Robotska manipulacija korištenjem računalnog vida i neuronske mreže Dex-Net

Stipić, Nikola

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

Sveučilišni studij

Robotska manipulacija korištenjem računalnog vida i neuronske mreže Dex-Net

Diplomski rad

Nikola Stipić

Osijek, 2022.

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 04.10.2022.

Ime i prezime studenta:	Nikola Stipić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1163R, 13.10.2020.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Robotska manipulacija korištenjem računalnog vida i neuronske mreže Dex-Net**

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

i sumentora Dr. sc. Petra Pejić

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

Potpis studenta:

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 17.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Nikola Stipić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1163R, 13.10.2020.
OIB studenta:	35233169948
Mentor:	Prof.dr.sc. Robert Cupec
Sumentor:	Dr. sc. Petra Pejić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Damir Filko
Član Povjerenstva 1:	Prof.dr.sc. Robert Cupec
Član Povjerenstva 2:	Doc.dr.sc. Ivan Vidović
Naslov diplomskog rada:	Robotska manipulacija korištenjem računalnog vida i neuronske mreže Dex-Net
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Eksperimentalno ispitati mogućnosti Dex-Net neuronske mreže za primjenu u robotskoj manipulaciji pomoću računalnog vida. Tema rezervirana za: Nikola Stipić Sumentor s FERIT-a: Petra Pejić
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	17.09.2022.

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

SADRŽAJ

1. UVOD

Veliki iskorak umjetne inteligencije u relativno kratkom vremenu omogućio je primjenu istog u realnom sektoru pa tako i u širem području robotike. Robotika je interdisciplinarna grana računalne znanosti i inženjerstva koja uz umjetnu inteligenciju predstavlja jedan od glavnih stupova Industrije 4.0. Od brojnih izazova s kojima se susrećemo u robotici kao što su navigacija, upravljanje, obrada ulaznih podataka, i drugi, u ovom radu obrađivati će se problem manipulacije.

Iako se robotska manipulacija može sastojati od jednostavne interakcije s objektom u okruženju manipulatora, težnja za širom primjenom i što većom autonomnošću dovela je do znatno složenijeg postupka. U ovom radu manipulacija će se sastojati od prihvaćanja i obrade podataka s RGB-D kamere u svrhu pronalaska objekta od interesa i njegovog hvatišta, te podizanja i postavljanja (*engl. pick and place)* istog pomoću paralelne hvataljke.

Računalni vid uz umjetnu inteligenciju omogućava adaptivniju primjenu manipulacije s objektima koji nisu uvijek isto postavljeni (*engl. binpicking*, razvrstavanje hrpe objekata i sl.), ali je to vrlo složen proces koji ima puno prostora za daljnji razvitak. Jedan od takvih izazova je hvatanje objekata koji nisu prethodno poznati sustavu. Temeljna ideja istraživačkog projekta Dex-Net je da na osnovu dubinske slike objekta pomoću prethodno naučene neuronske mreže odredi najbolje hvatište tog objekta bez prethodne klasifikacije istog. [1]

Zadatak ovog diplomskog rada je eksperimentalno opisati mogućnosti primjene DexNet-a u stvarnom okruženju. Budući da je za rješenje problema potrebno objediniti više komponenata sa svojim sklopovljem i softverom potrebno je koristiti središnje računalo koje će uz to, također vršiti komunikaciju čovjek-robot. Kako bi se pojednostavila komunikacija računala i komponenata, obrada podataka te ljudsko upravljanje cijelim procesom, poželjno je koristiti softversko rješenje kao što je ROS(*engl. Robot Operating System)* koje služi kao poveznica istih.

U sljedećem poglavlju obradit će se trenutni pregled područja tog problema te obilježja ostalih pristupa. Zatim, u 3. poglavlju, detaljno će biti opisan princip rada Dex-Net te njegova softverska integracija u ROS-u uz analizu samog ROS-a. Sve komponente sustava, uz obradu njihovog sklopovlja i ROS implementacije, će biti opisane u 4. poglavlju. Provedeni pokusi te rezultati i zapažanja istih će biti opisani u 5. i 6. poglavlju. Zaključak i osvrt rada je dan u posljednjem, 7., poglavlju.

2. PREGLED PODRUČJA

Najzastupljenije metode pristupa problemu robotskog hvatanja objekata se mogu podijeliti na dvije skupine, analitičku i empirijsku.

Ideja analitičkih postupaka je na osnovu već poznatog mjesta objekta i pozicije hvatišta na samom objektu odrediti kvalitetu hvatišta. Problem predstavlja na temelju čega odrediti kvalitetu hvatišta te kako to uspješno upotrijebiti na stvarnom robotu. GraspIt! [2], alat namijenjen za istraživanja u robotskom hvatanju, metodu za izračun kvalitete hvatišta temelji na skupu podataka 3D objekata koji sadrže hvatišta te odgovarajuću kvalitetu tog hvatišta. Primjena ovakvog pristupa u stvarnom okruženju temelji se na podudaranju objekta od interesa s objektom iz skupa podatka u obliku oblaka točaka te odabiru hvatišta s najvećom kvalitetom. Uvođenjem mjere robusnosti, tj. očekivane vrijednosti kvalitete hvatišta pod utjecajem nesigurnosti koju nose senzori i upravljanje robotskim manipulatorom, otvorilo se područje za daljnja istraživanja, a upravo se na ovakvom istraživanju temelji ovaj rad. Budući da je tema ovog rada neuronska mreža Dex-Net 2.0 [3], koja predstavlja proširenje prethodne metode Dex-Net 1.0 [4], dotaknut ćemo se izvorne metode. Ideja Dex-Net 1.0 je da skup podataka za svako hvatište sadrži procijenjenu vjerojatnost prisilnog zatvaranja šake uzimajući u obzir nepouzdanost položaja i trenja u odnosu objekta i šake. Vjerojatnost prisilnog zatvaranje šake je mjerilo kvalitete hvatišta koje je u radu Dex-Net 2.0 zamijenjeno s robusnošću hvatišta Q_θ te je detaljnije opisano u potpoglavlju *3.1.2.*

Empirijski postupci, za razliku od analitičkih, koriste strojno učenje koje bi pomoću nadgledanog učenja rezultiralo zadovoljavajućim modelom. Ovakav model se uči na osnovu podataka za učenje koje su ili ljudi označili kao uspješan, tj. neuspješan, ili na temelju stvarnih pokusa hvatanja. Kao i kod analitičkih postupaka veliki problem predstavlja stvaranje dovoljno velikog i primjenjivog skupa podataka. Većina istraživanja u ovom području se temelji na korištenju konvolucijskih neuronskih mreža kao što je „*Real-Time Grasp Detection Using Convolutional Neural Networks"*. [5]

Iako ovakve metode omogućavaju i izvedbu u stvarnom vremenu, ograničene su za stvarnu primjenu jer se objekti na sceni moraju klasificirati prije određivanja zadovoljavajućeg hvatišta, tj. njihova upotrebljivost na objektima koji se ne nalaze u skupu podataka za učenje se znatno smanjuje. Ovo ograničenje dovelo je do istraživanja koja se ne oslanjaju na prepoznavanje objekata nego na oblik i pozu objekta, kao što je rad „*Robotic Grasping of Novel Objects using Vision"* iz 2008. godine učen na sintetičkom skupu podataka. [6]

Tema ovog diplomskog rada sadrži kombinaciju analitičkih i empirijskih metoda koje će detaljno biti opisane u sljedećem poglavlju.

3. DEX-NET

3.1. Dex-Net

Dex-Net projekt opisan je u nizu znanstvenih radova koji se sastoje od skupa podataka i metoda za određivanje hvatišta objekata. Ovaj diplomski rad bavit će se drugom verzijom Dex-Net-a, tj. Dex-Net 2.0.

3.1.1. Skup podataka

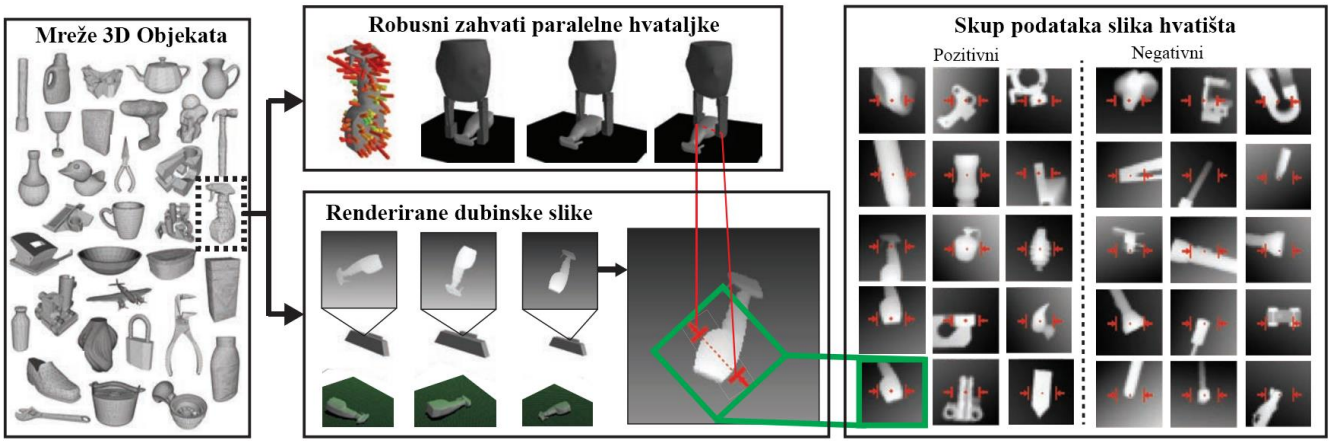
Budući da je Dex-Net 2.0 proširenje na prethodni rad, Dex-Net 1.0, oba skupa podatka sadržavaju velike sličnosti. Izvorni skup podataka se sastojao od 10 000 jedinstvenih 3D modela objekata sa 2.5 milijuna paralelnih hvatišta. Kao što je spomenuto u pregledu područja, svako hvatište sadrži mjeru robusnosti.

