SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2127

PRILAGODLJIVO UPRAVLJANJE VIRTUALNIM ROBOTOM KORIŠTENJEM SUSTAVA UNITY NA PRIMJERU PREMAZIVANJA ZIDOVA

Antun Mesar

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2127

PRILAGODLJIVO UPRAVLJANJE VIRTUALNIM ROBOTOM KORIŠTENJEM SUSTAVA UNITY NA PRIMJERU PREMAZIVANJA ZIDOVA

Antun Mesar

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

DIPLOMSKI ZADATAK br. 2127

Pristupnik:	**Antun Mesar (0036492785)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	doc. dr. sc. Mirko Sužnjević

Zadatak: **Prilagodljivo upravljanje virtualnim robotom korištenjem sustava Unity na primjeru premazivanja zidova**

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 različitih površina te ih oličiti. U procesu ličenja potrebno je minimalizirati dvostruke prijelaze po istoj površini. Prikažite funkcionalnost na primjeru dinamički generirane prostorije u kojoj će se nalaziti zidovi 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.

Sadržaj

Uvod

Ideja robota je fascinirala čovjeka još od doba antičke Grčke. U mitovima se spominje kako je Grčki bog Hefest stvorio mehaničke djeve od zlata, a i navodi se da je oko 400. godine pr. Kr. Grčki filozof Arthitas od Tarentuma izgradio mehaničkog goluba pogonjenog parom koji je mogao letjeti.

Moderna robotika je svoj procvat doživjela u 20. stoljeću dijelom popularizacije dijela znanstvene fantastike. Sama riječ robot se prvi put navodi u drami „R.U.R“ (Rossumovi univerzalni roboti) češkog književnika Karela Čapeka u kojem opisuje robota kao: „Stroj vješt u radu, a ponaša se slično čovjeku te ponekad ispunjava funkcije čovjeka.“.

Prvi pravi industrijski roboti korišteni su u Fordovim tvornicama radi povećavanja učinkovitosti proizvodnje automobilskih dijelova. Bili su to numerički upravljani strojevi za obradu metala, u početku programirani pomoću bušenih kartica, a kasnije elektroničkim računalom. No primjena robota nije bila dovoljna za povećanje učinkovitosti proizvodnje. Javio se problem vremenskih zastoja robota zato što se prijenos dijelova između strojeva obavljao ručno. Taj problem riješio je američki izumitelj George C. Devol, koji je 1954. godine prijavio patent za „programirani prijenos dijelova“. U suradnji s J. Engelbergerom izradio je 1958. godine i prvi robot, Unimate.

Za izradu robota potrebna je izrada simulacije. Današnji simulatori robota nude fizičke sustave za robusno testiranje i dizajniranje robota, ali su vrlo kompleksni, skupi i generalno nedostupni široj populaciji. Sustavi za razvoj digitalnih igara (*eng. game engine*) su iznimno popularni, a oba područja se značajno preklapaju u zahtjevima. Oboje struke trebaju način vizualizacije trodimenzionalnih objekata, simulaciji fizike, napredne algoritme kao što su inverzna kinematika, algoritam pronalaženja puta, umjetna inteligencija itd. Unity je jedan od najpopularnijih sustava za razvoj digitalnih igara današnjice s oko 5,000,000 korisnika koji posjeduje svoju trgovinu Asset store na kojoj korisnici dijele alate među kojima ima i alata potrebnih za razvoj robota kao npr. sustav inverzne kinematike [2].

Cilj ovoga rada je prikazati uporabljivost Unity sustava za kretanje simulacije robota. Ovaj rad se sastoji od pet poglavlja. Poglavlje jedan sastoji se od osnovnog teorijskog uvoda u robotiku, te karakteristika robota. Drugo poglavlje pokriva tri algoritma inverzne

kinematike. To su metoda Jacobianovog inverza, metoda cikličkog koordinatnog spusta (*eng. cyclic coordinate descent* (CCD)) i metoda unaprijednog i unazadnog dosega (*eng. Forward and Backward Reaching Inverse Kinematics* (FABRIK)). Poglavlje tri je teorijski uvod u problem planiranja puta alata robota. Unutar njega definiraju se pojmovi putanje i trajektorije gibanja robota, problem gibanja manipulatora od točke do točke, gibanje manipulatora kontinuirano po putanji i problem pravocrtnog gibanja. U četvrtom poglavlju opisujemo alat Unity koji smo koristili za implementaciju. Peto poglavlje opisuje implementaciju robota. Sastoji se od implementacije robota u grafu scene, implementaciji inverzne kinematike, načinu generiranja zidova, tehnici bojanja zida, pomicanja baze robota, planiranja puta alata te dobivenih rezultata.

1. Robotika

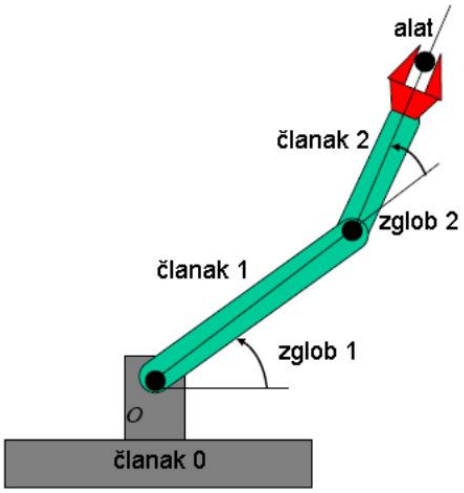
Robotika je interdisciplinarno znanstveno područje između računalne znanosti i inženjerstva. Robotika uključuje dizajn, konstrukciju, rad, uporabu i održavanje robota. Cilj robotike je dizajn inteligentnog stroja koji će pomoći ljudima u svakodnevnom životu i zamijeti čovjeka pri obavljanju zamornih, jednoličnih, odnosno opasnih ili po zdravlje štetnih poslova. Temelji se na dostignućima informatičkog inženjerstva, računalnog inženjerstva, strojarstva elektroničkog inženjerstva i drugih.

Roboti su automatizirani strojevi višestruke namjene koji se sastoje od konstrukcije s pripadajućim pogonskim uređajima, senzora i upravljačkog uređaja, dijele se po stupnju pokretljivosti (statički i mobilni roboti), strukturi konstrukcije (mehatronički, biotehnički i bioroboti), namjeni (industrijski, medicinski, edukacijski, podvodni, roboti za istraživanje svemira, vojni roboti, osobni roboti), veličini (makroroboti, mikroroboti i nanoroboti). Inteligentni roboti posjeduju sposobnost učenja, rasuđivanja i donošenja zaključaka te imaju visok stupanj funkcionalne, organizacijske i mobilne autonomnosti [1].

Na globalnoj razini prodaja robota raste ponajprije zahvaljujući potražnji automobilske i elektroničke industrije te se očekuje opći porast svjetske robotike. U svijetu je 2007. bilo oko 6,5 milijuna, a 2011. oko 18 milijuna robota. U automobilskoj industriji Japan rabi oko 350 000 robota, EU oko 350 000, ostale Europske zemlje oko 10 000, SAD oko 130 000, te Azija i Australija oko 75 000 robota [1].

1.1. Osnovni dijelovi robota

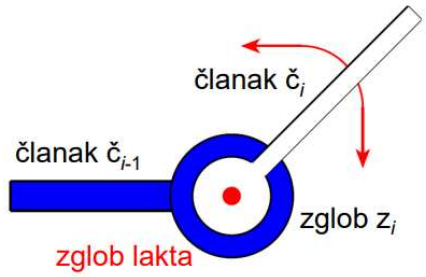
Robot se općenito sastoji od n zglobova, n+1 članaka u koje se ubraja nulti članak baza i n-ti članak alat (*eng. effector*). Pravilo je da se članak $č_i$ veže na zglob z_i tj zglob z_i se nalazi između članaka $č_{i-1}$ i $č_i$. Zglobovi određuju konfiguraciju i radni prostor robota. Dva su osnovna tipa zgloba u robotu: rotacijski zglob i translacijski zglob koji se dijele na podvrste opisane u nastavku (Slika 1.1).



