SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2129

PRILAGODLJIVO UPRAVLJANJE VIRTUALNIM ROBOTOM KORIŠTENJEM SUSTAVA UNITY NA PRIMJERU REZANJA 3D OBJEKTA

Luka Abramušić

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2129

PRILAGODLJIVO UPRAVLJANJE VIRTUALNIM ROBOTOM KORIŠTENJEM SUSTAVA UNITY NA PRIMJERU REZANJA 3D OBJEKTA

Luka Abramušić

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

DIPLOMSKI ZADATAK br. 2129

Pristupnik:	**Luka Abramušić (0036485591)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	doc. dr. sc. Mirko Sužnjević

Zadatak: **Prilagodljivo upravljanje virtualnim robotom korištenjem sustava Unity na primjeru rezanja 3D objekta**

Opis zadatka:

Tržište video igara raste velikom brzinom posljednjih godina te se na tržištu pojavio i veliki broj sustava za razvoj igara. Unity sustav za izradu igara je jedan od trenutno najpopularnijih. Osim za igre ovakvi sustavi se koriste za razvoj aplikacija različitih svrha među kojima je i fizikalna simulacija upravljanja robotima. Vaš zadatak je proučiti Unity sustav za izradu igara te u njemu razviti sustav koji omogućuje kontrolu virtualnog robota. Robot treba moći detektirati karakteristike trodimenzionalnih oblika te ih presjeći na pola. Kreirajte i evaluacijski sustav za procjenu učinkovitosti robota. Prikažite funkcionalnost na primjeru dinamički generiranih trodimenzionalnih objekata različitih dimenzija i karakteristika poput zakrivljenosti i ispupčenosti. Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Rok za predaju rada: 30. lipnja 2020.

Zahvaljujem se svojoj obitelji i svim prijateljima koji su mi pružali moralnu podršku i pomogli da održim razum i nađem volje da dovršim ovaj rad u izolaciji tijekom pandemije.

Sadržaj

Uvod 1

1. Slični radovi 2

2. Opis sustava 4

2.1. Rezanje objekta 5

2.2. Putanja kretanja robotske ruke 7

2.3. Model robotske ruke 8

2.4. Algoritam inverzne kinematike 12

3. Programsko rješenje 15

3.1. Korišteni alati 15

3.2. Struktura projekta 16

3.2.1. Klasa SimulationManager 16

3.2.2. Klasa IKManager 18

3.3. Korisničko sučelje 21

4. Rezultati i diskusija 24

Zaključak 31

Literatura 32

Sažetak 33

Summary 34

Uvod

Robotska manipulatorska ruka se koristi u većini današnjih industrija, poput automobilske, prehrambene, kemijske ili skoro svake industrije s pokretnom trakom. Uobičajeni industrijski zadaci su manipulacija, premještanje i obavljanje određenih zadataka na predmetima, uključujući i zadatke na mjestima opasnim za čovjeka (radijacija, visoke temperature). Radi smanjenja troškova i povećanja učinkovitosti industrije koriste računalne simulacije za vizualizaciju ili nadgledanje industrijskih procesa. Postojeća softverska rješenja za simulacije robota pružaju visoko precizne fizičke simulacije, no zahtijevaju detaljno razrađene zahtjeve i vrijeme za učenje softverskog alata.

Unity je sustav za izradu igara koji podržava velika aktivna zajednica. U posljednje vrijeme se zbog svoje pristupačnosti i jednostavnosti korištenja koristi u sve više industrija za simulacije industrijskih procesa.

Cilj ovog diplomskog rada je simulirati zadatak rezanja proizvoljnog virtualnog objekta na pola virtualnom robotskom rukom koristeći Unity sustav za izradu igara. Zadatak rezanja objekta je čest u industriji, no obično se zadaju određeni oblici koje je potrebno izrezati. Računanje polovišta proizvoljnog objekta ovisi o obliku i obilježjima objekta, poput zakrivljenosti, ispupčenosti itd. Virtualni objekti su u programima predstavljeni modelom definiranog mrežom trokutova (engl. mesh).

1. Slični radovi

Postojeća softverska rješenja za simulacije robota poput programa Gazebo (Koenig i Howard, 2004) ili ABB Robot Studio pružaju visoko precizne fizičke simulacije za robusno testiranje i dizajn robota i njihovih primjena. Međutim, za izradu robusnih simulacija su potrebni detaljno razrađeni zahtjevi i vrijeme koje inženjeri trebaju uložiti da nauče koristiti softverski alat.

Posljednji napreci u tehnologijama proširene stvarnosti (engl. Augmented Reality – AR) i virtualne stvarnosti (engl. Virtual Reality – VR) otvorili su nove mogućnosti za interakciju čovjeka s robotom. Razvojem multimodalne opreme za virtualnu stvarnost, korisnici mogu primiti sve više informacija o fizičkom sustavu i samo iskustvo interakcije s digitalnim svijetom postaje sve više uranjajuće za korisnika (Giorgio i ostali, 2017). Sustav za izradu igara Unity jedan je od najpopularnijih alata za izradu AR i VR aplikacija i podržava ga velika aktivna zajednica. U posljednje vrijeme se zbog svoje pristupačnosti i jednostavnosti korištenja koristi u sve više industrija za simulacije industrijskih procesa. U preglednom radu (Wang i ostali, 2015) su predstavili razvoj i funkcionalnost nekoliko projekata iz metalne i energetske industrije u kojima je korišten Unity za vizualizaciju i komunikaciju rezultata simulacije između korisnika. Simulacije su korištene u svrhu mogućnosti pregleda opreme i kreiranja interaktivnih scenarija obuke radnika. Postoji nekoliko radova koji su istražili potencijal korištenja sustava Unity za komunikaciju s robotskim operacijskim sustavom (ROS) i oblikovanje sučelja između čovjeka i robota (engl. Human Robot Interface – HRI). U radu (Bartneck i ostali, 2015) sva komunikacijska logika i HRI implementirani su unutar Unity okruženja. Unity su opisali prikladnim za programiranje ponašanja i interakcija robota zbog dostupnosti skupa alata za animiranje i vizualno programiranje korištenih u razvoju igara. Pristup za simulaciju robotskog zadatka varenja u sustavu Unity predstavljen je u (Pan i ostali, 2016). Razvijen je programski okvir za simulaciju nadzora industrijskog procesa u sustavu Unity koji komunicira s robotskim hardverom preko operacijskog sustava robota ROS (Sita i ostali, 2017). Programski okvir prima informacije o stanju sustava od ROS u stvarnom vremenu i koristi ih za sinkronizaciju animiranog 3D okruženja, ali ne može utjecati na operaciju varenja. Autori ističu da je Unity, za razliku od simulacijskog softvera, kao sustav za razvoj igara osmišljen za vizualizaciju modela i interakcija u 3D okruženju i da mu nedostaju prikladni alati za precizno reproduciranje scenarija iz stvarnog svijeta s obzirom na dinamiku i fiziku sustava. Tvrtka Osaro, koja se bavi implementacijom umjetne

inteligencije za robote korištene u industrijskoj automatizaciji, koristi sučelje implementirano u sustavu Unity za planiranje putanje robota i simulaciju obavljanja zadatka kupljenja predmeta za različite modele robota. Glavna prednost sustava Unity leži u lakoći korištenja i fleksibilnosti dizajniranja korisničkog sučelja i interakcije sa simulacijom. Ovisno od svrhe simulacije, ovaj kompromis može biti dovoljno isplativ razlog za korištenje sustava Unity umjesto simulacijskog softvera.

2. Opis sustava

Sustav se sastoji od robotske ruke, predmeta koji je potrebno izrezati i korisničkog sučelja koje korisnik koristi za upravljanje robotom. Koristeći sučelje, korisnik može odabrati jedan od trodimenzionalnih predmeta za rezanje. Na zahtjev korisnika robotska ruka mijenja položaj alata za rezanje i reže predmet na dva jednaka dijela. Nakon završetka rezanja, polovice objekta se odvajaju i na korisničkom sučelju se ispisuje učinkovitost robota.

Prethodno je potrebno definirati način pokretanja robotske ruke i odrediti ravninu po kojoj robot treba sjeći kako bi prepolovio objekt. Za ostvarenje simulacije rezanja potrebno je pomicati alat za rezanje duž granica presjeka ravnine rezanja s objektom i odvojiti mrežu trokutova koja gradi objekt na dvije polovice odvojene ravninom. Kôd 2.1 opisuje pseudokod algoritma za rezanje objekta na visokoj razini:

```
CutObject(object) {
  plane = FindCuttingPlane(object.mesh)
  addedVertices[], leftMesh, rightMesh =
    CutMesh(object.mesh, plane)
  trajectory[] = GetTrajectory(addedVertices)
  for (i = 0; i < trajectory.length; ++i) {
    jointAngles[] = InverseKinematics(trajectory[i])
    RotateJoints(jointAngles)
  }
  GenerateHalves(leftMesh, rightMesh)
}
```

Kôd 2.1 – Pseudokod općenitog algoritma sustava za rezanje objekta

Najprije se odredi ravnina koja odvaja mrežu objekta na dvije polovice. U ovu svrhu se koristi algoritam optimizacije parametara ravnine kako bi se odredila ravnina koja što preciznije odvaja polovice. Potom se dobivena ravnina koristi kako bi se svaki trokut mreže objekta dodijelio jednoj od polovina. U ovom koraku se generiraju i dodatni vrhovi koji upotpunjuju trokutove mreža polovina presječenih ravninom. Ovi vrhovi čine rub mreža polovica na mjestima presjeka s ravninom. Isti vrhovi čine putanju robotske ruke, poredani u smjeru kazaljke na satu od središta objekta. Kako bi se postigla simulacija rezanja, robotska ruka pokreće vrh alata za rezanje redoslijedom točkama putanje. Za svaku točku putanje, algoritmom inverzne kinematike izračunava se ciljna orijentacija kutova zglobova robota kako bi vrh alata dospio u zadanu točku, uzevši u obzir ograničenja kretanja. Konačno, nakon

što je robotska ruka obišla zadanu putanju, na prethodno odijeljene polovice objekta se primjenjuje simulacija fizike krutog tijela i polovice se fizički odvajaju.

U nastavku su detaljnije opisani pojedini algoritmi korišteni u prethodno opisanom pseudokodu. Najprije su opisani algoritmi računanja polovišne ravnine i odvajanja mreže objekta po dobivenoj ravnini. Potom je ukratko opisan način računanja putanje robotske ruke. Na kraju je opisan model robotske ruke korištene u ovom radu i način određivanja orijentacije zglobova robota algoritmom inverzne kinematike.

