Planiranje hvatanja i robotska manipulacija

Vučković, Jurica

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Jurica Vučković

Zagreb, 2022. godina.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

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

Zagreb, 2022. godina.

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

Zahvaljujem se mentoru doc. dr. sc. Marku Švaci na zadavanju teme i pruženoj pomoći tijekom izrade rada. Također se zahvaljujem kolegama Branimiru Ćaranu i Janu Jakovljeviću na usmjerenju u temu.

Veliko hvala obitelji, prijateljima i curi Eni na podršci tijekom studija i izrade završnog rada.

Jurica Vučković

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

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

ZAVRŠNI ZADATAK

Student: **Jurica Vučković** JMBAG: **0035219927**

Naslov rada na hrvatskom jeziku: **Planiranje hvatanja i robotska manipulacija**

Naslov rada na engleskom jeziku: **Grasp planning and robotic manipulation**

Opis zadatka:

Fizička interakcija robota s okolinom te mogućnost istraživanja okoline (eng. „exploration“) jedan je od osnovnih preduvjeta za inteligentno ponašanje robota. Današnje napredne robotske sustave moguće je opremiti s višeosnim i antropomorfnim hvataljkama (mehaničke i robotske šake) inspirirane ljudskim šakama. Na taj način robote je moguće uključiti u rad u okolini koja je orijentirana čovjeku a ne strojevima bez velike prilagodbe. Iako danas postoje različite istraživačke ali i komercijalno dostupne robotske šake (npr. „Shadow hand“, „Schunk SVH“, itd.) njihovo upravljanje s ciljem robusnog hvatanja i manipulacije predmetima različitog oblika i karakteristika još uvijek predstavlja velik izazov. S ciljem upoznavanja s različitim metodama i principima planiranja hvatanja (eng. „grasp planning“) koristeći robotske hvataljke i robotske šake potrebno je proučiti postojeća dostupna softverska rješenja te odabrati jednu od dostupnih platformi („GraspIt“, „ROS“, „Isaac“ i sl.).

U završnom radu potrebno je:

- Odabrati barem jednu postojeću metodu hvatanja te ju implementirati u scenariju fizikalne simulacije manipulacije koristeći dostupni fizikalni simulator,
- Dati kritički osvrt na implementaciju i dobivene rezultate odabrane metode hvatanja,
- Prema dobivenim rezultatima predložiti vlastitu metodu planiranja hvatanja poznatih i/ili nepoznatih predmeta koristeći proizvoljnu vrstu robotske hvataljke.

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

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2021.	**1. rok:** 24. 2. 2022. **2. rok (izvanredni):** 6. 7. 2022. **3. rok:** 22. 9. 2022.	**1. rok:** 28. 2. – 4. 3. 2022. **2. rok (izvanredni):** 8. 7. 2022. **3. rok:** 26. 9. – 30. 9. 2022.

Zadatak zadao:
Doc. dr. sc. Marko Švaco

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA I KRATICA

Kratica	**Puni naziv na izvornom jeziku**	**Tumačenje na hrvatskom jeziku**
GPD	*engl. Grasp Pose Detection*	identifikacija poze hvatanja
ROS	*engl. Robot Operating System*	robotski operativni sustav
SE(3)	*engl. Special Euclidean Group SE(3) ili engl. The Lie Group SE(3)*	Specijalna Euklidska grupa
OS	*engl. Operating System*	operativni sustav
PCL	*engl. Point Cloud Library*	biblioteka oblaka točaka
URDF	*engl. Unified Robotic Description Format*	ujedinjeni robotski deskriptivni format
SDF	*engl. Simulation Description Format*	simulacijski deskriptivni format
PID	*engl. Proportional-Integral-Derivative*	proporcionalni integralni derivacijski
CRTA	*hrv. Regionalni centar izvrsnosti za robotske tehnologije*	Regionalni centar izvrsnosti za robotske tehnologije
API	*engl. Application Programming Interface*	aplikacijsko programsko sučelje
RL	*engl. Reinforcement Learning*	podržano učenje
SL	*engl. Supervised Learning*	nadzirano učenje
CNN	*engl. Convolutional Neural Network*	konvolucijska neuronska mreža

SAŽETAK

Problem robotskog univerzalnog hvatanja i manipulacije nepoznatih predmeta već se desetljećima pokušava riješiti, ali unatoč naporima brojnih istraživača još uvijek je neriješen. Zbog kompleksnosti problema i velikog broja poremećaja (npr. greška pozicije kontrole robota, šum u slici kamere) koji se javljaju u procesu hvatanja robotskom rukom, teško je analitički definirati i riješiti problem. Zbog toga se zajednica robotičara okrenula na metode strojnog učenja za rješenje problema hvatanja.

U sklopu ovog završnog rada izabrana je metoda pronalaženja prostornog položaja hvata na nepoznatim objektima zasnovana na strojnom učenju zvana GPD *(engl. Grasp Pose Detection)*. Izabrana metoda je implementirana na robotski sustav razvijen u simulacijskom okruženju. U radu je detaljno objašnjena konfiguracija i postavljanje svih programa i paketa korištenih za obavljanje zadatka hvatanja nepoznatih objekta koristeći robotsku ruku. Robotski sustav u sklopu ovog završnog rada je razvijen koristeći robotski operativni sustav (ROS) za komunikaciju i povezivanje njegovih funkcionalnih elemenata.

Na kraju rada se objašnjava razvijeni upravljački program koji rezultate paketa GPD koristi za uspješno izuzimanje predmeta u simulacijskom okruženju, na temelju dubinske slike tih predmeta pomoću simulirane kamere. Upravljački kod je sinteza svega naučenog i implementiranog u ovome završnom radu.

Ključne riječi: ROS, GPD, *MoveIt*, *Gazebo*, hvatanje, robotska manipulacija, strojno učenje

SUMMARY

Universal robotic grasping and manipulation is still an unsolved problem in robotics, despite the efforts of researchers in the field which span decades. Because of the complexity of the problem and a lot of unknown variables that can disrupt the grasping process (ex. positional errors of robot arm control, noise in the camera image) of a robotic arm, it is hard to analytically define and solve the problem. Because of that, the robotics community turned to grasping methods based on machine learning algorithms to try and solve the problem.

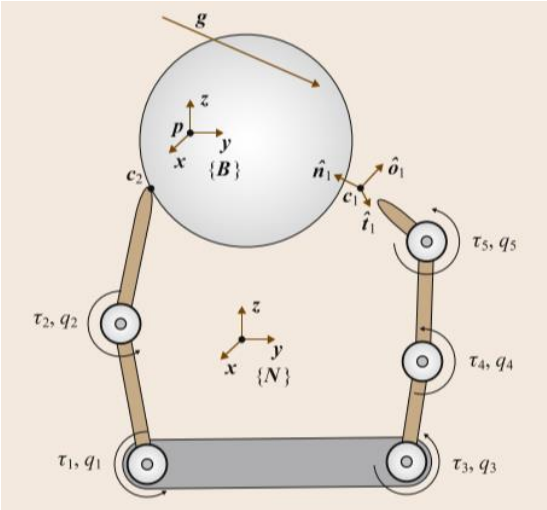
In this undergraduate thesis an approach of finding grasp poses on unknown objects using machine learning techniques called GPD *(Grasp Pose Detection)* was chosen. The chosen method was implemented on a robotic system developed inside a simulation environment. This thesis explains the configuration and setup of all the used programs and packages for the task of grasping unknown objects using a robotic arm. The robotic system developed for the purposes of this thesis uses the Robot Operating System (ROS) for connections and communication between its functional elements.

In the end of this thesis a developed program that uses the results from the GPD package for successful picking of objects inside a simulation environment is explained. This code is the synthesis of everything learned and implemented inside this undergraduate thesis.

Key words: ROS, GPD, *MoveIt*, *Gazebo*, grasping, robot manipulation, machine learning

1. UVOD

Ljudima je proces manipulacije i hvatanja predmeta trivijalan. Bebe u prvih šest mjeseci života počinju hvatati i uzimati stvari iz okoline i započinju proces učenja, a već između devetog i dvanaestog mjeseca života im se dovoljno razvila koordinacija ruku i očiju *(engl. hand-eye coordination)* da im radnja postaje lagana [1]. Za robote je taj zadatak iznimno težak te unatoč naporima i interesu znanstvene zajednice i industrije posljednjih desetljeća i dalje je zadatak univerzalnog hvatanja nepoznatih predmeta neriješen [2].



Slika 1.1 Analiza hvatanja [4]

Slika 1.2 „Farma robota“ - *Google* [5]

Istraživači u području robotskog hvatanja su najprije pristupali problemu koristeći analitičke metode zasnivane na prvim principima iz fizike. Koristili su se brojni modeli, često zasnivani na teoriji vijaka *(engl. screw theory)* koja se javila u kinematici i dinamici krutih tijela [3]. Na slici 1.1 prikazane su glavne vrijednosti i parametri koji se koriste u analizi hvatanja. Iako su analitički modeli fizikalno zasnivani i matematički vrlo intenzivni, rezultati koje proizvode su razočaravajući za estimaciju poza hvatanja kod kompleksnijih predmeta i neuređenu okolinu. Glavni razlog loših rezultata modela su inherentne pogreške u samim modelima (primjerice kaos u trenju zbog nelinearnosti pojave), poremećaji u senzorima i greške u kontroli robota.

Zbog loših rezultata, istraživači i zajednica robotičara se okrenula pristupu zasnivanom na strojnom učenju. Metode strojnog učenja se rapidno razvijaju zbog velikog interesa istraživača u računalnoj znanosti. Interes je nastao zbog potrebe da se iz velike količine podataka koje prikupljaju tehnološke kompanije ekstrahiraju korisni uzorci i zaključci. Metode zahtijevaju

veliki broj podataka i primjera kako bi se neuronske mreže naučile davati dobre rezultate za dani problem.

Osim za potrebe analize velike količine podataka, razvijene metode strojnog učenja se mogu koristiti za čitav niz problema u raznim područjima, uključujući i robotiku. Jedan od primjera gdje se koristi strojno učenje u robotici je upravljanje animatroničkih šaka koje su projektirane po uzoru na ljudsku šaku. Zbog velikog broja stupnjeva slobode gibanja u šaci je nemoguće analitički opisati njenu dinamiku i kinematiku pa se za taj zadatak koriste metode strojnog učenja.

Za problem hvatanja se također koristi strojno učenje, a podatci potrebni za treniranje neuronskih mreža mogu se sakupljati unutar simulacije ili koristeći prave robotske ruke. Slika 1.2 prikazuje poznatu „farmu robota“ u *Google*-u, gdje je tim istraživača koristio 14 robotskih ruku koje su u svrhu prikupljanja podataka hvatale predmete godinu dana. Na posljetku je rezultat bio 800.000 sakupljenih podataka hvatanja u stvarnom svijetu [5]. Ovakav pristup je najbliži stvarnim situacijama, ali je izuzetno skup, a budući da većina istraživačkih timova nema resursa kao *Google*, podatci se češće prikupljaju unutar simulacija.

U sklopu ovog završnog rada će se opisati dostupne metode hvatanja i njihov razvoj kroz povijest te će se izabrati i implementirati jedna metoda hvatanja u fizikalnoj simulaciji. Fizikalni simulatori su dobri za razvoj i isprobavanje algoritama u sigurnom okruženju prije korištenja istih na pravome robotskom sustavu.

2. RAZVOJ METODA HVATANJA U ROBOTICI

Robotsko hvatanje je od početka robotike kao grane inženjerske znanosti bila aktivna tema istraživanja i razvoja, budući da je hvatanje jedna od temeljnih, ali i najizazovnijih vještina robota. Hvatanje zahtijeva istovremenu koordinaciju kontrole, percepcije i planiranja robotskog sustava. Usprkos interesu i brojnim znanstvenim radovima na temu hvatanja i dalje su sposobnosti hvatanja najboljih razvijenih algoritama znatno lošije od ljudskih mogućnosti, pogotovo kada je riječ o nestrukturiranoj okolini [3].

U ovom poglavlju dati će se kratki osvrt povijesti razvoja metoda hvatanja u robotici i neka trenutna dostignuća u području. Najprije se daje osvrt na analitičke metode hvatanja, a zatim na metode koje se koriste strojnim učenjem.

2.1. Formulacija problema

Pronalaženje poza hvatanja na objektu *(engl. grasp synthesis)* znači pronalazak odgovarajućih konfiguracija aktuatora robota, povezanih s trenutnim stanjem objekta hvatanja, koje rezultiraju stabilnim hvatom [3]. Ovaj problem se može definirati na više načina, gdje se klasične formulacije fokusiraju na mehanička svojstva predmeta hvatanja, a moderni pristupi na vizualna svojstva slike istog.

Prije nego se krenu opisivati same metode hvatanja, potrebno je opisati načine reprezentacije hvata *(engl. grasp representation)*.

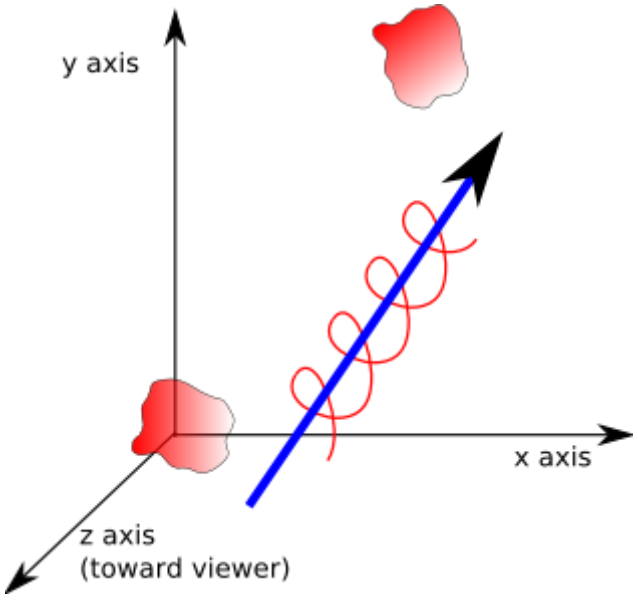
2.1.1. Reprezentacije hvata

Iako postoje i drugi načini reprezentacije hvata, u sklopu ovog poglavlja će se definirati samo dva koja se koriste u opisanim metodama hvatanja. Riječ je o reprezentaciji hvata pomoću kontaktnih točaka i 6D reprezentaciji hvata.

2.1.1.1. Reprezentacija kontaktnim točkama

Ova reprezentacija se koristi najčešće kod analitičkih metoda hvatanja. Zasnivana je na *teoriji vijaka (engl. screw theory)*, čiji će temeljni pojmovi biti ukratko opisani kako bi se mogla definirati ova reprezentacija hvata.

Teorija vijaka predstavlja prostorni pomak krutog tijela *(engl. screw displacement)* pomoću rotacije i translacije oko definiranog pravca u prostoru. Izometrijske transformacije krutog tijela u prostoru se mogu u potpunosti definirati koristeći jednu rotaciju oko pravca u prostoru, uz određeni smjer rotacije i korak *(engl. pitch)* zavojnice oko tog pravca [6]. Ako je korak jednak nuli tada se objekt samo rotira, a ako teži u beskonačnost onda se samo translatira u prostoru. Takav princip je analogan principu navoja kod vijaka pa je po tome i dodijeljeno ime teoriji. Na slici ispod se nalazi primjer takve transformacije.



Slika 2.1 Translacija i rotacija krutog tijela - *teorija vijaka* [6]

Vijak (engl. screw) je 6-dimenzionalan vektor koji se sastoji od para 3-dimenzionalnih vektora (primjerice sila i momenta ili linearne i kutne brzine) koji se javljaju tijekom promatranja kretanja krutih tijela u prostoru [4]. Komponente ovog vektora definiraju Plücker koordinate pravca (način opisivanja pravca u prostoru koristeći šest homogenih koordinata) i norme vektora pomaka po duljini pravca i momenta oko tog pravca [6].