Izvedeni skup podataka koristio je 1500 mrežnih (engl. *mesh*) modela iz izvornog skupa podataka te se svaki objekt skalirao unutar 5 centimetara da bi odgovarao širini paralelne hvataljke te mu je postavljena težina od 1 kilograma. S obzirom na pretpostavku da se kamera nalazi okomito iznad objekta, također su određene stabilne poze objekta na osnovu njihovih fizikalnih svojstava.

Pristup određivanja hvatišta se sastoji od generiranja do 100 hvatišta na površini objekta koji su rezultat uzorkovanja odbijanjem antipodalnih parova. Svakom hvatištu se pridružuje mjera kvalitete na osnovu poze objekta, poze hvataljke te koeficijenta nesigurnosti trenja dobivenih Monte-Carlo uzorkovanjem. Za svaku stabilnu pozu objekta određuju se hvatišta paralelna stolu te hvatišta koja ne izazivaju koliziju hvataljke.

Zatim se za svaki objekt u svakom od njegovih stabilnih položaja renderiraju dubinske slike iz oblaka točaka pomoću nasumično generirane poze pinhole kamere. Kako bi sintetički kreirani skup podataka što bolje predstavljao stvarne scene, primjenjuju se multiplikacijski šum i Gaussovo procesni šum. Svako prethodno uzorkovano hvatište se povezuje s odgovarajućim pikselima na renderiranim slikama te se kao završni korak svaka slika translatira, rotira i reže tako da se hvatište nalazi u središtu slike. Vizualizacija ovog postupka nalazi se na slici *3.1.*

Rezultat ovog postupka kreiranja skupa podataka su 6.7 milijuna slika sa 32 piksela visine i širine.

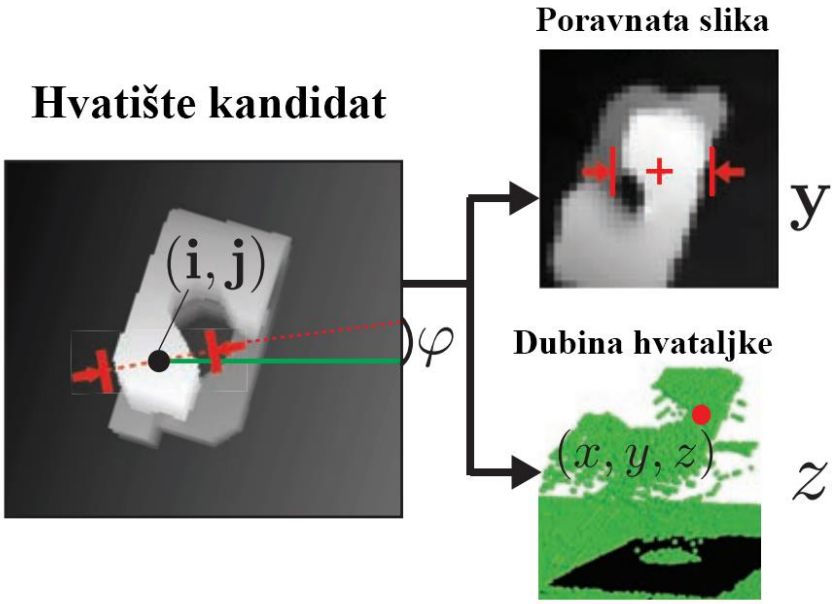


Slika 3.1. *Prikaz postupka kreiranja skupa podataka za učenje mreže*

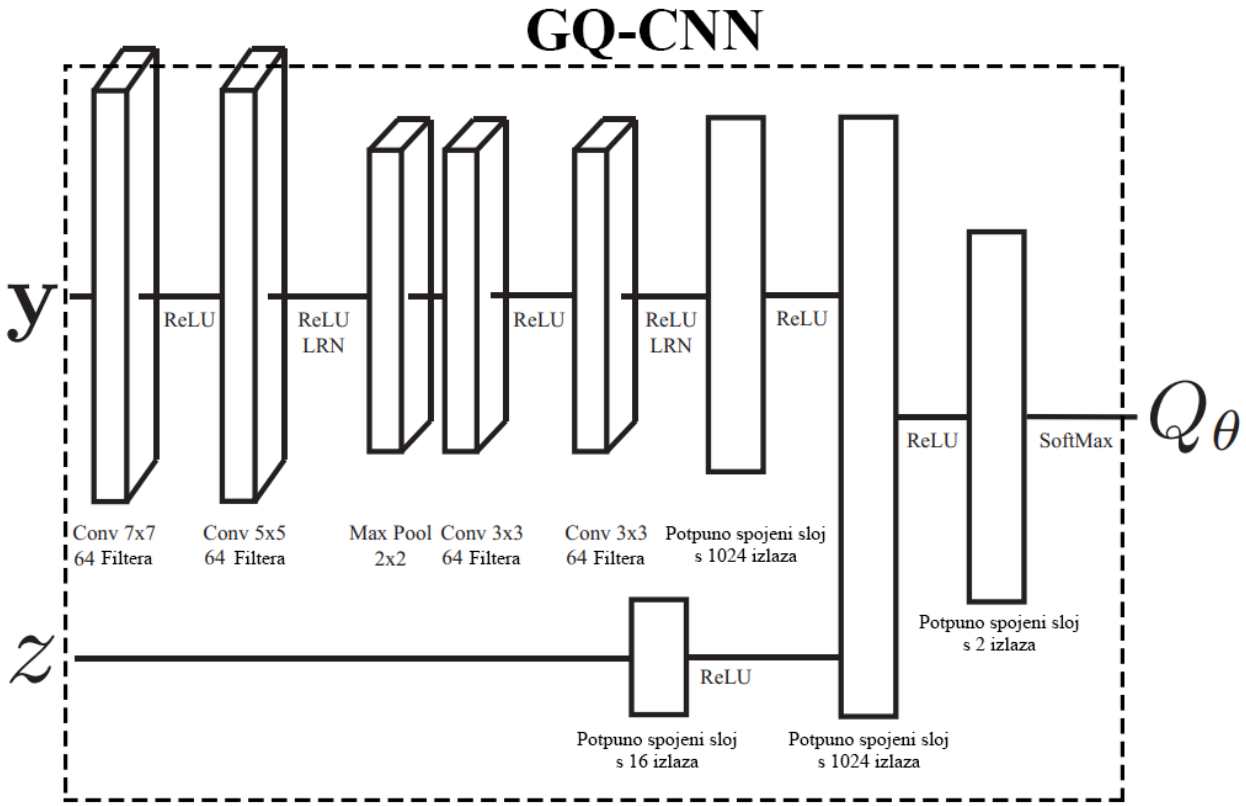
3.1.2. Konvolucijska neuronska mreža za kvalitetu hvatišta

Arhitektura Konvolucijske neuronske mreže za kvalitetu hvatišta (engl. *Grasp Quality Convolutional Neural Network = GQ-CNN*) sastoji se od 4 konvolucijska sloja u parovima razdvojenim ReLu nelinearnim aktivacijskim funkcijama iza kojih slijede 3 potpuno povezana sloja te dodatnim ulazom. Konačni rezultat Q_θ se dobiva primjenom SoftMax funkcije nad posljednjim potpuno povezanim slojem. Q_θ predstavlja robusnost hvatišta, tj. mjeru kvalitete uspješnog hvatišta u intervalu od 0-1. Arhitektura neuronske mreže prikazana je na slici *3.3.*

Ulazni prostor GQ-CNN-a se sastoji od dubinskih slika centriranih predloženih hvatišta kojima je os hvatišta φ (pravac koji povezuje dodirne točke hvatišta) poravnata s osi kamere, a na dodatni ulaz se dovodi udaljenost *z* od hvatišta do kamere. Prikaz ulaznih podataka nalazi se na slici *3.2.* Poravnanje osi kamere i hvatišta rezultira mogućnošću procjene hvatišta s bilo kojom orijentacijom. Ulazni podatci se normaliziraju oduzimanjem srednje vrijednosti i dijeljenjem sa standardnom devijacijom podataka za učenje.



Slika 3.2. *Ulazni podatci GQ-CCN-a*



Slika 3.3. *Prikaz arhitekture GQ-CCN mreže*

3.1.3. Učenje neuronske mreže

Učenje GQ-CNN-a se odvija na prethodno opisanom skupu podatka koji se sastoji od dubinskih slika s centriranim i poravnatim hvatištem objekta. Pri učenju potrebno je definirati parametar praga kvalitete, tj. prag robusnosti hvatišta δ, koji razdvaja uspješnu provedbu hvatanja objekta od neuspješne.

Iako je učenje GQ-CNN-a izvedeno na 4 tipa skupova podataka, najuspješniji rezultat je ostvaren učenjem na cijelom Dex-Net skupu podataka. Uz parametar praga kvalitete postavljen na 0.002 te standardnom devijacijom prethodno opisanog Gaussovog procesnog šuma od 0.005 izvršeno je učenje na 5 epoha cijelog Dex-Net skupa podataka.