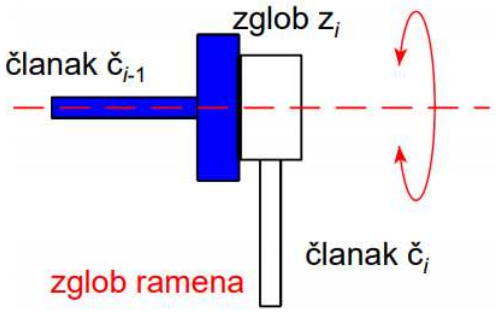
Slika 1.1 Osnovni dijelovi robota (preuzeto iz [3])

Vrste zglobova robota:

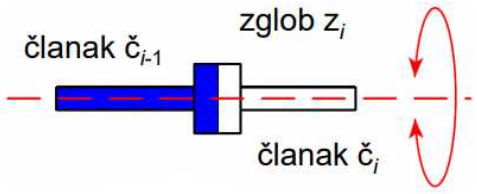
- Rotacijski zglob: Rotacijska os zgloba z_i okomita je na članak $č_{i-1}$ (Slika 1.2),
- Revolucijski zglob: Rotacijska os zgloba z_i u smjeru osi članka $č_{i-1}$, a članak $č_i$ okomit na članak $č_{i-1}$ (Slika 1.3),
- Torzijski zglob: Rotacijska os zgloba z_i paralelna s osima članaka $č_{i-1}$ i $č_i$ (Slika 1.4),
- Tranzicijski zglob: Translacijska os zgloba z_i u smjeru osi članka $č_{i-1}$, a članak $č_i$ okomit na članak $č_{i-1}$ (Slika 1.5)



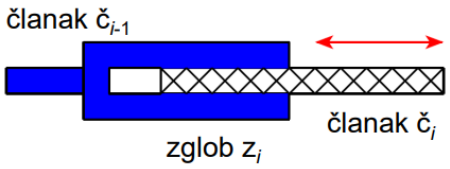
Slika 1.2. Rotacijski zglob (preuzeto iz [3])



Slika 1.3. Revolucijski zglob (preuzeto iz [3])



Slika 1.4. Torzijski zglob (preuzeto iz [3])



Slika 1.5. Translacijski zglob (preuzeto iz [3])

1.2. Karakteristike robota

U karakteristike robota se ubraja: broj osi, nosivost, maksimalna brzina, dohvat, hod, orijentacijalata alata, ponovljivost, preciznost, točnost i radna okolina [3].

1.2.1. Broj osi

Za svaki robot karakterističan je broj osi za rotacijska ili translacijsko gibanje njegovih članaka. Kako se gibanje robota odvija u trodimenzionalnom prostoru, prve tri osi najčešće se koriste za određivanje položaja ručnog zgloba robota. Različite kombinacije zglobova prvih triju osi određuje tip konfiguracije koje se označavaju tipom zgloba, R za rotacijski, a T za translacijski. Korištene kombinacije su: pravokutna (TTT), cilindrična (RTT), sferna (RRT), rotacijska (RRR) i robot tipa SCARA (*eng. selective assembly robot arm*) – RTR, TRR ili RRT građe. Preostale osi određuju orijentaciju završnog mehanizma [3].

Općeniti manipulator ima šest osi te može dovesti prihvatnicu u bilo koji položaj i orijentaciju unutar radnog prostora. Pri tome se mehanizam otvaranja i zatvaranja prstiju ne smatra nezavisnom osi jer ne utječe niti na položaj niti na orijentaciju alata.

Ako manipulator ima više od šest osi, tada se redundantne osi mogu koristiti za izbjegavanje prepreka unutar radnog prostora, optimiranje gibanja. Također se može povećati vještina i spretnost gibanja.

1.2.2. Nosivost i brzina

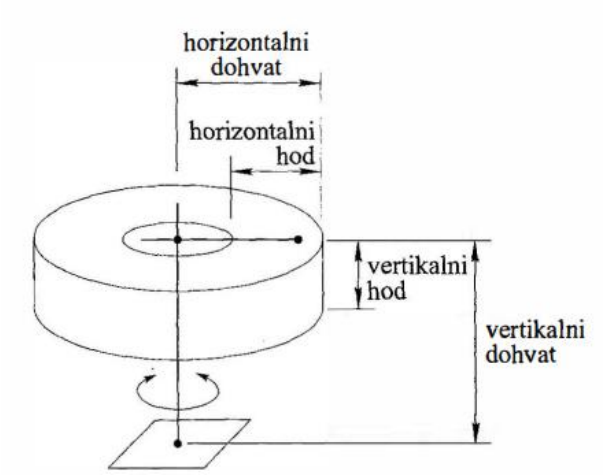
Maksimalna masa tereta koju robot može prenijeti i maksimalna brzina gibanja robota veoma ovise o tipu robota i njegovoj primjeni (nosivost može biti od nekoliko kilograma do nekoliko tona, a vrh alata može se kretati brzinama od 10 cm/s do 10 m/s) [3].

Važno mjerilo brzine robota je vrijeme radnog ciklusa, tj. vrijeme potrebno za izvođenje periodičnog gibanja kao npr. jednostavna operacija podizanja i spuštanja predmeta na određeno mjesto. Tada se uz poznatu duljinu putanje može izračunati prosječna brzina kretanja manipulatora.

1.2.3. Dohvat i hod

Veličina radnog prostora robota može se približno odrediti pomoću dohvata i hoda. Horizontalni dohvat je maksimalna udaljenost koju može dosegnuti ručni zglob, mjerena od vertikalne osi oko koje robot rotira. Horizontalni hod je ukupna udaljenost od vertikalne osi po kojoj se ručni zglob može kretati. Razlika između horizontalnog dohvata i hoda je minimalna udaljenost ručnog zgloba od glavne vertikalne osi, a kako je ta veličina pozitivna, dohvat je uvijek veći ili jednak hodu. Vertikalni dohvat robota maksimalna je udaljenost ručnog zgloba robota od baze, a vertikalni se hod slično može definirati kao ukupna vertikalna udaljenost po kojoj se ručni zglob može gibati. Pri tome je vertikalni hod robota manji ili jednak vertikalnom dohvatu (Slika 1.6) [3].

Kod rotacijskih robota dohvat je često jednak hodu, pa takvi roboti imaju puni radni prostor. Pri tome je potrebno paziti na zaštitu od robot od samooštećivanja, jer se rotacijski robot može isprogramirati tako da udari sam sebe ili da se sudari s predmetima u svojoj radnoj okolini.



Slika 1.6. Dohvat i hod (preuzeto iz [3])

1.2.4. Orijentacija alata

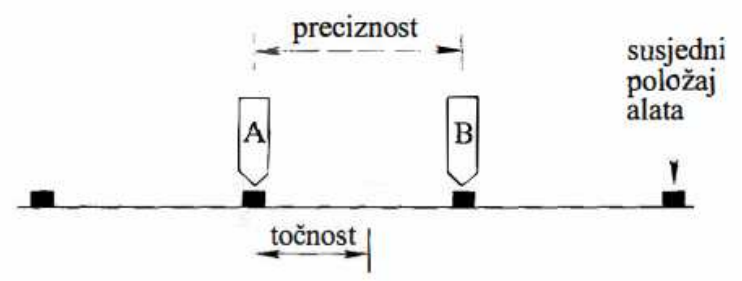
Prve tri osi robota osim oblika radnog prostora određuju položaj kraja podlaktice i pripadajuću orijentaciju alata. Dodatne rotacijske osi kod neredundantnih robota (robot koji može postići određenu poziciju na više različitih načina [3]) određuju orijentaciju završnog mehanizma. U uvjetima bez ograničenja, s tri dodatne osi alat može tijekom gibanja robota postići bilo koju orijentaciju u trodimenzionalnom prostoru.