Ključ (engl. wrench) je naziv vijka koji sadrži vektore sile i momenta. Budući da sila ima točku primjene i pravac gdje djeluje, njezine Plücker koordinate opisuju pravac u prostoru, a norma vektora pomaka po pravcu je nula. Moment nije vezan za pravac pa je vijak s beskonačnim korakom. Omjer normi vektora sile i momenta definira ukupan korak *vijka* [6].

Nakon definiranih pojmova iz teorije vijaka se napokon može navesti definicija reprezentacije kontaktnim točkama. Formalno, za skup kontaktnih točki $\mathbf{p} = \{p_i\}_{i=1}^{N}$, jedan *ključ (engl. wrench)* $\omega_i = (f_i, \tau_i)$ je nametnut na svaku odgovarajuću kontaktnu točku [3]. Gdje je f_i primijenjena sila na točki p_i, a τ_i je moment oko površinske normale kontaktne točke.

Napokon, hvat je definiran kao $g_j = (\omega_1, \omega_2, \ldots, \omega_N)$, gdje je j ∈ [1 ... N], a svi *ključevi*, točke i hvatovi su definirani u referentnom koordinatnom sustavu objekta hvatanja.

Ako se na objektu primijenjuje neki vanjski *ključ* ω_e uvjet da hvat bude u ravnoteži je da je suma svih *ključeva* hvata jednaka nametnutom vanjskom ključu ω_e.

Ovakva reprezentacija je široko korištena u analitičkim metodama hvatanja, a koristila se i u početcima u metodama hvatanja korištenjem strojnog učenja [3].

2.1.1.2. 6D reprezentacija hvata

Zbog raširenosti dvoprstih paralelnih hvataljki, moguće je koristiti pojednostavljene reprezentacije hvata. Budući da je kinematika dvoprstih paralelnih hvataljki jednostavna, kontaktne točke na objektu hvatanja su u potpunosti određene 6D pozom hvataljke (točka i orijentacija u prostoru). Poza hvataljke je definirana kao $g = (x, y, z, r_x, r_y, r_z) \in \mathrm{SE}(3)$.

SE(3) je specijalna Euklidska grupa, odnosno grupa homogenih transformacijskih matrica u $\mathbb{R}^3$. Drugim riječima, to je skup svih 4 x 4 realnih matrica oblika:

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

gdje je R matrica orijentacija koordinatnih osi transformacije, a p vektor položaja transformacije (s obzirom na neki referentni koordinatni sustav) [7].

6D reprezentacija hvata je česta kod metoda hvatanja koje koriste 3D percepciju i strojno učenje [3].

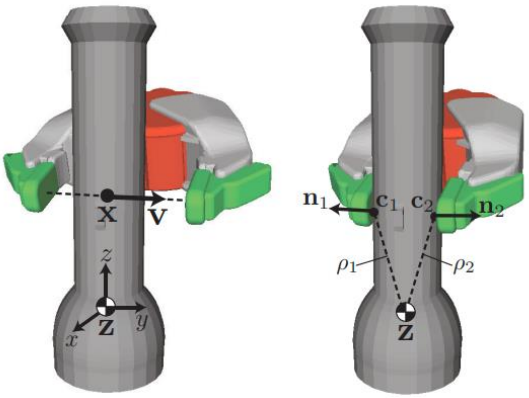
2.2. Analitičke metode

Kao što je već spomenuto, analitičke metode se uglavnom oslanjaju na principe iz mehanike tijela. Reprezentacija hvatova koju koriste je skup kontaktnih točaka i odgovarajućih *ključeva* koji na tim točkama djeluju. Obično se razlikuju tri kontaktna modela: kontakt bez trenja, kontakt s trenjem i kontakt "mekih prstiju" *(engl. soft-finger)*. Prva dva kontaktna modela su najviše zastupljena u robotici, dok je područje meke robotike *(engl. soft robotics)* još relativno novo i nije široko zastupljeno pa tako nema dovoljno radova na temu s takvim kontaktnim modelom [3].

Rani znanstveni radovi iz područja su promatrali problem pronalaženja poza hvatanja na geometrijski primitivnim tijelima (npr. cilindar, kocka, kvadar). Takvi modeli su obično bili zasnivani na *grasp-closure* značajki hvata. *Grasp-closure* je termin koji se uveo za evaluaciju kvalitete hvata, a on označava može li se ili ne hvat na objektu oduprijeti bilo kojem eksternom poremećajnom *ključu.* Iako su te metode teoretski optimalne i valjane, u praksi su se oslanjale na pojednostavljenja kontaktnih modela i geometrije objekata, što uvelike ograničava njihovu primjenu u stvarnom svijetu [3].

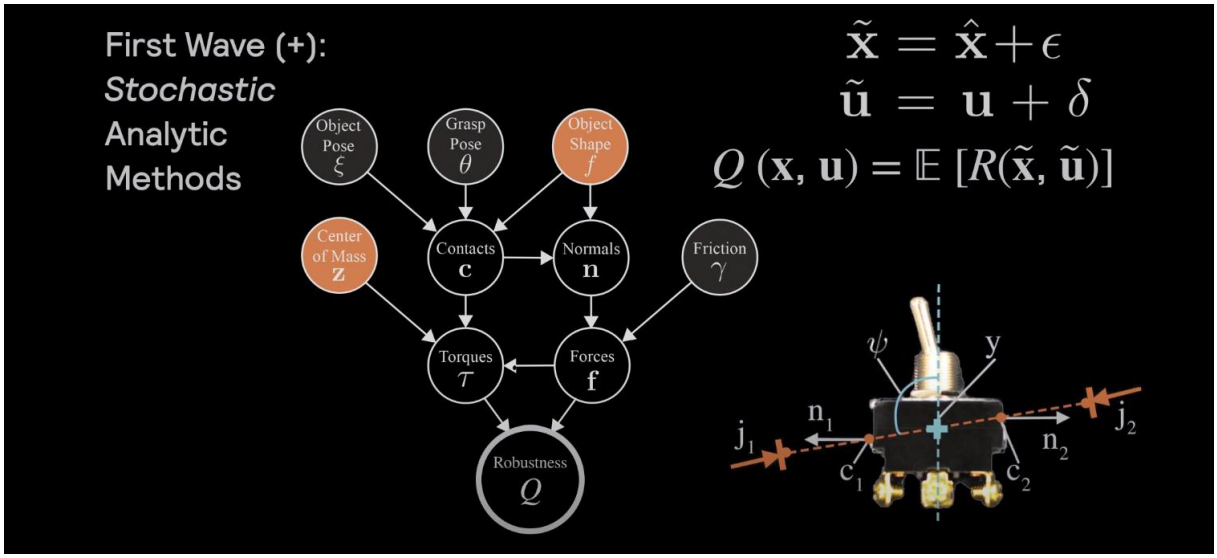
Budući da se problemi analitičkih metoda već javljaju na najjednostavnijim geometrijskim oblicima, ove metode ne djeluju obećavajuće za pronalaženje hvatova na nepoznatim kompleksnim objektima. Ipak, niz timova istraživača je pokušalo razviti algoritme hvatanja za nepoznate objekte analitičkim pristupom problemu.

Jedan od takvih primjera je *Dex-Net 1.0* razvijen na sveučilištu *Berkeley* 2016. godine. Cilj *Dex-Net 1.0* algoritma je pronaći hvat g* koji maksimizira *grasp-closure* kao mjerilo kvalitete hvata, s obzirom na nesigurnost u estimaciji stanja objekta, okruženja i robota [8].



Slika 2.2 Reprezentacija hvatova u *Dex-Net 1.0* [8]

Hvatovi se reprezentiraju kontaktnim točkama kao što prikazuje slika 2.2, a kontaktni model je s trenjem. Na slici 2.3 se prikazuju izvori nesigurnosti u modelu koji koristi *Dex-Net 1.0.* Istraživači su pretpostavili da je oblik objekta hvatanja nesiguran, da je nesigurna vrijednost koeficijenta trenja u kontaktnoj točki i nesigurna lokacija težišta. Zbog svega toga, robusnost odnosno mjera kvalitete hvata pada. Način na koji su tu inherentnu nesigurnost istraživači htjeli kompenzirati je koristeći Monte Carlo integraciju (tehnika numeričke integracije slučajnih varijabli), pretpostavljajući da su nesigurne vrijednosti slučajne varijable. S tako pretpostavljenim stanjem predmeta, uspoređuju ga s bazom *3D CAD modela* i na temelju sličnosti pretpostavljaju poklapanje geometrije promatranog tijela s modelom iz baze podataka.



Slika 2.3 Nesigurnost u modelu - Dex-Net 1.0 [9]

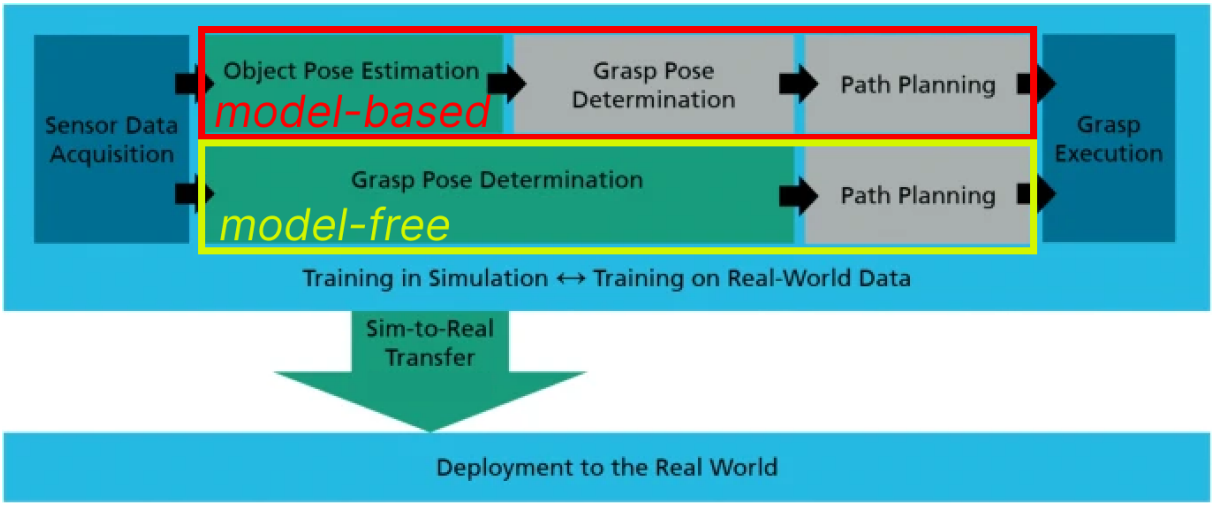
Rezultati koje daje *Dex-Net 1.0* su prema zaključku autora loši, gdje smatraju da je razlog taj što su ignorirali nesigurnost u percepciji objekta. Pretpostavka s kojom su kreirali algoritam je da promatraju stanje objekta, a zapravo promatraju stanje slike objekta [9]. Zbog toga su u sljedećim iteracijama *Dex-Net* algoritama koristili direktnu snimku objekta za pronalazak poza hvatanja umjesto korištenja snimke za estimaciju poza i stanja objekta. Sve iduće iteracije *Dex-Net* algoritma koriste metode strojnog učenja za pronalazak poza hvatanja te se napušta analitički pristup [9].

Analitički pristup se dakle koristio u početcima područja za hvatanje primitivnih geometrijskih tijela, a pokušavalo se primijeniti principe i na kompleksnije geometrije, ali neuspješno. U nastavku slijedi opis iduće etape istraživanja hvatanja pomoću metoda strojnog učenja *(engl. data-driven approach)*.

2.3. Metode strojnog učenja

Zbog neuspješnosti analitičkih metoda, zajednica se u posljednjim godinama okrenula metodama koje koriste algoritme strojnog učenja. Osim zbog neuspješnosti analitičkih metoda, razlog zašto se raširio ovaj pristup je zbog veće pristupačnosti velikim bazama podataka (nužno za učenje mreža), jačoj procesnoj moći računala i poboljšanja u samim algoritmima strojnog učenja. Pristupi koji se koriste za ostvarivanje zadatka hvatanja su razni, gdje se najčešće dijele na pristupe koji koriste modele objekta *(engl. model-based approaches)* i pristupe koji ne

koriste modele objekta *(engl. model-free approaches)* [10]. Na slici 2.4 prikazan je proces hvatanja za oba modela.



Slika 2.4 Proces hvatanja za pristupe s estimacijom modela i bez estimacije [10]

Može se primijetiti da se pristupi razlikuju jedino u dodatnom koraku estimacije poze objekta hvatanja. *Model-based* pristupi u tom koraku pokušavaju identificirati poznati objekt i smjestiti ga u prostor te na temelju toga pronaći poze hvatanja, dok *model-free* pristupi direktno iz slike senzora pokušavaju pronaći poze hvatanja na objektu [10]. Lako je zaključiti da *model-free* pristupi omogućuju generalizaciju na nove i nepoznate objekte, dok *model-based* ostaju ograničeni na objekte za koje postoji dostupan *CAD model* u bazi podataka algoritma. Budući da je cilj ovog rada implementirati metodu koja može hvatati nepoznate objekte, promatrat će se samo metode koje koriste *model-free* pristup.

Dodatni kriterij po kojem se često dijele modeli je tip korištenog strojnog učenja, odnosno je li sustav treniran koristeći nadzirano učenje *(engl. supervised learning - SL)* ili podržano učenje (engl. reinforcement learning - RL). *SL* algoritmi zahtijevaju veliku količinu označenih *(engl. labeled)* podataka učenja, a *RL* algoritmi ne trebaju označene skupove podataka već sami označuju korištene podatke koristeći razne metode [10].

U tablici ispod navedene su neke metode hvatanja predmeta uz kategorizaciju prema opisanim kriterijima i nekim dodatnim kriterijima koji nisu opisani, ali su jasni bez dodatnog pojašnjenja.

Tablica 1 Pregled nekih metoda hvatanja koje koriste algoritme strojnog učenja [10, 11]

Naziv	**Pristup**	**Poznati/ nepoznati predmeti?**	**Tip strojnog učenja**	**Treniranje u stvarnom svijetu ili simulacija?**	**Hvatanje u gustom neredu?**	**Tip ulaznog podatka (senzor)**	**Uspješnost hvatanja**
TossingBot	*model-free*	nepoznati	*SL*	oboje	da	RGB-D slika	64-76%
Dex-Net 2.0	*model-free*	nepoznati	*SL*	simulacija	da	dubinska slika	do 98%
GPD	*model-free*	nepoznati	*SL*	stvarni svijet	da	oblak točaka	93%
QT-Opt	*model-free*	nepoznati	*RL*	stvarni svijet	da	RGB slika	76-96%

U tablici je prikazan osvrt na četiri metode hvatanja. Svaki promatrani pristup je *model-free* zbog uvjeta generalizacije na nepoznate objekte. Može se vidjeti da se gotovo svi modeli poklapaju po većini promatranih kategorizacija, no ipak se uspješnost hvatanja algoritama znatno razlikuje. Najveću uspješnost postiže *Dex-Net 2.0*, koji je iduća iteracija opisanog analitičkog *Dex-Net 1.0* modela. Nažalost, *Dex-Net* nije javno dostupan te ga zbog toga nije moguće primijeniti u sklopu ovog rada.

Iduća najbolja metoda po uspješnosti hvatanja je GPD *(engl. Grasp Pose Detection)*. GPD postiže (relativno) visoku uspješnost hvatanja i javno je dostupan te je zbog toga izabran za implementaciju na robotski sustav koji će se razviti u sklopu ovog rada. U nastavku slijedi opis GPD algoritma hvatanja.