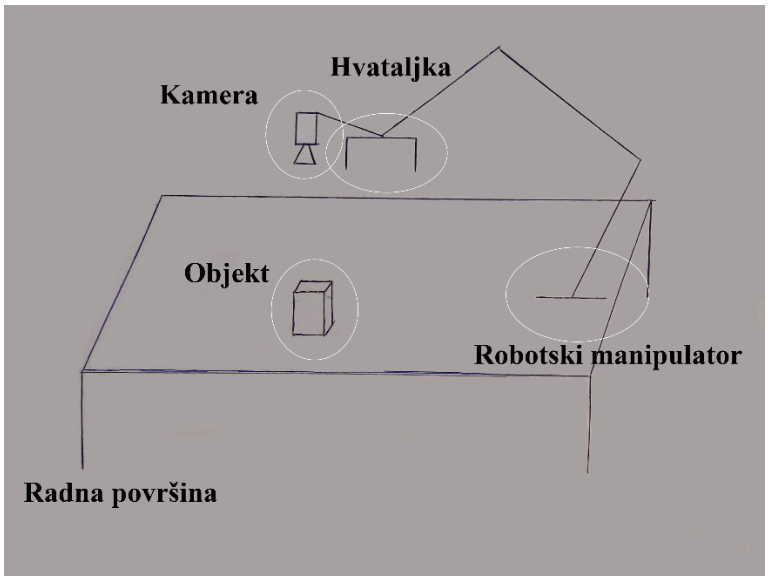
Programska izvedba neuronske mreže ostvarena je pomoću Tensorflow biblioteke s „batch" veličinom 128. Optimizacija parametara mreže odvija se pomoću metode povratne pogreške s eksponencijalno propadajućom stopom učenja koraka 0.95 te momentom 0.9. Prvotne težine se uzorkuju Gaussovom razdiobom simetričnom oko srednje vrijednosti (engl. „*zero-mean Gaussian noise*") s varijancom $\frac{2}{n_i}$. n_i predstavlja broj ulaza i-tog sloja neuronske mreže. U sklopu učenja mreže se provodi augmentacija podataka za učenje njihovim zrcaljenjem te rotiranjem za 180°. Već spomenuto primjenjivanje šuma na sintetičkim podatcima u potpoglavlju *3.1.1.* odvija se adaptivno tijekom učenja da bi se izbjeglo pohranjivanje više vrsta iste slike. Kako bi se ubrzao postupak uzorkovanja šuma, Gaussov procesni šum se aproksimira bilinearnom interpolacijom niza nekoreliranog „*zero-mean Gaussian noise*". Naučeni konvolucijski filteri vizualizirani su na slici *3.5.*

Učenje koristeći grafičku karticu Nvidia GeForce GTX 1080 trajalo je 48 sati.

Slika 3.4. *Naučeni konvolucijski filteri GQ-CNN mreže*

4. SKLOPOVLJE

Praktični dio ovog rada sastojao se od procesa primjene DexNet-a u stvarnom okruženju. Kako bi se mogao izvesti cijeli postupak hvatanja objekata bilo je potrebno softver i sklopovlje spojiti u cjelinu. Na slici *4.1.* prikazana je inicijalna skica potrebnih komponenti te radnog okruženja koji bi uz odgovarajući softver omogućio pokus hvatanja objekta. Sklopovlje komponenata će biti opisano u ovom poglavlju, dok će programska podrška komponenata te njihovi odnosi biti opisani u sljedećem poglavlju.



Slika 4.1. *Skica planiranog radnog okruženja*

4.1. UR5

Universal Robots UR5 je industrijski robotski manipulator napravljen od strane proizvođača Universal Robots namijenjen obavljanju opetovanih zadataka s maksimalnom nosivošću od 5 kilograma. Pripada obitelji visoko fleksibilnih robotskih manipulatora pod nazivom CB3-serija koja se sastoji od UR3, UR5 i UR10 kojima broj u nazivu odgovara i maksimalnoj nosivosti. [7] Na slici *4.2.* prikazan je UR5 robot s postavljenom Robotiq hvataljkom s 2 prsta.

Svojom težinom od 18.4 kilograma, radijusom operabilnosti od 850 milimetara te sa 6 zglobova odličnih je specifikacija za rad na pakiranju, sastavljanju i testiranju.

Jedna od posljednjih tehnologija u robotici su kolaborativni roboti koji su sigurni pri radu uz čovjeka. UR obitelj robota opremljena je osjetljivim senzorima koji robotu omogućuju da osjeti koliziju s okolinom , tj. da ako se robot prekine u njegovom izvršavanju zadatka, npr. primjenom

sile na robotsku ruku, obustavit će sve radnje te prijeći u sigurni način rada za razliku od tradicionalnih robota koji bi nastavili svoj rad te potencijalno ozlijedili osobu.

Universal Robots roboti sadrže URCap platformu koja omogućava jednostavnu instalaciju softvera na robotski manipulator u obliku sličnom kao instalacija aplikacija na pametni telefon. Instalacija se izvodi putem dlanovnika za upravljanje robotom te pokretanjem datoteke s *.urcap* ekstenzijom.

Svojim manjim dimenzijama, nižim cijenama te jednostavnijom programibilnošću omogućuju pristup razvoju i upotrebi industrijskih robota mnogo širem broju tvrtki.

Slika 4.2. *UR5 robot s Robotiq hvataljkom s dva prsta*

4.2. Intel RealSense L515

Intel RealSense LiDAR Camera L515 jedna je od najmanjih dubinskih kamera temeljenih na LiDAR-u visoke rezolucije. Ključna stavka ove kamere za njezinu upotrebu u robotici je njezina dubinska komponenta temeljena na LiDAR tehnologiji. Svojim malim dimenzijama, pristupačnom cijenom, visokom preciznošću od 0.25 do 9 metara te 1024x768 rezolucijom dubinske slike, predstavlja odlično rješenje za robotska okruženja. Relativno malom težinom obzirom na LiDAR tehnologiju, od 100 grama, postaje odlično rješenje za oko-u-ruci (engl. *eye-in-hand*) način postavljanja kamere u odnosu na robotsku ruku. Uz dubinsku komponentu, L515 pruža i RGB sliku rezolucije 1920x1080 uz brzinu slike (engl. *frame rate*) od 30 slika po sekundi. [8] Fizičke karakteristike kamere vidljive su na slici *4.3*.

Uz fizičke specifikacije važno je napomenuti opremu za razvoj programske podrške Intel RealSense SDK 2.0 otvorenog koda, koja pruža veliku fleksibilnost za primjenu kamere u okruženjima kao što je ROS.

Ograničenja s kojima se suočava su osjetljivost na refleksiju, tipična za ostale dubinske kamere, te minimalna udaljenost od 25 centimetara. Zbog kompaktnosti i složenosti kamere dolazi do pregrijavanja što dodatno narušava njezinu pouzdanost i ispravan rad.

Slika 4.3. *RealSense L515 kamera s odgovarajućom ambalažom*

4.3. Robotiq hvataljka

Adaptivna robotska hvataljka s 3 prsta proizvodnje tvrtke Robotiq sa svojim prilagodljivim prstima pruža višestruki pristup hvatanju objekata. Budući da se svaki prst može upravljati pojedinačno, hvataljka sastoji se od 4 načina hvatanja. Način štipaljke, široki način rada, način rada kao škare te osnovni način rada. [9] Otvorena hvataljka prikazana je na slici *4.4.* dok se fizičke specifikacije hvataljke nalaze u tablici *4.1*.

Hvataljka sadrži softver potreban za upravljanje istom putem dlanovnika UR5 robota u obliku URCap-a.

Slika 4.4. *Prikaz Robotiq hvataljke s 3 prsta*

Tablica 4.1. *Fizičke specifikacije Robitq hvataljke s tri prsta*

Otvor hvataljke	0 do 155mm
Težina hvataljke	2.3 kg
Promjer objekta	20 do 155mm
Maksimalna preporučena nosivost (obuhvatno prianjanje)	10 kg
Maksimalna preporučena nosivost (prianjanje vrhovima prstiju)	2.5 kg
Sila pritiska (prianjanje vrhovima prstiju)	30 do 70 N

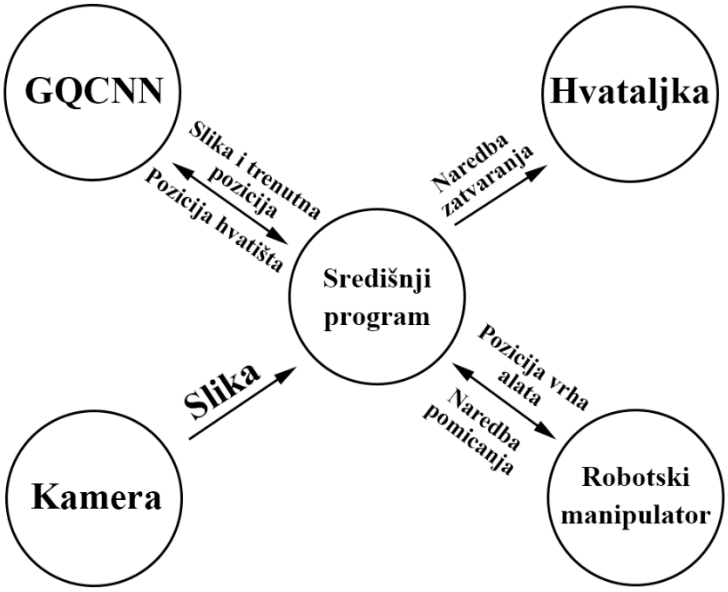
5. IMPLEMENTACIJA

5.1. Ideja tijeka rada

Na osnovu skice komponenata na slici *4.1.* potrebno je bilo odrediti komponente u obliku programske podrške te njihove relacije u ROS-u. Na dijagramu odnosa na slici *5.1.* vidljivi su funkcionalni čvorovi te njihovi odnosi sa središnjim upravljačkim čvorom.

Prvi korak je dohvaćanje slike scene s objektom pomoću čvora kamere koja se šalje u središnji čvor koji tu sliku obrađuje te prosljeđuje u čvor GQCNN-a. Da bi GQCNN mogao izračunati poziciju objekta u prostoru potrebna mu je informacija o poziciji kamere u prostoru. Pozicija kamere u odnosu na bazu robota dobiva se tako da čvor robotskog manipulatora (MoveIt) [10] šalje trenutnu poziciju vrha alata u središnji čvor gdje se uračunava odstupanje pozicije kamere u odnosu na vrh alata. Uz informaciju o poziciji kamere te sliku scene, GQCNN algoritam određuje položaj hvatišta na objektu u odnosu na poziciju kamere. Upravljački program šalje robotski manipulator na izračunatu poziciju hvatišta uz uračunavanje udaljenosti vrha hvataljke od vrha alata robotskog manipulatora te kada se ta kretnja izvrši, šalje hvataljci naredbu da se zatvori.