2.1. Rezanje objekta

Za rezanje objekta potrebno je odrediti ravninu po kojoj robot treba sjeći kako bi prepolovio objekt i odvojiti mrežu trokutova koja gradi objekt na dvije polovice odvojene ravninom. Algoritam koji računa polovišnu ravninu objekta za mrežu zadanog objekta generira ravninu koja prolazi ishodištem objekta i polovi objekt na dva dijela približno jednakih volumena. Najprije je potrebno odrediti volumen zadanog objekta iz njegove mreže. Mreža je izgrađena od trokutova od kojih je svaki definiran s tri vrha. Prema algoritmu predstavljenom u radu (Zhang i Chen, 2001), volumen objekta je moguće izračunati iz njegove mreže trokutova sumiranjem algebarskih izraza koji ovise o koordinatama tih trokutova, po izrazu (1).

$$V = \sum_i \frac{1}{6}(-x_{i3}y_{i2}z_{i1} + x_{i2}y_{i3}z_{i1} + x_{i3}y_{i1}z_{i2} - x_{i1}y_{i3}z_{i2} - x_{i2}y_{i1}z_{i3} + x_{i1}y_{i2}z_{i3}) \quad (1)$$

Nakon što je izračunat volumen objekta, potrebno je odrediti polovišnu ravninu. Polovišna ravnina polovi objekt na dvije mreže jednakih volumena tako da vrijedi $V = V_1 + V_2$ i $|V_1 - V_2| = 0$. U ovu svrhu je korišten postupak pronalaženja optimuma funkcije s ciljem minimizacije funkcije $f = |V_1 - V_2|$, uz određeni broj iteracija. Rezultat algoritma su parametri jednadžbe ravnine (A, B, C) koja prolazi kroz ishodište (D = 0) i polovi zadani objekt. Kako bi smanjili prostor pretrage, zbog simetričnosti prostora rješenja možemo fiksirati jedan parametar (npr. C) kao pozitivan (C > 0).

Za postupak pronalaženja optimuma funkcije korišten je optimizacijski postupak s ograničenjima po Boxu (Box, 1965), zasnovan na ideji simpleks postupka (Nelder i Mead, 1965). Postupak koristi simpleks strukturu zatvorene petlje od $k = 2n$ točaka, gdje je n broj parametara koji se optimiraju. Simpleks struktura se koristi točkom centroida, koja predstavlja središnju točku petlje. Početne točke simpleksa se proizvoljno generiraju tako da zadovoljavaju postavljena ograničenja, a konačno rješenje se postiže primicanjem najgorih

točaka strukture prema centroidu operacijom refleksije, u nekoliko iteracija. Algoritam se zaustavlja kada simpleks uzastopnim kontrakcijama postane dovoljno malen, pronađeno je egzaktno rješenje ili ako se vrijednost funkcije cilja nije poboljšala zadnjih stotinu iteracija, kada algoritam divergira. Za razliku od originalnog algoritma, u ovom radu je za konačno rješenje uzeta najbolja točka simpleksa umjesto centroida konačnog simpleksa. Parametri ravnine su eksplicitno ograničeni na proizvoljne cjelobrojne vrijednosti, a kao implicitno ograničenje je postavljeno ograničenje $A \neq 0 \wedge B \neq 0 \wedge C \neq 0$, budući da takva konfiguracija parametara ne čini ravninu.

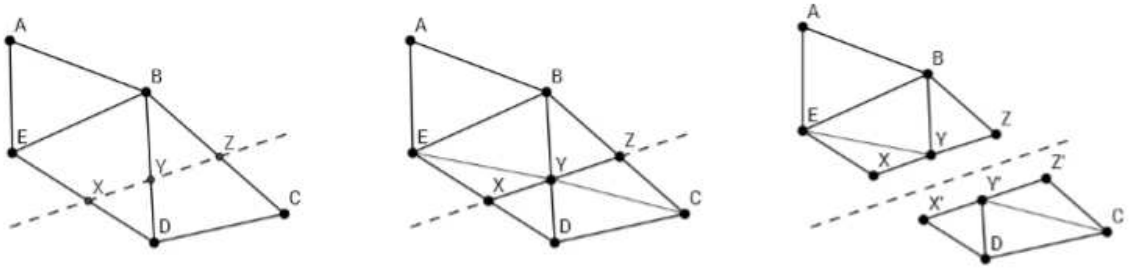
Dobivena polovišna ravnina odvaja trokutove koji čine mrežu objekta. Za svaki trokut mreže se provjerava kojoj polovici mreže odvojene ravninom pripada. Trokutovi mreže u cijelosti iznad ravnine su dodani u lijevu mrežu, a trokutovi u cijelosti ispod ravnine u desnu mrežu. Za točku $T(x, y, z)$ iznad ravnine vrijedi $Ax + By + Cz > 0$, a za točku ispod ravnine $Ax + By + Cz < 0$.

Iz vrhova trokutova koje ravnina presijeca na dva dijela i točaka presjeka ravnine s bridovima trokutova su konstruirani novi trokutovi i dodijeljeni odgovarajućim polovinama objekta. Svaki od presječenih trokutova je presječen tako da se jedan od njegovih vrhova nalazi na jednoj strani ravnine, a druga dva na suprotnoj strani. Drugim riječima, svaki od tih trokutova sadrži dva brida koje ravnina presijeca na dva dijela, dok je treći brid u potpunosti na jednoj strani ravnine. Novi vrhovi se generiraju na mjestima gdje ravnina presijeca ta dva brida. Ovaj proces prikazuje Slika 2.1. Normaliziranu udaljenost točke presjeka ravnine s bridom moguće je dobiti prema izrazu (2), gdje $\vec{n}$ predstavlja normalu ravnine, $\vec{p}$ točku presjeka ravnine s bridom, a $\vec{a}$ i $\vec{b}$ vrhove presječenog brida trokuta (Pregun, 2017).

$$d = \frac{(\vec{p} - \vec{a})\,\vec{n}}{(\vec{b} - \vec{a})\,n} \tag{2}$$

Normalizirana udaljenost daje vrijednost između 0 i 1 za presječeni brid te je pomoću nje moguće jednostavno konstruirati novi vrh trokuta linearnom interpolacijom između vrhova trokuta. Novi vrh se interpolira prema izrazu (3).

$$\vec{x} = d\,\vec{b} + (1 - d)\,\vec{a} \tag{3}$$



Slika 2.1 Proces rezanja mreže trokutova

Konačno, algoritmom triangulacije novonastalih poligona na ravnini presjeka se generiraju novi trokutovi koji pokrivaju strane mreže gdje je napravljen presjek. Susjedni vrhovi nastalih poligona jednostavno su povezani sa središtem tog poligona, čineći jednak broj trokutova. Ovime je cijela geometrija objekta podijeljena na dvije odvojene mreže trokutova.

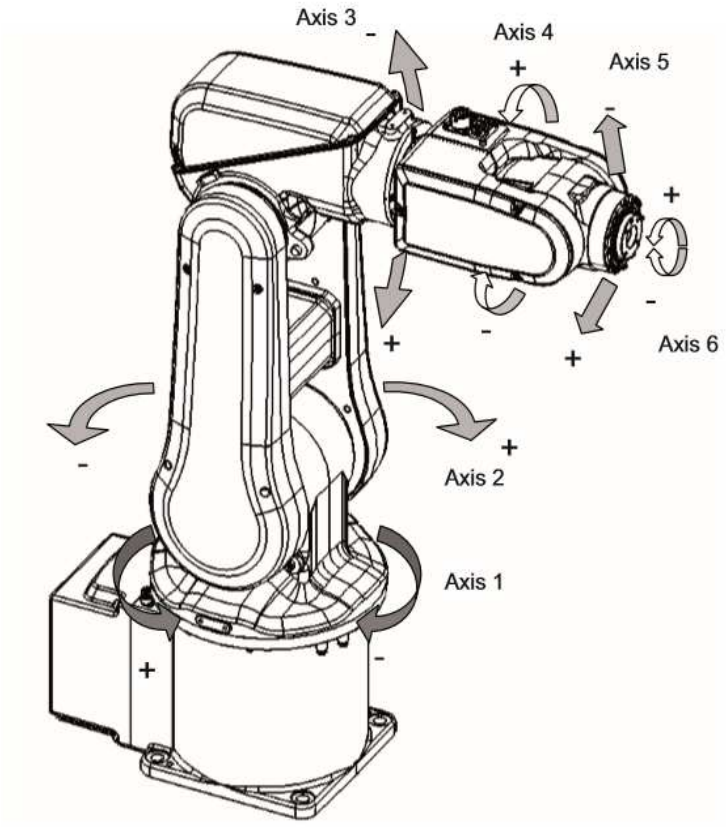
2.2. Putanja kretanja robotske ruke

Prethodno dobiveni vrhovi trokutova na presjeku ravnine s mrežom objekta se koriste kao diskretne točke između koji se interpolira kako bi se generirale kontinuirane točke putanje robotske ruke u stvarnom vremenu simulacije. Robot mijenja poziciju vrha alata za rezanje iz trenutne u sljedeću točku krećući se po bridovima poligona dobivenog presjekom ravnine s objektom.

Za dobivanje prirodnije putanje kretanja, točke putanje moraju biti poredane redoslijedom kojim je očuvan poredak susjednih točaka u poligonu. Tijekom rezanja objekta, zapamćeni su svi bridovi koji nastaju presijecanjem trokutova mreže ravninom. Ovi bridovi su spojeni zajedničkim vrhovima u jednu ili više zatvorenih petlji, tvoreći time planarne poligone koji leže na ravnini rezanja. Na kraj putanje se ponovno dodaje početna točka kako bi se robot nakon završetka rezanja vratio u početnu poziciju prije nego je započeto rezanje. Zbog ovakvog načina definiranja putanje, robot će ispravno odrediti putanju od više zatvorenih petlji u slučajevima presjeka ravnine s konkavnim dijelovima objekta. Tada je potrebno generirati nekoliko vanjskih točaka između petlji kako bi robot glatko promijenio poziciju s jedne na drugu petlju.

2.3. Model robotske ruke

Robotska ruka korištena u ovom radu simulirana je prema modelu robotske ruke IRB 120 proizvođača ABB Robotics. Ovaj model je industrijski robot najnovije generacije sa 6 rotacijskih stupnjeva slobode gibanja (DOF, Degrees of Freedom). Osmišljen je specifično za proizvodne industrije kojima je potrebna fleksibilna automatizacija robotima (Product specification IRB 120, 2019). Prva tri rotacijska zgloba upravljaju pozicijom alata, a druga tri upravljaju njegovom orijentacijom. Model robotske ruke IRB 120 s označenim smjerovima rotacije pojedinih zglobova prikazuje Slika 2.2 (Product specification IRB 120, 2019).



Slika 2.2 Model robota IRB 120 (Product specification IRB 120, 2019)

Informacije o zglobovima prikazuje Tablica 2.1 (Product specification IRB 120, 2019).
Ostali tehnički podaci preuzeti iz specifikacije proizvoda su nabrojani u tablici Tablica 2.2.