2.4. GPD (engl. Grasp Pose Detection)

Cilj autora rada je bio razviti metodu hvatanja koja poboljšava rezultate na nepoznatim objektima (u usporedbi s drugim znanstvenim radovima na temu). Dodatno, razvijena metoda treba raditi bez precizne segmentacije objekta iz okoline i mora stvarati hvatove na bilo kojoj snimljenoj vidljivoj površini objekta.

Za postizanje navedenih ciljeva, razvijena metoda koristi prethodno spomenuti *model-free* pristup i *6D reprezentaciju hvata* (pretpostavka korištenja dvoprste hvataljke) [11]. Za označavanje kvalitete hvata, koristi se *grasp-closure* opisan u poglavlju 2.2. GPD algoritam prati korake navedene u tablici ispod.

Tablica 2 *Grasp Pose Detection* algoritam [11]

Algoritam *Grasp Pose Detection*
Ulaz: oblak točaka, $\mathbb{C}$; područje pretraživanja hvatova, $\mathscr{R}$; geometrija korištene hvataljke, Θ; pozitivan cijeli broj, N
Izlaz: skup 6D kandidata hvatanja (u području pretraživanja hvatova)
1. $\mathbb{C}'$ = predprocesiranjeOblakaTočaka ($\mathbb{C}$)
2. $\mathscr{R}$ = područjePretraživanjaHvatova ($\mathbb{C}'$)
3. S = uzimanjePrimjeraHvatova ($\mathbb{C}'$, $\mathscr{R}$, Θ, N)
4. I = pretvorbaVišeKanala (S, $\mathbb{C}'$, $\mathscr{R}$, Θ)
5. H = rangiranje (I)
6. g = odabirHvata (S, H)

Ulaz u GPD algoritam je oblak točaka, područje pretraživanja hvatova (ovisno o koordinatnom sustavu kamere), geometrija korištene hvataljke i pozitivan cijeli broj N koji označava koliko se kandidata hvatanja generira u području pretraživanja hvatova. Izlaz algoritma je skup 6D kandidata hvatanja.

Prvi korak algoritma vrši pred-procesiranje ulaznog oblaka točaka da se smanji količina buke i grešaka u snimljenoj slici. Oblak točaka se kompresira, miču se statistički izuzetci i vrše se ostali standardni koraci smanjivanja grešaka snimljene slike [11].

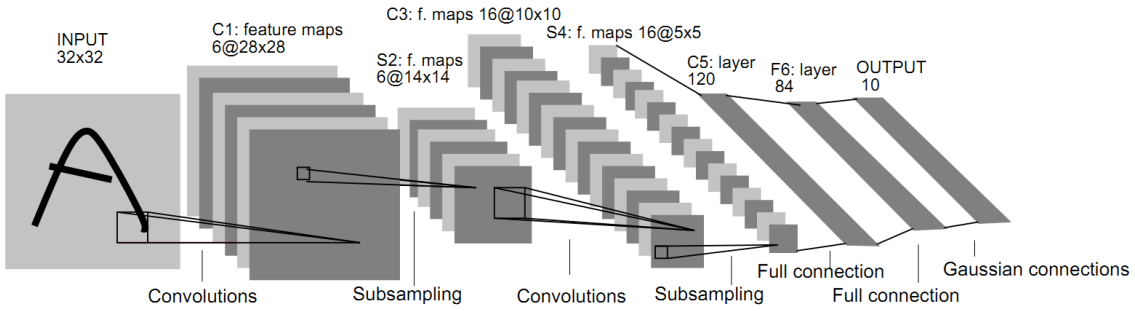
Drugi korak algoritma identificira područje interesa odnosno pretraživanja hvatova u prethodno procesiranom oblaku točaka. Bitno je naglasiti da ovaj korak ne znači nužno segmentaciju objekta hvatanja od njegove pozadine, iako i taj korak može biti uključen u algoritam [11].

Treći korak algoritma kreira kandidate za hvatanje, gdje je svaki kandidat 6D poza hvataljke. Algoritam nasumično uzima N uzoraka iz područja oblaka točaka koje je u definiranom području interesa hvatova $\mathscr{R}$. Zatim se za to područje računa referentni koordinatni sustav koristeći svojstvene vektore i provodi se lokalno pretraživanje područja za hvatove koji zadovoljavaju minimalne kriterije mogućih hvatova prema [11]. Slika 2.5 prikazuje rezultate ovog koraka algoritma u obliku pronađenih mogućih hvatova na oblaku točaka.

Slika 2.5 Pronađeni kandidati za hvatanje objekta - korak 3 GPD algoritma [11]

Četvrti korak algoritma pretvara svakog kandidata za hvatanje i njegovo okolno područje oblaka točaka u više-kanalnu sliku koja će se koristiti kao ulaz u četvero-slojnu konvolucijsku neuronsku mrežu *(engl. Convolutional Neural Network - CNN)* [11].

Peti korak algoritma je prolazak pretvorenih kandidata hvatanja kroz *CNN*. GPD koristi četvero-slojni *CNN* koji ima strukturu kao LeNet-5 prikazan na slici ispod [11].



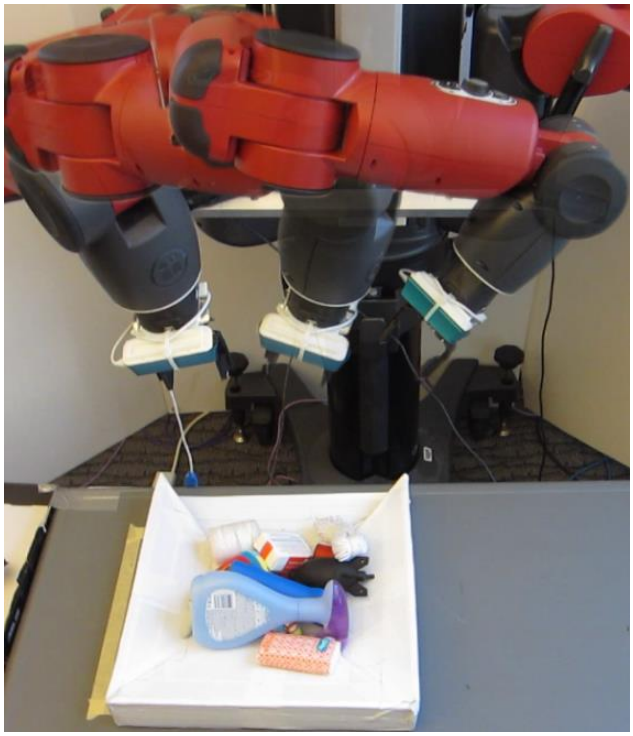
Slika 2.6 Arhitektura LeNet-5 *CNN* [12]

Svi korišteni konvolucijski slojevi GPD-a su isti kao LeNet-5, gdje je razlika jedino u ulazu u *CNN*. GPD mreža je trenirana koristeći stohastički gradijentni spust *(engl. stohastic gradient descent)* s faktorom učenja 0,00025 [11].

Šesti i zadnji korak algoritma je odabir najboljih hvatova na objektu koristeći rezultate petog koraka rangiranja i dodatne uvjete postavljene ulazom u algoritam [11].

Za treniranje *CNN*-a korištenog u GPD-u autori su koristili 1,5 milijuna primjeraka izvučenih iz BigBird skupa podataka [11]. BigBird skup podataka je u vrijeme pisanja rada GPD-a sadržavao 125 primjera snimljenih oblaka točaka objekta. Autori su iz tog skupa podataka uzeli 55 objekta te su za svaki od tih objekta kreirali dodatne oblake točaka za različite smjerova gledanja predmeta. Na kraju su generirali oko 50 tisuća označenih podataka (s obzirom na *grasp-closure*) za svaki objekt [11].

Na kraju rada su autori razvijeni algoritam testirali na pravom hardveru. Kao eksperimentalni postav korištena je robotska ruka sa sedam stupnjeva slobode zvana *Baxter Research Robot* s paralelnom dvoprstom hvataljkom [11]. Za objekte hvatanja su nasumično izabrali 10 predmeta (od ukupno 27 dostupnih) i stavili u kutiju koju su zatim promiješali i stavili ispred robota. Slika 2.7 prikazuje sliku eksperimentalnog postava autora GPD-a.



Slika 2.7 Eksperimentalan postav autora GPD-a [11]

Na tako opisanom eksperimentalnom postavu su izvršili 300 pokušaja hvatanja i postigli ukupnu uspješnost izuzimanja predmeta iz gustog nereda od 93% [11]. Postignuta uspješnost još uvijek nije dovoljna u svrhu praktične primjene algoritma hvatanja, ali predstavlja poboljšanje ako se usporedi s ostalim algoritmima hvatanja koji postižu slične uspješnosti u scenarijima blagog nereda *(engl. light clutter)* ili izoliranih objekata.

Kao što je već spomenuto, ovaj algoritam hvatanja je javno dostupan na *GitHub*-u autora rada i koristit će se za pronalazak poza hvatanja na predmetima snimljenih simuliranom dubinskom kamerom u ovome završnom radu.

3. SIMULACIJSKI POSTAV I KONFIGURACIJA SUSTAVA ZA HVATANJE

U ovom radu će se koristiti ranije opisani paket GPD za detektiranje poza hvatanja na nepoznatim predmetima. Cilj je razviti okruženje u kojem paket radi, daje dobre rezultate i na posljetku da se koristeći dobivene rezultate izradi program upravljanja robota do točke hvatanja. Budući da je GPD integriran s robotskim operativnim sustavom (ROS – *engl. Robot Operating System*; opisan kasnije), odabrani simulator, ali i sve ostale komponente robotskog sustava moraju također raditi unutar ROS-a. U nastavku slijede opisi korištenih programa i paketa te detaljno opisana konfiguracija razvijenog robotskog sustava za potrebe ovog rada.

3.1. Robotski operativni sustav - ROS

Kao temelj sustava koristit će se prethodno spomenuti robotski operativni sustav (ROS). ROS je fleksibilno softversko okruženje koje sadrži razne alate i biblioteke koda za robotske sustave [13]. Pruža nekoliko snažnih značajki (primarno standardizacija poruka) koje omogućuju ponovno korištenje razvijenog koda, dijeljenje biblioteka i implementaciju najnovije razvijenih algoritama korisnika ROS-a diljem svijeta. Iako ROS nije operativni sustav, nego skup softverskih okvira za razvoj robotskog softvera, on pruža usluge poput apstrakcije hardvera, kontrole uređaja, slanja poruka i vođenje paketa što jesu neke od značajki stvarnog operativnog sustava. Za detalje o korištenju ROS-a vidjeti [14].

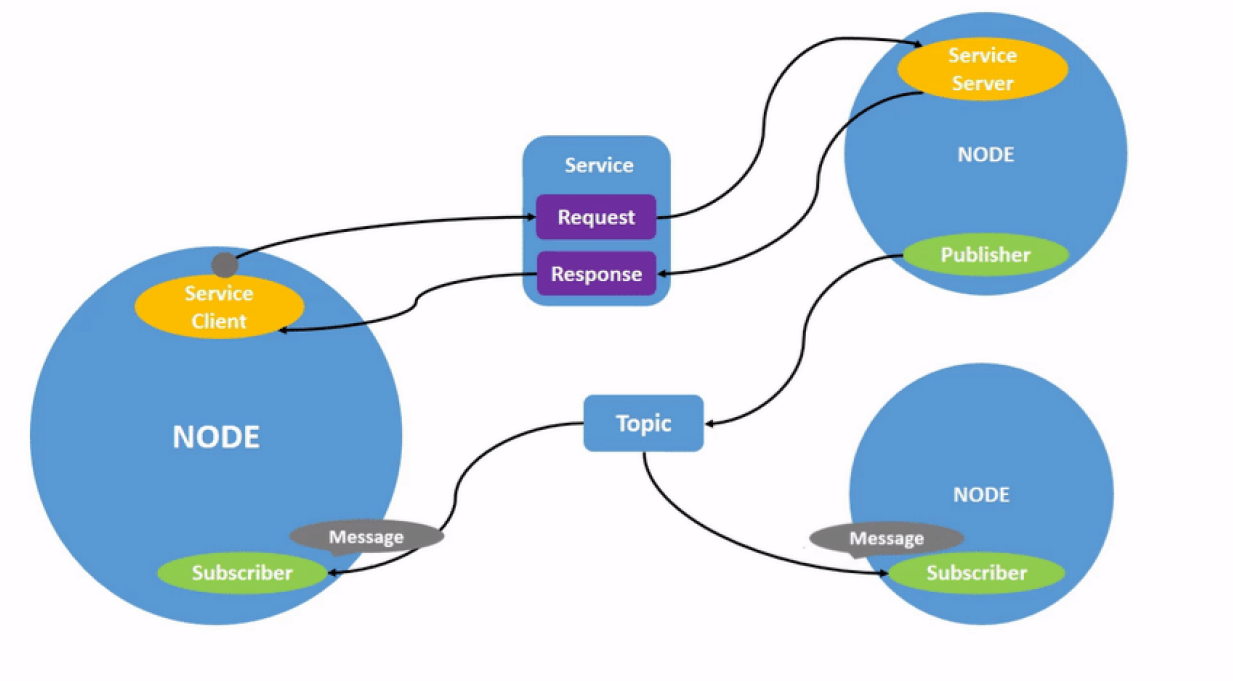
U ROS-u se procesi prikazuju u arhitekturi grafova, gdje se procesiranje podataka odvija u čvor *(engl. node)* elementima koji mogu primati *(engl. subscribe)*, objavljivati *(engl. publish)* i slagati *(engl. multiplex)* poruke iz senzora, aktuatora, kontrolera i drugih sustava unutar robota. Čvorovi također mogu interaktirati koristeći servise *(engl. services)* koji rade na principu pitanje i odgovor. Kako bi se čvorovi povezali u funkcionalnu cjelinu i uopće omogućila njihova komunikacija, potreban je *master*. *Master* funkcionira kao DNS server, gdje se u procesu pokretanja čvor najprije poveže na *master* da bi mogao komunicirati s ostalim čvorovima. Sustav koristi TCP/IP protokol gdje je varijablom *ROS_MASTER_URI* definirana IP adresa *mastera* preko koje se čvorovi spajaju na njega [15].

Unutar ROS okoline se komunikacija između čvorova vrši slanjem poruka *(engl. messages)* na određene teme *(engl. topics)*. Čvorovi mogu primati ili objavljivati poruke iz/na teme. Unutar ROS-a postoji i server koji služi čvorovima za spremanje varijabli zvan *parameter*

server (hrv. server parametara). Spremanje varijabli se vrši koristeći ključ-brava princip, što omogućava čvorovima spremanje, brisanje i modificiranje varijabli spremljenih u serveru. Primjer strukture ROS sustava se nalazi na slici 3.1. Na slici se vide tri čvora. Dva čvora objavljuju poruku na određenu temu dok treći čvor prima poruku. Lijevi čvor ima i funkcionalnost server-klijenta i može slati zahtjeve drugom čvoru i/ili primati odgovor.

Glavna prednost ROS-a je njegova velika i brzo rastuća zajednica. Zbog velike zajednice, modularnosti, hardverske apstrakcije i brojnih drugih značajki, ROS koristi i većina visoko-tehnoloških kompanija u području robotike.

Mana ROS-a je njegova strma krivulja učenja za nove korisnike. Robotičari koji žele razvijati softver u ROS-u prvo moraju naučiti veliki broj novih koncepata da bi se uopće mogli početi služiti alatima unutar okruženja.



Slika 3.1 Jednostavan prikaz graf arhitekture ROS-a [16]

3.2. GPD

Za dobivanje poza hvatanja na nepoznatim objektima, koristit će se GPD, koji je detaljno opisan u prethodnom poglavlju. Čitav kod paketa je javno dostupan na *GitHub*-u autora [17]. U *GitHub* repozitoriju paketa se nalaze instrukcije kako se instalira paket, no te instrukcije su zastarjele i kod kompiliranja paketa se može javiti problem zbog zahtijevanih zavisnih *(engl. dependency)* programa. Zbog toga će se u nastavku poglavlja opisati i instalacija paketa na korištenom operativnom sustavu te konfiguracija parametra korištenih kod pokretanja istoga.