GQCNN funkcionalni čvor predstavljen je *grasp_planning_service* ROS čvorom opisanim u potpoglavlju *5.3.1.* Funkcionalni čvor kamere predstavljen je *rs_camera* ROS čvorom opisanim u potpoglavlju *5.3.2.* Čvor robotskog manipulatora predstavljen je pomoću ROS čvorova *ur5_bringup* opisanog u potpoglavlju *5.3.3.* te *ur5_moveit_planning_execution* i *robotManipulation.py* ROS čvorova opisanih u potpoglavlju *5.3.4.* Upravljanje hvataljkom izvedeno je pomoću *Robotiq3FGripperSimpleController* ROS čvora opisanog u potpoglavlju *5.3.5.* Središnji program koji povezuje sve ostale funkcionalne čvorove opisan je u potpoglavlju *5.3.6.* te će uz čvor *robotControl.py* biti popraćen opisanim programskim kodom.



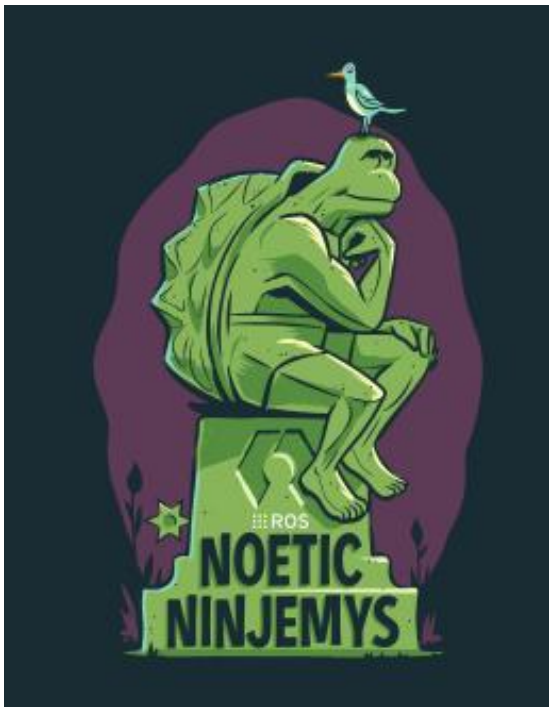
Slika 5.1. *Prikaz funkcionalnih čvorova i njihovih odnosa*

5.2. ROS

Robot Operating System (ROS) je otvoreni kod softverskih okvira, koji unatoč imenu ne predstavlja operativni sustav nego okruženje programske podrške za razvoj robota. ROS pruža usluge kao što su apstrakcija sklopovlja, upravljanje uređaja niske razine, prijenos poruka između procesa, upravljanje paketima te implementaciju često korištenih funkcionalnosti u razvoju robotske programske podrške. Iako je potreba za izvođenje u stvarnom vremenu pri upravljanu robota jako velika, ROS nije operativni sustav stvarnog vremena te je to dovelo do razvoja ROS-a 2. Podržani i preporučeni operativni sustav ROS-a je Ubuntu Linux zbog ROS-ove naklonjenosti Unix sustavima zbog velike ovisnosti o programskoj podršci otvorenog koda. Operativni sustavi poput macOS-a, Microsoft Windows-a te ostalih verzija Linux-a označeni su kao eksperimentalni od strane ROS-a, ali uz veliku podršku zajednice ROS postaje izvediv i na tim operativnim sustavima. Jedan od glavnih principa razvoja ROS-a je fleksibilnost te je sukladno tome omogućeno korištenje više programskih jezika za razvoj programske podrške, ali su C++ i Python najzastupljeniji. C++ ima veću naklonost zbog boljih performansi i bliže povezanosti sa sklopovljem, za razliku od Python-a kojeg krasi jednostavnije pisanje koda.

Jedna od značajnijih koristi ROS-a je njegova reproduktivnost u obliku paketa (engl. *package*) koji omogućuju jednostavan prijenos i instalaciju programske podrške, temeljene na principima Unix sustava. [11][12]

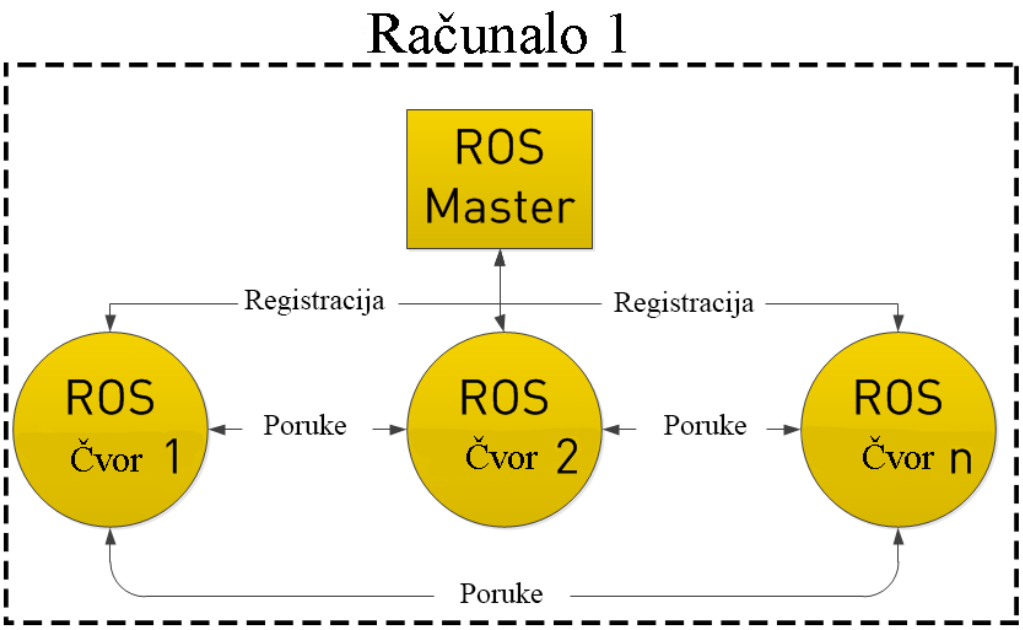
ROS 1 trenutno sadrži 13 distribucija dok prva ROS distribucija seže iz 2010. godine pod nazivom ROS Box Turtle. ROS distribucije su verzionirani skupovi ROS paketa koji u pravilu izlaze svakih godinu dana. Najnovije verzije su ROS Noetic te ROS Melodic koji i dalje primaju podršku od strane ROS razvojnog tima. Ikona ROS Noetic-a prikazana je na slici *5.2.*



Slika 5.2. *Ikona korištene distribucije ROS-a u ovom radu*

5.2.1. Struktura ROS-a

Procesi u ROS-u predstavljeni su kao čvorovi u strukturi spojeni temama (engl. *topic*). Čvorovi prenose poruke jedni drugima, pružaju servise (engl. *service*) drugim čvorovima, postavljaju ili dohvaćaju podatke iz zajedničke baze podataka pod nazivom poslužitelj parametara (engl. *parameter server*) putem tema. Iako ROS *master,* tj. glavni proces osposobljava rad ROS-a vezujući sve čvorove na sebe, poruke i servisi se ne razmjenjuju kroz *master* nego *peer-to-peer* komunikacijom između čvorova. Struktura i odnosi čvorova prikazani su na slici *5.3.* Ovakav koncept komunikacije dobro je prilagođen robotima koji se često sastoje od podmreža računala.



Slika 5.3. *Prikaz strukture ROS procesa*

5.3. Implementacija čvorova

Implementacija programske podrške izvedena je u ROS Noetic okruženju na Ubuntu 20.04 operacijskom sustavu. Budući da je GQ-CNN izvorno pisan u Python programskom jeziku, zbog jednostavnosti su i ROS čvorovi pisani u Pythonu i to verziji 3.6.

5.3.1. GQ-CNN paket

Budući da je GQ-CNN već prilagođen za korištenje u ROS-u, njegova instalacija se sadržavala od kloniranja službenog Github repozitorija u *src* direktoriji radnog prostora ROS-a te izgradnjom (engl. *build*) koristeći *catkin_make* funkciju ROS-a.

Ovaj paket sadrži potrebne modele, ROS čvor za planiranje hvatanja, te Python biblioteke potrebne za pronalaženje hvatišta, tj. za upotrebu GQ-CNN modela na stvarnom robotu i okruženju. *Grasp_planning_service.launch* datoteka zaslužna je za pokretanje *grasp_planner* čvora koji prima parametre modela DexNet-a. Pokretanje čvora izvodi se naredbom prikazanom na slici *5.4.*

```
$ roslaunch gqcnn grasp_planning_service.launch
```

Slika 5.4. *Naredba za pokretanje GQCNN čvora*

5.3.2. ROS omotač (engl. *wrapper*) za Intel RealSense uređaje

Instalacija ROS omotača za Inter RealSense kameru L515 izvediva je na 2 načina. Jednostavniji način je korištenjem APT korisničkog sučelja za instalaciju programske podrške na Debian Linux distribucijama, dok je robusniji način izgradnjom iz izvornog koda koji se nalazi na Github repozitoriju. [13]

Datoteka za pokretanje L515 kamere je *rs_camera.launch* koja pokreće *realsense2_camera* čvor uz parametar *align_depth* koji je zaslužan za poravnanje RGB slike s dubinskom. Pokretanje čvora izvodi se naredbom prikazanom na slici *5.5.*

Nakon pokretanja čvora kamere dobivamo pristup raznim ROS temama, a one korištene u ovom radu su tema za intrinzične parametre kamere te tema za dohvaćanje dubinske slike.

```
$ roslaunch realsense2_camera rs_camera.launch align_depth:=true
```

