fd:
            config_yaml = yaml.safe_load(fd)
        self.config = AttrDict(config_yaml)


        self.captures_dir = '/home/robot/cleargrasp/data/captures/exp-000'
        rospy.init_node('live_demo_node')
        rgb_image_sub = message_filters.Subscriber('/camera/color/image_raw', Image)
        depth_image_sub = message_filters.Subscriber('/camera/aligned_depth_to_color/image_raw', Image)
        info_sub = message_filters.Subscriber('/camera/color/camera_info', CameraInfo)


        # Initialize Camera
        print('Running live demo of depth completion. Make sure realsense camera is streaming.\n')


        # # Create directory to save captures
        runs = sorted(glob.glob(os.path.join(self.config.captures_dir, 'exp-*')))
        prev_run_id = int(runs[-1].split('-')[-1]) if runs else 0
        self.captures_dir = os.path.join(self.config.captures_dir, 'exp-{:03d}'.format(prev_run_id))
        if os.path.isdir(self.captures_dir):
            if len(os.listdir(self.captures_dir)) > 5:
                # Min 1 file always in folder: copy of config file
                self.captures_dir = os.path.join(self.config.captures_dir, 'exp-{:03d}'.format(prev_run_id + 1))
                os.makedirs(self.captures_dir)
        else:
            os.makedirs(self.captures_dir)

        # # Save a copy of config file in the logs
        shutil.copy(CONFIG_FILE_PATH, os.path.join(self.captures_dir, 'config.yaml'))

        print('Saving captured images to folder: ' + termcolor.colored('"{}"'.format(self.captures_dir), 'blue'))
        print('\n Press "c" to capture and save image, press "q" to quit\n')

        # Initialize Depth Completion API
        outputImgHeight = int(self.config.depth2depth.yres)
        outputImgWidth = int(self.config.depth2depth.xres)
        self.depthcomplete = depth_completion_api.DepthToDepthCompletion(normalsWeightsFile=self.config.normals.pathWeightsFile,
                                                                         outlinesWeightsFile=self.config.outlines.pathWeightsFile,
                                                                         masksWeightsFile=self.config.masks.pathWeightsFile,
                                                                         normalsModel=self.config.normals.model,
                                                                         outlinesModel=self.config.outlines.model,
                                                                         masksModel=self.config.masks.model,
                                                                         depth2depthExecutable=self.config.depth2depth.pathExecutable,
                                                                         outputImgHeight=outputImgHeight,
                                                                         outputImgWidth=outputImgWidth,
                                                                         fx=int(self.config.depth2depth.fx),
                                                                         fy=int(self.config.depth2depth.fy),
                                                                         cx=int(self.config.depth2depth.cx),
                                                                         cy=int(self.config.depth2depth.cy),
                                                                         filter_d=self.config.outputDepthFilter.d,
                                                                         filter_sigmaColor=self.config.outputDepthFilter.sigmaColor,
                                                                         filter_sigmaSpace=self.config.outputDepthFilter.sigmaSpace,
                                                                         maskinferenceHeight=self.config.masks.inferenceHeight,
                                                                         maskinferenceWidth=self.config.masks.inferenceWidth,
                                                                         normalsInferenceHeight=self.config.normals.inferenceHeight,
                                                                         normalsInferenceWidth=self.config.normals.inferenceWidth,
                                                                         outlinesInferenceHeight=self.config.normals.inferenceHeight,
                                                                         outlinesInferenceWidth=self.config.normals.inferenceWidth,
```

```
                                                    min_depth=self.config.depthVisualization.minDepth,
                                                    max_depth=self.config.depthVisualization.maxDepth,
                                                    tmp_dir=self.captures_dir)
        self.capture_num = 0


        ts = message_filters.TimeSynchronizer([rgb_image_sub, depth_image_sub, info_sub], 10)
        ts.registerCallback(self.callback)
        cv2.destroyAllWindows()
        rospy.spin()

    def callback(self, rgb_msg, depth_msg, camera_info_msg):
        self.main(rgb_msg, depth_msg, camera_info_msg)

    def main(self, rgb_msg, depth_msg, camera_info_msg):
        rgb_image = bridge.imgmsg_to_cv2(rgb_msg, desired_encoding='passthrough') # desired_encoding='passthrough')
        input_depth = bridge.imgmsg_to_cv2(depth_msg, desired_encoding='passthrough').astype(np.float32)

        input_depth /= 1000.0
        color_img = rgb_image
        print("Trebam prikazati sliku")
        cv2.imshow('Live Demo', rgb_image)

        keypress = cv2.waitKey(10) & 0xFF
        print("HELLO")
        if keypress == ord('q'):
            pass
        elif keypress == ord('c'):
            print("I am here!")