Dodatno, GPD je integriran s ROS okruženjem i ta integracija je također javno dostupna na *GitHub*-u.

3.2.1. GPD-ROS

GPD-ROS je *wrapper* glavnog paketa za korištenje u ROS okruženju. *Wrapper* je termin koji se koristi za čvorove koji služe za spajanje koda koji nije napisan u ROS okruženju s ROS-om. Potrebno je instalirati i ovaj paket za integraciju GPD-a s robotskim sustavom koji se razvija u ovom radu.

3.3. Gazebo simulacijsko okruženje

Za simulaciju robotskog sustava, koristit će se *Gazebo*. *Gazebo* je više robotski fizikalni simulator za kompleksne simulacije u zatvorenom i otvorenom prostoru [13]. U *Gazebo* svijetu se osim robota može simulirati niz senzora i 3D predmeta u okolini. Glavne komponente okruženja su:

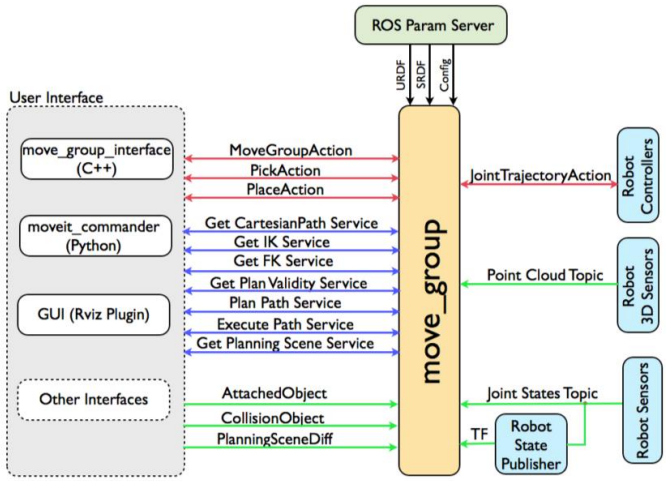
- *datoteke svijeta (engl. world files)* – sadrže sve elemente simulacije, uključujući robote, svijetlo, senzore i statičke objekte
- *modeli (engl. models)* – individualni elementi unutar *Gazebo* svijeta
- *gzserver* – baza *Gazeba*, pokreće fiziku svijeta i informacije senzora
- *gzclient* – grafičko sučelje, vizualizacija simulacije
- *plugin* – dodaju funkcionalnosti u *Gazebo* uz modularnost; *C++* programi koji su neovisni o ROS-u

Primjer *Gazebo* svijeta je prikazan na slici 3.14, koji je kreiran i objašnjen kasnije u radu, a za stvaranje svijeta se koristio izvor [18].

3.4. MoveIt!

Kako bi se cijeli sustav spojio u funkcionalnu cjelinu, potreban je još jedan čvor, odnosno element koji će se pobrinuti za kretanje robotske ruke do identificiranog hvata GPD paketa. Za to će se u ovom radu koristiti *MoveIt!*. *MoveIt!* je skup paketa i alata za mobilne manipulacije robota unutar ROS okruženja. *MoveIt!* sadrži rješenja za provjeravanje kolizija, 3D percepciju, kinematiku, inverznu kinematiku, kontrolu i navigaciju robota.

Paket ima više sučelja: *RViz* plugin za vizualizaciju, *Setup Assistant* za postavljanje novog robota u sustav, upravljanje preko terminala te *C++* i *Python* sučelje korištenjem *move_group* čvora [13]. Radi razumijevanja *MoveIt!*-a, bitno je opisati njegovu arhitekturu, prikazanu na slici ispod.



Slika 3.2 Arhitektura MoveIt! paketa [13]

Može se reći da je *move_group* čvor srce *MoveIt!*-a, budući da on integrira razne komponente robota i pruža servise/akcije ovisno o zahtjevima korisnika. Iz slike se vidi da čvor sakuplja informacije iz ROS okruženja pretplatom na razne teme. Iz ROS *servera parametara* dobiva *URDF*, *SRDF* i konfiguracijske datoteke robota. Kada *MoveIt!* primi sve informacije prikazane na arhitekturi, može se reći da je adekvatno konfiguriran [13].

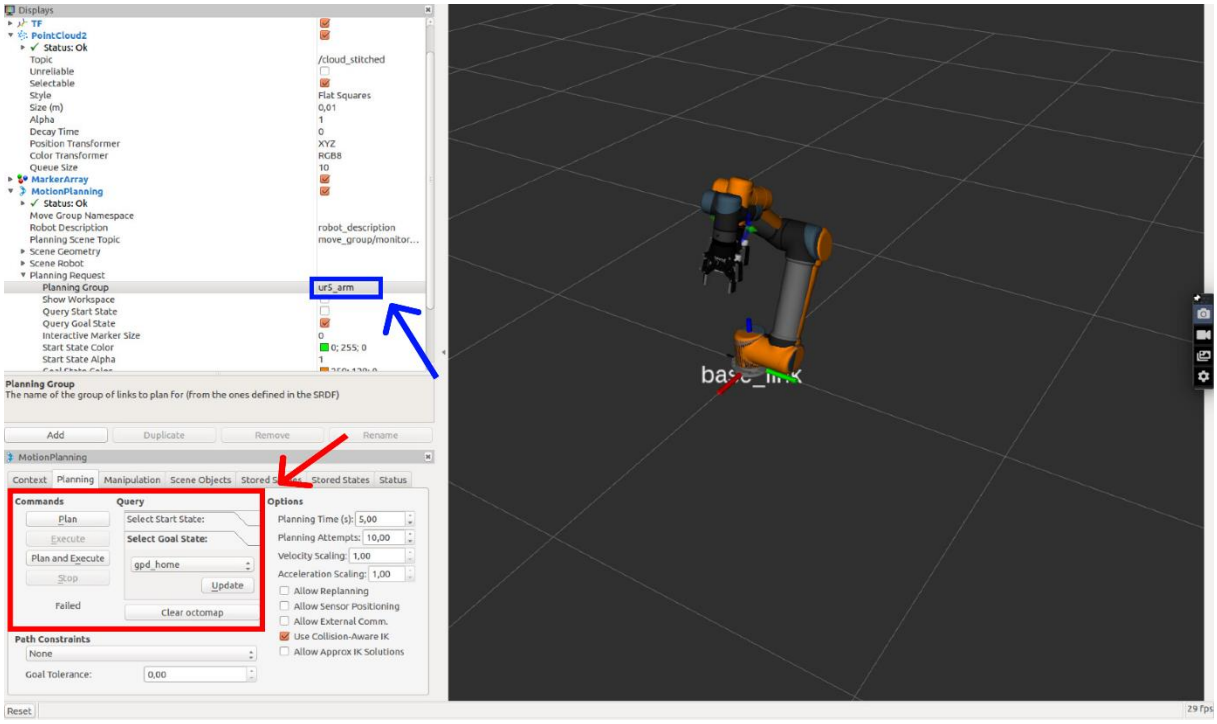
Čvor *move_group* je kao što je prethodno spomenuto samo integrator, on ne pokreće algoritme planiranja kretanja direktno, nego prikuplja sve informacije i povezuje u funkcionalnu cjelinu. Za postizanje zadataka koji su nabrojani u početku poglavlja, *MoveIt!* koristi *plugin*-ove koji su (opet) integrirani preko *move_group* čvora. Tako je bitan *plugin* koji se treba konfigurirati za robota njegov kontroler. U nastavku poglavlja će biti opisana konfiguracija *MoveIt!*-a za robotski sustav korišten u ovome radu.

3.4.1. MoveIt! plugin za RViz

RViz je 3D vizualizacijski alat unutar ROS okruženja. Veliki broj tema koje se objavljuju u ROS-u se može vizualizirati u *RViz*-u (vidjeti primjerice sliku 3.15 gdje je vizualiziran snimljen oblak točaka). Neki paketi imaju i dodatnu integraciju koristeći programske dodatke *(engl. plugin)*. *MoveIt!* je primjer takvog paketa, gdje je njegovo sučelje unutar *RViz*-a prikazano na slici 3.3.

Može se vidjeti da sučelje ima veliki broj opcija i značajki, od prikazivanja modela robota unutar *RViz*-a do planiranja kretanja. Plavo je označeno mjesto gdje se bira grupa planiranja *(engl. planning group)* robota, a crveno je označeno mjesto gdje se definira željena imenovana poza robota i naredbe da robot izvrši gibanje do te poze.

RViz plugin je zgodan alat jer omogućava kontrolu robota bez pisanja koda, no limitiran je u odnosu na mogućnosti koje pruža *Python* ili *C++* sučelje. Zbog toga se unutar ovog rada koristi samo zbog vizualizacije planiranih kretnji, ali ne i za kontrolu kretanja robota.



Slika 3.3 *MoveIt! plugin* za *RViz*

Za pokretanje *RViz*-a potrebno je samo u *terminal* unijeti naredbu niže, a onda u samom programu konfigurirati željene vizualizacije.

```
$ rviz
```

3.4.2. Python sučelje (MoveItCommander)

Cijela kontrola kretnji razvijenog koda za „Pick and Place“ zadatak koristeći rezultate GPD paketa se je vršila pomoću *Python MoveIt!* sučelja. U centru sučelja, leži *moveit_commander* paket.

Paket je podijeljen na nekoliko prostora imena[1] *(engl. namespaces)*, od kojih će se u razvijenom kodu koristiti:

- *move_group* – u njemu se nalazi *MoveGroupCommander* klasa kojom se kontroliraju grupe planiranja (primjerice grupa hvataljke ili robotske ruke); u nastavku poglavlja će se kreirati *MoveIt!* grupe planiranja za robotski sustav
- *planning_scene_interface* – sadrži klasu *PlanningSceneInterface* koja služi kao sučelje robota s njegovom okolinom (unutar razvijenog koda se koristi za definiranje stola, kako bi se planiralo kretanje bez kolizije s njime)
- *roscpp_initializer* – sadrži metode *roscpp_initialize* i *roscpp_shutdown* koje se koriste za inicijalizaciju *moveit_commander*-a (API sa *C++* kodom), odnosno gašenje istoga.

Primjer korištenja *Python* sučelja s *MoveIt!*-om je priloženi kod koji će se detaljnije opisati u idućem poglavlju (lokacija datoteke: *ur5_gripper_moveit_config → scripts → pnp_jurica.py*). Za dodatne informacije o korištenju paketa vidjeti izvor [19].

3.5. Konfiguracija korištenih programa i paketa

U nastavku ovog potpoglavlja se najprije navodi korišteno računalo i verzije softvera za izradu ovog završnog rada. Kako bi se robotski sustav mogao koristiti unutar simulacije potrebno je konfigurirati sve elemente sustava, što je opisano u sklopu ovog poglavlja. Dodatno, opisani je način instalacije paketa GPD i *GPD-ROS* budući da njihova instalacija odstupa od službenih uputa autora paketa. Za ostale programe i pakete nije naveden proces instalacije.

[1] prostor imena unutar Python-a je kolekcija definiranih simboličkih imena zajedno s informacijom o objektima koje svako ime referencira

3.5.1. Korišteno računalo i softversko okruženje

Kao što je već spomenuto, kao baza robotskog sustava korištenog u ovom radu uzima se ROS. ROS je osmišljen za rad na *Ubuntu Linux* operativnom sustavu pa je i računalo na kojem je rađen ovaj rad koristilo *Linux* (dual-boot s *Windows*-om). Verzija odabranog operativnog sustava je *Ubuntu Linux 16.04*, koja je starija, ali je odabrana zbog toga što je GPD paket napisan i testiran u njoj.

U ROS zajednici su neki korisnici uspješno instalirali paket i na drugim distribucijama *Linux*-a i ROS-a. Računalo koje je korišteno za izradu rada je *Gigabyte Aero 15X V8*.
Radi kompletnosti informacija, u tablici 3 se navodi hardver korištenog računala, a u tablici 4 verzije korištenih paketa, programa i operativnih sustava.

Tablica 3 Karakteristike korištenog računala [20]

CPU	*Intel® Core™ i7-8750H Processor (2,2GHz-4,1GHz)*
GPU	*NVIDIA® GeForce® GTX 1070 GDDR5 8GB, Max-Q*
RAM	*16GB DDR4 @ 2666MHz*
Memorija	*Eksterni hard disk: WD Elements Portable 1000GB, USB 3.0*

Tablica 4 Softversko okruženje

OS	*Ubuntu Linux 16.04.7 LTS (Xenial Xerus)*
ROS	*Kinetic Karne (release: May 23rd, 2016)*
Gazebo	*Gazebo 7*
GPD	*2.0.0*
MoveIt	*0.9.18*
PCL	*1.9*
Eigen	*3.4.0*
OpenCV	*3.4*
RViz	*1.12.17*

3.5.2. Instalacija GPD paketa i GPD-ROS wrapper-a

Za instalaciju paketa za softversko okruženje prikazano u tablici 4 potrebno je ručno instalirati *OpenCV*, *Eigen* i *PCL* zavisne pakete. To je potrebno napraviti jer se u službenim repozitorijima *Ubuntu OS 16.04* nalaze starije verzije paketa od onih traženih, budući da je ta distribucija *Linux*-a već zastarjela i više nije podržana. Instalacija *OpenCV 3.4.0* zavisnog paketa se je izvršila prema [21], a instalacija GPD-a prema izvoru [22].

3.5.3. Konfiguracija GPD paketa

Kako bi se konfigurirao GPD, potrebno je urediti konfiguracijsku datoteku koja se nalazi unutar glavnog GPD paketa.

U konfiguracijskoj datoteci (u priloženom *GitHub* repozitoriju lokacija: *gpd_ros → cfg*) se nalaze parametri koji definiraju geometriju hvataljke robota, geometriju kamere, lokaciju težina koje su dobivene nakon treniranja GPD mreže i razne parametre za procesiranje ulaznih podataka u paket.