Slika 5.5. *Naredba za pokretanje čvora kamere*

5.3.3. Universal Robots ROS driver

ROS programska podrška potrebna da bi se upravljalo robotom UR5, nalazi se u paketu pod nazivom Universal_Robots_Ros_Driver kojeg je potrebno izgraditi iz izvornog koda. Izvorni kod se nalazi na službenom Github repozitoriju tvrtke Universal Robots. [14]

Nakon uspješne instalacije na računalu, potrebno je konfigurirati robota za vanjsko upravljanje (engl. *external control*). Na robot u jednom trenu može biti spojeno samo jedno računalo, a to računalo se definira u konfiguraciji robota pomoću lokalne IP adrese i porta računala domaćina. Komunikacija s računalom se izvodi pomoću robotskog programa *Vanjsko Upravljanje* (engl. *external control*) koje mora biti pokrenuto na robotu dok se na računalu treba pokrenuti datoteka *ur5_bringup.launch* uz parametar lokalne IP adrese robota koja inicijalizira *robot_state_publisher* čvor. Pokretanje čvora izvodi se naredbom prikazanom na slici *5.6.*

Nakon uspješnog uspostavljanja komunikacije, robot je spreman za prihvaćanje naredbi.

```
$ roslaunch ur_robot_driver ur5_bringup.launch robot_ip:=192.168.22.14
```

Slika 5.6. *Naredba za pokretanje UR5 čvora*

5.3.4. MoveIt paket

Nakon uspostave komunikacije između računala i robota, potrebno je slati naredbe za upravljanje koje se sastoje u pomicanju zglobova robotske ruke. Jedno od najpoznatijih sučelja otvorenog koda za planiranje kretanja (engl. *motion planning*) je MoveIt. Instalacija se izvodi pomoću APT korisničkog sučelja za instalaciju programske podrške na Debian Linux distribucijama.

Iako MoveIt sadrži niz implementacija kao što su planiranje scene, MoveIt hvatanje objekata, MoveIt *Pick and Place* za hvatanje i premještanje objekata i drugi, u ovom radu koristit će se Pokretna Grupa (engl. *Move Group*) korisničko sučelje za pomicanje robota uz korisničko sučelje planiranja scene. [15]

Kako bi se olakšala komunikacija između MoveIt čvorova i UR5 robota, Universal Robots ROS driver dolazi s potrebnom konfiguracijom parametara za svaki robotski manipulator iz UR obitelji. Pokretanje čvora izvodi se naredbom prikazanom na slici *5.7.*

Kako bi komunikacija korisničkih ROS čvorova koji koriste MoveIt sučelje bila moguća, potrebno je imati uključen *move_group* čvor.

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

Slika 5.7. *Naredba za pokretanje MoveIt čvora*

Budući da je za izvođenje pokusa ključno poznavanje inicijalne pozicije robotskog manipulatora, a ista se može dobiti pomoću MoveIt paketa, potrebno je imati korisnički čvor koju će tu informaciju stalno objavljivati. Korisnički čvor za poziciju robotskog manipulatora nalazi se u *robotManipulaiton.py* dokumentu. Pokretanje čvora izvodi se naredbom prikazanom na slici *5.8.*

```
$ rosrun gqcnn robotManipulation.py
```

Slika 5.8. *Naredba za pokretanje robotManipulation čvora*

Na slici *5.9.* nalazi se kod koji opisuje inicijalizaciju *robot_manipulation* čvora uz objavljivače (engl.*publishers*) za trenutnu poziciju vrha alata u obliku trodimenzionalne koordinate (*robot_position_publisher*) te u obliku kutova zglobova robota (*robot_joints_publisher*). Tipovi ROS poruka koje se koriste za ove objavljivače su *geometry_msgs.msg.Pose* te *std_msgs.msg.Float32MultiArray*. Trenutna pozicija vrha alata dohvaća se putem MoveIt objekta *MoveGroupCommander* korištenjem metode *get_current_joint_values()* za kutove zglobova, a pozicija vrha alata metodom *get_current_pose()*.

```
if __name__ == "__main__":
    rospy.init_node("robot_manipulation")
    pub = rospy.Publisher('robot_manipulation_publisher', Pose, queue_size=10)
    pub_joints = rospy.Publisher('robot_joints_publisher', Float32MultiArray, queue_size=10)

    robot = moveit_commander.RobotCommander()
    scene = moveit_commander.PlanningSceneInterface
    rate = rospy.Rate(10)
    group_name = 'manipulator'
    move_group = moveit_commander.MoveGroupCommander(group_name)

    while not rospy.is_shutdown():
        pub.publish(move_group.get_current_pose().pose)
        msg = Float32MultiArray()
        msg.data = move_group.get_current_joint_values()
        msg.layout.data_offset = 0
        msg.layout.dim = [MultiArrayDimension()]
        msg.layout.dim[0].label = "joint_values"
        msg.layout.dim[0].size = 6
        msg.layout.dim[0].stride = 0
        pub_joints.publish(msg)
        rate.sleep()
    rospy.spin()
```

Slika 5.9. *Inicijalizacija robot_manipulation čvora te objavljivača*

Datoteka *robotManipulation.py* također sadrži funkciju zasluženu za slanje naredbe za pozicioniranje robotskog manipulatora, prikazanu na slici *5.10.*

Parametri potrebni za izvođenje pozicioniranja vrha alata su odredišna pozicija i orijentacija. Pozicija se izražava kao trodimenzionalna koordinata, a orijentacija kao kvaternion. Budući da je u izvođenju ovog dijela rada MoveIt imao poteškoće s pozicioniranjem putem odredišnih koordinata, korišteni su kutovi zglobova robota. Metoda *get_ik* objekta IK (*Inverse Kinematics)* na temelju trenutnih kutova zglobova i odredišnih parametara izračunava konačne kutove

zglobova. Izvođenje same kretnje robotskog manipulatora postiže se metodom *MoveGroupCommander-a go()*.

```
def move_to_pose(position, orientation):
    ik = IK('base_link', 'tool0', solve_type="Distance")
    robot, scene, move_group = initial_workplace()
    current_joints = move_group.get_current_joint_values()
    x = position[0]
    y = position[1]
    z = position[2]
    qx = orientation[1]
    qy = orientation[2]
    qz = orientation[3]
    qw = orientation[0]
    for joint in current_joints:
        if (joint > 3.14):
            joint = joint - 3.14
        elif (joint < -3.14):
            joint = joint + 3.14
    goal_joints = ik.get_ik(current_joints, x, y, z, qx, qy, qz, qw)
    joint_goal = move_group.get_current_joint_values()
    joint_goal[0] = goal_joints[0]
    joint_goal[1] = goal_joints[1]
    joint_goal[2] = goal_joints[2]
    joint_goal[3] = goal_joints[3]
    joint_goal[4] = goal_joints[4]
    joint_goal[5] = goal_joints[5]
    move_group.go(joint_goal, wait=True)
    move_group.stop()
    move_group.clear_pose_targets()
```

Slika 5.10. *Funkcija za pozicioniranje robotskog manipulatora*

5.3.5. Robotiq paket

ROS korisničko sučelje za upravljanje hvataljkama proizvođača Robotiq potrebno je izgraditi iz izvornog koda koji se nalazi na Github repozitoriju. Za komuniciranje s hvataljkom postoje 2 komunikacijska protokola: Modbus RTU i Modbus TCP. [16] Pokretanje čvora izvodi se naredbom prikazanom na slici *5.11*.

```
$ rosrun robotiq_3f_gripper_control Robotiq3FGripperSimpleController
```

Slika 5.11. *Naredba za pokretanje Robotiq čvora*

5.3.6. Središnji čvor

Kako bi se postigla očekivana funkcionalnost potrebno je sve prethodne pakete povezati u jednu smislenu cjelinu s korisničkim sučeljem. Da bi središnji čvor radio ispravno, potrebno je pokrenuti sve prethodno navedene čvorove te pokrenuti naredbu u terminalu sa slike *5.12.*

```
$ rosrun gqcnn centralController.py
```

Slika 5.12. *Naredba za pokretanje središnjeg čvora*

Kao i u potpoglavlju *5.3.4.* kod *robot_manipulation.py* datoteke, potrebno je inicijalizirati čvor te pojedine objavljivače. U ovom slučaju, kako je prikazano i na slici *5.13.* moguće je proslijediti tri argumenta koji odgovaraju širini hvataljke, imenskom prostoru (engl. *namespace*) te vizualizaciji hvatišta objekta. U slučaju izvedenih pokusa, korišteni su zadani argumenti, tj. širina hvataljke od 5cm, gqcnn imenski prostor te vizualizacija hvatišta objekta.

```
parser = argparse.ArgumentParser(
   description="Run a grasping from realsense camera.")
parser.add_argument("--gripper_width",
                    type=float,
                    default=0.05,
                    help="width of the gripper to plan for")
parser.add_argument("--namespace",
                    type=str,
                    default="gqcnn",
                    help="namespace of the ROS grasp planning service")
parser.add_argument("--vis_grasp",
                    type=bool,
                    default=True,
                    help="whether or not to visualize the grasp")
args = parser.parse_args()
gripper_width = args.gripper_width
namespace = args.namespace
vis_grasp = args.vis_grasp
```

Slika 5.13. *Dio koda zaslužan za definiranje argumenata pri pokretanju čvora*

Korišteno ime čvora isto je kao i ime datoteke. Budući da se softver hvataljke oslanja na ROS, također je bilo potrebno inicijalizirati objavljivač pod imenom *Robotiq3FGripperRobotOutput* koji objavljuje poruku tipa *robotiq_3f_gripper_articulated_msgs.msg. Robotiq3FGripperRobotOutput*.