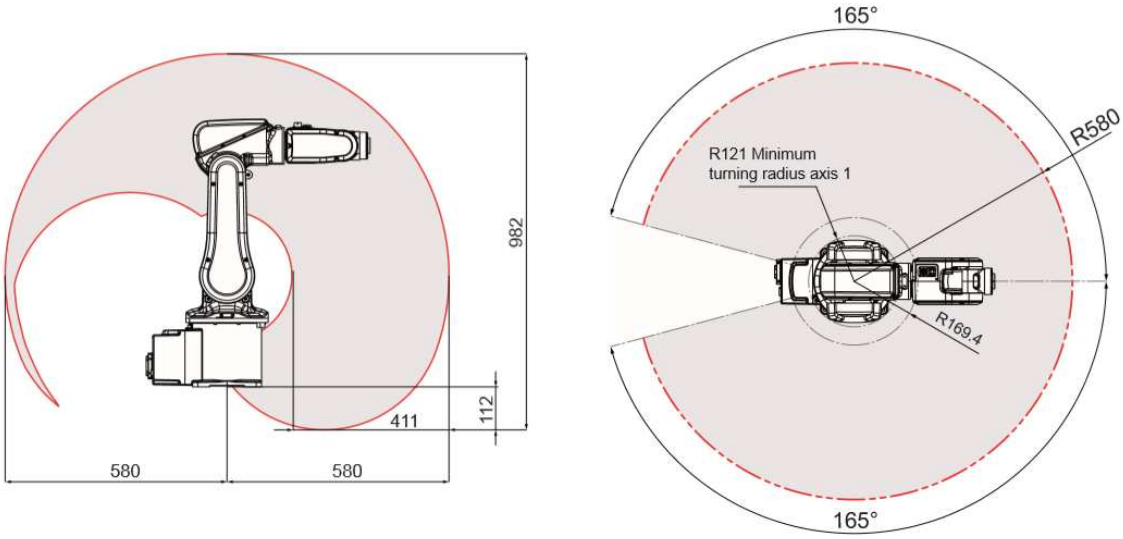
Tablica 2.1 Informacije o zglobovima

Os	Kretanje	Radni opseg	Brzina zgloba
1	struk	±165°	250 °/s
2	rame	±110°	250 °/s
3	lakat	+70° do -110°	250 °/s
4	zavrtanje	±160°	320 °/s
5	nagibanje	±120°	320 °/s
6	zakretanje	±400° (proizvoljno)	420 °/s

Tablica 2.2 Tehnička obilježja robota IRB 120

Broj stupnjeva slobode gibanja	Masa	Nosivost	Radni doseg
6	25 kg	3 kg	0,58 m

Radni prostor robota određuje njegov prostorni doseg ograničen njegovim dimenzijama i dizajnom. Radni prostor korištenog modela IRB 120 prikazuje Slika 2.3 (Product specification IRB 120, 2019). Iz radnog prostora ovog modela je vidljivo da robot ima širok pojas okretanja struka i fleksibilan visinski domet. Međutim, robot nema mogućnost fleksibilnog savijanja članaka prema sebi niti rotacije vršnih članaka oko y-osi. O ovim ograničenjima će biti potrebno voditi računa pri postavljanju objekta u prostor kako bi izračunata putanja robota ostala unutar radnog prostora.



Slika 2.3 Radni prostor modela robota IRB 120 (Product specification IRB 120, 2019)

Kinematički model robota opisuje vezu između zglobova i članaka. Zadatak direktne kinematike je odrediti položaj vrha manipulatora robota (alata) u odnosu na referentni mirni koordinatni sustav (baza robota) iz zadane konfiguracije orijentacije zglobova robota. Položaj alata podrazumijeva njegovu poziciju i orijentaciju, a određen je trenutnom rotacijskom konfiguracijom zglobova. Kako bi se odredilo rješenje direktne kinematike, potrebno je pridružiti desno orijentirane koordinatne sustave vrhu alata i svim zglobovima. Osnovni princip rješenja sastoji se od uzastopnih transformacija koordinata vrha alata iz koordinatnog sustava alata do sustava pridruženog bazi robota (Kovačić, 2008). Za definiranje veze koordinatnih sustava zglobova robota uobičajeno je koristiti Denavit-Hartenbergovu (DH) notaciju. Pristup na osnovi transformacije homogenih koordinata omogućava rješavanje problema direktne kinematike u odnosu prema drugim koordinatnim sustavima.

Kinematički parametri u Denavit-Hartenberg-ovoj notaciji:

1. θ_k – kut između susjednih članaka $k-1$ i k oko osi rotacije zgloba k (Kut zgloba),
2. d_k – duljina duž osi rotacije zgloba k između susjednih članaka $k-1$ i k (udaljenost zgloba),
3. a_k – duljina okomice na osi rotacije zglobova k i $k+1$ (duljina članka),
4. α_k – kut između osi rotacije zglobova k i $k+1$ oko članka k (zakret članka)

Kinematičku konfiguraciju n-osnog manipulatora čini ukupno 4n kinematičkih parametara. Za svaku os rotacije tri parametra su konstantna, a četvrti je varijabla zgloba q_k. Za rotacijski zglob vrijedi $q_k = \theta_k$. Vrijednosti konstanti su određene mehaničkom konstrukcijom robota.

Denavit-Hartenbergova (DH) metoda određivanja kinematičkih parametara (Denavit i Hartenberg, 1995) je način sustavnog pridruživanja desno orijentiranih koordinatnih sustava svakom članku otvorenog kinematičkog lanca. Pridruživanjem ortogonalnih koordinatnih sustava se relativni položaj i orijentacija dva susjedna koordinatna sustava mogu jednoznačno opisati s dva spiralna gibanja oko osi zglobova. Spiralno gibanje predstavlja kombinaciju linearnog pomaka po osi s kutnim pomakom oko iste osi. DH parametre modela robota opisanog u ovom radu prikazuje Tablica 2.3. Veličine članaka robota su preuzete iz službenih tehničkih specifikacija.

Tablica 2.3 DH parametri robota IRB 120

Zglob	**θ [rad]**	**d [mm]**	**a [mm]**	**α [rad]**
1	θ_1	290	0	$-\frac{\pi}{2}$
2	$\theta_2 - \frac{\pi}{2}$	0	270	0
3	θ_3	0	70	$-\frac{\pi}{2}$
4	θ_4	302	0	$+\frac{\pi}{2}$
5	θ_5	0	0	$-\frac{\pi}{2}$
6	θ_6	72	0	0

Kinematičkim parametrima određene su matrice transformacija homogenih koordinata sustava, čijim se uzastopnim množenjem dobije matrica složene homogene transformacije koordinata ili matrica manipulatora (ruke). Matrica manipulatora kao rezultat rješavanja problema direktne kinematike predstavlja podlogu za dobivanje analitičkog rješenja inverznog kinematičkog problema.

2.4. Algoritam inverzne kinematike

Kako bi se robotska ruka mogla kretati po prethodno određenoj putanji, potrebno je pozicionirati vrh alata za rezanje na niz točaka prostora na objektu. Problem pokretanja robotske ruke može se definirati kao iterativni problem pronalaska kutova orijentacije zglobova ruke za koje se postiže pomicanje vrha alata u zadanu točku. Tako definiran problem moguće je riješiti rješenjem problema inverzne kinematike.

Rješenja inverzne kinematike robota dijele se na analitičke i numeričke metode. Analitičke metode daju precizna i brza rješenja, ali postojanje njihovih rješenja ovisi o kinematičkoj strukturi robota. Korišteni model robotske ruke IRB 120 slijedi tzv. „321 kinematičku strukturu“, odnosno sadrži šest rotacijskih zglobova i tri uzastopna zgloba čije se osi rotacije sijeku. Po (Pieper, 1968), tako oblikovani industrijski roboti imaju analitičko rješenje problema inverzne kinematike u zatvorenom obliku.

Numeričke metode se oslanjaju na iterativnu optimizaciju u svrhu pronalaska približnog rješenja, pojednostavljujući postupak izračuna rješenja. Heurističke numeričke metode imaju niske troškove računanja i jednostavne su za implementirati, a brzo dolaze do konačnog rješenja.

Iako za model robota korištenog u ovom radu postoji analitičko rješenje, zbog složenosti postupka dobivanja točnog rješenja u odnosu na činjenicu da numeričke metode daju dovoljno dobre rezultate na jednostavniji način, u ovom radu je za rješavanje problema inverzne kinematike korištena popularna heuristička metoda Cyclic Coordinate Descent ili CCD (Wang, 1991). Ova metoda je jednostavna za implementaciju i prikladna za korišteni model robota jer podržava jednostavno postavljanje ograničenja na osi rotacije i kutove rotacije zglobova.

Osnovna ideja algoritma je iterativno rotirati svaki od članaka robota jedan po jedan, počevši od posljednjeg članka (članak na koji je pričvršćen alat) pa sve do početnog članka (struk), sve dok nije postignuta zadovoljavajuća kvadratna udaljenost pozicije alata od ciljne točke ili je dostignut maksimalan broj iteracija algoritma. Budući da je alat pričvršćen na posljednji članak tako da im se osi rotacije poklapaju, alat nije potrebno dodatno rotirati. Pseudokod rotacije pojedinog članka algoritmom CCD uz postavljena ograničenja na osi rotacije i kutove rotacije zglobova opisan je pseudokodom Kôd 2.2. Funkcija `RotateBone` prima reference na objekte `effector` (alat) i `bone` (članak koji se rotira) i vektor

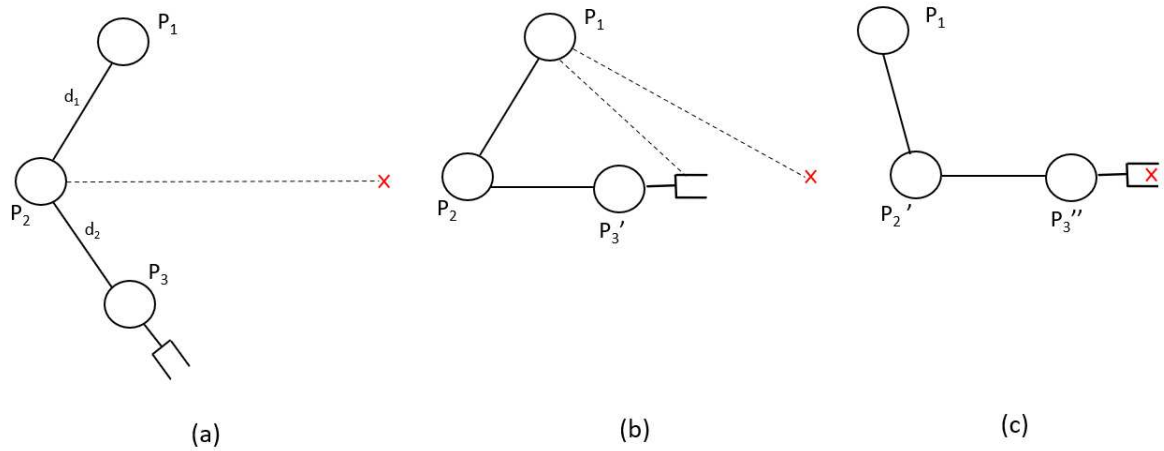
`targetPosition` (ciljna točka). Član `position` podrazumijeva vektor koji predstavlja trenutnu poziciju objekta, a član `rotation` kvaternion koji predstavlja zapis trenutne orijentacije objekta ili operatora rotacije koji se primjenjuje. Funkcija `Q(v1, v2)` konstruira kvaternion koji predstavlja rotaciju iz vektora `v1` u vektor `v2`. Ograničenje na os rotacije zadano je vektorom `axis` usmjerenom u smjeru jedne od koordinatnih osi, a `perpendicular` predstavlja vektor okomit na tu os. Ograničenja na kutove rotacije u oba smjera zadana su vrijednostima `minAngle` $[-\pi, 0]$ i `maxAngle` $[0, \pi]$. Funkcija `ConstrainToNormal` opisana je u nastavku ovog potpoglavlja.

```
RotateBone(effector, bone, targetPosition) {
  // pomakni alat prema cilju
  boneToEffector = effector.position – bone.position
  boneToTarget = targetPosition – bone.position
  Q1 = Q(boneToEffector, boneToTarget)
  bone.rotation = Q1 * bone.rotation

  // primijeni ograničenje na os rotacije
  Q2 = Q(bone.rotation * axis, bone.parent.rotation * axis)
  bone.rotation = Q2 * bone.rotation

  // primijeni ograničenja na kutove rotacije
  perpendicular = axis.perpendicular
  d1 = bone.rotation * perpendicular
  d2 = bone.parent.rotation * perpendicular
  Q3 = Q(d1, ConstrainToNormal(d1, d2, minAngle, maxAngle))
  bone.rotation = Q3 * bone.rotation
}
```