1.2.5. Ponovljivost, preciznost i točnost

Ponovljivost je mjera sposobnosti robota da vrh prihvatnice ponovno dovede u isti položaj. Pogreška koja pri tome može nastati najčešće je manja od 1 mm, a javlja se zbog zazora zupčanika i elastičnosti članaka.

Preciznost je mjera razlučivosti kojom se prihvatnica može pozicionirati u radnom prostoru (Slika 1.7.). Ako je vrh alata doveden u točku A i ako je točka B sljedeći najbliži položaj u koji može doći, tada preciznost predstavlja udaljenost između točaka A i B.

Točnost je, za razliku od preciznosti, razlika između mogućeg i željenog položaja, tj. mjera sposobnosti robota da dovede prihvatnicu u proizvoljan položaj radnog prostora [3].



Slika 1.7. Preciznost i točnost (preuzeto iz [3])

1.2.6. Radna okolina

Radna okolina robota uglavnom će ovisiti o zadatku koji on mora izvršiti. Roboti često rade u opasnoj ili onečišćenoj okolini, npr. prilikom prijevoza ili premještaja radioaktivnog materijala potrebnog za rad nuklearnih postrojenja, zatim pri bojenju, zavarivanju ili radu u ljevaonicama. Takvi se roboti posebno opremaju kako bi mogli podnijeti rad pri vrlo visokim temperaturama i uz prisutnost različitih onečišćenja u zraku. Drugi ekstremni tip radne okoline jesu vrlo čiste radne prostorije, kakve se susreću npr. u poluvodičkoj industriji, u kojima se vrlo pažljivo prate onečišćenja, temperatura i vlažnost zraka. U tom slučaju i sam robot mora biti vrlo čist kako bi se onečišćenje radnog materijala svelo na minimum [3].

2. Inverzna kinematika

U robotici i računalnim animacijama inverzna kinematika je matematički proces računanja parametara zglobova nužnih za postavljanje alata u na određenu poziciju relativno na početni članak. Parametri zgloba uključuju lokalnu poziciju definiranu trodimenzionalnim vektorom zgloba relativno poziciji prijašnjeg zgloba i lokalnu orijentaciju zgloba definiranu kvaternionom relativno orijentaciji prijašnjeg zgloba. Ako znamo parametre zglobova možemo robota postaviti u određeni položaj. Takav način postavljanja se naziva direktna kinematika. Međutim inverzna operacija je generalno mnogo zahtjevnija [8].

U nastavku ću opisati tri metode računanja inverzne kinematike: metoda Jacobianovog inverza, metoda cikličkog koordinatnog spusta (*eng. cyclci coordinate descent (CCD)*) i metoda unaprijednog i unazadnog dosega (*eng. forward and backward reaching inverse kinematic (FABRIK)*)

2.1. Metoda Jacobianovog inverza

Jedno od prvih rješenja za problem inverzne kinematike su metoda Jacobianovog inverza. To je skup metoda koji koriste Jacobianovu matricu parcijalnih derivacija za određivanje novog položaja zglobova. Ove metode su vrlo moćne, ali potencijalno računalno zahtjevne i nestabilne [9].

Jacobianove metode se sastoje od tri glavna koraka:

1. Pronalaženje ciljne konfiguracije zglobova: *T,*
2. Izračun promijene u rotaciji: *dO,*
3. Izračun Jacobianove matrice: *J*

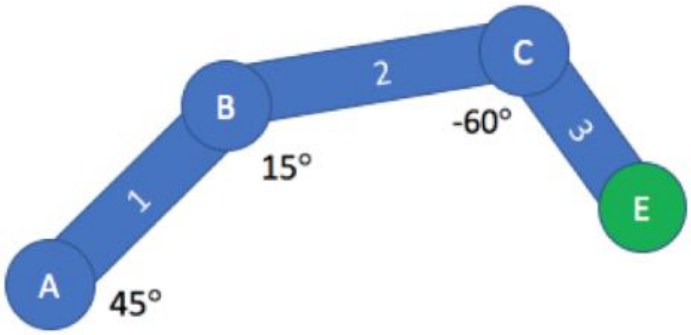
2.1.1. Pronalaženje ciljne konfiguracije zglobova

Da bi se pomakli od početnog položaja do ciljne lokacije alata definirat ćemo formulu:

$$T = O + dO$$

Pretpostavimo da su svi zglobovi rotacijski, onda O predstavlja vektor koji odgovara početnoj orijentaciji svakog zgloba. T je vektor koji predstavlja ciljnu orijentaciju svakog

zgloba, takvu da pozicija alata odgovara poziciji ciljne točke. *dO* je vektor koji predstavlja promijenu u orijentaciji svakog zgloba. Na primjeru u dvodimenzionalnom prostoru na Slika 2.1. *O* bi bio vektor [45, 15, -60]



Slika 2.1. Dvodimenzionalna robotska ruka (preuzeto iz [8])

Jacobianove metode koriste iterativni postupak za izračun *dO*. Iterativno pronalazimo rješenja za *dO*, tako da svako sljedeće rješenje bolje odgovara finalnom rješenju. Svakom iteracijom nalazimo lokalni minimum funkcije pogreške. Modificirat ćemo početnu formulu da odgovara iterativnom pristupu:

$$T = O + dO * h$$

Gdje je h korak simulacije koji se može podešavati. Dobra početna vrijednost za h se smatra 0.01 [8].

2.1.2. Izračun promjene u rotaciji

Za izračun promijene *dO* nam je potrebna Jacobianova matrica korištenjem formule:

$$V = J * dO$$

Gdje je V vektor promjene položaja alata tj. razlika između pozicije alata i ciljnog položaja:

$$V = T - E$$

A Jacobianova matrica predstavlja odnos položaja alata u ovisnosti o rotaciji svakog zgloba. *dO* izrazimo kao:

$$J^{-1}V = J^{-1}JdO$$
$$J^{-1}V = dO$$
$$dO = J^{-1}V$$

Iz formule vidimo da nam je za izračun promjene *dO* potreban inverz Jacobianove matrice. Računanje inverza je složena operacija i nije garantirano da inverz postoji. Postoji par načina da se zaobiđe taj problem [10]. Neke od predloženih metoda su korištenje transverzalne matrice kao aproksimaciju inverza ili korištenje Moore-Penrosevog pseudoinverza:

$$A^{+} = (A^{T}A)^{-1}A^{T}$$

2.1.3. Izračun Jacobianove matrice

Zadnji korak koji nam je preostao je definirati Jacobianovu matricu:

$$J = \begin{matrix} \frac{\partial p_x}{\partial \theta_A} & \frac{\partial p_x}{\partial \theta_B} & \frac{\partial p_x}{\partial \theta_C} \\ \frac{\partial p_y}{\partial \theta_A} & \frac{\partial p_y}{\partial \theta_B} & \frac{\partial p_y}{\partial \theta_C} \\ \frac{\partial p_z}{\partial \theta_A} & \frac{\partial p_z}{\partial \theta_B} & \frac{\partial p_z}{\partial \theta_C} \end{matrix}$$

Svaki član matrice predstavlja kako promjena jednog zgloba (označeni s A,B,C kao na Slika 2.1) mijenja položaj alata na kraju lanca. Pozicija alata je ovdje definirana kao [p_x, p_y, p_z]. Na primjer prvi član pokazuje koliko bi se alat pomaknuo po x osi, ako se zglobu A kut rotacije promijeni za diferencijalnu količinu, npr. 0.00001 [8]. Važno je napomenuti da gore navedena jacobianova matrica sadrži tri retka jer robotska ruka na Slika 2.1. ima samo tri zgloba. Za kompleksnije robote raste veličina matrice.