Kako bi se ostvarila veza s gqcnn čvorom pod imenom *grasp_planner*, potrebno je servis *gqcnn/grasp_planner* povezati s lokalnom varijablom. Tip servisa koji povezujemo je *GQCNNGraspPlanner* koristeći metodu *ServiceProxy* iz objekta *rospy*. *rospy* predstavlja aplikacijsko programsko sučelje (API) za ROS izveden u Pythonu, dok je za C to biblioteka *ros/ros.h*. Za obradu poruke slike koju zaprimimo od čvora kamere potrebno je inicijalizirati objekt klase *CvBridge()* iz biblioteke OpenCV. [17] Opisani postupak nalazi se na slici *5.14.*

```
# Initialize the ROS node.
   rospy.init_node("central_controller")
   pub = rospy.Publisher('Robotiq3FGripperRobotOutput', Robotiq3FGripperRobotOutput,
queue_size=10)

   # Wait for grasp planning service and create service proxy.
   rospy.wait_for_service("%s/grasp_planner" % (namespace))
   plan_grasp = rospy.ServiceProxy("%s/grasp_planner" % (namespace),
                        GQCNNGraspPlanner)
   cv_bridge = CvBridge()
```

Slika 5.14. *Inicijalizacija čvor, objavljivača i ostvarivanje veze s gqcnn-om*

Uz sliku, pri obradi iste, potrebno je pronaći intrinzične parametre kamere. Parametri kamere se mogu učitati iz datoteke, ali se u slučaju *realsense2* kamere mogu pretplatiti na ROS temu pod imenom */camera/depth/camera_info* iz čvora kamere. Kao i za parametre kamere, sama slika dobiva se putem pretplate na temu pod nazivom */camera/depth/image_rect_raw* kao što je prikazano na slici *5.15*. Metoda za dohvaćanje jedne poruke s određene teme nalazi se također u *rospy* biblioteci pod imenom *wait_for_message*. Tipovi poruke su *sensor_msgs.msg CameraInfo* te *Image*. Budući da je za daljnju upotrebu intrinzičnih parametara kamere potrebno koristiti sučelje *CameraIntrinsics* iz biblioteke *autolab_core*, kreirana je funkcija *to_camera_intrinsics*, prikazana na slici *5.16.* koja ROS poruku preslikava u već spomenuto sučelje.

Autolab Core predstavlja modul koji sadrži skup korisnih uslužnih (engl. *utilities*) programa za robotske zadatke, kreiran od strane Berkley AutoLab-a, koji je također kreator DexNet-a. [18]

```
camera_intr_topic = '/camera/depth/camera_info'
camera_info = rospy.wait_for_message(camera_intr_topic, CameraInfo)
camera_intr = to_camera_intrinsics(camera_info)
image_topic = "/camera/depth/image_rect_raw"
img = rospy.wait_for_message(image_topic, Image)
```

Slika 5.15. *Veza s čvorom kamere*

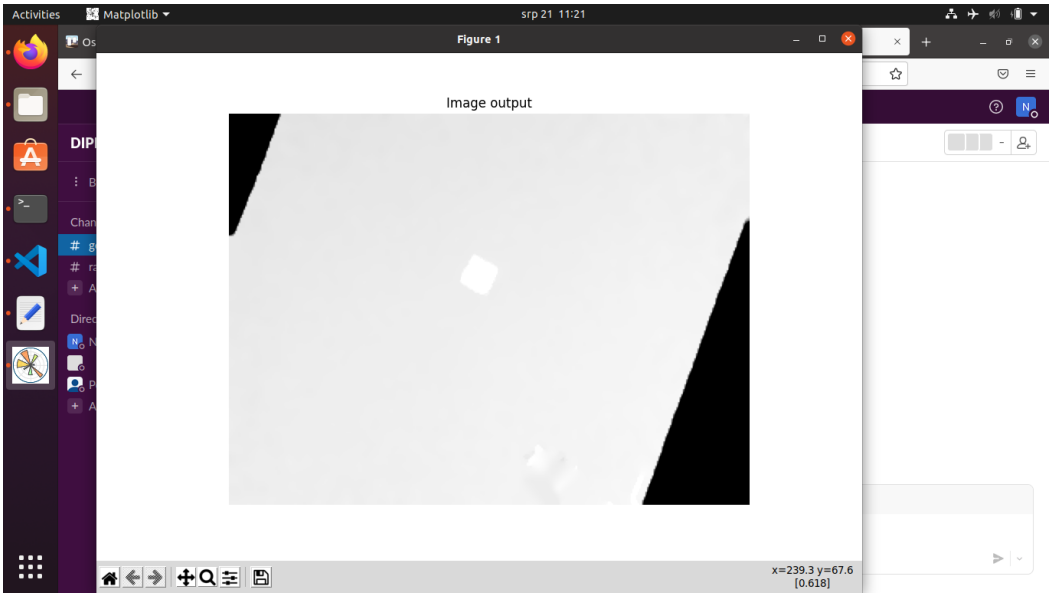
```
def to_camera_intrinsics(msg):
   frame = msg.header.frame_id
   fx = msg.K[0]
   fy = msg.K[4]
   cx = msg.K[2]
   cy = msg.K[5]
   skew = 0.0
   height = msg.height
   width = msg.width
   return CameraIntrinsics(frame, fx, fy, cx, cy, skew, height, width)
```

Slika 5.16. *Funkcija za preslikavanje intrinzičnih parametara kamere*

Nakon skeniranja scene, dobivenu sliku je potrebno obraditi. Obrada slike sastoji se od pretvaranja ROS poruke slike u OpenCV sliku putem već spomenutog CvBridge-a. Budući da GQCNN koristi RGB-D ulaz, ali ne koristi podatke iz RGB-a, za RGB koristimo praznu matricu s istim dimenzijama kao i dubinska slika. Na dubinsku sliku moguće je primijeniti „*inpainting*“ koji služi kako bi se svi pikseli s vrijednostima nula interpolirali s okolnim pikselima. Funkcija zaslužena za *„inpainting”* postupak nalazi se na slici *5.17.* Nakon *„inpainting-a“* vizualizira se dubinska slika scene prije prosljeđivanja GQCNN-u, a primjer iste prikazan je na slici *5.18.*

```
im = cv_bridge.imgmsg_to_cv2(msg, desired_encoding="passthrough").astype(np.float32) / 1000
depth_im = DepthImage(im, frame=camera_intr.frame)
color_im = ColorImage(np.zeros([depth_im.height, depth_im.width, 3]).astype(np.uint8),
                      frame=camera_intr.frame)
depth_im = depth_im.inpaint()
if vis_grasp:
    vis.figure(size=(10, 10))
    vis.imshow(depth_im, vmin=0.6, vmax=0.9)
    vis.title("Image output")
    vis.show()
```

Slika 5.17. *Obrada slike scene te vizualizacija dubinske slike scene*



Slika 5.18. *Prikaz vizualizacije scene nakon obrade slike*

Prosljeđivanje potrebnih podataka izvodi se putem već spomenutog servisa u varijabli *plan_grasp*. Rezultat određivanja hvatišta GQCNN-a potrebno je pohraniti u sučelje *Grasp2D* iz biblioteke *gqcnn.grasping*. Budući da rezultat određivanja hvatišta sadrži sve potrebne podatke o hvatištu, *Grasp2D* sučelju prosljeđujemo podatke o središtu hvatišta, udaljenosti odnosno dubini hvatišta, kutu hvatišta te širini hvataljke i intrinzičnim parametrima kamere. Matricu položaja objekta u

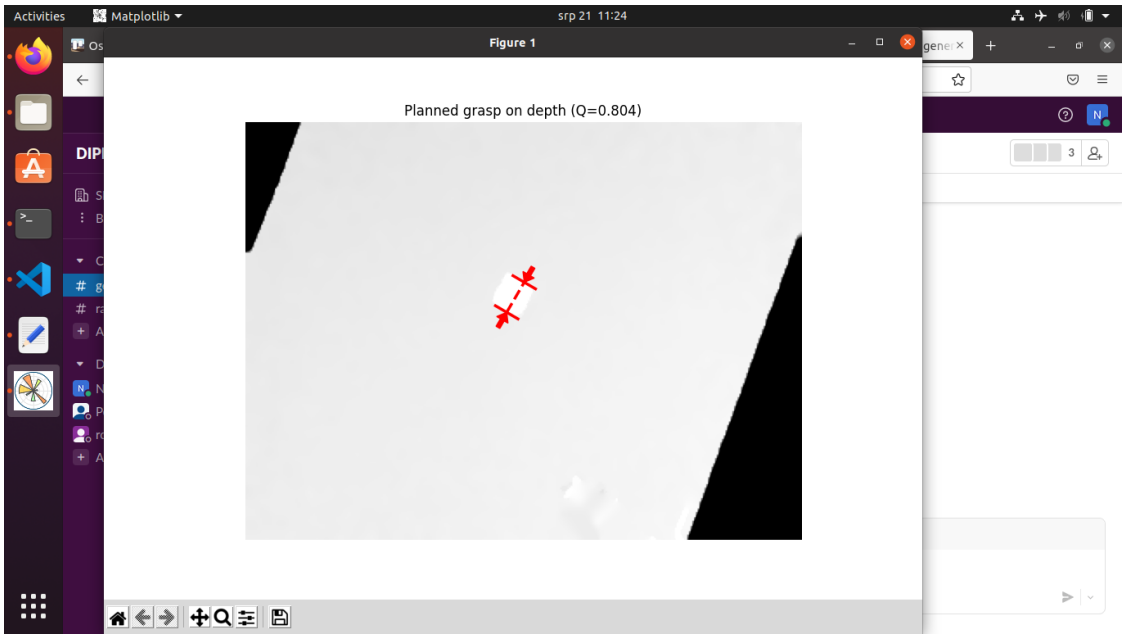
odnosu na kameru dobivamo putem metode *pose()* sučelja *Grasp2D*, a dobivena transformacijska matrica je inkapsulirana u sučelju *RigidTransform* iz već spomenute biblioteke Autolab Core.