Kako bi se bolje shvatili parametri i što oni znače unutar programa GPD se može pokrenuti i samostalno bez spajanja na ROS. Pokretanje programa direktno uz navođenje lokacije datoteke koja sadrži oblak točaka proizvoljnog predmeta prikazano je u *terminal* naredbi niže. Ovakvo pokretanje programa je dobro za evaluaciju rezultata paketa u idealnom scenariju (izolirani objekt u oblaku točaka). Rezultat pokretanja naredbe na nekoliko oblaka točaka predmeta vidljiv je na slikama 3.6 a) – e).

```
$ ./detect_grasps ../cfg/eigen_params.cfg ../tutorials/krylon.pcd
```

U nastavku se navode i objašnjavaju konfiguracijski parametri GPD-a:

a) datoteka *ros_eigen_params.cfg*:

- *hand_geometry_filename* – lokacija datoteke koja opisuje geometriju hvataljke
- *image_geometry_filename* – lokacija datoteke koja opisuje geometriju volumena i slike ulaza u GPD
- *weights_file* – lokacija datoteke koja sadrži parametre trenirane neuronske mreže
- *voxelize* – pretvara li se slika u voksel grafiku

- *remove_outliers* – miču li se statistički izuzetci iz rezultata hvatanja
- *workspace* – dimenzije kvadra koje odgovaraju mogućem dosegu korištene robotske ruke
- *camera_position* – prostorna pozicija kamere u odnosu na koordinatni sustav oblaka točaka
- *sample_above_plane* – uzima samo objekte koji se nalaze iznad ravnine podloge; vrlo bitan parametar (vidjeti sliku 3.4 što se događa ako je postavljen na 0)
- *num_samples* – broj uzoraka koji se uzima iz oblaka točaka; čim veći to je vjerojatnije da će biti bolji rezultat hvatanja, ali će algoritam zahtijevati više vremena za izvršenje
- *num_threads* – broj dretvi koje koristi GPD
- *nm_radius* – radijus kojim se radi pretraživanje susjedstva točaka
- *hand_axes* – oko kojih se osi rotira susjedstvo točaka (slika 3.5 prikazuje vektore i njihove identifikacijske brojeve za parametar); moguće navesti jednu, dvije ili sve tri osi
- *deepen_hand* – gura li se hvataljka prema predmetu da se poveća površina prianjanja
- *min_aperature* – koliko je minimalna širina koju hvataljka može postići
- *max_aperature* – maksimalna širina hvataljke
- *workspace_grasps* – granice kvadra (s obzirom na koordinatni sustav oblaka točaka) gdje se promatraju i uzimaju kandidati za hvatanje
- *filter_approach_direction* – filtrira li se u odnosu na neki vektor (s obzirom na ishodište oblaka točaka) smjer pronađenih hvatova
- *min_inliers* – minimalan broj elemenata po identificiranom klasteru
- *num_selected* – koliko se hvatova izabire za vizualizaciju i slanje poruke gpd_ros paketa
- *plot_X* – vizualizira li se X podatak

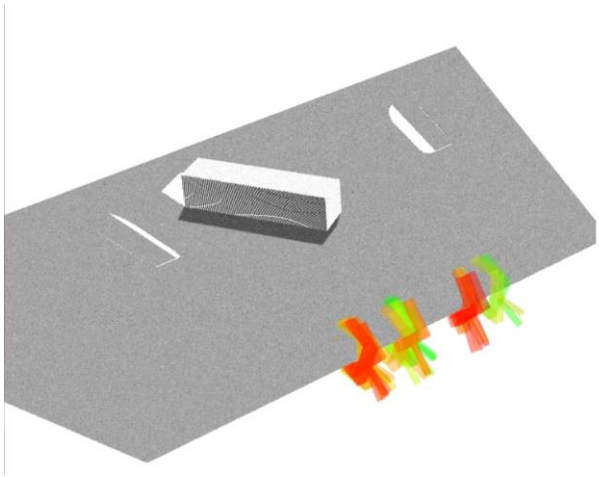
b) datoteka *hand_geometry.cfg*:

- *finger_width* – širina prsta hvataljke

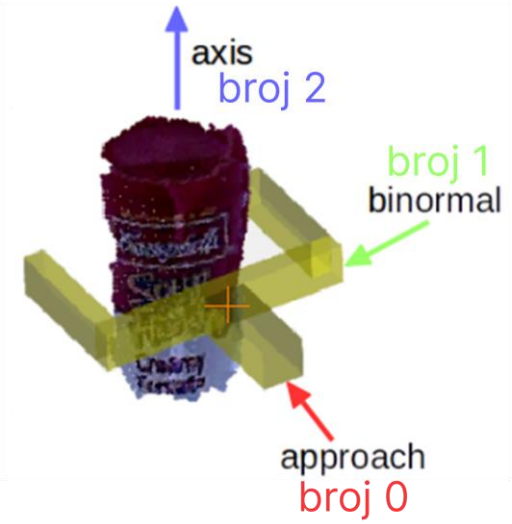
- *hand_outer_diameter* – maksimalna širina hvataljke + 2 * širina prsta
- *hand_depth* – dužina prsta
- *hand_height* – visina ruke
- *init_bite* – koliki udio (snimljene) površine objekta mora biti pokriven hvataljkom nakon hvata

c) datoteka *image_geometry.cfg*:

- *volume_width* – širina kocke koja stane u hvataljku
- *volume_depth* – dubina kocke u hvataljki
- *volume_height* – visina kocke u hvataljki
- *image_size* – veličina slike (širina i visina)
- *image_num_channels* – broj kanala slike

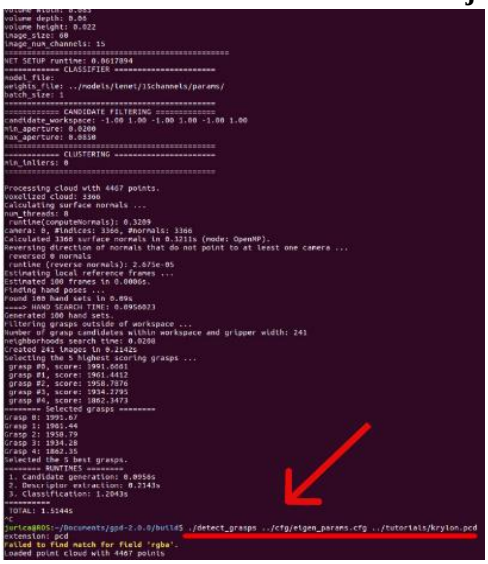


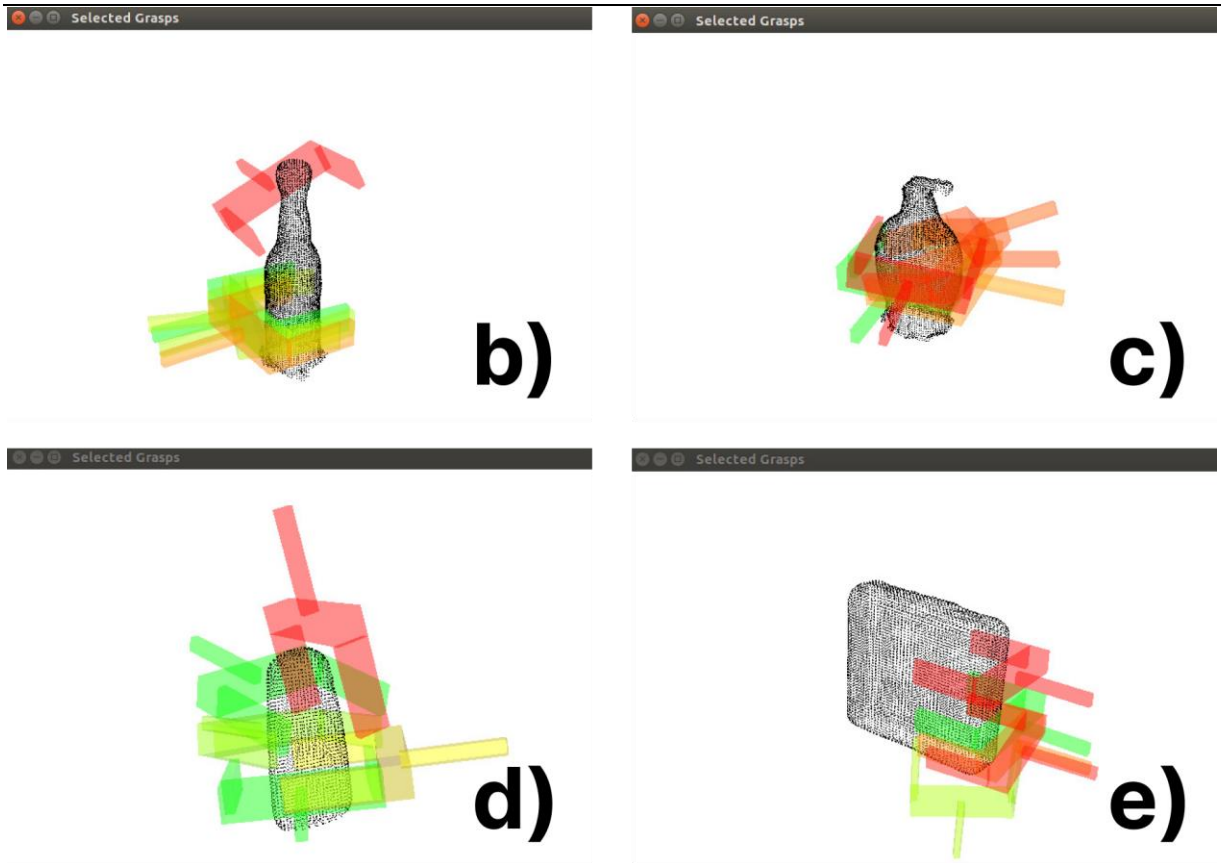
Slika 3.4 Rezultati GPD-a bez uključenog parametra filtriranja iznad ravnine



Slika 3.5 Označene osi hvatanja i njihovi identifikacijski brojevi

a)





Slika 3.6 a) Pokretanje GPD paketa direktno prikazanom naredbom, b) – e) Rezultati pokretanja GPD-a na različitim .pcd datotekama svakodnevnih predmeta

Nakon definiranja parametara, još je potrebno promijeniti *.launch* datoteku *gpd_ros* paketa da reflektira promjene konfiguracijskih parametara. Ispod je dana nova *.launch* datoteka GPD-a, gdje je promijenjen put do konfiguracijske datoteke i naziv teme iz koje dolazi oblak točaka. Naziv teme oblaka točaka na koju se pretplaćuje GPD će se morati poklapati s nazivom teme na koju objavljuje kamera robotskog sustava.

```
<launch>
  <node name="detect_grasps" pkg="gpd_ros" type="detect_grasps"
output="screen">
    <param name="cloud_type" value="0" /> <!-- 0: PointCloud2, 1:
CloudIndexed, 2: CloudSamples -->
    <param name="cloud_topic" value="/gpd_input" />
    <param name="config_file" value="/home/jurica/Documents/gpd-
2.0.0/cfg/ros_eigen_params.cfg" />
    <param name="rviz_topic" value="plot_grasps" />
  </node>
</launch>
```

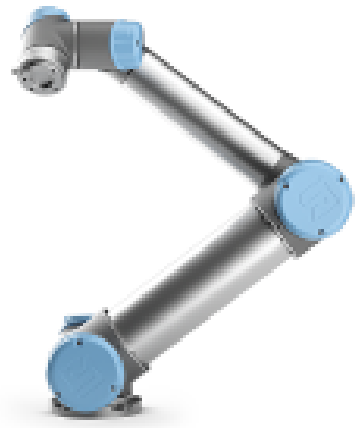
NAPOMENA: sve vrijednosti konfiguracijskih parametra korištene u radu dostupne su u priloženom *GitHub* repozitoriju na lokaciji: *gpd_ros → cfg → "Ime datoteke"*

3.5.4. Robotski sustav (UR5e, Robotiq 2F-85 i Intel Realsense)

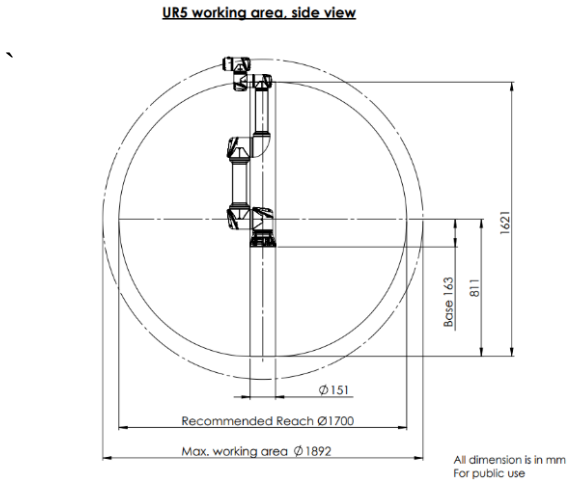
Kod konfiguracije GPD paketa je vidljivo da je potrebno definirati parametre koji su vezani uz robotsku ruku, hvataljku i kameru na kojima se GPD koristi. Stoga je potrebno izabrati komponente robotskog sustava koje će se simulirati u *Gazebu*.

Za simulaciju je odabrana kolaborativna robotska ruka *UR5e* tvrtke *Universal Robots*, prikazana na slici 3.7. Ova robotska ruka je izabrana zbog dobrih značajki prikazanih u tablici 5, njezine sigurnosti (kolaborativni robot namijenjen za rad u istom radnom prostoru s čovjekom), raširenosti u ROS zajednici (dobra podrška) i na posljetku dostupnosti u *Regionalnom centru izvrsnosti za robotske tehnologije (CRTA)* što daje mogućnost kasnijeg testiranja implementiranog paketa u stvarnom svijetu.

Radni prostor *UR5e* robota je prikazan na slici 3.8. Može se vidjeti da je radni prostor sfernog oblika s preporučenim dosegom radijusa 850 mm. Ostale značajke robotske ruke se nalaze u tablici ispod.



Slika 3.7 Robotska ruka UR5e [23]



Slika 3.8 Radni prostor *UR5e* robota [24]

Tablica 5 Specifikacije robotske ruke UR5e [25]

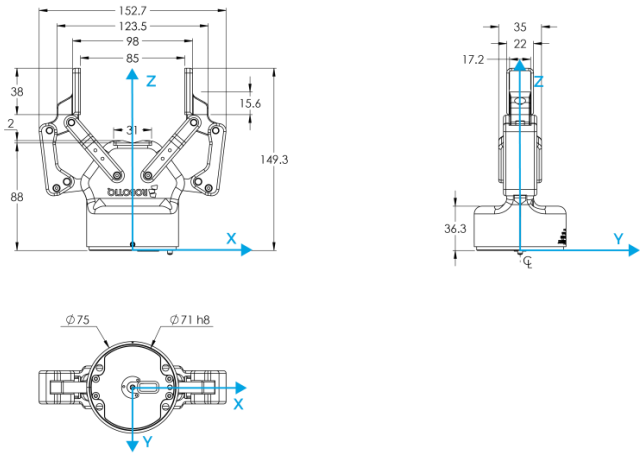
Nosivost	5 kg
Doseg	850 mm
Stupnjevi slobode	6
Masa	20,6 kg

Uz robotsku ruku, izabrana je i dvoprstna hvataljka *Robotiq 2F-85* prikazana na slici 3.9. Ovaj model hvataljke je izabran zbog dobrih značajki navedenih u tablici 6, kompatibilnosti s *UR5e* robotskom rukom, dostupnih konfiguracijskih datoteka za simulaciju i velike podrške u ROS zajednici. Dodatno, ova hvataljka je dostupna i u *CRTA*-i[2] pa isto kao i s robotskom rukom daje mogućnost isprobavanja razvijenog sustava hvatanja na pravom robotu.

Dimenzije hvataljke se nalaze na slici 3.11 a korisne su za definiranje parametara GPD paketa (objašnjenih ranije).

Slika 3.9 *Robotiq 2F-85* hvataljka [26]

Slika 3.10 *Intel Realsense D435* [27]



Slika 3.11 Dimenzije hvataljke *Robotiq 2F-85* [28]

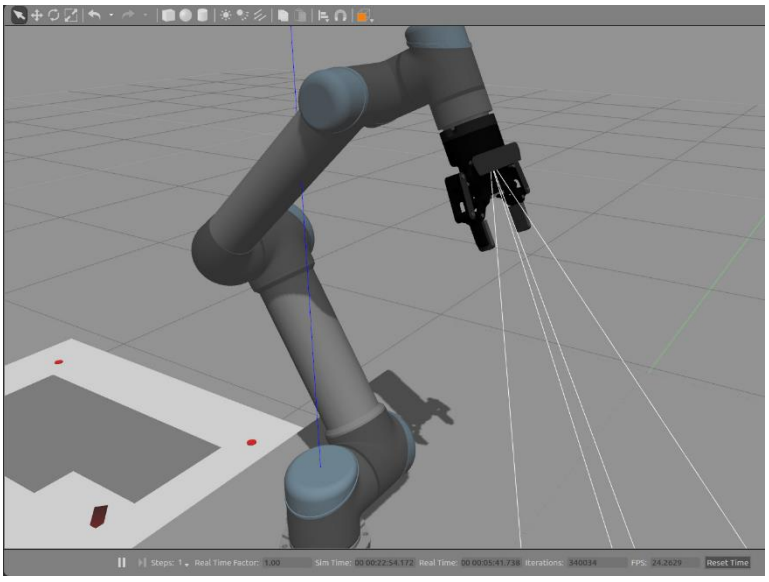
[2] u *CRTA*-i je dostupna hvataljka *Robotiq 2F-140*, no razlika između hvataljki je efektivno u udaljenosti između prstiju, što se vrlo lako promijeni u konfiguraciji GPD paketa

Tablica 6 Tehničke značajke hvataljke *Robotiq 2F-85* [28]

Širina hvataljke	85 mm
Težina	0,925 kg
Sila hvatanja	20…235 N
Brzina prstiju	20…150 mm/s
Rezolucija pozicije	0,05 mm
Nosivost	5 kg
Kontroler	Integrirani vlastiti kontroler; funkcionalnost prepoznavanja izuzetog objekta i njegovih dimenzija

Paket GPD zahtjeva kao ulaz u algoritam hvatanja 3D informacije o objektu . Zbog toga je potrebno koristiti dubinsku kameru. Izabrana kamera treba biti kompaktna, davati kvalitetnu snimku oblaka točaka i biti precizna kako bi GPD mogao davati zadovoljavajuće rezultate. U svrhu zadovoljavanja tih zahtjeva, izabrana je kamera *Intel Realsense D435*. Kamera ima doseg snimanja do 10 m, kompaktna je, ima široko vidno polje i dobru vidljivost u scenarijima s malo osvjetljenja [27].