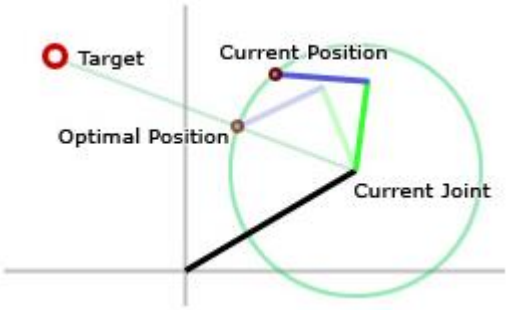
2.2. Ciklički koordinatni spust

Algoritam cikličkog koordinatnog spusta (*eng. cyclic coordinate descent* (CCD)) prvi put je predložen 1991. Algoritam je baziran na konceptu kombinirane optimizacije. Koristi ciklički koordinatni spust za brzo pronalaženje točke koja se nalazi blizu ciljne točke. Algoritam također implicitno rješava problem ograničenja u rotaciji zglobova pošto definira kutove rotacija zglobova, a ne njihove krajnje pozicije [11]. Algoritam je iterativan kao i metoda Jacobianovog inverza. Svaki korak iteracije optimiziramo udaljenost alata od ciljnog položaja pronalaženjem najboljeg kuta za svaki zglob počevši od vrha prema bazi

robota. Algoritam se ponavlja dok razlika između alata i cilja postane dovoljno malena ili algoritam odradi prethodno zadani broj iteracija.

2.2.1. Optimizacija zgloba

Za svaki zglob želimo minimizirati udaljenost alata(*eng. effector*) i cilja (*eng. target*) formulom: $\min|e-t|$ gdje je *e* pozicija alata, a *t* pozicija cilja. Ovisno o tipu zgloba koriste se različite jednadžbe za pronalazak optimalnog rješenja. Na primjer za rotacijski zglob (Slika 2.2) mijenjamo položaj alata po kružnici tako da se točka u kojoj je najmanja udaljenost alata i cilja nalazi na presjeku pravcu koji povezuje centar kružnice, tj. položaj trenutno gledanog zgloba i položaj cilja s kružnicom po kojoj pomičemo alat [12].



Slika 2.2. Optimizacija jednog zgloba (preuzeto iz [12])

Da bi pomakli alat na optimalni položaj moramo rotirati zglob za nepoznati kut α. Znamo položaj trenutnog zgloba *j*, trenutnu poziciju alata *e* i poziciju cilja *t*. Da bi pronašli α tj. kut između vektora (*e* - *j*) i (*t* - *j*). Kut pronalazimo formulom za kut između dva vektora:

$$\cos\alpha = \frac{(e-j)\cdot(t-j)}{|e-j||t-j|}$$

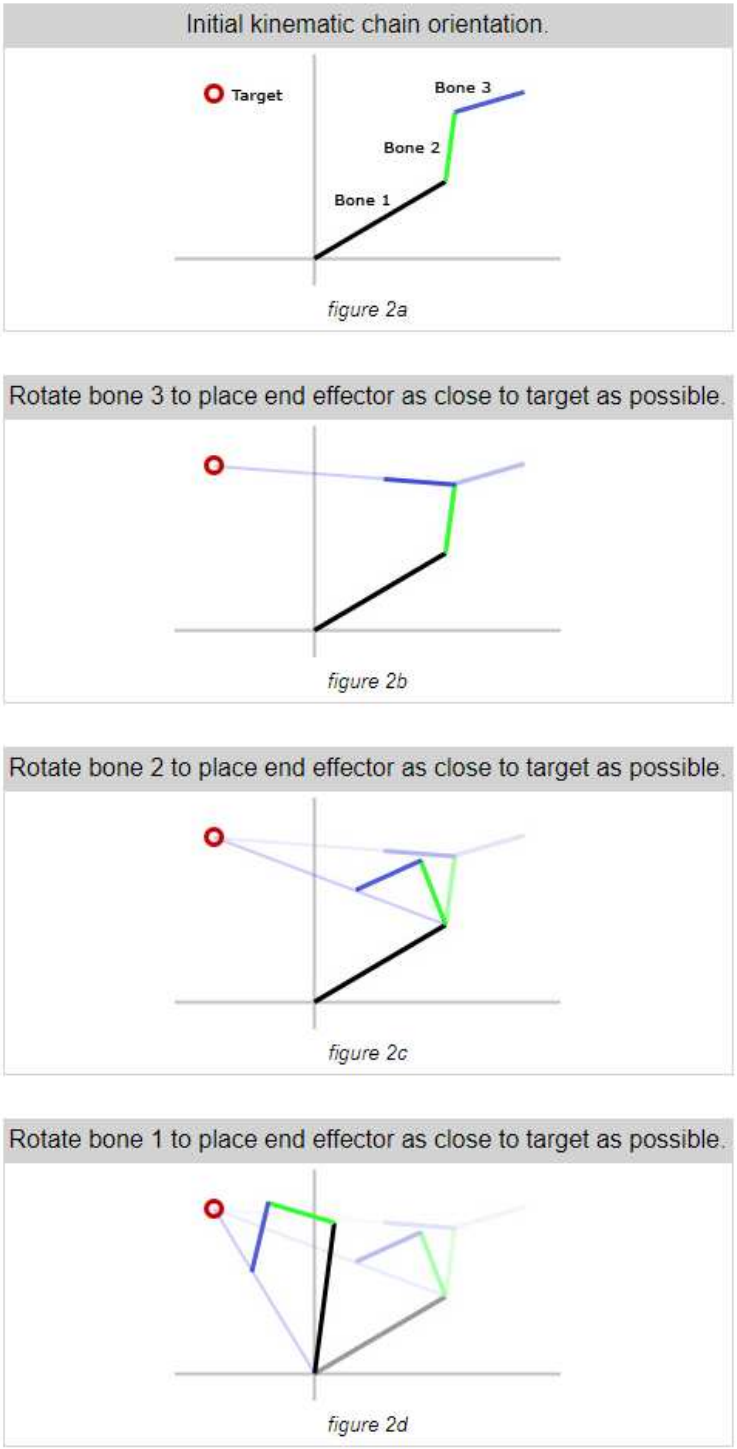
Gdje u brojniku razlomka koristimo skalarni produkt (*eng. dot product*).

Kosinus nam daje veličinu α ali ne i predznak. Da bi dobili predznak moramo izračunati $\sin\alpha$. Puni izvod za izračun sinusa u dvodimenzionalnom prostoru je opisan u [12]. Ovdje je navedena samo krajna funkcija:

$$\sin\alpha = \frac{(e_x - j_x)(t_y - j_y) - (e_y - j_y)(t_x - j_x)}{|e-j||t-j|}$$

2.2.2. Vizualizacija CCD algoritma

U nastavku je vizualizirana jedna iteracija CCD algoritma (Slika 2.3). Počinjemo od zgloba najbližeg ciljnom položaju i rotiramo ga za kut dobiven računicom iz prethodnog poglavlja. Postupak ponavljamo za svaki zglob [12].



Slika 2.3 Vizualizacija CCD algoritma (preuzeto iz [12])

2.3. Metoda unatrijednog i unazadnog dosega

Metoda unaprijednog i unazadnog dosega (*eng. Forward and Backward Reaching Inverse Kinematics* (FABRIK)) koristi unaprijedni i unazadni iterativni pristup rješavanju problema inverzne kinematike. Algoritam pronalazi pozicije za svali zglob pronalaženjem točke na liniji. Kako algoritam pronalazi pozicije, a ne rotacije ograničavanje kuta rotacije zgloba nije implicitno kao u CCD algoritmu, ali postoji metoda za nadogradnju algoritma za rješavanje problema s ograničenjima. Također algoritam podržava robote s više točaka alata istovremeno i pronalazi rješenje u manje iteracija od drugih sličnih metoda [13].