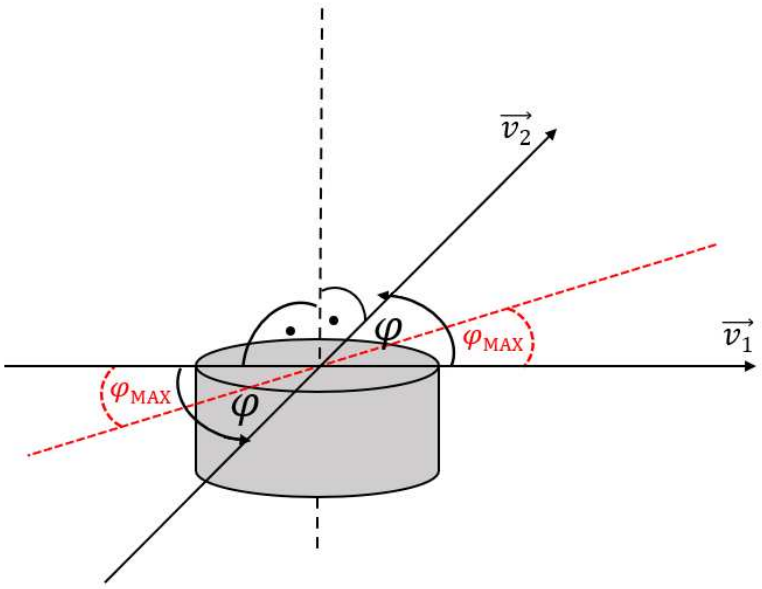
Kôd 2.2 – Pseudokod algoritma rotacije članka algoritmom CCD

Članak se najprije rotira tako da se alat približi cilju. Ova rotacija se računa kao rotacija koju je potrebno napraviti tako da se vektor u smjeru od članka do alata orijentira u smjeru vektora od članka do ciljne točke. Ovaj postupak ilustrira Slika 2.4. U prvoj iteraciji se vršni članak d_2 rotira tako da je alat orijentiran prema spojnici zgloba koji vrši rotaciju i cilja. U idućoj iteraciji članak koji mu prethodi rotira za kut kojim se postiže pomicanje alata u ciljnu točku. Tako dobivena rotacija se ograniči na dozvoljenu os rotacije članka rotacijom koja vraća pomaknutu os rotacije na ispravnu orijentaciju. Za ograničenje kutova rotacije koriste se vektori okomiti na os rotacije članka prije i poslije primjene rotacija, budući da je između tih vektora moguće izračunati kutni pomak. Funkcija `ConstrainToNormal` prima dva

vektora između kojih računa kutni pomak. Kut između vektora moguće je izračunati formulom $\varphi = \arccos\left(\vec{a} \cdot \vec{b}\right)$. Ovisno o predznaku njihovog skalarnog produkta, primjenjuje se ograničenje na kut u pozitivnom ili negativnom smjeru rotacije. Vektor dobiven rotacijom sferno se interpolira prema vektoru prije rotacije za vrijednost relativnog odstupanja trenutnog kuta od najvećeg dozvoljenog. Ovime je određena rotacija koja rotira članak tako da njegov kut rotacije ne prelazi postavljena ograničenja. Opisani postupak ograničenja kutova rotacije ilustrira Slika 2.5. Os rotacije zgloba predstavljena je isprekidanom crnom linijom. Vektor $\overrightarrow{v_2}$ dobiven rotacijom vektora $\overrightarrow{v_1}$ u pozitivnom smjeru za kut φ orijentira se prema smjeru određenom kutom φ_{MAX}.



Slika 2.4 Osnovna ideja CCD algoritma

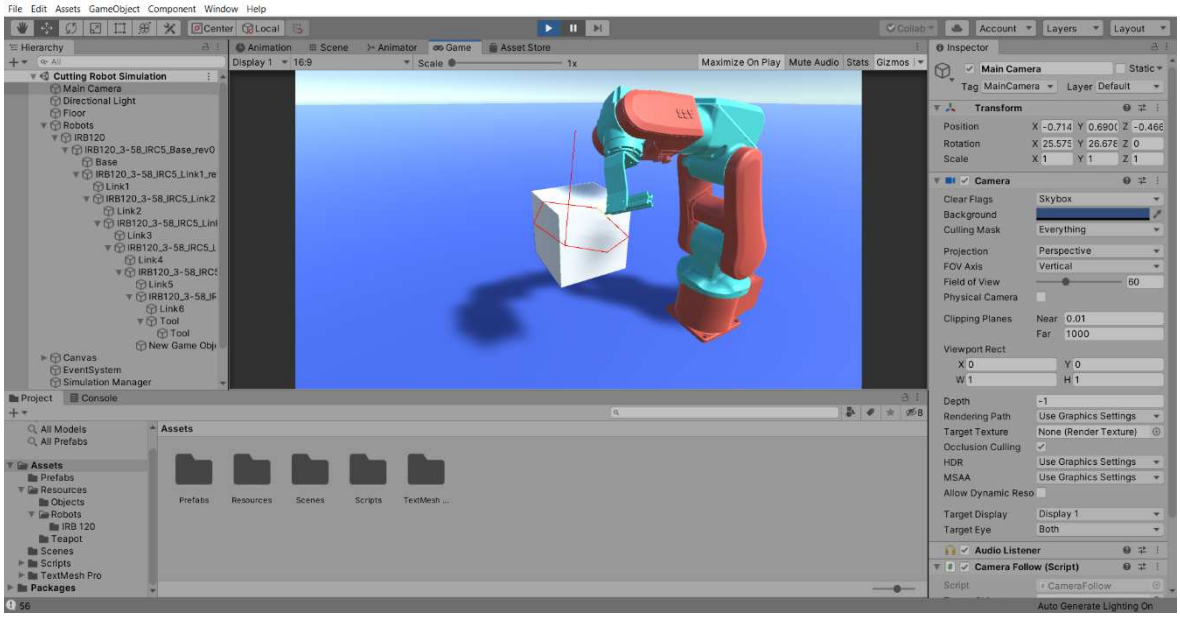


Slika 2.5 Ograničenje kuta rotacije zgloba

3. Programsko rješenje

3.1. Korišteni alati

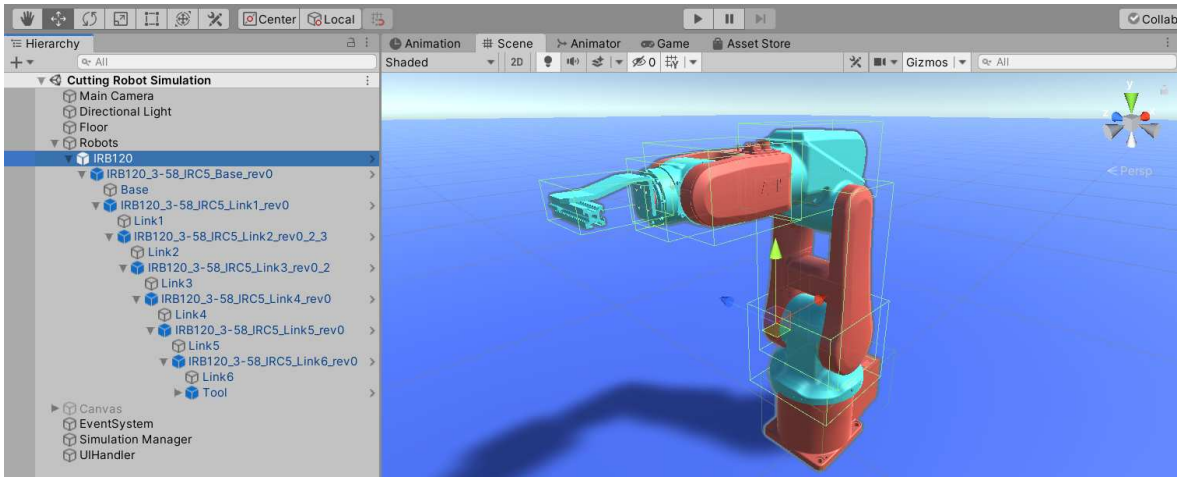
Sustav za upravljanje virtualnim robotom razvijen je koristeći sustav za izradu igara Unity i programski jezik C#. Unity omogućuje izradu igara i simulacija u trodimenzionalnom ili dvodimenzionalnom prostoru, kao i virtualnoj ili proširenoj stvarnosti. Igre izrađene pomoću sustava Unity izgrađene su od više scena koje čine razine igre ili zasebne virtualne svjetove. Svaka scena sadrži objekte koji grade okruženja i korisnička sučelja igre. Objekti (GameObjects) su osnovni građevni element od kojih su izgrađene scene. Sustav Unity omogućuje jednostavno manipuliranje svojstvima objekata poput njihovog položaja, orijentacije te veličine povlačenjem modela objekata u sceni. Primjer izgleda otvorenog projekta u sustavu Unity prikazuje Slika 3.1.



Slika 3.1 Primjer otvorenog projekta u sustavu Unity

Objektima je moguće dodijeliti komponente koje im daju određena fizikalna svojstva. Unity sadrži veliki broj ugrađenih komponenti korisnih pri grafičkim izračunima i fizičkim simulacijama. Vrijednosti parametara komponenti je moguće postaviti izmjenom vrijednosti u prozorima alata Unity. Izgradnja sustava se svodi na sastavljanje objekata poput komponenti, koje su ponovno izgrađene od manjih komponenti. Ovakav način izgradnje sustava znatno olakšava proces izrade simulacija i ubrzava testiranje funkcionalnosti. Komponente objekata mogu biti i korisnički definirane programske skripte napisane u jeziku

C#. Skriptama je definirano posebno ponašanje objekta. Koristeći tu mogućnost je definirano ponašanje sustava simulatora, dok je model objekta robota učitan izravno u scenu. Izgled scene s robotom prikazuje Slika 3.2.



Slika 3.2 Izgled scene s robotom

3.2. Struktura projekta

Standardna struktura projekta u sustavu Unity sadrži direktorij `Assets` u kojem su pohranjeni svi materijali projekta, uključujući skripte, modele objekata, teksture, zvukove i slične materijale. Unutar tog direktorija, programske skripte su pohranjene u direktoriju `Scripts` , a direktorij `Resources` objekte koje je moguće dinamički učitati i modele ostalih objekata. Sustav simulatora na početku simulacije dinamički učitava modele objekata za rezanje iz `.obj` datoteka i pohranjuje ih uz ostale informacije o objektu u Prefab datoteke. Unity koristi ovaj format datoteke za pohranu objekata sa svim svojim komponentama kako bi se mogli lako ponovno koristiti. Modeli objekata se učitavaju iz direktorija `Resources/CuttingObjects`, gdje ih je potrebno prethodno pripremiti. Direktorij `Scripts` sadrži skripte koje pokreću simulaciju i pomoćne klase za obavljanje izračuna. U nastavku su opisane glavne klase koje upravljaju simulacijom.

3.2.1. Klasa SimulationManager

`SimulationManager` je glavna klasa zadužena za upravljanje fazama simulacije i organizaciju komponenti sustava. Pri pokretanju simulacije, dinamički učitava objekte za rezanje iz direktorija `Resources/CuttingObjects` i čeka zahtjev za rezanjem. Na zahtjev korisnika započinje proces rezanja trenutno odabranog objekta.