            try:
                output_depth, filtered_output_depth = self.depthcomplete.depth_completion(
                    color_img,
                    input_depth,
                    inertia_weight=float(self.config.depth2depth.inertia_weight),
                    smoothness_weight=float(self.config.depth2depth.smoothness_weight),
                    tangent_weight=float(self.config.depth2depth.tangent_weight),
                    mode_modify_input_depth=self.config.modifyInputDepth.mode)
            except depth_completion_api.DepthCompletionError as e:
                print('Depth Completion Failed:\n  {}\n  ...skipping image {}'.format(e, i))
                pass


            color_img = self.depthcomplete.input_image
            input_depth = self.depthcomplete.input_depth
            surface_normals = self.depthcomplete.surface_normals
            surface_normals_rgb = self.depthcomplete.surface_normals_rgb
            occlusion_weight = self.depthcomplete.occlusion_weight
            occlusion_weight_rgb = self.depthcomplete.occlusion_weight_rgb
            outlines_rgb = self.depthcomplete.outlines_rgb
            mask = self.depthcomplete.mask_predicted
            mask_rgb = cv2.cvtColor(mask,cv2.COLOR_GRAY2RGB)
            # Display Results in Window
            input_depth_mapped = utils.depth2rgb(input_depth, min_depth=self.config.depthVisualization.minDepth,
                                                 max_depth=self.config.depthVisualization.maxDepth,
                                                 color_mode=cv2.COLORMAP_JET, reverse_scale=True)
            output_depth_mapped = utils.depth2rgb(output_depth, min_depth=self.config.depthVisualization.minDepth,
                                                  max_depth=self.config.depthVisualization.maxDepth,
                                                  color_mode=cv2.COLORMAP_JET, reverse_scale=True)
            filtered_output_depth_mapped = utils.depth2rgb(filtered_output_depth,
                                                           min_depth=self.config.depthVisualization.minDepth,
                                                           max_depth=self.config.depthVisualization.maxDepth,
                                                           color_mode=cv2.COLORMAP_JET, reverse_scale=True)
            color_img = cv2.cvtColor(color_img, cv2.COLOR_RGB2BGR)
            surface_normals_rgb = cv2.cvtColor(surface_normals_rgb, cv2.COLOR_RGB2BGR)
            outlines_rgb = cv2.cvtColor(outlines_rgb, cv2.COLOR_RGB2BGR)
            occlusion_weight_rgb = cv2.cvtColor(occlusion_weight_rgb, cv2.COLOR_RGB2BGR)

            grid_image1 = np.concatenate((color_img, surface_normals_rgb, outlines_rgb, occlusion_weight_rgb), 1)
            grid_image2 = np.concatenate((input_depth_mapped, output_depth_mapped, filtered_output_depth_mapped,
                                          mask_rgb), 1)
            grid_image = np.concatenate((grid_image1, grid_image2), 0)

            # Saving grid
            # cv2.imwrite(os.path.join(path , 'result_grid.jpg'), grid_image)

            cv2.imshow('Live Demo Capture', grid_image)
            cv2.waitKey(0)
            # Save captured data to config.captures_dir
            self.depthcomplete.store_depth_completion_outputs(self.captures_dir,
                                                              self.capture_num,
                                                              min_depth=self.config.depthVisualization.minDepth,
                                                              max_depth=self.config.depthVisualization.maxDepth)
            print('captured image {0:06d}'.format(self.capture_num))
            self.capture_num += 1

if __name__ == '__main__':
    cgn = ClearGraspNode()
```

P.5.1. Izmijenjeni kod – live_demo.py

```
1:  <launch>
2:    <include file="$(find realsense2_camera)/launch/rs_camera.launch">
3:       <arg name="enable_pointcloud" value="true" />
4:       <arg name="depth_width" value="640" />
5:       <arg name="depth_height" value="480" />
6:       <arg name="depth_fps" value="15" />
7:       <arg name="color_width" value="640" />
8:       <arg name="color_height" value="480" />
9:       <arg name="color_fps" value="15" />
10:      <arg name="align_depth" value="true" />
11:   </include>
12:   <include file="$(find ur_robot_driver)/launch/ur5_bringup.launch">
13:     <arg name="robot_ip" value="192.168.22.14"/>
14:     <arg name="kinematics_config" value="/home/$(env
15:      USER)/my_robot_calibration.yaml"/>
16:   </include>
17:   <include file="$(find ur5_moveit_config)
18:    /launch/ur5_moveit_planning_execution.launch"/>
19:   <include file="$(find ur5_moveit_config)/launch/moveit_rviz.launch"/>
20:   <node name="gripper_control" pkg="robotiq_3f_gripper_control"
21:    type="Robotiq3FGripperTcpNode.py" args="192.168.22.11"/>
22:   <node pkg="cleargrasp" name="check_current_phase_server"
23:    type="handle_experiment.py" output="screen"/>
24:   <node pkg="cleargrasp" name="live_demo_node" type="live_demo.py"
25:    output="screen"/>
26:   <node pkg="cleargrasp" name="get_center_and_dimensions"
27:    type="get_center_and_dimensions.py" output="screen"/>
28:   <node pkg="labus" name="move_robot" type="move_robot.py"
29:    output="screen"/>
30: </launch>
```

P.5.2. Primjer implementacije čekanja programa na njihov red unutar skripte live_demo.py