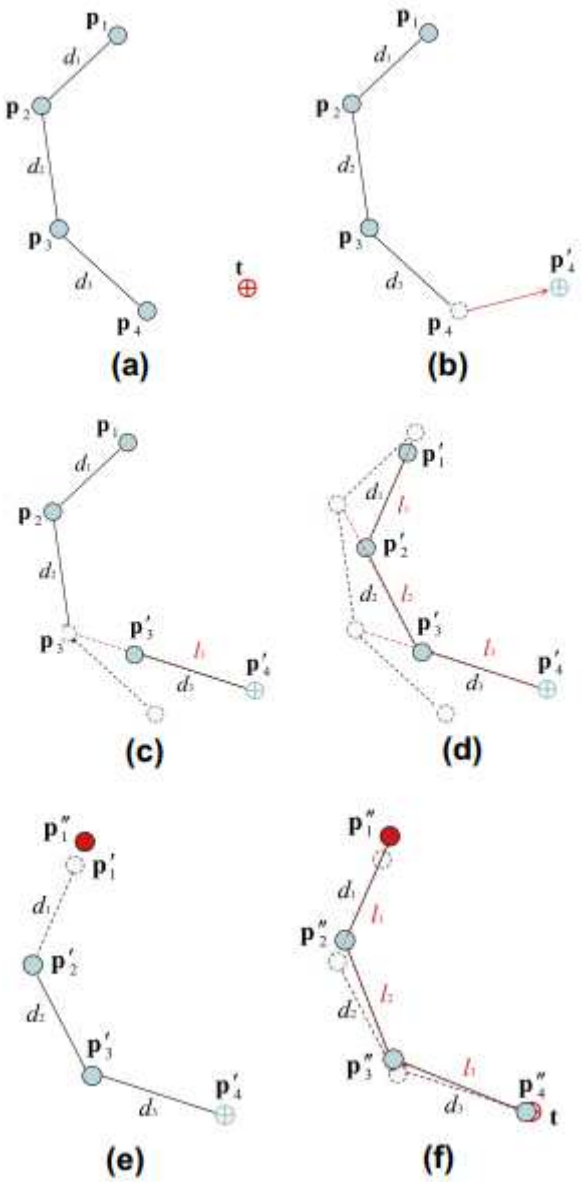
Slično kao u CCD metodi algoritam se izvršava dok udaljenost između cilja i alata nije manja od zadane greške ili dok nije izvršen zadan maksimalni broj iteracija. Svaka iteracija se sastoji od dva koraka: Korak dosega prema naprijed (*eng. forward reach*) i korak dosega prema natrag (*eng. backward reach*).

2.3.1. Iteracija FABRIK algoritma

Kao primjer uzet ćemo dvodimenzionalnu robotsku ruku s četiri zgloba p_1, p_2, p_3 i p_4. Članke između zglobova označit ćemo s d_1, d_2, d_3 . Ciljnu točku označujemo s t (Slika 2.4). (a)). Prvo radimo korak dosega prema naprijed tako da pomaknemo alat (zglob p_4) u ciljnu točku (b)). Zatim prethodni zglob (p_3) pomičemo po dužini između zglobova p_3 i p_4 dok udaljenost od p_3' *do* p_4' nije jednaka početnoj dužini članka d_3 (c). Taj postupak ponavljamo za svaki prethodni zglob (d). Time smo izvršili korak dosega prema naprijed. Rezultat je da smo dosegli ciljnu točku, ali smo pomakli početni zglob p_1 iz njegove originalne pozicije (e). Slijedi korak dosega prema natrag kojime vraćamo početni zglob na originalnu poziciju. Korak je identičan dosegu prema naprijed samo ovaj put prvo pomičemo početni zglob i onda za njim sve sljedeće zglobove (f). Time smo dovršili jednu iteraciju FABRIK algoritma i vidimo da se alat pomakao prema cilju, a početni zglob je ostao na svome mjestu. Također možemo primijetiti da se nema neprirodnih pomaka u jednoj iteraciji (Slika 2.3) za razliku od CCD algoritam koji se neprirodno savija, u FABRIK-u se svakom iteracijom pomakne relativno prirodno [13].

Mana algoritma je što na početku moramo provjeriti je li cilj doseziv što nije implicitno riješeno kao u CCD algoritmu gdje samo mijenjamo rotacije, a ne položaje zglobova.

Prednost je brzina izvođenja jer u iteracijama nema kompleksnih računalnih operacija kao što su inverz matrice ili sinus i kosinus koji su potrebni za prethodne algoritme.



Slika 2.4. Vizualizacija FABRIK algoritma (preuzeto iz [13])

3. Planiranje putanje alata

Da bi robot mogao obaviti zamišljeni zadatak, potrebno je zadati niz točaka u prostoru kroz koje vrh alata manipulatora mora proći. U tom pogledu moguće je robotske sustave podijeliti na [3]:

1. sustave koji ostvaruju gibanje „od točke do točke“ i sustave koji ostvaruju gibanje „kontinuirano po putanji“,

2. sustave s upravljanjem u otvorenoj petlji i sustave s upravljanjem u zatvorenoj petlji,

3. sustave s različitom građom manipulatora (kartezijski, cilindrični, sferni i artikulirani)

Najprije je potrebno definirati dva osnovna pojma: putanju i trajektoriju gibanja.

3.1. Putanja i trajektorija gibanja

Željena putanja pri upravljanju kontinuiranim gibanjem vrha alata robota može se definirati u prostoru konfiguracije alata pomoću vektora konfiguracije alata w koji predstavlja položaj i orijentaciju. Putanja alata se definira kao krivulja u prostoru konfiguracije alata tj. prostoru koji alat doseže. Ako se pri tom zadaju trenutci u kojima alat mora biti u odgovarajućim točkama putanje, tada putanja postaje trajektorija.

Iz toga slijedi da je potrebno naći funkciju koja će opisati vezu između točaka na putanji i zadanih trenutaka dolaska alata u te točke. To je moguće učiniti pomoću funkcije raspodjele brzine $s(t)$ čija derivacija $\dot{s}(t)$ predstavlja trenutnu brzinu vrha alata manipulatora u vremenu. Dakle, zadavanjem funkcije raspodjele brzine zadaje se brzina gibanja alata po željenoj krivulji [3].

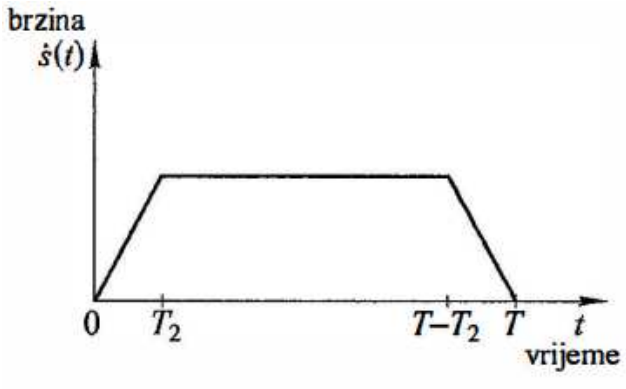
Najjednostavniji primjer funkcije $s(t)$ je:

$$s(t) = \frac{t}{T}, 0 \leq t \leq T$$

Gdje je *T* ukupno vrijeme.