Komunikacija s klasom `IKManager` ostvarena je pomoću zajedničkog automata stanja s prijelazima ovisnim o trenutnoj fazi simulatora. Simulator neće nastaviti s izvođenjem iduće faze simulacije dok prethodna faza nije u potpunosti izvršena. Ovakav način komunikacije je potreban kako bi `SimulationManager` mogao pričekati da se ekran osvježi dovoljan broj puta kako bi se simulacija kretanja uspjela prikazati u cijelosti prije idućeg izračuna. Time je cijela simulacija podijeljena u nekoliko faza. Iako su u implementaciji ove faze podijeljene u više metoda, zbog preglednosti Kôd 3.1 sažeto prikazuje pozive funkcija svih faza simulacije u funkciji `CutObject`.

```
public void CutObject()
{
    // 1 - Calculate halving plane
    var meshFilter =
cuttingObject.GetComponent<MeshFilter>();
    var MeshStruct = new MeshStruct(meshFilter);
    Cutter.OriginalMesh = MeshStruct;
    var Volume =
VolumeDifferenceFunction.VolumeOfMesh(meshFilter);
    f = new GoalFunction(new
VolumeDifferenceFunction(MeshStruct, Volume));
    new Thread(new ThreadStart(FindPlaneNormal)).Start();

    // Find plane normal
    var x0 = new List<double>() { 0.5f, 0.5f, 0.5f };
    var lowerBound = new List<double>() { -5, -5, 0 };
    var upperBound = new List<double>() { 5, 5, 5 };
    var implicitConstraint = new ImplicitConstraint();
    var normal = f.Box(x0, lowerBound, upperBound,
implicitConstraint);
    plane = new Plane(normal.normalized, 0);
    Cutter.Cut(cuttingObject, plane);

    // 2 - Move robot
    IKManager.Instance.StartMoving(plane, cuttingObject);

    // 3 - Separate meshes
    Cutter.GenerateMeshes(cuttingObject);
    StartCoroutine(SeparateMeshes());
}
```

Kôd 3.1 – Sažeti prikaz funkcija za izvršavanje faza simulacije rezanja

U početnoj fazi simulacije pripremaju se strukture koje sadrže vrijednosti iz mreže objekta za rezanje potrebne za izračun normale polovišne ravnine. Zbog potencijalne vremenske zahtjevnosti izračuna, posao pronalaska ravnine se obavlja na odvojenoj dretvi. U svakoj iteraciji algoritma optimizacije, pri izračunavanju vrijednosti ciljne funkcije za trenutne parametre ravnine se iznos volumena mreže trokutova računa za samo jednu polovicu objekta. Pritom su u mrežu trokutova te polovice uključeni svi trokutovi s te strane ravnine, kao i trokutovi nastali rezanjem postojećih trokutova mreže originalnog objekta na dva dijela. Volumen druge polovice se dobije kao razlika izračunatog volumena prve polovice od ukupnog volumena objekta. Ovime je izbjegnuto nepotrebno dvostruko računanje volumena u svakoj iteraciji i postignuto djelomično ubrzanje algoritma. Dodatno ubrzanje je postignuto time što se izračunate vrijednosti funkcije cilja za svaku točku pohranjuju u tablicu sažimanja tako da se za iste točke umjesto ponovnog računanja dohvaćaju izračunate vrijednosti iz tablice. Nakon što je pronađena ravnina, `IKManager` može nastaviti s idućom fazom simulacije, pokretanjem robotske ruke po putanji. Ova faza je opisana nešto kasnije u kontekstu klase `IKManager`. Treća faza simulacije počinje kada `IKManager` dojavi da je robot završio s kretanjem i da se objekt može prepoloviti. Generirane polovice objekta se odvajaju jedna od druge u smjeru normale ravnine rezanja i time je simulacija završena.

3.2.2. Klasa IKManager

Ova klasa je zadužena za generiranje putanje robota iz presjeka ravnine sa zadanim objektom i za pokretanje robota po izračunatoj putanji. Pri početku druge faze simulacije, `IKManager` iz prethodno sačuvanih parova bridova nastalih rezanjem mreže trokutova objekta povezuje njihove vrhove u zatvorene petlje oko površine modela objekta. Ako razmak između susjednih točaka prelazi određeni prag, u točke putanje između njih se dodaje niz interpoliranih točaka kako bi se robot glatko kretao.

Ako je dostupna iduća točka do koje se robot treba pokrenuti, algoritam inverzne kinematike računa valjanu orijentaciju članaka kako bi usmjerio vrh alata prema točki putanje. Pritom se članci rotiraju prema ciljnoj točki u svakoj iteraciji. Stoga se njihova trenutna orijentacija vraća na prethodnu i rotacija se glatko animira interpoliranjem do iduće orijentacije. Prije računanja svake iduće orijentacije članaka, trenutna orijentacija se pohranjuje kao početna orijentacija za iduću rotacije. Završetkom izračuna algoritma inverzne kinematike, završna orijentacija se pohranjuje kao ciljna orijentacija i robot se trenutno postavlja na svoju prethodnu orijentaciju. Tada se robotska ruka rotira u idući položaj interpolacijom između

početne i ciljne orijentacije. Nakon što je animacija promjene položaja obavljena u cijelosti, iduća ciljna točka putanje postaje dostupna i proces se ponavlja.

Algoritam inverzne kinematike koji računa orijentaciju članaka robota iz zadane ciljne točke slijedi pseudokod CCD algoritma opisanog u potpoglavlju 2.4, uz nekoliko implementacijski specifičnih funkcionalnosti. Od implementacijskih razlika istaknuti su orijentiranje alata prema definiranom smjeru i ugrađeno izbjegavanje sudara robota s rezanim objektom.

Prilikom orijentacije robota, zbog prirode simulacije je prikladno podesiti smjer igle za rezanje na alatu okomito na površinu objekta. Definirano načinom dizajna korištenog modela robota, orijentaciju alata za rezanje u zadani vektor je moguće podesiti rotiranjem posljednja tri vršna članka robotske ruke. Stoga se posljednja tri članka robota rotiraju prema prikladnom smjeru okomitom na objekt, umjesto rotiranja izravno prema cilju. Okomiti smjer se računa kao vektorski produkt normale ravnine rezanja i smjera kretanja točaka putanje. Kôd funkcije koja rješava problem inverzne kinematike robota prikazuje Kôd 3.2.

```
public void InverseKinematics(Vector3 target)
{
  Vector3 targetPosition = target;
  Vector3 effectorPosition = GetTooltip(Bones[0]);

  int iterationsCount = 0;
  float sqrDistance;

  do
  {
    for (int i = 1; i < Bones.Count; ++i)
    {
    RotateBone(GetTooltip(Bones[0]), tooltipDir, Bones[i],
targetPosition, Joints[i], rotateToDirection = i < 4);
    sqrDistance = (GetTooltip(Bones[0]) -
targetPosition).sqrMagnitude;
    if (sqrDistance <= sqrDistError)
      break;
    }

    // Find the closest point
    for (int i = Bones.Count - 1; i >= 0; --i)
    {
```

```
      var boxCollider =
Bones[i].GetComponentInChildren<BoxCollider>();
      Vector3 jointPoint =
boxCollider.ClosestPoint(cuttingObject.transform.position);
      if (jointPoint.Equals(cuttingObject.transform.
position)) {
        jointPoint = Bones[i].position;
      }
      Vector3 offset = jointPoint -
cuttingObject.transform.position;

      Vector3 meshHitPoint;
      RaycastHit hit;
      Ray ray = new Ray(cuttingObject.transform.position,
offset.normalized);
      ray.origin = ray.GetPoint(100f);
      ray.direction = (-ray.direction).normalized;
      bool didHit =
cuttingObject.GetComponent<MeshCollider>().Raycast(ray, out
hit, Mathf.Infinity);
      if (!didHit) continue;

      var objectOffsetLen = (hit.point -
cuttingObject.transform.position).magnitude;
      if (offset.magnitude < objectOffsetLen)
      {
        nearestPoint.parent = Bones[j].transform;
        nearestPoint.position = jointPoint;
        displacementTarget.position =
nearestPoint.transform.position + (offset.normalized *
(objectOffsetLen - offset.magnitude));

        // Move away from object
        for (int j = i; j < Bones.Count; ++j)
        {
          // Skip effector
          if (j == 0) continue;
          RotateBone(nearestPoint.position, tooltipDir,
Bones[j], displacementTarget.position, Joints[j]);
        }
      }
```

```
        }

        sqrDistance = (GetTooltip(Bones[0]) -
targetPosition).sqrMagnitude;
        ++iterationsCount;
      }
      while (sqrDistance > sqrDistError && iterationsCount <=
Iterations);
      StoreAndResetRotations();
}
```

Kôd 3.2 – Implementacija algoritma inverzne kinematike

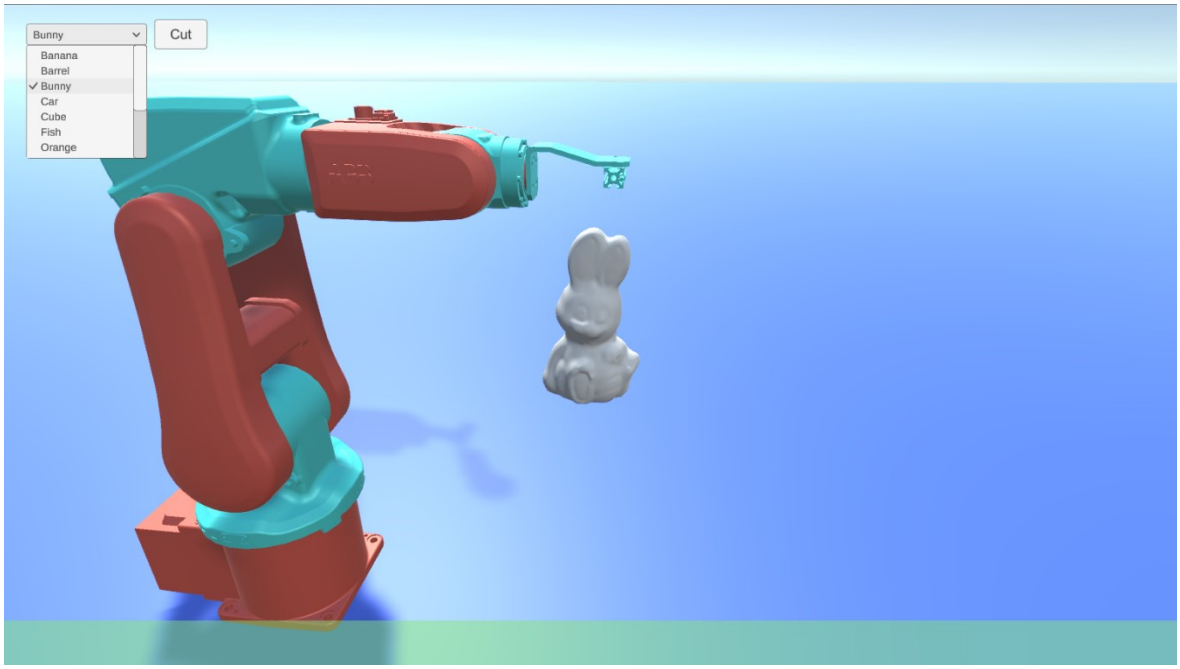
Podaci o člancima robota su pohranjeni u polje suprotnim redoslijedom, s alatom kao početnim članom, jer se istim redoslijedom članci rotiraju u ciljnu orijentaciju. Rotacija članka se obavlja na način opisan ranije u pseudokodu Kôd 2.2, s razlikom da se posljednja tri članka prvom rotacijom orijentiraju prema smjeru okomitom na rezani objekt.