Nadalje, kamera ima mogućnost brzog i jednostavnog kalibriranja bez korištenja mete. Ove značajke je čine popularnim izborom za primjene u robotici te uzročno-posljedično i dobro podržanom u ROS zajednici. Kamera je prikazana na slici 3.10, a cjelokupan robotski sustav na slici 3.13 (slikano u *CRTA*-i).

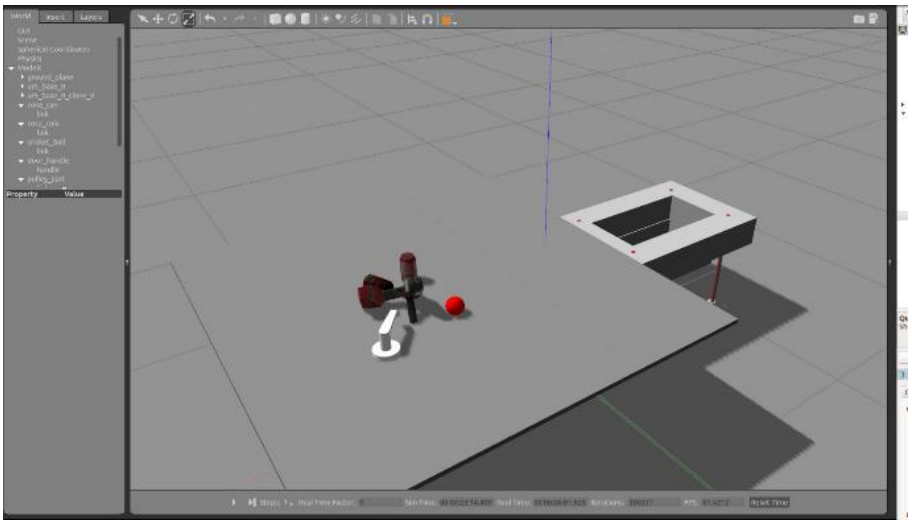


Slika 3.12 Robotski sustav - konfiguriran za *Gazebo* simulaciju

Slika 3.13 Robotski sustav - stvarni

3.5.5. Kreiranje Gazebo svijeta (eng. world)

Za potrebe ovog rada, izrađen je jednostavni *Gazebo* svijet u kojemu se odvija simulacija. Svijet je prikazan na slici ispod. U svijetu se nalaze tri stola, na jednom se nalazi robot, na drugom niz svakodnevnih predmeta čija je svrha testiranje algoritma GPD-a dok je trećem stolu svrha ostavljanje izuzetih predmeta. U svijet je još potrebno smjestiti izabranu robotsku ruku, hvataljku i kameru.



Slika 3.14 Kreirani *Gazebo* svijet

3.5.6. Konfiguracija UR5e ruke

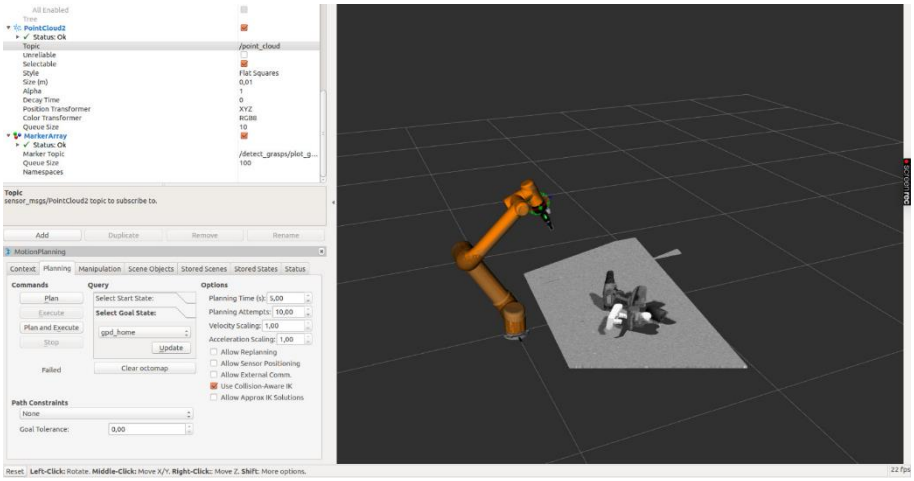
Kako bi se robotski sustav prenio u *Gazebo* simulaciju, potrebno je kreirati *URDF* datoteku. *URDF (engl. Unified Robotic Description Format)* je XML format koji se koristi u ROS okruženju za definiranje elemenata unutar robota. *URDF* je koristan i standardiziran format za opisivanje robota unutar ROS-a, ali mu nedostaju neke značajke potrebne za fizikalnu simulaciju. Primjerice, u izvornom *URDF* formatu ne postoji podrška za opisivanje više robota odjednom ili inercijskih parametara (npr. trenje). Zbog toga se za korištenje unutar *Gazeba* razvija novi format nazvan *SDF (engl. Simulation Description Format)* detaljno opisan u [29, 30].

Na sreću, za većinu robota su već kreirane *URDF* datoteke koje se mogu preuzeti i koristiti unutar ROS okruženja. Za konfiguriranje *UR5e* robotske ruke, preuzeti je *URDF* paket od *Universal Robot* kompanije [31]. Svi paketi korišteni u ovom završnom radu su dostupni na *GitHub* repozitoriju koji se nalazi u prilogu rada. Zbog toga, neće se detaljno opisivati način preuzimanja gotovih datoteka, nego će se samo navoditi dodatci i izmjene koje su napravljene za prilagođavanje paketa ovom radu.

U *GitHub* repozitoriju (prilog) u datoteci *universal_robot → ur_description → urdf* dodana je nova datoteka nazvana *ur5_robotiq85_gripper.urdf.xacro* u kojemu je opisana cijela *URDF* konfiguracija robotskog sustava praćenjem prvog dijela izvora [32].

U *Gazebo* svijet i konfiguraciju *URDF*-a robota potrebno je dodati *Robotiq 2F-85* hvataljku i *Intel Realsense* kameru. Za simulaciju kamere u *Gazebu* koristi se *plugin* dostupan na [33]. Kako bi se *plugin* uključio u model robota, potrebno je dodati linije koda koje definiraju spoj kamere na robota te naziv i lokaciju spremljenog *plugin*-a kamere u preuzetu *URDF* datoteku *ur5_robotiq85_grippper.urdf.xacro*. Promjene koje su napravljane kako bi se uključila kamera u model robota se mogu vidjeti u priloženom *GitHub*-u. Bitno je naglasiti da se mora uključiti u *RealSense* pluginu opcija *pointCloud* jer će nam trebati za konfiguraciju paketa koji se koristi za hvatanje.

Nakon napravljenih izmjena, robotski sustav je spreman za simulaciju. Rezultat izmjena *URDF* formata za robotski sustav je prikazan na slici 3.12. Spajanjem *RealSense plugin*-a u *Gazebo* dobiva se oblak točaka *(engl. point cloud)*, koji je prikazan na slici ispod.



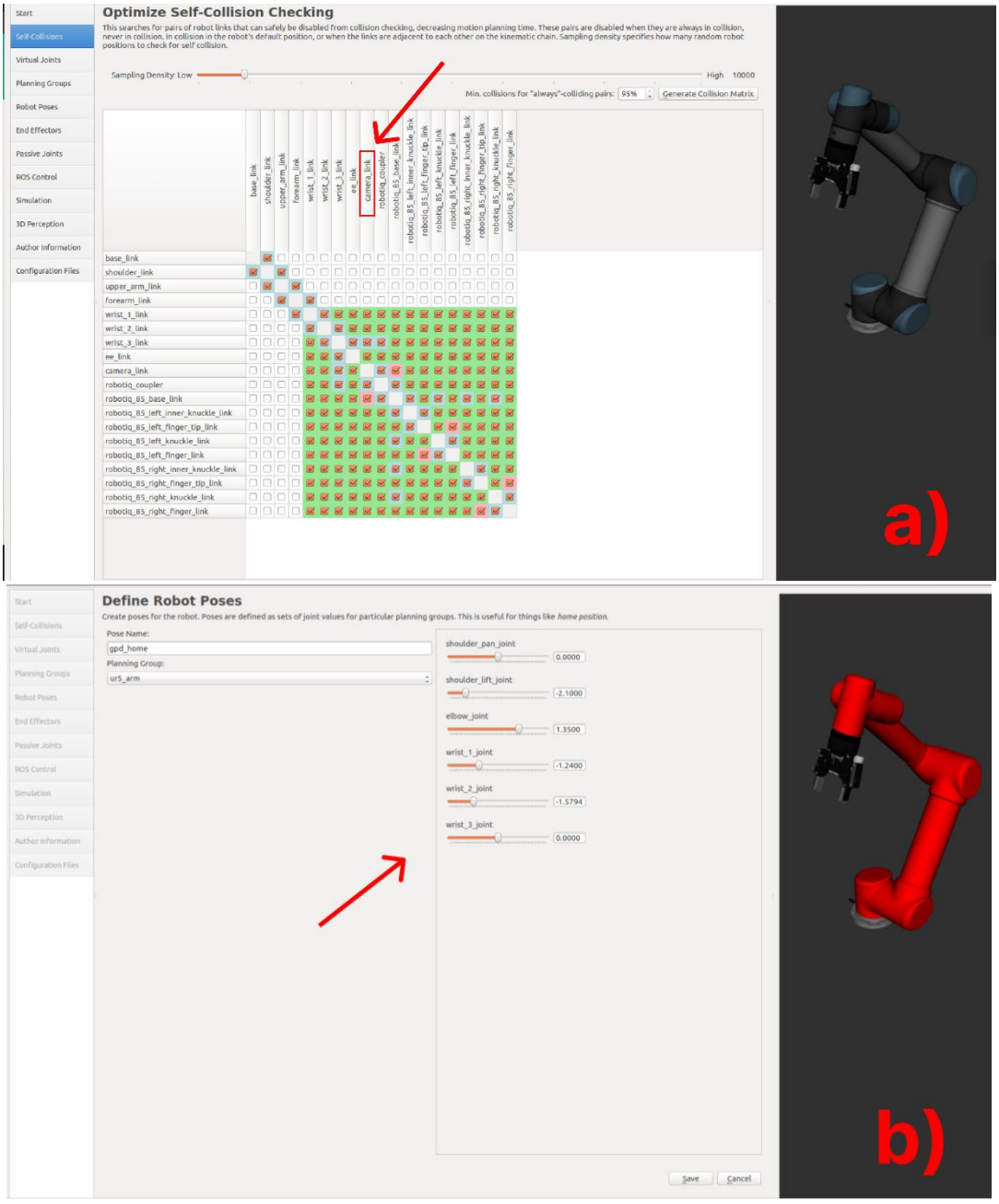
Slika 3.15 Dubinska slika nastala korištenjem plugin-a za *Intel Realsense D435* kameru unutar Gazeba

Pokretanje *Gazebo* simulacijskog svijeta i *RViz* vizualizacije se vrši naredbom u *terminal*-u navedene niže.

```
$ roslaunch ur5_gripper_moveit_config demo_gazebo.launch
```

3.5.7. Konfiguracija MoveIt! paketa

Za upravljanje robotom unutar simulacije potrebno je konfigurirati i *MoveIt!* paket.Za konfiguriranje *MoveIt!* paketa na robotski sustav, prati se postupak objašnjen u izvoru [32]. Od izvora se razlikuje *URDF* datoteka definirana u *MoveIt! Setup Assistant*-u, gdje je za konfiguraciju ovog *MoveIt!* paketa izabrana datoteka definirana u poglavlju 3.5.6. Također, razlikuju se i definirane pozicije robota i matrica kolizija (posljedica uključivanja kamere u sustav). Sve promjene su prikazane na slikama ispod.



Slika 3.16 a) Dodano ignoriranja kolizija spoja kamere, b) Dodana gpd_home pozicija robota

3.5.8. Transformacija oblaka točaka kamere u koordinatni sustav robota

Na posljetku, prije korištenja GPD-a s upravljačkim kodom (opisan u idućem poglavlju), potrebno je još transformirati koordinatni sustav oblaka točaka kamere. Razlog zašto se transformira koordinatni sustav je jer su parametri GPD-a definirani u odnosu na koordinatni sustav oblaka točaka i jer su rezultati koje paket daje u istom koordinatnom sustavu. Za transformaciju, koristi se čvor koji je dostupan na *GitHub*-u [34]. Potrebno je samo promijeniti dva parametra u *demo.launch* datoteci unutar paketa. Ispod su navedeni promijenjeni parametri unutar *.launch* datoteke.

```
    <param name="target_frame" value="base_link" />
    <remap from="point_cloud_transformed" to="/cloud_stitched" />
```

Parametar *target_frame* definira ciljani *tf2* koordinatni sustav u koji želimo transformirati oblak točaka. Druga linija definira ime teme na koju se želi objavljivati transformirani oblak točaka. Može se primijetiti da to nije isto ime teme oblaka točaka koje sluša GPD (definirano u *.launch* datoteci na kraju poglavlja 3.5.3). Razlog zašto se ne koristi isto ime teme je jer se u glavnom upravljačkom kodu za „Pick and Place“ zadatak (priloženi *GitHub* repozitorij datoteka: *ur5_gripper_moveit_config* → *scripts* → *pnp_jurica.py*) uvodi još jedan čvor za pretplaćivanje i objavljivanje oblaka točaka.

Taj čvor se pretplaćuje na transformirani oblak točaka teme */cloud_stitched* i objavljuje isti oblak točaka na temu */gpd_input*, ali samo kada se to zatraži unutar koda i točno jednu poruku. Razlog zašto se tako okolno daje ulaz u GPD je jer u protivnom algoritam stalno prima ulazne oblake točaka i konstantno računa, iako nije u željenoj poziciji snimanja i ne daje korisne rezultate, a troši računalno vrijeme procesora. Objavljivanje transformiranog oblaka točaka se radi frekvencijom 0,2 Hz.

4. ANALIZA DOBIVENIH REZULTATA I DALJNJE MOGUĆNOSTI RAZVOJA

Za analizu i evaluaciju dobivenih rezultata GPD paketa i razvijene strategije upravljanja kretanja robota korištenjem istih, koristi se simulacijsko okruženje razvijeno u prethodnom poglavlju ovog rada. Simulacija je koristan alat za testiranje algoritama u robotici prije korištenja istih na stvarnoj opremi jer na siguran i ekonomski isplativ način provjerava njihovu uspješnost i učinkovitost. Simulacija dakako ima i svoje mane, budući da su rezultati dobiveni na stvarnom hardveru uvijek bolji pokazatelj uspješnosti upravljačkog algoritma.