Graf funkcije $\dot{s}(t)$ naziva se profil brzine. Tipičan profil brzine na kojoj se može vidjeti da robot na početku trajektorije, tj. u vremenskom intervalu $[0, T_2]$, ubrzava svoje gibanje do maksimalne brzine, zatim se određeno vrijeme giba tom brzinom te na kraju trajektorije, tj.

u intervalu $[T - T_{2,}T]$, usporava kretanje i zaustavlja se u krajnjoj točki zadane krivulje (Slika 3.1).



Slika 3.1. Profil brzine (preuzeto iz [3])

3.2. Gibanje manipulatora od točke do točke

Može se pronaći vrlo veliki broj operacija koje manipulator, odnosno vrh alata, obavlja uz gibanje od točke do točke. Takve su operacije točkasto zavarivanje, bušenje, podizanje i spuštanje itd. Osnovno načelo gibanja od točke do točke uključuje ove korake [3]:

1. pomak u zadani položaj,
2. zaustavljanje u zadanom položaju,
3. obavljanje zadatka,
4. pomak u sljedeći položaj

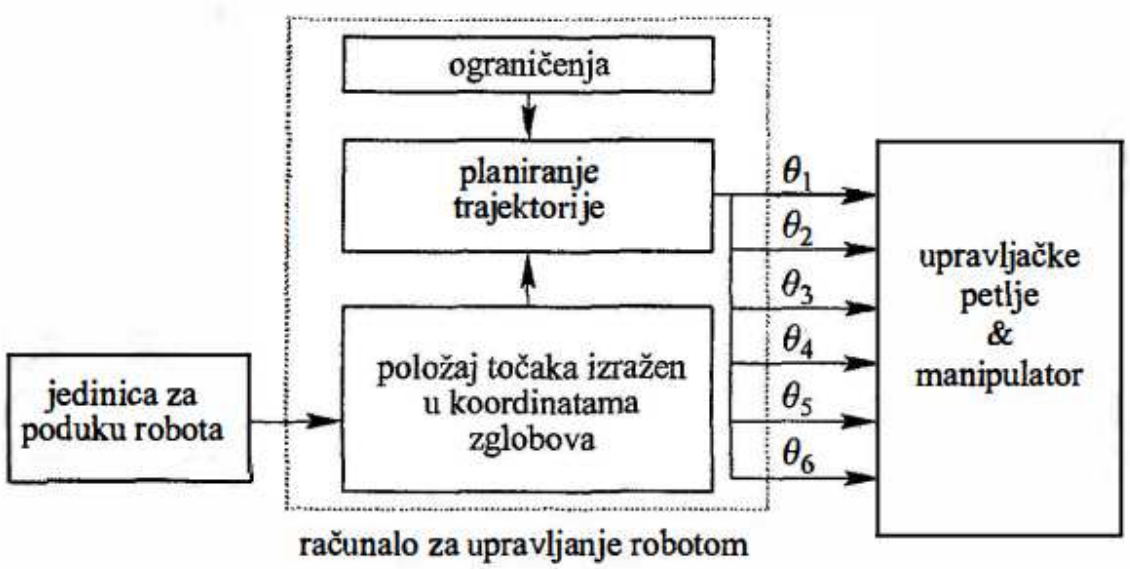
U velikom broju slučajeva pri gibanju od točke do točke trajektorija robota i brzina gibanja zapravo su nevažne. Pri tome se načelno gibanje robota može odvijati na dva osnovna načina [3]:

1. svaka os zasebno kreće se maksimalnom brzinom
2. sve osi završavaju gibanje istovremeno, što znači da se maksimalnom brzinom kreće ona os koja mora prevaliti najveću udaljenost dok se ostale osi gibaju sporije.

Pod pojmom planiranja trajektorije razumije se proračun ubrzanja, usporenja, sinkronizacija gibanja među osima (usklađivanje brzina) te izračunavanje položaja zglobova svih osi gibanja u skladu sa zadanom putanjom gibanja vrha alata robota.

Prilikom zadavanja uzastopnih položaja taj se postupak ciklički ponavlja putem odgovarajućeg programa, koji izvršava računalo za vođenje robota. Uobičajeni način zadavanja položaja jest korištenjem jedinice za poduku robota.

Pri planiranju trajektorije nužno je u proračune uključiti ograničenja koja kod robota objektivno postoje zbog geometrije robota i ograničenja radnog prostora te zbog realnih karakteristika konstrukcije robota (čvrstoća) i njegovih pogona (maksimalno dopušteno ubrzanje i brzina gibanja svakog zgloba). Načelna shema planiranja trajektorije za gibanje od točke do točke (Slika 3.2).



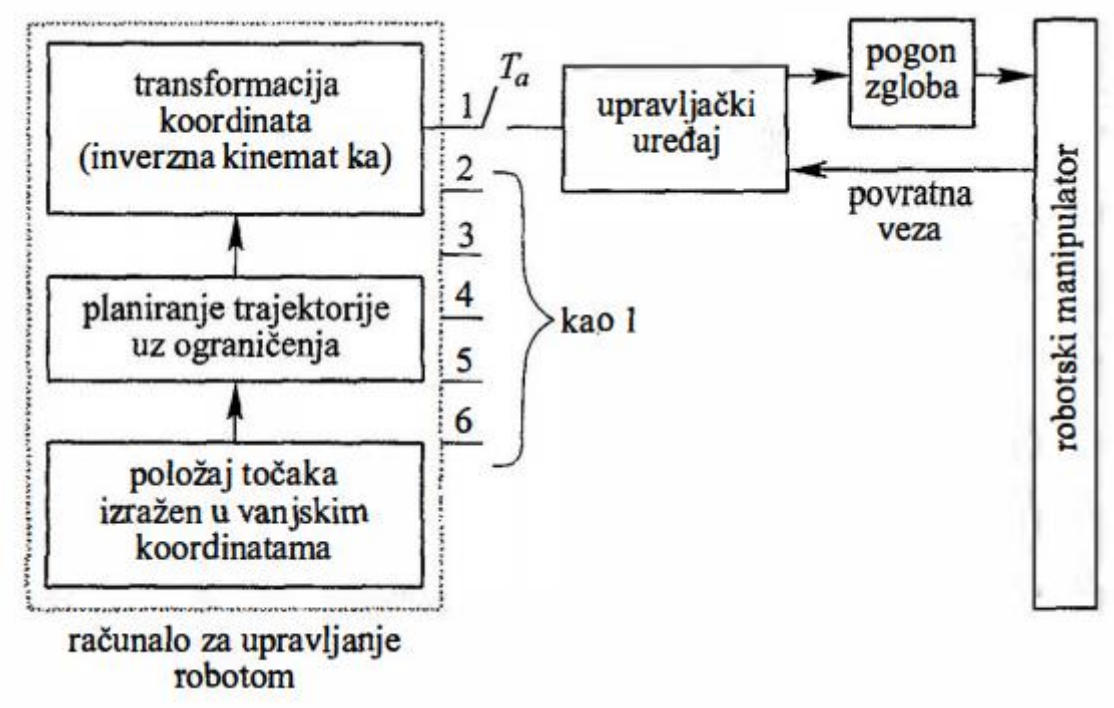
Slika 3.2. Planiranje trajektorije za gibanje rotacijskog robota od točke do točke (preuzeto iz [3])

3.3. Gibanje manipulatora kontinuirano po putanji

Kao i kod gibanja od točke do točke, tako se i za gibanje vrha alata manipulatora kontinuirano po putanji može pronaći velik broj operacija koje zahtijevaju takav karakter gibanja. Takve su operacije npr. zavarivanje, lijepljenje, inspekcija, skidanje srha, bojenje raspršivačem, složeni zadaci montaže itd.