Kako bismo vizualizirali hvatište na objektu u snimljenoj sceni, potrebno je proslijediti podatke o hvatištu te dubinsku sliku scene dobivenu od strane GQCNN-a u *GraspAction* sučelje. Vrijednost kvalitete hvatišta se također prikazuje u naslovu vizualizacije putem *GraspAction.q_value* svojstva (engl. *property)*. Cijeli postupak prikazan je na slici *5.19.*

```
grasp_resp = plan_grasp(color_im.rosmsg, depth_im.rosmsg,
                        camera_intr.rosmsg)
grasp = grasp_resp.grasp
center = Point(np.array([grasp.center_px[0], grasp.center_px[1]]),
               frame=camera_intr.frame)
grasp_2d = Grasp2D(center,
                   grasp.angle,
                   grasp.depth,
                   width=gripper_width,
                   camera_intr=camera_intr)
T_grasp_camera = grasp_2d.pose()

thumbnail = DepthImage(cv_bridge.imgmsg_to_cv2(
          grasp.thumbnail, desired_encoding="passthrough"),
          frame=camera_intr.frame)
action = GraspAction(grasp_2d, grasp.q_value, thumbnail)
if vis_grasp:
    vis.figure(size=(10, 10))
    vis.imshow(depth_im, vmin=0.6, vmax=0.9)
    vis.grasp(action.grasp, scale=2.5, show_center=False, show_axis=True)
    vis.title("Planned grasp on depth (Q=%.3f)" % (action.q_value))
    vis.show()
```

Slika 5.19. *Proces određivanja položaja objekta u odnosu na kameru i prikaz hvatišta*



Slika 5.20. *Prikaz vizualizacije hvatišta u sceni, os hvatišta prikazana je crvenom bojom*

Nakon uspješnog određivanja položaja objekta u odnosu na kameru, primjer prikazan na slici *5.20.*, potrebno je odrediti transformacijsku matricu koja predstavlja položaj u koji trebamo dovesti vrh alata robotskog manipulatora prije zatvaranja hvataljke.

Kako bismo odredili tu matricu, potrebne su nam transformacijske matrice vrha alata u odnosu na bazu robotskog manipulatora, vrha alata u odnosu na kameru te vrha alata u odnosu na vrh hvataljke. Za lakše rukovanje istih, svaka će biti inkapsulirana u sučelje *RigidTransform*.

Trenutnu poziciju vrha alata u odnosu na bazu robota dobivamo putem već spomenutog čvora *robot_manipulation* i poruke iz objavljivača *robot_manipulation_publisher* te ju spremamo kao varijablu *current_tcp_position*. Postupak dohvaćanja trenutne pozicije vrha alata prikazan je na slici *5.21.*

```
current_pos = rospy.wait_for_message('robot_manipulation_publisher', Pose)
   current_pos_rigid = RigidTransform(translation=np.array([current_pos.position.x,
current_pos.position.y,
                                        current_pos.position.z], np.float32),
                        from_frame='tcp',
                        to_frame='base'
                        )
   current_pos_rigid.rotation =np.array([current_pos.orientation.w,
                                    current_pos.orientation.x,
                                    current_pos.orientation.y,
                                    current_pos.orientation.z,
                                    ], np.float)
   current_tcp_position = current_pos_rigid
```

Slika 5.21. *Dohvaćanje trenutne pozicije vrha alata u odnosu na bazu robota*

Vrijednosti korištene pri izradi matrica položaja vrha hvataljke u odnosu na vrh alata te položaja kamere u odnosu na vrh alata su ručno mjerene te su aproksimativne vrijednosti. Veliki problem u cijelom postupku određivanja položaja objekta u odnosu na bazu robotskog manipulatora je nepreciznost mjerenja položaja kamere u odnosu na vrh alata te sklonost kamere pomicanju koje rezultira osjetnim greškama u krajnjem rezultatu. Izmjerene vrijednosti prikazane su na slici *5.22.*

```
measured_tooltip_to_tcp = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.24]], np.float32)
measured_camera_to_tcp = np.array([[-0.99499865, 0, -0.09988833, 0],[0, -1, 0, 0.190],[-
0.09988833, 0, 0.99499865, -0.015]], np.float32)
T_tooltip_tcp = RigidTransform(translation=measured_tooltip_to_tcp[:, 3],
                         rotation=measured_tooltip_to_tcp[:3, :3],
                         from_frame='tooltip',
                         to_frame='tcp')
T_tcp_camera = RigidTransform(translation=measured_camera_to_tcp[:, 3],
                        rotation=measured_camera_to_tcp[:3, :3],
                        from_frame='tcp',
                        to_frame='camera_depth_optical_frame')
```

Slika 5.22. *Postavljanje matrica na temelju izmjerenih vrijednosti*

Izračun same vrijednosti konačnog položaja vrha alata u odnosu na bazu robotskog manipulatora sastoji se od transformacija potrebnih da bi se hvatište objekta prikazalo u baznom sustavu, tj. položaj objekta u odnosu na bazu robotskog manipulatora. Nakon određivanja spomenute, potrebno je rotirati vrh alata da prsti hvataljke budu sukladni osi hvatišta objekta, prikazanoj na slici *5.20.* Budući da se objektu uvijek prilazi po osi *z*, tj. okomito na površinu stola, potrebno je uračunati dužinu hvataljke u konačni položaj vrha alata *end_tcp_position*. Izračun konačnog položaja vrha alata prikazan je na slici *5.23.*

```
end_tcp_position = current_tcp_position * T_tcp_camera.inverse() * T_grasp_camera
y = np.cross(current_pos_rigid.z_axis, end_pos.y_axis)
y = y / np.linalg.norm(y)
x = np.cross(y, current_pos_rigid.z_axis)

end_tcp_position.rotation = np.c_[x, y, current_pos_rigid.z_axis]

gripper_rotation_offset_angle = 45
gripper_rotation_offset = np.array(
   [np.cos(np.radians(gripper_rotation_offset_angle)), -
np.sin(np.radians(gripper_rotation_offset_angle)), 0,
np.sin(np.radians(gripper_rotation_offset_angle)),np.cos(np.radians(gripper_rotation_offset_
angle)), 0, 0, 0, 1])

end_tcp_position.rotation = np.matmul(end_tcp_position.rotation,
gripper_rotation_offset.reshape((3,3)))

end_tcp_position.translation = end_tcp_position.translation + T_tooltip_tcp.translation
```

Slika 5.23. *Izračun konačnog položaja vrha alata*

Izvođenje samog hvatanja objekta odvija se putem *robotManipulation* sučelja iz potpoglavlja *5.3.4.* Prije pozicioniranja vrha alata hvataljku je potrebno otvoriti te nakon postavljanja istog na konačnu poziciju, hvataljku zatvoriti. Komunikacija s hvataljkom prikazana je na slici *5.24.* Otvaranje i zatvaranje hvataljke realizirano je putem funkcija *open_gripper()* te *close_gripper()* prikazanih na slici *5.25*.

```
open_gripper()
robotManipulation.move_to_pose(end_tcp_position.position, end_tcp_position.quaternion,
rospy.wait_for_message('robot_joints_publisher', Float32MultiArray).data)
close_gripper()
```

Slika 5.24. *Kod za izvođenje hvatanja objekta*

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

    start_time = time()

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

        end_time = time()

        if float(end_time - start_time) >= 3.0:
            break
def close_gripper():
    command = Robotiq3FGripperRobotOutput()
    command.rACT = 1
    command.rGTO = 1
    command.rSPA = 255
    command.rFRA = 150
    command.rATR = 0
    command.rMOD = 1
    command.rPRA = 100

    start_time = time()

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

        end_time = time()

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

Slika 5.25. *Implementacija funkcija za upravljanje hvataljkom*

6. EKSPERIMENTALNA EVALUACIJA I REZULTATI

6.1. Pokusi

Budući da je nit vodilja ovog rada bila ostvariti hvatanje objekata u stvarnom okruženju te spoj svih sustava u jednu smislenu cjelinu, pokusi su također odrađeni u stvarnom okruženju.

Izbor objekata nad kojima će se vršiti pokus je ograničen na objekte koji se mogu uhvatiti širinom hvataljke do 5 centimetara, te reflektivnošću površine objekta zbog ograničenja dubinske kamere izazvanih iznimnom osjetljivošću na refleksiju.

Pozicija objekata ograničena je samim Dex-Net-om, tj. uvjetima pod kojima je nastao skup za učenje na kojem je rađeno učenje mreže. Idealni uvjeti za objekt su: stabilni položaj objekta na ravnoj plohi udaljenoj oko 50 centimetara od kamere te objekt pozicioniran direktno ispod kamere.

Testiranje se sastojalo od ukupno 10 testova s jednim objektom na različitim pozicijama na površini radne plohe.

Uzroci koji su rezultirali neuspješnim izvođenjem pokusa su:

- Neprepoznavanje objekta od strane DexNet-a
- Neprecizno određeno hvatište na samom objektu od strane DexNet-a
- Netočno određen položaj objekta
- Netočno izvršeno planiranje puta robotskog manipulatora

Distribucija uzroka koji su rezultirali neuspješnim izvođenjem pokusa kroz sve provedene pokuse nalazi se u trećem stupcu tablice *6.1.*

Tablica 6.1. *Tablica rezultata sprovedenih pokusa i njihova uspješnost*

Broj pokusa	Rezultat pokusa	Uzrok pogreške
1	Pozitivan	-
2	Negativan	Neprecizno određeno hvatište
3	Pozitivan	-
4	Negativan	Netočno određen položaj objekta
5	Negativan	Neprepoznavanje objekta
6	Negativan	Netočno izvršeno planiranje puta robotskog manipulatora

7	Pozitivan	-
8	Pozitivan	-
9	Negativan	Netočno određen položaj objekta
10	Negativan	Netočno određen položaj objekta

S obzirom na velik broj neuspješnih pokusa, efektivnost DexNet-a bi se mogla dovesti u pitanje, ali s obzirom da su samo dva pokusa uzrokovana izravnom greškom DexNet-a, efektivnost istoga bi se trebala procjenjvati u boljim uvjetima radnog okruženja.