4.1. Analiza rezultata GPD paketa

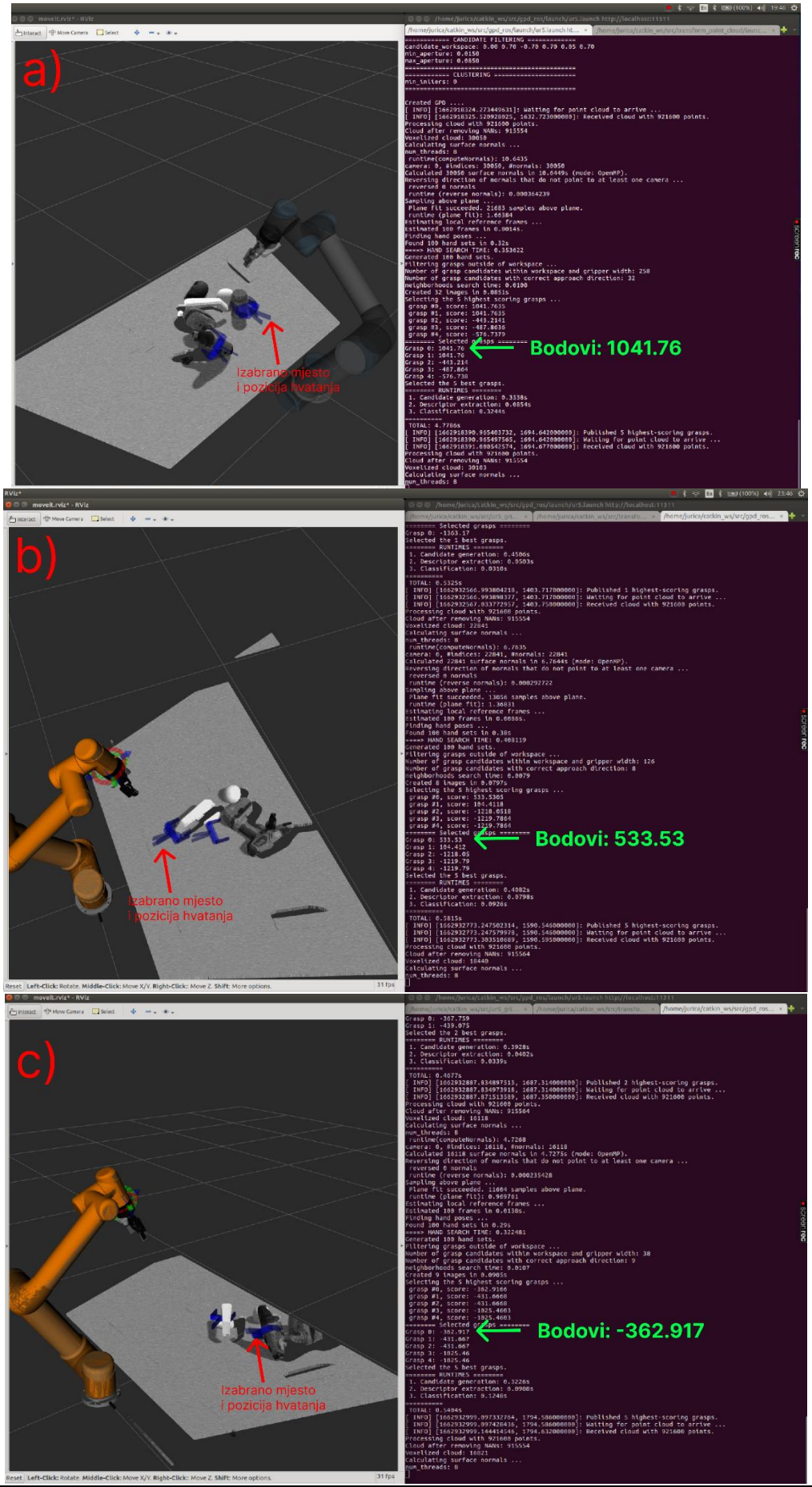
Kao test paketa, koristit će se *Gazebo* svijet prikazan na slici 3.14. Konfiguracijski parametri paketa su dostupni u priloženom repozitoriju, gdje su lokacije datoteka navedene u prethodnom poglavlju.

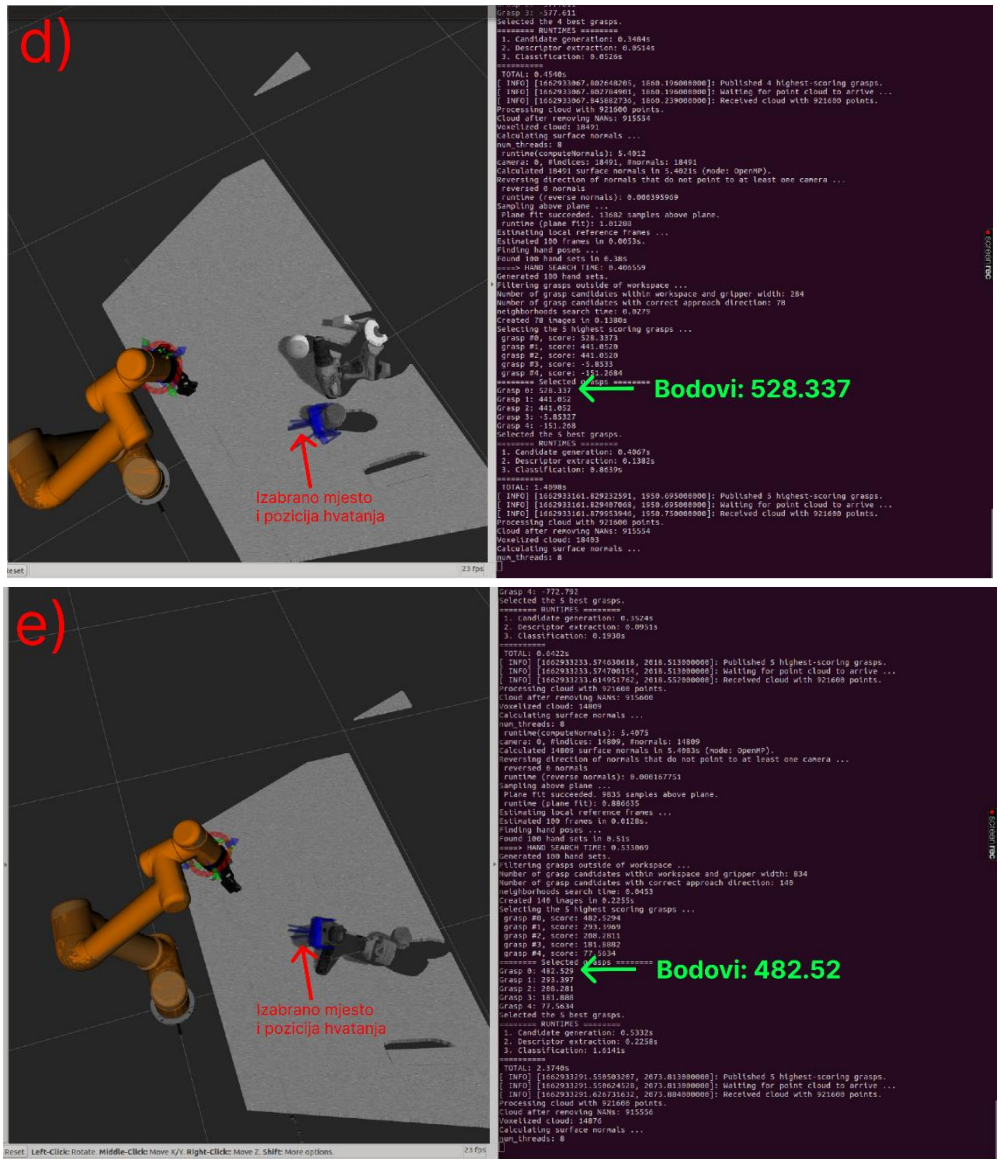
U svrhu testiranja rezultata, predmeti koji su smješteni na stolu će se nasumično izmiješati i postaviti u pozicije unutar radnog prostora robota. Udaljenost između predmeta će biti mala, kako bi se imitirao scenarij izuzimanja predmet u gustom neredu *(engl. dense clutter picking)*, za koji je po tvrdnjama autora paketa uspješnost hvatanja 93% [11].

Za evaluaciju rezultata će se najprije koristiti heuristična procjena kvalitete poza hvatanja na temelju iskustva hvatanja predmeta stečenih u vlastitom životu. Ljudi su kao što je spomenuto u uvodu izuzetno dobri u ovome zadatku, tako da ova heuristika nije loša za prvu aproksimaciju kvalitete rezultata.

Nakon toga će se koristiti rezultati GPD-a da se unutar *Gazebo* simulacije pokušaju podići predmeti koristeći razvijen upravljački kod. U upravljačkom kodu će se ujedno i svi do sada opisani elementi spojiti u funkcionalnu cjelinu.

U nastavku slijede rezultati koje daje GPD algoritam za pet scenarija nasumično postavljenih predmeta po stolu iz *Gazebo* simulacije.


a)
Izabrano mjesto
i pozicija hvatanja
Bodovi: 1041.76
b)
Izabrano mjesto
i pozicija hvatanja
Bodovi: 533.53
c)
Izabrano mjesto
i pozicija hvatanja
Bodovi: -362.917



Slika 4.1 a) - e) Testiranje izlaza GPD-a za 5 različitih pozicija i orijentacija predmeta na stolu

Iz slika 4.1 a) - e) se vide razlike u rezultatima koje daje GPD. Algoritam koristi rangiranje kvalitete (odnosno pouzdanja) hvatova po nekom bodovnom rangu *(engl. score)*. Što je algoritam sigurniji u točnost hvata, to je veći broj dodijeljenih bodova. Tako se bodovi dobiveni za hvatove prikazane na slikama iznad kreću u rasponu vrijednosti od -362,87 do 1041,76.

Prva dva hvata su dobro rangirana i heuristično se čine kao dobra mjesta hvatanja objekata. Treći hvat je najlošije rangiran, jer algoritam nije siguran koliko je dobro mjesto hvatanja na objektu (bušilici), vjerojatno jer je dio oblaka točaka izvan dometa dubinske kamere. Preostala dva hvata su dobro rangirana i djeluju kao dobro mjesto hvatanja objekata.

4.2. Razvijen kod za „Pick and Place“ zadatak koristeći rezultate GPD-a

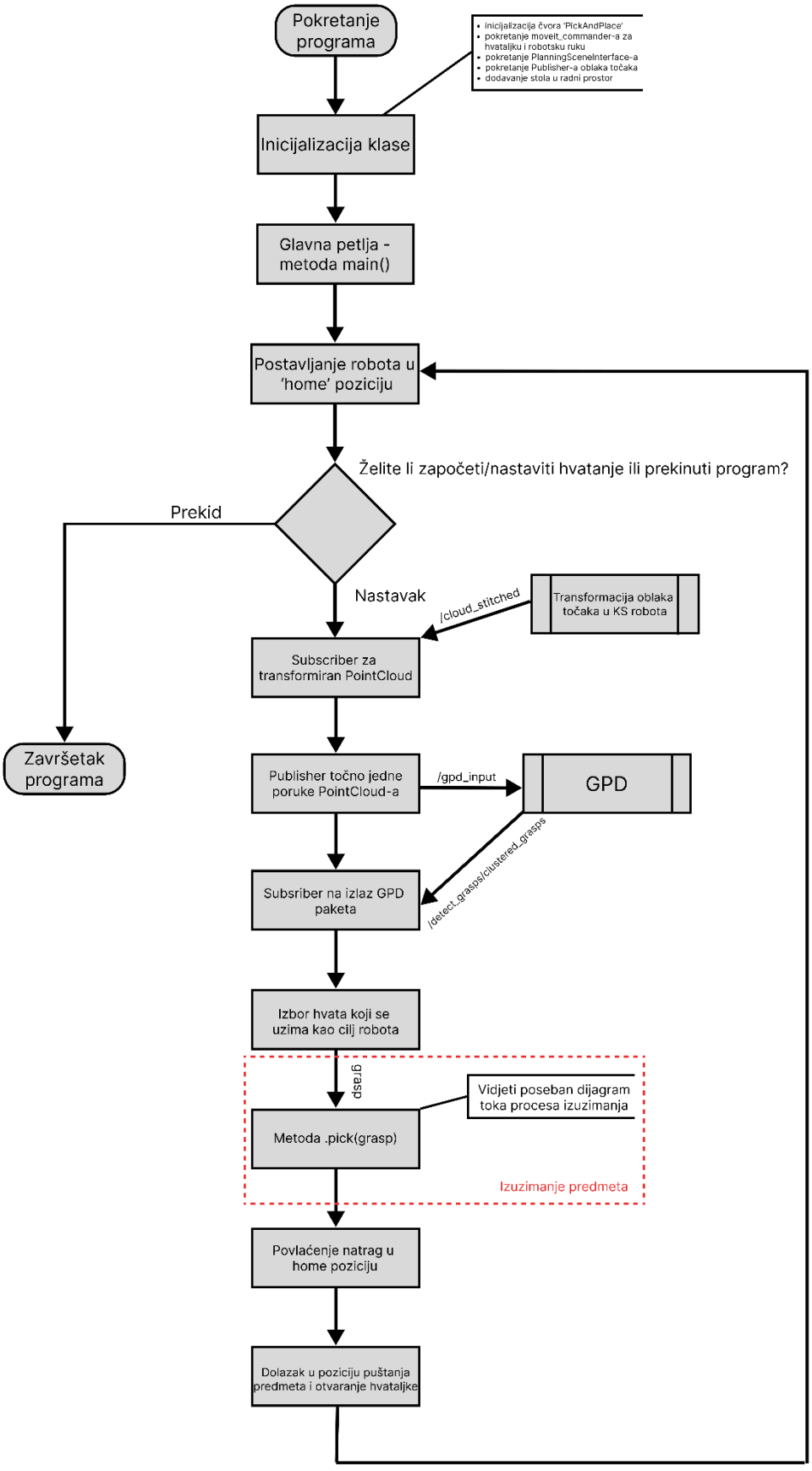
U prethodnom poglavlju su predstavljeni rezultati GPD-a i vidi se da sustav radi i daje neke rezultate koji djeluju zadovoljavajuće. Da ne ostane sve na teorijskoj razini, potrebno je rezultate osim heuristično, provjeriti i dovođenjem robota do poze hvata. To će se postići koristeći vlastito razvijen kod u ROS okruženju u kojem će se svi dijelovi koji su do sada opisani spojiti u funkcionalnu cjelinu s ciljem izvršavanja zadatka.

Slika 4.2 prikazuje dijagram toka razvijenog „Pick and Place“ programa, dok slika 4.3 prikazuje dijagram toka metode izuzimanja predmeta (označena crveno na slici 4.2). Najprije je potrebno pokrenuti program koristeći naredbu prikazanu ispod (nakon što su pokrenuti svi ostali paketi i programi opisani u prethodnom poglavlju).