Bitno obilježje robota koji se gibaju kontinuirano po putanji jest da se kretanje po putanji i obavljanje zadatka odvija istovremeno. To podrazumijeva da su brzine kretanja različite i da je, općenito, proračun referentnih veličina složeniji nego kod robota koji se gibaju od točke do točke. Ciljni položaj zadan je vanjskim koordinatama *x,y,z* u odnosu prema koordinatnom sustavu baze robota. To znači da zadane vanjske koordinate predstavljaju referentne položaje za odgovarajuće zglobove jedino u slučaju kartezijskog robota [3]. Za

ostale vrste robota, koji se građom razlikuju, potrebno je obaviti transformaciju koordinata i interpolaciju, za što je potrebno računalo. Načelna shema planiranja trajektorije za gibanje kontinuirano po putanji pri čemu T_a označava trajanje algoritma (Slika 3.3).



Slika 3.3. Planiranje trajektorije za gibanje robota kontinuirano po putanji (preuzeto iz [3])

3.4. Pravocrtno gibanje

Pravocrtno gibanje jest kretanje vrha alata robota po krivulji koja ima najmanju duljinu između dviju točaka u radnom prostoru. Takav tip gibanja koristi se pri radu s tekućom vrpcom ili pri zavarivanju.

Kod takvog problema potrebno je odrediti trajektoriju u prostoru zglobova, koja daje pravocrtnu krivulju u prostoru konfiguracije alata, pri čemu je moguće upotrijebiti jednadžbe inverzne kinematike.

Neka w^0 označava početnu, a w^1 završnu točku u prostoru konfiguracije alata. Tada se pravocrtna trajektorija vrha alata robota može definirati na sljedeći način, pri čemu T određuje ukupno vrijeme gibanja [3]:

$$w(t) = [1 - s(t)] \cdot w^0 + s(t) \cdot w^1, \qquad 1 \leq t \leq T$$

Jedna od važnih pravocrtnih putanja za većinu robota jest gibanje vrha alata po liniji koja je određena pomoću konstantnog vektora približavanja (takav tip kretanja potreban je npr.

pri operaciji umetanja objekata). Kod peteroosnog rotacijskog robota s poniranjem i valjanjem alata može se upravljati gibanjem po bilo kojem vektoru približavanja koji je dostupan robotu, pri čemu vrh alata mora ostati unutar radnog prostora.

Ako se brzine zglobova robota ne mogu neovisno mijenjati, tada je moguće upravljati gibanjem alata od točke do točke. Pri tome je potrebno pronaći drugačiji način upravljanja kontinuiranim kretanjem, kako bi se aproksimiralo pravocrtno gibanje.

4. Korištene tehnologije

4.1. Unity

Unity 3D je popularan game engine zbog relativne jednostavnosti izrade projekata i podrške za jako širok spektar platformi (trenutno podržava 25 različitih). Može se koristiti za izradu dvodimenzionalnih ili trodimenzionalnih igara, kao i igara u proširenoj i virtualnoj stvarnosti. Uz izradu igara koristi se i za simulacije i u raznim drugim industrijama kao što su filmska, automobilna, građevinska, robotika itd. Za razvoj se koristi programski jezik C# koji se za izvođenje prevodi u C++. U prošlosti su bili podržani i jezici Boo koji je ukinut s verzijom unity 5 koja je izašla 2016 i JavaScript koji su ukinuli verzijom 2017.1. Trenutno je aktualna verzija 2019.3 s konstantnim nadogradnjama.

Velika prednost Unityja naspram drugim game engineima je veličina zajednice. Trenutno ima oko 5,000,000 registriranih korisnika, posjeduje trgovinu Asset Store s raznim alatima za razvoj aplikacija kao što su modeli, animacije ili čak cijeli programski sistemi [2].

Potrebe simulatora za robotiku su slične potrebama video igre. Oboje trebaju način vizualizacije trodimenzionalnih objekata, simulaciji fizike, napredne algoritme kao što su inverzna kinematika, algoritam pronalaženja puta, umjetna inteligencija [2].

Prednost korištenja Unityja je i brzo prototipiranje koje uz pomoć gotovih sistema za fiziku, vizualizaciju, kolizije, napredni unos naredbi (*eng. input*) itd. Uz nativne Unityeve sisteme postoje i mnoga proširenja od strane korisnika koja su dostupna putem Unity Asset Storea ili GitHuba.

5. Implementacija

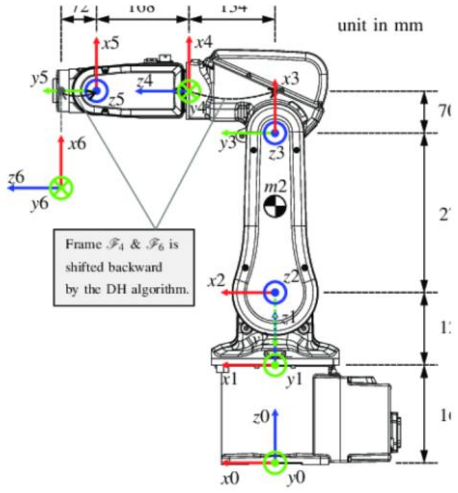
5.1. Robot

Robot korišten za implementaciju je multifunkcionalna robotska ruka ABB IRB 120 (Slika 5.2). Robot sa 6 osi slobode relativno male mase od 25 kg, dosega 0.58 m. Nosivost mu iznosi 3 kg, a vrijeme ubrzanja od 0 do 1 m/s postiže za 0.07 s. Ponovljivost mu iznosi 0.01 mm. Ograničenja rotacijskih zglobova su prikazane na Slika 5.1 [14].

Movement

Axis movement	Working range	Velocity IRB 120
Axis 1 rotation	+165° to -165°	250°/s
Axis 2 arm	+110° to -110°	250°/s
Axis 3 arm	+70° to -110°	250°/s
Axis 4 wrist	+160° to -160°	320°/s
Axis 5 bend	+120° to -120°	320°/s
Axis 6 turn	Default: +400° to -400° Max. rev: +242 to -242	420°/s

Slika 5.1. Ograničenja rotacijskih zglobova (preuzeto iz [14])



Slika 5.2. ABB IRB 120 (preuzeto iz [15])

5.1.1. Implementacija robota

Robot je implementiran kao skup zglobova na koje su dodani članci. U Unityevoj hijerarhiji objekata koja se ponaša kao graf scene u kojoj objekti na dnu pozicijsku ovise o objektima iznad sebe tj. izvršavanjem određene transformacije (promjena položaja, rotacije ili veličine) nad objektom roditelja automatski mijenjamo karakteristike svih objekata djece. Time postižemo da rotacija ili pomak jednog zgloba utječe na sve zglobove nakon njega.

Robot kao bazni objekt sadrži djecu zglobove (Slika 5.3). Zglobovi (u hijerarhiji imenovani: Base, First, Second, ..., Effector) Na sebi sadrže skriptu `RotationgJointWith` Bone koja definira parametre rotacijskog zgloba. Svaki zglob kao dijete ima sljedeći zglob i članak. Članci su imenovani FirstMesh, SecondMesh, ..., SixthMesh. Oni na sebi imaju komponente `MeshFilter` i `MeshRenderer`. U `MeshFilter` -u se određuje koji model će se koristiti za prikaz, a `MeshRenderer` definira kako će se objekt prikazati na zaslonu uporabom programa za sjenčanje (*eng. shader*) kao i interakciju objekta s izvorima svjetlosti. Članci ne posjeduju nikakvu logiku na sebi.

Zglob Effector uz skriptu `RotatingJointWithBone` posjeduje i skriptu `Effector` u kojoj se izvršava bojanje objekata koje će biti objašnjeno u kasnijem poglavlju.