Prilikom računanja rješenja inverzne kinematike, u implementaciji je potrebno uzeti u obzir i izbjegavanje sudara robotske ruke s objektom koji se reže. Iako sustav Unity podržava detekciju sudara ugrađenim komponentama, ugrađene komponente za detekciju sudara je moguće koristiti samo ako se objekti čiji sudar testiramo ponašaju kao kruta tijela, što u ovom slučaju daje neželjene rezultate simulacije. Stoga je razvijeno algoritamsko rješenje koje prilikom računanja rješenja inverzne kinematike, nakon rotiranja članaka prema cilju, pokušava odmaknuti članke robota od rezanog objekta tako da površinu objekta dodiruje isključivo vrhom alata za rezanje. Algoritam za svaki članak računa točku na članku najbližu središtu objekta i točku na mreži trokutova rezanog objekta na pravcu usmjerenom od središta objekta prema izračunatoj točki na članku. Ako je udaljenost od središta objekta do točke na članku manja od udaljenosti od središta objekta do točke na mreži objekta, članak je u sudaru s predmetom. Tada se taj članak i svi prethodni članci rotiraju tako da se točka dodira na članku pomakne za potrebnu razliku udaljenosti, na isti način na koji se prethodno pozicionira vrh alata u ciljnu točku.

3.3. Korisničko sučelje

Simulirana scena sadrži robota i trenutno odabrani objekt za rezanje. Korisničko sučelje omogućuje odabir objekta s popisa objekata prethodno pripremljenih u direktoriju `Resources/CuttingObjects`. Odabirom objekta s popisa se iscrtani objekt

automatski mijenja u odabrani. Odabir novog objekta je onemogućen dok traje prethodno pokrenuta simulacija rezanja. Izgled simulatora uz mogućnost odabira objekta za rezanje s popisa prikazuje Slika 3.3.

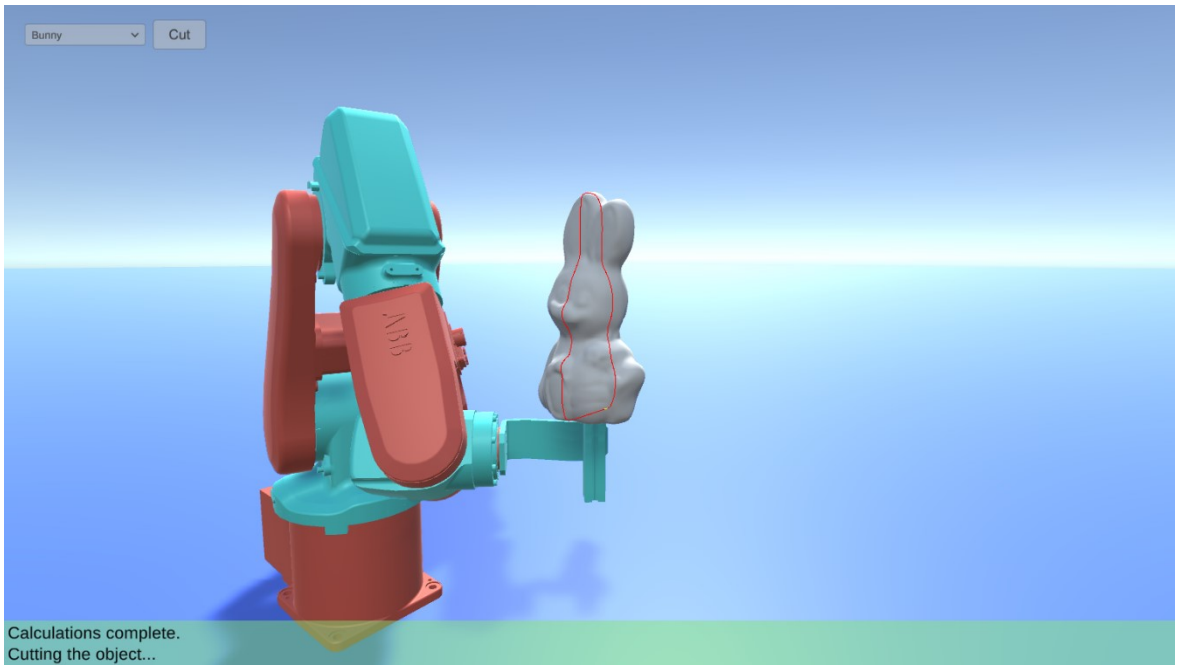


Slika 3.3 Izgled simulatora

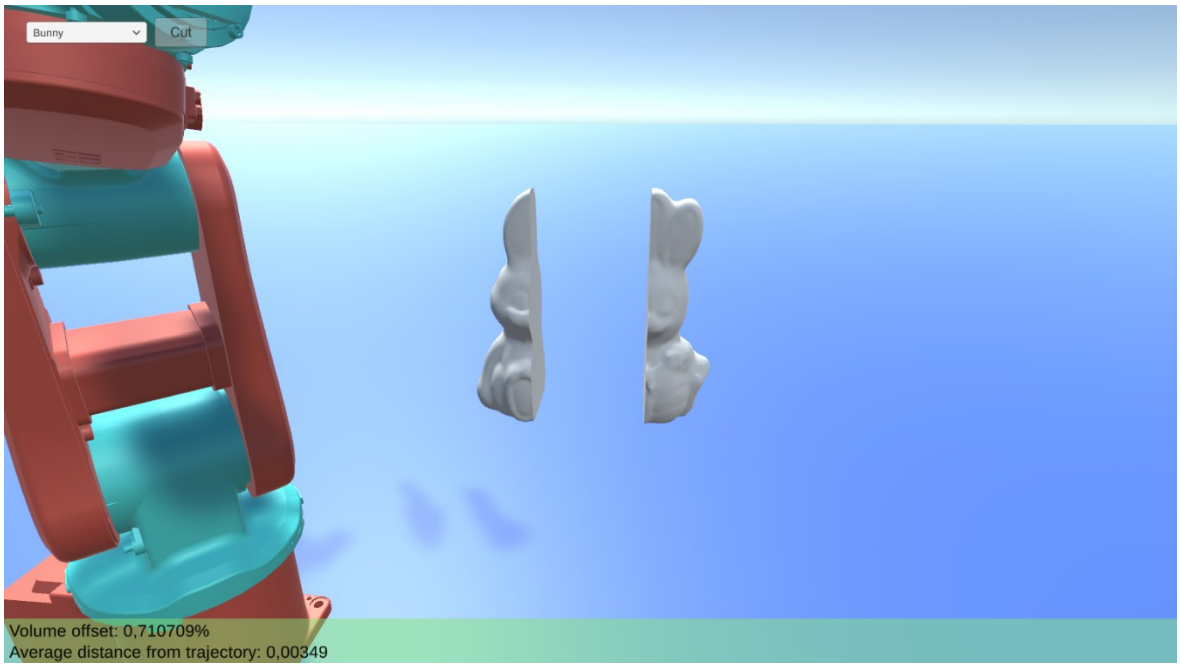
Statusna traka je prikazana zelenim transparentnim okvirom na dnu prozora. Tekst statusne trake se mijenja završetkom pojedinih faza simulacije kako bi korisnik mogao pratiti trenutnu fazu simulacije. Nakon izvršenja simulacije, statusna traka prikazuje metrike koje opisuju uspješnost robota.

Pokretanje simulacije rezanja odabranog objekta započinje pritiskom na gumb s oznakom „Cut“. Nakon računanja normale ravnine, simulator počinje pokretati robota. Robotska ruka se pokreće po putanji iz okvira u okvir dok ne dovrši petlju i vrati se u početni položaj. Pri testiranju simulacije unutar sustava Unity putanja robotske ruke je iscrtana crvenom linijom. Pokrenuti proces rezanja kretanjem robota po putanji prikazuje Slika 3.4. Po završetku rezanja, polovice mreže objekta se odvajaju i odmiču dalje od druge u smjeru normale ravnine rezanja. Primjer prepolovljenog objekta prikazuje Slika 3.5. Nakon završetka simulacije rezanja je moguće odabrati drugi objekt s liste objekata i ponovno pokrenuti simulaciju.

Tijekom simulacije je u svakom trenutku moguće mijenjati kut gledanja na objekt mijenjanjem položaja kamere na koordinatnim osima. Položaj kamere je moguće mijenjati pomoću tipkovnice koristeći tipke sa strelicama te tipke `W` i `S`.



Slika 3.4 Proces rezanja objekta u tijeku



Slika 3.5 Prepolovljeni objekt kao rezultat simulacije

4. Rezultati i diskusija

Simulator je testiran na nekoliko objekata različitih složenosti mreža trokutova i obilježja oblika poput zakrivljenosti i ispupčenosti. Vrednovanje učinkovitosti robota na pojedinom objektu ostvareno je mjerenjem učinkovitosti algoritma optimizacije ravnine i preciznosti pozicioniranja vrha alata na površinu objekta.

Kao metrike za mjerenje učinkovitosti algoritma optimizacije ravnine su korišteni relativno odstupanje razlike volumena polovica objekta te vrijeme računanja parametara ravnine u ovisnosti o složenosti strukture mreže trokutova objekta. Relativno odstupanje razlike volumena RE_V je izračunato kao apsolutna razlika volumena polovica u odnosu na ukupni volumen prema izrazu (4) i izraženo je u postotcima.

$$\mathrm{RE}_V = \frac{|\mathrm{V}_1 - \mathrm{V}_2|}{V} \tag{4}$$

Vrijeme potrebno za računanje parametara ravnine koja polovi objekt je mjereno od trenutka poziva funkcije optimizacije do trenutka kada je rješenje pronađeno. Kako se volumen objekta računa sumiranjem algebarskih izraza koji ovise o trokutovima mreže objekta, po izrazu (1), vremenska složenost računanja volumena objekta u jednoj iteraciji linearno raste s brojem trokutova u njegovoj mreži. Stvarno vrijeme izračuna parametara ravnine jednako je vremenu potrebnom optimizacijskom algoritmu da konvergira prema zadovoljavajućem rješenju, što će još ovisiti o simetričnosti i sličnim obilježjima oblika pojedinog objekta.

Za mjerenje preciznosti pozicioniranja vrha alata na površinu objekta je promatrana prosječna dostignuta udaljenost vrha alata od ciljne pozicije. Udaljenost vrha alata od ciljne točke u putanji se računa na kraju rješenja inverzne kinematike za svaku točku putanje. Po završetku simulacije izračunat je njihov prosjek, prema izrazu (5).

$$\bar{d} = \frac{\sum_i d_i}{n} \tag{5}$$

Vrijednosti duljina s kojima sustav Unity računa izražene su u virtualnim mjernim jedinicama specifičnim za Unity. Pretpostavljena vrijednost za jednu jedinicu duljine u sustavu Unity je jedan metar. Ista pretpostavka je korištena u prikazu rezultata u ovom radu.