Neka od izvedivih poboljšanja radnog okruženja osim položaja kamere su kvalitetnija RGB-D kamera, prostorija s boljim osvjetljenjem, nerefklektivne površine, efektivniji algoritam planiranja puta.

Na sljedećim slikama, *6.1.-6.6.,* prikazano je radno okruženje u raličitim koracima izvođenja pokusa te primjeri uspješnih i neuspješnih pokusa. Uz prikaze radnog orkuženja, posljednja slika, *6.7.* prikazuje izgled radnog okruženja na osobnom računalu te vizualizaciju hvatišta izračunatim DexNet-om.

Slika 6.1. *Prikaz radnog okruženja prilikom izvođenja pokusa*

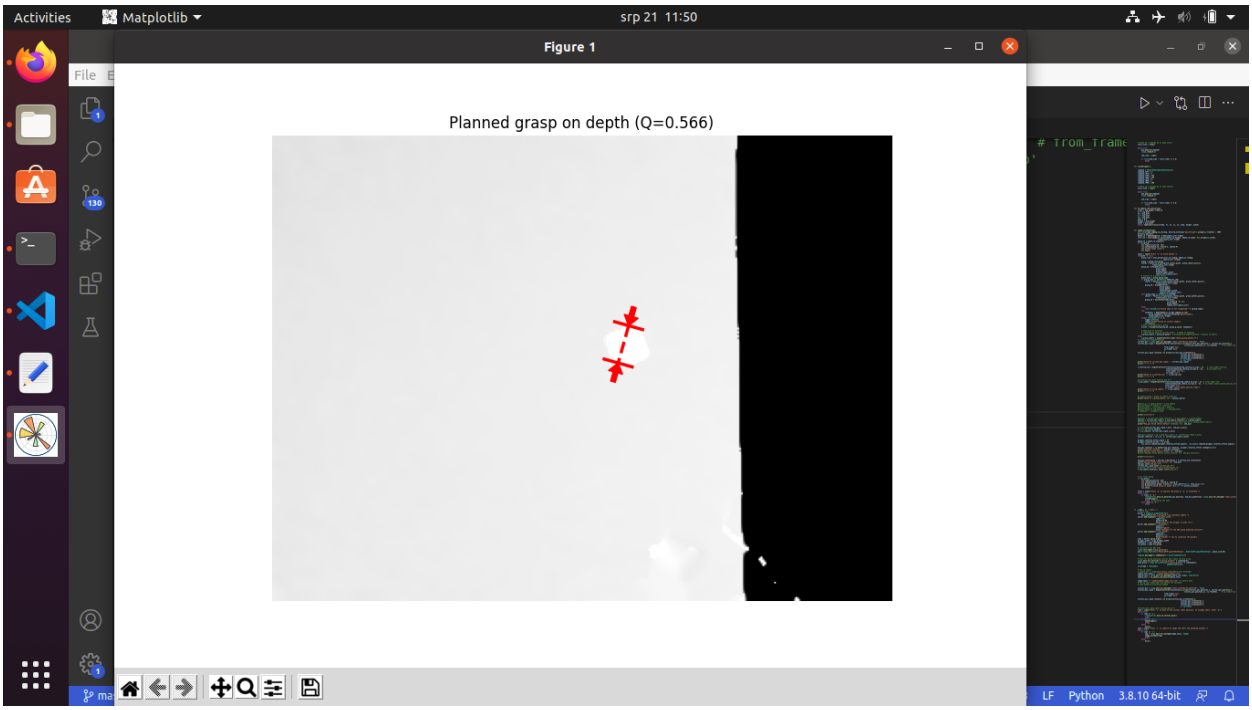
Slika 6.2. *Inicijalni položaj robotskog manipulatora pri skeniranju scene*

Slika 6.3. *Prikaz uspješno izvedenog pokusa*

Slika 6.4. *Prikaz neuspješno izvedenog pokusa*

Slika 6.5. *Prikaz neuspješno izvedenog pokusa*

Slika 6.6. *Prikaz uspješno izvedenog pokusa prije podizanja objekta*



Slika 6.7. *Prikaz izračunatog hvatišta koje je rezultiralo neuspješnim izvođenjem pokusa*

7. ZAKLJUČAK

Iako rezultati nisu u potpunosti pokazali da je Dex-Net dobro rješenje za realni sektor, ovaj primjer u njegovoj ograničenoj izvedbi daje dovoljno dobru smjernicu i motiv za daljnje istraživanje ove metode. Za očekivati je da bi se uz poneke adaptacije neuronske mreže i postupka njezinog učenja, fleksibilnost primjene u industriji povećala te bi se tako napravio iskorak prema univerzalnoj metodi za hvatanje potpuno novih objekata. Ograničenost raspoloživim resursima i vremenom pri postavljanju radnog okruženja u ovom radu, znatno je utjecalo na rezultate samih pokusa.

Kao najveći problem pri izvršavanju eksperimenata pokazala se osjetljivost držača kamere na samom robotskom manipulatoru. Mala odstupanja u rotaciji stalka i od par stupnjeva rezultirala su greškama i do par centimetara u konačnom izračunu položaja objekta. Svako izvođenje eksperimenta uzrokovalo je male pomake u položaju kamere, ali dovoljno velike da eksperiment ne uspije. Alternativa trenutnom rješenju bila bi postavljanje kamere na fiksnu poziciju iznad radne površine, ali bi to zahtijevalo modifikaciju radnog okruženja postavljanjem stalka dovoljno stabilnog da vanjski utjecaji ne interferiraju s položajem kamere. Druga alternativa je stabilniji stalak na samom robotskom manipulatoru koji bi osigurao isti položaj kamere kroz trajanje svih eksperimenata, ali da je pri tome omogućeno dovoljno precizno mjerenje same rotacije i translacije kamere u odnosu na vrh alata. S obzirom na trenutna iskustva, minimalno milimetarska odstupanja bi rezultirala uspješnim izvršavanjem eksperimenta.

Ograničenost na jedan objekt pri testiranju je rezultat nedostatka objekata unutar granice 5 centimetara širine hvataljke, a da nisu iznimno reflektivni. Nakon osiguravanja stabilnosti kamere i preciznosti izračuna položaja iste, eksperimenti bi se trebali ponovno izvesti s više objekata, po mogućnosti 3D printanih kako bi odgovarali zahtjevima ciljnog objekta za hvatanje.

Uz industrijski izvedeno radno okruženje te prikladniji skup podataka za učenje na objektima koje pronalazimo u određenim industrijama, DexNet bi mogao naći svoju praktičnu primjenu u industriji.

LITERATURA

[1] BerkleyAutomation, Dex-Net, https://berkeleyautomation.github.io/dex-net

[2] Andrew Miller and Peter K. Allen. „Graspit!: A Versatile Simulator for Robotic Grasping“. IEEE Robotics and Automation Magazine, No.4, V. 11, pp. 110-122, 12. 2004.

[3] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, K. Goldberg, „Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics.“, Proc. Robotics: Science and Systems (RSS), 2017.

[4] J. Mahler, F. T Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kroger, J. Kuffner, K. Goldberg „Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi armed bandit model with correlated rewards“, Proc. IEEE Int. Conf. Robotics and Automation (ICRA), pp. 1957-1964 IEEE, 2016.

[5] J. Redmon, A. Angelova, „Real-time grasp detection using convolutional neural networks“, Proc. IEEE Int. Conf. Robotics and Automation (ICRA), pp. 1316–1322, IEEE, 2015.

[6] A. Saxena, J. Driemeyer, A. Y Ng, „Robotic grasping of novel objects using vision“, The International Journal of Robotics Research, pp. 157–173, 2008.

[7] UR5 collaborative robot arm, https://www.universal-robots.com/products/ur5-robot/

[8] LiDAR Camera L515, https://www.intelrealsense.com/lidar-camera-l515/

[9] 3-Finger Adaptive Robot Gripper, https://robotiq.com/products/3-finger-adaptive-robot-gripper

[10] moveit_tutorials Noetic documentation, https://ros-planning.github.io/moveit_tutorials/

[11] Robot Operating System - Wikipedia, https://en.wikipedia.org/wiki/Robot_Operating_System

[12] Documentation – ROS Wiki, http://wiki.ros.org/Documentation

[13] ROS Wrapper for Intel® RealSense™ Devices, https://github.com/IntelRealSense/realsense-ros

[14] GitHub - Universal_Robots_ROS_Driver, https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

[15] Moveit1 Source Build: Linux, https://moveit.ros.org/install/source

[16] GitHub - jr-robotics/robotiq: Robotiq packages, https://github.com/jr-robotics/robotiq

[17] OpenCV, https://opencv.org/

[18] GitHub - BerkeleyAutomation/autolab_core: Core utilities for Berkeley AutoLab., https://github.com/BerkeleyAutomation/autolab_core

SAŽETAK

U ovom diplomskom radu, eksperimentalno su ispitane mogućnosti primjene neuronske mreže Dex-Net, koja određuje prikladno mjesto na površini predmeta za hvatanje od strane robota na osnovu fizičkih obilježja objekta koristeći računalni vid, s primjenom u robotskoj manipulaciji. Sva programska podrška izvedena je u sustavu ROS te primijenjena u stvarnom okruženju. Robotski sustav korišten u eksperimentalnoj fazi ovog rada sastoji se od UR5 robotskog manipulatora, Intel RealSense L515 kamere, Robotiq hvataljke s 3 prsta te osobnog prijenosnog računala.

Ključne riječi: Dex-Net, robotska manipulacija, računalni vid, ROS

ABSTRACT

In this thesis, the possibilities of using the neural network Dex-Net, which determines suitable grasps based on the physical characteristics of the object using computer vision, with application in robotic manipulation, were experimentally investigated. All software was implemented using ROS and was applied in a real environment. Robotic system used in the experimental phase of this thesis consists of a UR5 robotic manipulator, an Intel RealSense L515 camera, a Robotiq 3-finger gripper and a personal laptop.

Keywords: Dex-Net, robotic manipulation, computer vision, ROS