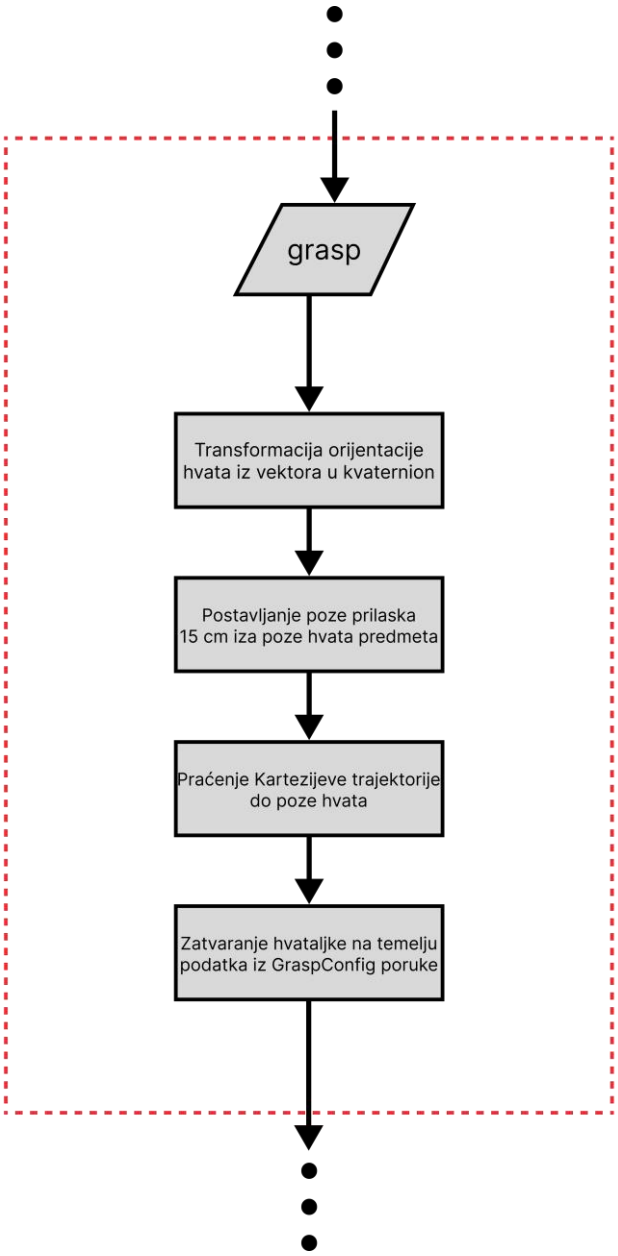
```
$ rosrun ur5_gripper_moveit_config pnp_jurica.py
```

Kada se pokrene program, kreće se s inicijalizacijom klase. Unutar klase, poziva se niz objekata koji su navedeni kao komentar u oblaku spojenom s „Inicijalizacija klase“ u dijagramu toka. Nakon toga se pokreće glavna petlja programa i robot se postavlja u *gpd_home* poziciju iz koje se snima oblak točaka za ulaz u GPD algoritam i traži se naredba korisnika želi li pokrenuti program hvatanja ili izaći van iz programa. Ovisno o odgovoru korisnika, kreće prikupljanje informacija i međusobna komunikacija čvorova, što je prikazano u dijagramu toka. Transformirani oblak točaka se šalje GPD paketu te se generiraju poze hvatanja na snimljenim objektima. GPD vraća rangiranu listu mogućih hvatova i unutar programa se koristi jednostavna heuristika za odabir hvata (uzima se najbolje rangirani iz liste pronađenih poza hvatanja).

Nakon odabira hvata, zove se metoda izuzimanja predmeta (*pick* metoda unutar programa). Tu metodu opisuje dijagram toka na slici 4.3. Ukratko, metoda uzima izabrani hvat i izvlači podatke pozicije i orijentacije hvata u prostoru i transformira tu poruku u kvaternion za lakšu manipulaciju u kodu. Zatim robotska ruka dobiva naredbu da dođe do točke 15 cm iza točke hvatanja (u točku prilaska) u smjeru vektora označenog brojem 0 na slici 3.5. Kada robot dođe u točku prilaska, stvara se trajektorija da robot dođe do točke prihvata objekta.



Slika 4.2 Dijagram toka razvijenog "Pick and Place" programa (izuzimanje dodatno prikazano na drugom dijagramu toka)

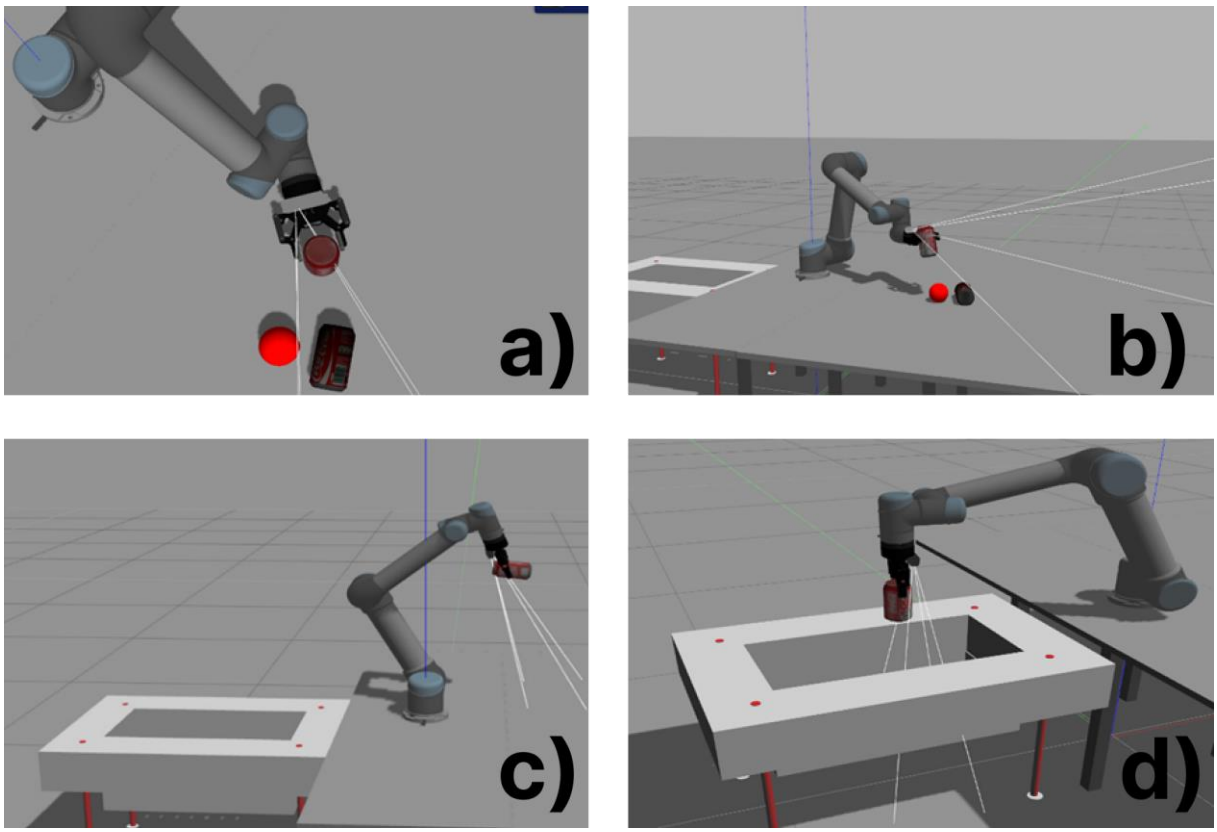


Slika 4.3 Dijagram toka metode za izuzimanje predmeta unutar glavnog programa

Budući da je korišteni kontroler u *Gazebu* dosta loše podešen, kontrola robotske ruke nije precizna. Zbog toga se i definirana trajektorija koristi kao dodatna mjera poboljšanja preciznosti, jer je definirana s obzirom na željenu točku prilaska, a ne onu u koju je robot zapravo došao. Praktički je efekt toga da robot popravi većinu odstupanja praćenjem ove trajektorije i dobiva se zadovoljavajuća preciznost. Na posljetku se šalje naredba hvataljki da se zatvori i nastavlja se izvođenje glavnog toka programa.

4.3. Rezultati upravljačkog programa

Testiranje razvijenog programa se provodi u *Gazebo* simulacijskom svijetu prikazanom na slici 3.14 i korištenjem kontrolera robotske ruke konfiguriranih prema poglavlju 3.5.7. Rezultati su prikazani na slikama ispod, a video demonstracije su dostupne u prilogu II. i III.



Slika 4.4 a) Dolazak u točku prihvata GPD-a, b) Izuzimanje predmeta, c) Vraćanje u „home“ poziciju, d) Odlaganje predmeta

Kao što je vidljivo iz slika i videa u prilogu, razvijeni program daje dobre rezultate što se tiče dolaženja do točke prilaska i točke prihvata. U nekim slučajevima se robot tijekom prilaženja predmetu po trajektoriji od točke prilaska do točke prihvata zna zaustaviti i prekinuti kretanje. Razlog tome je jer u *Gazebo* simulaciji najbolje rade kontroleri koji koriste *EffortJointInterface*, odnosno kontroleri koji zadaju referentnu vrijednost sile ili momenta umjesto pozicije za kontrolu robota. Takvi kontroleri daju lošiju preciznost u usporedbi s kontrolerima koji koriste referentne vrijednosti pozicije zglobova.

Ultimativni rezultat toga je da *MoveIt!* tijekom izvršavanja trajektorije zna javiti grešku i prekinuti kretanje zbog odstupanja trenutne vrijednosti pozicije robota u odnosu na traženu vrijednost. Ovaj problem se može riješiti povećavanjem tolerancije *MoveIt!*-a na vrijednost

veću od zadanih 0,01 mm koristeći *.set_goal_tolerance()* metodu *MoveGroupCommander* klase. Takvo povećanje tolerancije nije preporučljivo već je bolje poboljšati parametre PID regulatora.

Razvijeni program bi na pravom robotskom hardveru trebao davati dobre rezultate, budući da navedeni problemi nestanu kada se koristi optimiran kontroler, što *UR5* ima. Dodatno, ako se koristi *MoveIt!* kontroler za upravljanje pravim robotom, preciznost će također biti bolja zbog mogućnosti korištenja *PositionJointInterface* kontrolera umjesto *EffortJointInterface* kontrolera. *PositionJointInterface* kontroleri daju bolju preciznost od *EffortJointInterface* kontrolera kod praćenja trajektorija i dolaženja do željene poze u prostoru.

5. ZAKLJUČAK

U sklopu ovog završnog rada najprije je objašnjen povijesni razvoj metoda hvatanja u robotici. Navedeno je nekoliko metoda razvijenih u znanstvenoj zajednici s ciljem hvatanja i manipulacije nepoznatih predmeta u prostoru. Na posljetku je izabrana metoda zvana GPD, koja koristi strojno učenje kako bi iz dubinske slike dobivene iz senzora lokalizirala moguće hvatove na objektu bez potrebe estimacija poze objekta. GPD paket je javno dostupan i integriran je s ROS-om te je zbog toga razvijen robotski sustav u ROS okruženju.

Ostatak završnog rada opisuje sve potrebne korake za integraciju GPD paketa unutar ROS okruženja i korištenja njegovih rezultata za manipulaciju nepoznatih objekta u simulacijskom svijetu. Opisuju se potrebni koraci za instalaciju i konfiguraciju paketa korištenih u svrhu povezivanja sustava u funkcionalnu cjelinu.

Na kraju se opisuje razvijeni kod koji služi za izvođenje *„Pick and Place“* zadatka na nepoznatim predmetima u simulacijskoj sceni. Upravljački algoritam je sinteza svega naučenog i razvijenog u sklopu završnog rada.

Upravljački algoritam dobro dolazi u pronađenu pozu prihvata objekta, ali ga je još potrebno testirati na pravom robotskom hardveru kako bi se dobila prava slika kvalitete sustava. Simulacija pruža dobar, siguran i jeftin način za razvijanje algoritama unutar robotike, ali zbog ograničenja tehnologije i dalje je potrebno testirati algoritme na stvarnom hardveru.

LITERATURA

[1] https://www.babycentre.co.uk/a6578/developmental-milestones-graspings, Pristupljeno: 7.09.2022.

[2] Hodson R. : A gripping problem, Nature 2018.

[3] Zhang, H. et al. : Robotic Grasping from Classical to Modern: A Survey

[4] Siciliano B., Khatib O. : Handbook of Robotics, 2nd edition. Berlin: Springer; 2016.

[5] Levine, S. et al. : Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection

[6] https://www.euclideanspace.com/maths/geometry/affine/screwTheory/index.htmm, Pristupljeno: 18.09.2022.

[7] Lynh, K., Park F. Modern Robotics, Preprint edition. Cambridge: Cambridge University Press; 2017.

[8] Mahler, J. et al. : Dex-Net 1.0: A Cloud-Based Network of 3D Objects for Robust Grasp Planning Using a Multi-Armed Bandit Model with Correlated Rewards

[9] https://www.youtube.com/watch?v=ATDrSWZXuwk, Pristupljeno 17.09.2022.

[10] Kleeberger, K. et al. : A Survey on Learning-Based Robotic Grasping

[11] ten Pas, A. et al. : Grasp Pose Detection in Point Clouds

[12] https://www.kaggle.com/code/blurredmachine/lenet-architecture-a-complete-guide/notebook, Pristupljeno: 19.09.2022.

[13] Lentin J., Cacace J.: Mastering ROS for Robotics Programming, Second Edition. Birmingham-Mumbai: Packt Publishing Ltd; 2018.

[14] http://wiki.ros.org/ROS/Tutorials , Pristupljeno: 10.07.2022.

[15] Čaran, B. Planiranje kretanja i ispitivanje točnosti pozicioniranja mobilnog robota

[16] https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html, Pristupljeno: 17.09.2022.

[17] https://GitHub.com/atenpas/gpd, Pristupljeno: 10.09.2022.

[18] https://classic.gazebosim.org/tutorials, Pristupljeno: 10.07.2022.

[19] http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/index.html, Pristupljeno: 11.09.2022.

[20] https://www.gigabyte.com/Laptop/AERO-15X--i7-8750H#kf, Pristupljeno: 9.09.2022.

[21] https://web.archive.org/web/20190130231639/http://www.pytorials.com/how-to-install-opencv340-on-ubuntu1604/, Pristupljeno: 10.09.2022.

[22] https://blog.katastros.com/a?ID=01650-c9ca3fcd-dc23-42af-b420-094205e25e44, Pristupljeno: 10.09.2022.

[23] https://www.techvitas.com/en/ur5-ur5e/, Pristupljeno: 9.09.2022.

[24] https://www.universal-robots.com/articles/ur/application-installation/what-is-a-singularity/, Pristupljeno: 9.09.2022.

[25] https://www.universal-robots.com/media/50588/ur5_en.pdf, Pristupljeno: 9.09.2022.

[26] https://www.universal-robots.com/plus/products/robotiq/robotiq-2f-85/, Pristupljeno: 9.09.2022.

[27] https://www.intelrealsense.com/depth-camera-d435/, Pristupljeno: 9.09.2022.

[28] https://assets.robotiq.com/website-assets/support_documents/document/online/2F-85_2F-140_TM_InstructionManual_HTML5_20190503.zip/2F-85_2F-140_TM_InstructionManual_HTML5/Content/6.%20Specifications.htm, Pristupljeno: 17.09.2022.

[29] https://classic.gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros, Pristupljeno: 10.09.2022.

[30] http://wiki.ros.org/urdf/Tutorials, Pristupljeno: 10.09.2022.

[31] https://GitHub.com/ros-industrial/universal_robot, Pristupljeno: 11.09.2022.

[32] https://roboticscasual.com/ros-tutorial-how-to-create-a-moveit-config-for-the-ur5-and-a-gripper/, Pristupljeno: 06.09.2022.

[33] https://GitHub.com/pal-robotics/realsense_gazebo_plugin, Pristupljeno 9.09.2022.

[34] https://GitHub.com/lucasw/transform_point_cloud, Pristupljeno: 11.09.2022.

PRILOZI

I. Cjelokupno ROS radno okruženje *(engl. workspace)* ovog rada dostupno je na: https://GitHub.com/Ginkbel/zavrsni_ws

II. Primjer uspješnog hvata u *Gazebu*: https://youtu.be/be1sQWZDDsY

III. Primjer neuspješnog hvata u *Gazebu*: https://youtu.be/riQlwA25WZU