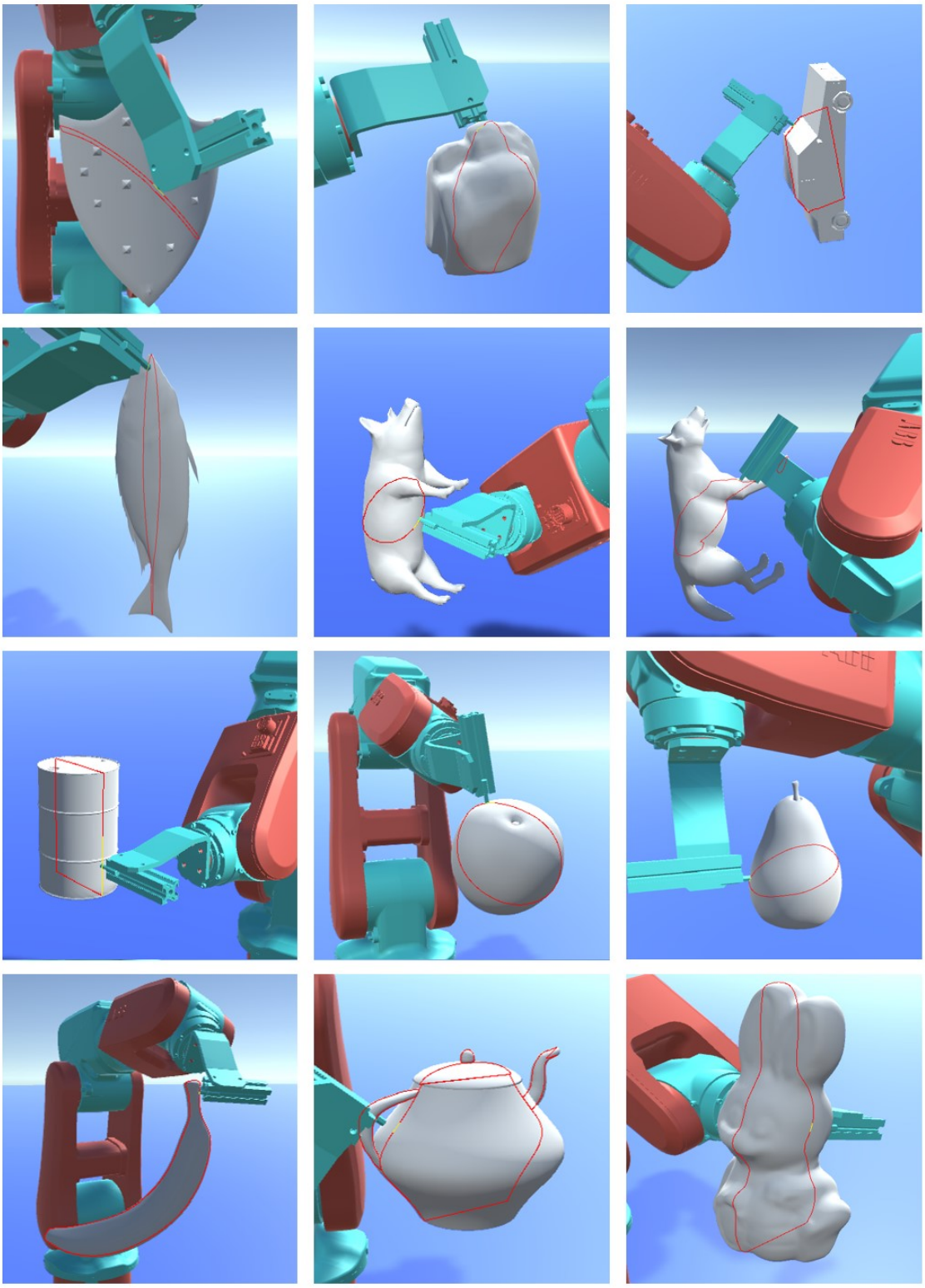
Kako bi algoritam rezanja ispravno prepoznao oblik vanjskog presjeka ravnine s mrežom objekta, objekti za rezanje moraju zadovoljavati određena svojstva. Izvorišnu točku objekta je potrebno podesiti u centar njegove mase koristeći neku od aplikacija za uređivanje modela objekata. Kako bi površina presjeka polovica bila ispravno osjenčana, normale i UV vrijednosti u vrhovima trokutova mreže moraju biti odgovarajuće definirane. Unutarnja struktura mreže trokutova objekata ne smije biti nepotrebno komplicirana kako bi presjeci ravnine s mrežom bili ispravno definirani. Trokutovi bi trebali biti međusobno spojeni u jednu cjelinu umjesto da su modeli sastavljeni od više spojenih dijelova. U suprotnom, presjek ravnine s mrežom trokutova će sadržavati nedostižne vrhove unutar objekta i robot će neuspješno pokušavati odvojiti spojene dijelove. Moguće rješenje za zaobilazak ovog problema bi bilo koristiti skenirajuću liniju koja prolazi presjekom i eliminira sve unutarnje vrhove iz putanje. Zbog prirode korištenog algoritma triangulacije pri popunjavanju mreže trokutova na ravninama presjeka polovica, algoritam neće davati sasvim ispravne rezultate za izrazito konkavne ili šuplje poligone presjeka. Za algoritam triangulacije je moguće koristiti algoritam rezanja uha (ElGindy i ostali, 1993) koji je kvadratne vremenske složenosti, ali daje ispravne rezultate za konkavne presjeke. Unatoč tome, poligoni sa šupljinama i dalje neće biti ispravno popunjeni. Za to bi bilo potrebno konstruirati hijerarhiju poligona, budući da svaki od njih može sadržavati više šupljina od kojih svaka može sadržavati dodatne poligone s još šupljina. Ako se u obzir uzme da neki modeli sadrže unutarnje poligone kao rezultat strukture njegove mreže, za ove poligone nije moguće jednoznačno odrediti trebaju li biti tretirani kao poligoni ili rupe. Ispravno rješenje će ovisiti o specifičnom modelu čiju je strukturu potrebno prethodno analizirati. Za izbjegavanje dodatnih komplikacija, preporučuje se koristiti jednostavnije modele pravilno strukturiranih mreža trokutova.

Iz navedenih razloga je teško pronaći besplatno dostupne odgovarajuće gotove modele objekata za svrhu demonstracije rada simulatora. Korišteni objekti za testiranje služe ilustrativnoj svrsi demonstracije rada robota na objektima različitih složenosti mreža trokutova i obilježja oblika poput zakrivljenosti i ispupčenosti. Popis korištenih testnih objekata uz pripadajuće rezultate vrednovanja učinkovitosti robota na pojedinom objektu prikazuje Tablica 4.1. Za polaznu točku usporedbe s testnim modelima je korišten model jednostavne kocke. Slika 4.1 prikazuje procese rezanja ostalih modela korištenih objekata redoslijedom navedenim u tablici. Pripadajuće dobivene izrezane polovice prikazuje Slika

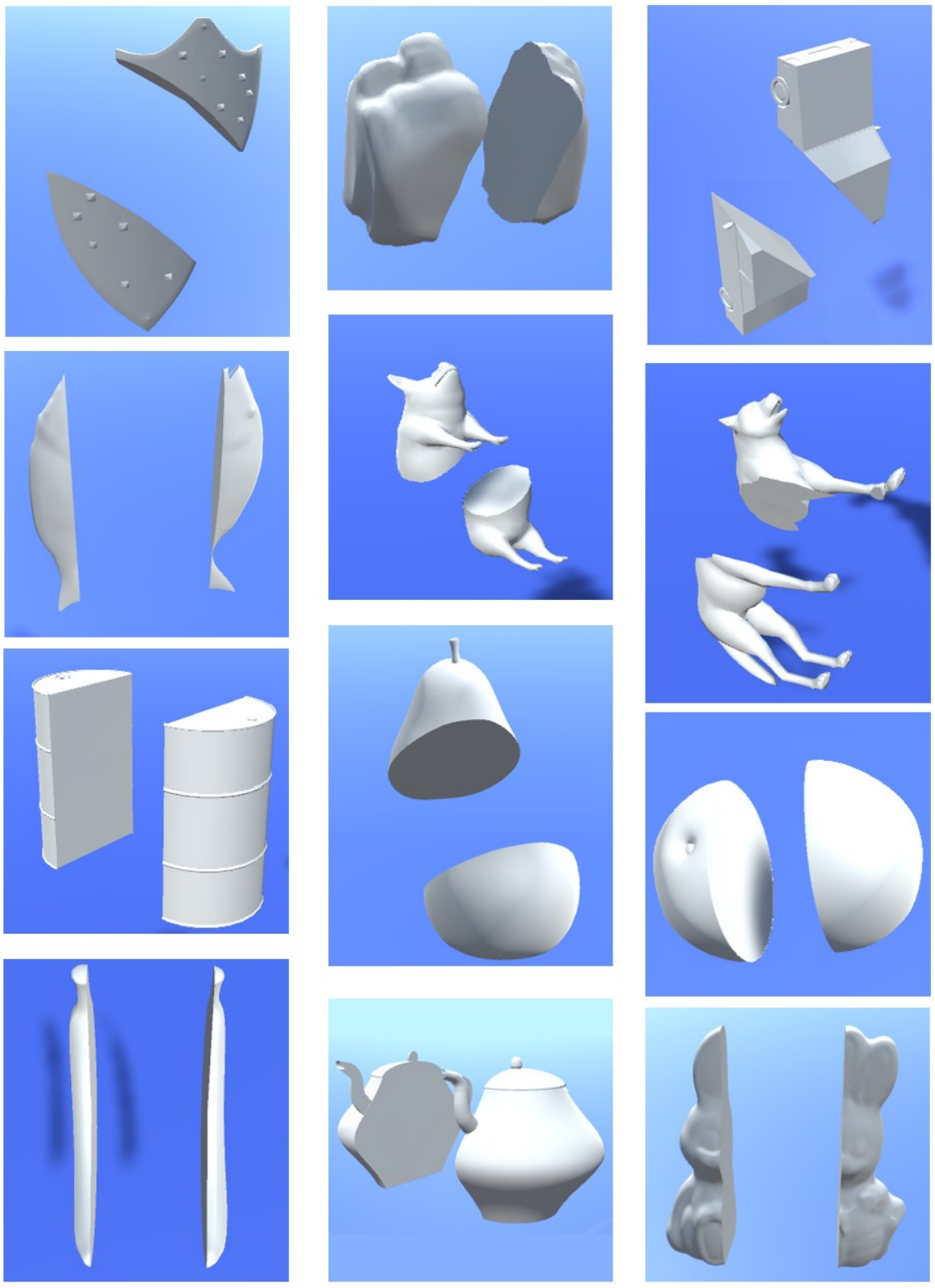
4.2. Vremena pronalazaka polovišne ravnine u provedenim mjerenjima u ovisnosti o broju trokutova mreža testnih objekata su interpolirana grafom koji prikazuje Slika 4.3.

Tablica 4.1 Rezultati vrednovanja učinkovitosti robota na testnim objektima

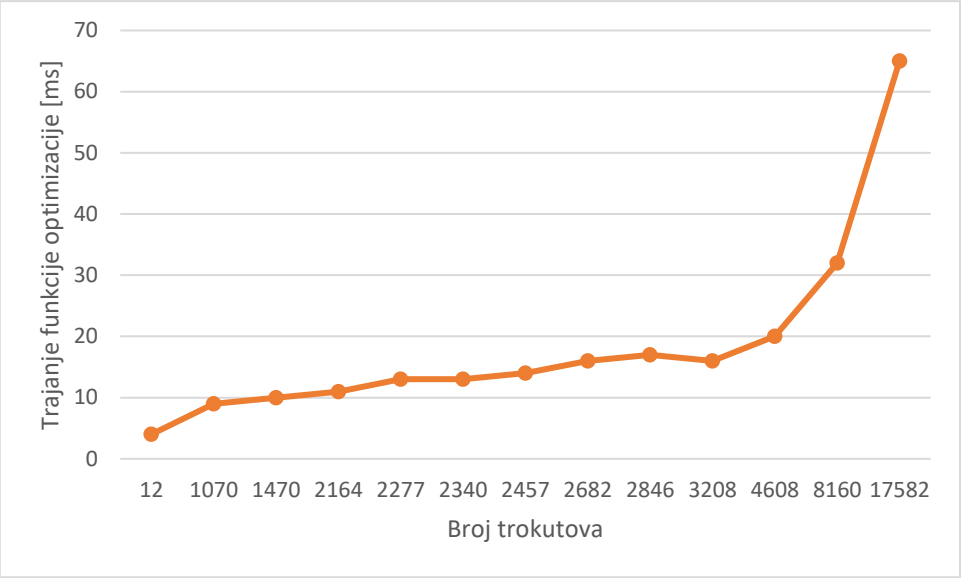
Objekt	**Broj trokutova**	**Prosječno vrijeme traženja ravnine [ms]**	**Relativna greška razlike volumena [%]**	**Prosječna udaljenost od cilja [mm]**
Kocka	12	4	0	4,40
Štit	1070	9	0,1738	2,74
Kamen	1470	10	0,0062	3,10
Automobil	2164	11	0,0296	4,42
Riba	2277	13	2,8369	2,84
Svinja	2340	13	0,8341	2,90
Vuk	2457	14	0,2266	4,64
Bačva	2682	16	0,0035	3,08
Naranča	2846	17	0,0539	2,85
Kruška	3208	16	0,0022	2,81
Banana	4608	20	0,1969	3,77
Čajnik	8160	32	0,0008	8,56
Zec	17582	65	0,7107	3,30



Slika 4.1 Proces rezanja testnih objekata



Slika 4.2 Izrezane polovice testnih objekata



Slika 4.3 Graf ovisnosti vremena traženja polovišne ravnine o broju trokutova mreže objekta

Rezultati za razlike volumena nisu iznenađujući, budući da je većina objekata za testiranje približno simetrična u odnosu na više ravnina simetrije. Većina predmeta u stvarnom svijetu posjeduje svojstvo približne simetrije u odnosu na neku ravninu i problem pronalaska ravnine koja dijeli objekt na približno jednake dijelove se uglavnom jednostavno rješava u par koraka. Nešto veća greška je primjetna na modelu ribe jer je korišteni model skoro plosnat pa se problem praktički svodi na pronalazak osi simetrije u dvodimenzionalnom prostoru. Za korišteni model bi precizniji presjek bio krivuljasta petlja sredinom tijela ribe.

Rezultati akumulirane prosječne udaljenosti od cilja su uglavnom posljedica aproksimativne prirode rješenja korištenog heurističkog algoritma inverzne kinematike. Kako bi se udaljenost približila nuli, potrebno je izračunati analitičko rješenje inverzne kinematike robota i koristiti jednu od dobivenih vrijednosti. Unatoč tome, postoji mogućnost da robot ne može precizno pozicionirati alat jer je ciljna točka izvan njegovog radnog dosega. Kako bi se izbjegao ovaj problem, testni modeli su podešeni na odgovarajuću veličinu i smješteni unutar radnog dosega robota. Ovo rješenje nije uvijek moguće primijeniti pa bi elegantnije rješenje bilo dinamički pomicati rezani objekt tako da su njegove nedostižne strane okrenute bliže robotu. Ciljna točka također može biti nedostupna robotskom manipulatoru ako se nalazi u konkavnom udubljenju objekta gdje vrh alata ne može dosegnuti. Ovaj problem je vezan uz specifične objekte i robot bi trebao pronaći presjek objekta s alternativnom ravninom ili drukčije izmijeniti putanju kretanja.

U slučaju modela čajnika, zbog načina konstrukcije mreže trokutova modela u više spojenih dijelova umjesto u jednu cjelinu, presjek ravnine s modelom sadrži više međusobno spojenih

petlji do kojih alat za rezanje ne može u potpunosti doseći bez pomicanja alata unutar objekta. Rezultat toga je često odvajanje alata od putanje ili ignoriranje sudara s objektom, te nešto veća prosječna udaljenost alata od točaka putanje.

Iz ovisnosti vremena traženja polovišne ravnine o broju trokutova mreže objekta prikazane grafom (Slika 4.3) moguće je primijetiti da vrijeme izračuna monotono raste porastom broja trokutova u mreži te da je pravilnost oblika objekta manje ključan faktor od složenosti mreže. Iako zbog malog broja podataka vrijednosti nisu prikazane na razmjernoj skali, iščitavanjem vrijednosti je moguće primijetiti skoro pa eksponencijalni rast vremena. Ovo je posljedica činjenice da korišteni Boxov algoritam optimizacije u svakoj iteraciji računa volumene prolazeći kroz čitavu mrežu trokutova za više točaka simpleks strukture. Također, apsolutne promjene vremena porastom složenosti mreže su neznatne, povećavajući se svakih tisuću trokutova za svega par sekundi.

Zbog načina detekcije sudara isključivo tijekom računanja rješenja inverzne kinematike, moguće je da se robot sudari s predmetom tijekom primjene interpolirane rotacije u orijentaciju prema idućoj točki. Ove promjene su najčešće kod promjene položaja alata od početne točke do površine objekta ili od jedne zatvorene petlje na objektu do druge petlje. Problem je moguće izbjeći prethodnim interpoliranjem u udaljeniju točku gdje je robot siguran od sudara te vraćanjem na sljedeću točku putanje. Budući da izbjegavanje sudara robota s objektom pokušava udaljiti samo najbližu točku na članku robota od objekta, u rijetkim slučajevima je moguće da određeni dijelovi alata prolaze kroz ispupčenja objekta.

Konačni rezultati pokazuju da ostvareni simulator za osnovnu implementaciju algoritma rezanja mreže objekta i općenite modele objekata s pravilnom strukturom mreže daje zadovoljavajuće rezultate. Uz nadogradnju korištenih algoritama rezanja i izbjegavanja sudara te detaljniju specifikaciju slučajeva korištenja simulatora, moguće je ostvariti simulator robota koji je spreman za uporabu u procesu industrijske automatizacije.

Zaključak

U ovom radu je predstavljeno idejno i programsko rješenje simulatora robotske ruke koja obavlja zadatak rezanja proizvoljnih virtualnih 3D objekata napola. Sustav simulatora se sastoji od robotske ruke, predmeta koje je potrebno izrezati i korisničkog sučelja koje korisnik koristi za upravljanje robotom. Objekt se dijeli na dvije polovice odvojene polovišnom ravninom. Parametri ravnine koja jednako dijeli objekt računaju se minimizacijom funkcije razlike volumena polovica određenih ravninom. Putanja kretanja robota je određena rubovima presjeka polovišne ravnine s objektom. Algoritmom inverzne kinematike određen je način izračuna orijentacije zglobova kako bi se robot kretao po putanji. Rješenje inverzne kinematike dobiveno je izmijenjenim algoritmom Cyclic Coordinate Descent (CCD).

Implementacija sustava je razvijena u sustavu za izradu igara Unity. Koristeći padajući izbornik u sučelju, korisnik može odabrati jedan od trodimenzionalnih predmeta za rezanje. Na zahtjev korisnika robotska ruka pomiče alat za rezanje po putanji i reže predmet na dva jednaka dijela. Nakon završetka rezanja, polovice objekta se odvajaju i statusna traka na korisničkom sučelju ispisuje metrike koje označavaju učinkovitost robota.

Simulator je testiran na nekoliko objekata različitih složenosti mreže trokutova i obilježja oblika poput zakrivljenosti i ispupčenosti. Rezultati su pokazali da za modele objekata s pravilnom strukturom mreže ostvareni sustav simulatora robotske ruke može precizno izrezati objekt napola.

Literatura

[1] Koenig, N., Howard, A. *Design and use paradigms for gazebo, an open-source multi-robot simulator*. Proceedings of the International Conference on Intelligent Robots and Systems, IEEE, (2004), str. 2149-2154.

[2] Giorgio, A., Romero, M., Onori, M., Wang, L. *Human-machine collaboration in virtual reality for adaptive production engineering*. 27th International Conference on Flexible Automation and Intelligent Manufacturing, Modena, (2017), str. 1279-1287.

[3] Wang, J., Phillips, L., Moreland, J., Wu, B., Zhou, C. *Simulation and Visualization of Industrial Processes in Unity*. Proceedings of the Conference on Summer Computer Simulation. Chicago, (2015), str. 1-7.

[4] Bartneck., C., Soucy, M., Fleuret, K., Sandoval, E. B. *The robot engine – Making the unity 3D game engine work for HRI*. 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), IEEE, (2015), str. 431-437

[5] Pan, J., Zhuo, Y., Hou, L., Bu, X. *Research on simulation system of welding robot in unity3d*. Proceedings of the 15th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry, New York, (2016), str. 107-110.

[6] Sita, E., Horvath, C. M., Thomessen, T., Korondi, P., Pipe, A. *ROS-Unity3D based system for monitoring of an industrial robotic process*. IEEE/SICE International Symposium on System Integration (SII), (2017), str. 1047-1052.

[7] *Product specification IRB 120*. IRC5, 3HAC035960-001, Rev. T, ABB, 2019

[8] Kovačić, Z. *Praktikum robotike*, Zagreb:FER, 2008.

[9] Denavit, J., Hartenberg, R. S. *A kinematic notation for lower-pair mechanisms based on matrices*, Trans. of the Asme. journal of Applied Mechanics, 22 (1995), str. 215-221

[10] Pieper, D. L. *The kinematics of manipulators under computer control.* Doktorska disertacija. Stanford University, Department of Mechanical Engineering, 1968.

[11] Zhang, C., Chen, T. *Efficient feature extraction for 2D/3D objects in mesh representation*. Proceedings 2001 International Conference on Image Processing, Thessaloniki, (2001), str. 935-938

[12] Pregun, B. *Three-dimensional mesh cutting in virtual scene*. Diplomski rad. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2017.

[13] Box, M. J. *A New Method of Constrained Optimization and a Comparison With Other Methods*. The Computer Journal, 8,1 (1965), str. 42-52

[14] Nelder, J. A., Mead, R. *A simplex method for function minimization*. The Computer Journal, 7 (1965), str. 308

[15] Wang, L.-C. T., Chen, C. C. *A combined optimization method for solving the inverse kinematics problems of mechanical manipulators*. IEEE Transactions on Robotics and Automation, 7,4 (1991), str. 489-499

[16] ElGindy, H., Everett, H., Toussaint, G. *Slicing an ear using prune-and-search*. Pattern Recognition Letters, 14,9 (1993), str. 719-722

Sažetak

Prilagodljivo upravljanje virtualnim robotom korištenjem sustava Unity na primjeru rezanja 3D objekta

U radu je predstavljena ideja i implementacija simulatora robotske ruke koja može izrezati 3D objekte napola. Proučeni su algoritmi rezanja mreže trokutova, minimizacija funkcija, inverzna kinematika i drugi pojmovi industrijske robotike. Algoritmi su povezani u sustav koji za odabrani objekt pomiče robota površinom objekta i reže ga napola. Sustav je implementiran u sustavu za izradu igara Unity. Učinkovitost robota je testirana na objektima različitih obilježja.

Ključne riječi: Unity, virtualni industrijski robot, simulacija robota, inverzna kinematika, rezanje 3D objekta

Summary

Adaptable control of virtual robot in Unity for a 3D cutting task

In this work we present a theoretical and practical approach to implementing an industrial robot simulator applied for the task of cutting 3D objects in half. Our research covers mesh cutting algorithms, function minimization, inverse kinematics algorithms and terminology related to industrial robotics. The covered algorithms are composed into a system which simulates movement of the robotic arm over a specified object's surface and cuts it in half. The system is implemented using the Unity game engine. The simulator's efficiency is tested on objects of various characteristics.

Keywords: Unity, virtual industrial robot, robot simulation, inverse kinematics, 3D mesh cutting