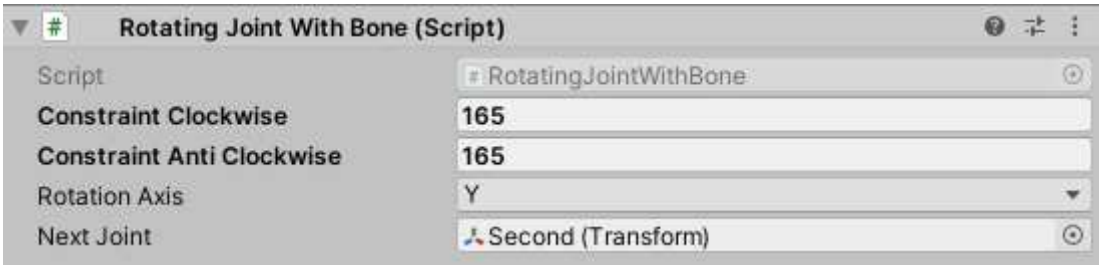
Bazni objekt Robot na sebi sadrži dvije skripte. Prva je skripta `CCD_IK` koja se brine za inverznu kinematiku, a druga je skripta `BaseMover` koja sadrži logiku za pomicanje baze robota kada je ciljna točka koju želimo dohvatiti izvan dosega.

Slika 5.3. Robot u hijerarhiji objekata

5.2. Implementacija Inverzne kinematike

Za rješavanje problema inverzne kinematike koristio sam algoritam cikličkog koordinatnog spusta (CCD) opisan u poglavlju 3.2. Odlučio sam se za taj algoritam jer su svi zglobovi robota ili rotacijski oko osi ili torzijski tako da implicitno podržava ograničenja rotacije. Razlika u odnosu na algoritam opisan u prethodnom poglavlju je potreba za rješavanjem problema u trodimenzionalnom prostoru umjesto dvodimenzionalnom. Osnovni koncepti su identični.

Za implementaciju algoritma sam koristio dvije klase: `RotatingJointWithBone` i `CCD_IK`. U `RotatingJointWithBone` definiram kutove ograničenja rotacije, os u lokalnom koordinatnom sustavu zgloba oko koje se odvija rotacija i duljinu članka (Slika 5.4). Referenca na sljedeći zglob je potrebna za izračun duljine članka. Sama skripta sadrži ne sadrži logiku.



Slika 5.4. Rotating Joint With Bone skripta

Skripta `CCD_IK` sadrži sva pravila za rješavanje problema inverzne kinematike. Za ulazne parametre potrebno je definirati listu zglobova, alat i objekt čija će se pozicija gledati kao ciljna točka (Slika 5.5).

Slika 5.5. CCD_IK skripta

Kako svaki zglob definira svoju lokalnu os rotacije tako da trodimenzionalni problem inverzne kinematike postaje dvodimenzionalan. Rotacija jednog zgloba se izvodi metodom (**Error! Reference source not found.**). Korištene varijable su: `ointToEffectorDirection` je vektor smjera od i-tog zgloba do alata, a `jointToTargetDirection` je vektor smjera od i-tog zgloba do ciljne točke. `RotationAxis` je lokalna os rotacije dobivena it `RotatingJointWithBone`, a `rotationAxisWorld` je ista ta os rotacije samo u globalnom koordinatnom sustavu scene. Unity definira funkciju `SignedAngle` koja prima vektore smjera od i do i globalnu os oko koje se računa kut s predznakom. Za rotaciju objekata u trodimenzionalnom prostoru se koriste kvaternioni.

```
private void RotateJointWithConstraints(int i)
{
    Vector3 jointToEffectorDirection = _joints.Last().position - _joints[i].position;
    Vector3 jointToTargetDirection = TargetPosition() - _joints[i].position;
    Vector3 rotationAxis = _joints[i].GetRotationAxis();
    Vector3 rotionAxisWorld = _joints[i].transform.localToWorldMatrix * rotationAxis;
    float angle = Vector3.SignedAngle(jointToEffectorDirection, jointToTargetDirection, rotionAxisWorld);
    Quaternion rotation = Quaternion.AngleAxis(angle, rotationAxis);
    Quaternion constrainedRotation = ConstrainRotation(_joints[i].localRotation * rotation, _joints[i]);
    _joints[i].localRotation = constrainedRotation;
}
```

Kod 1. Rotacija zgloba s ograničenjem

Prednost kvaterniona nad eulerovim kutovima za definiranje rotacije je karakteristika kvaterniona da rješavaju problem zastoja žiroskopskih osi (*eng. gimbal lock*) kojime se gubi os slobode ako se osi rotacije preklope. Nedostatak kvaterniona je neintuitivna reprezentacije kuta [16].

Funkcija `AngleAxis` prima kut i os rotacije, a vraća kvaternion koji odgovara toj rotaciji. Važno je napomenuti da ovdje koristimo lokalnu os rotacije jer zglob želimo rotirati u njegovom lokalnom koordinatnom sustavu. Funkcija `ConstrainRotation` na ulaz prima kvaternion rotacije i zglob u kojem su definirane granice rotacije. Ta metoda pretvara kvaternion u vektor eulerovih kuteva i provjerava jesu li kutevi unutar granica. Ako prelaze granicu postavljaju se na vrijednost granice. Zadnji red metode postavlja kvaternion kao lokalnu rotaciju zgloba.

Glavna metoda za rješavanje CCD algoritama je relativno jednostavna. U jednoj iteraciji rotiramo sve zglobove počevši od najudaljenijih od baze. Iteriramo dok alat ne dosegne cilj ili se izvrši određeni broj iteracija (Kod 2).

```
public void ResolveIK()
{
    int count = 0;
    if (_target == null) return;
    while (!HasReachedTarget())
    {
        if (count == MAX_ITERATIONS) break;
        for (int i = _joints.Count - 1; i >= 1; i--)
        {
            RotateJointWithConstraints(i);
        }
        count++;
    }
}
```

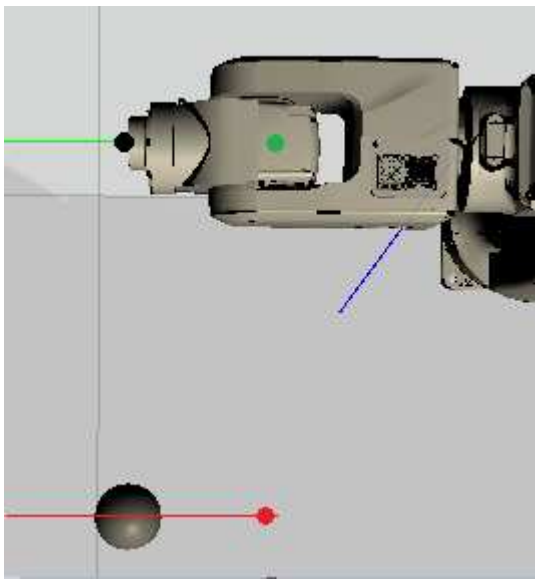
Kod 2. CCD algoritam inverzne kinematike

Nakon provođenja algoritma robot doseže ciljnu točku. Primijetimo da iteracija ne obuhvaća prvi bazni zglob jer se on ne bi trebao pomicati (Slika 5.6). Problem s ovim rješenjem je što orijentacija alata nije okomita na zid. Primijetimo da bi dobili traženu orijentaciju moramo rotirati zglob s alatom i njegov zglob roditelj. Kako mu je zglob roditelj torzijski zglob (zglob 4 na Slika 5.2.) znamo da je lokalna pozicija zgloba s alatom (zglob 5 na Slika 5.2) neovisna o rotaciji četvrtog zgloba. To znači da ako kao cilj algoritma inverzne kinematike definiramo minimizaciju udaljenosti pozicije četvrtog zgloba (Slika 5.7 zelena točka) i pomaka ciljne točke za duljinu petog članka u smjeru vektora okomitog na zid (Slika 5.7 crvena točka). Kada se te dvije točke poklope moramo još samo odrediti rotaciju četvrtog i petog zgloba.

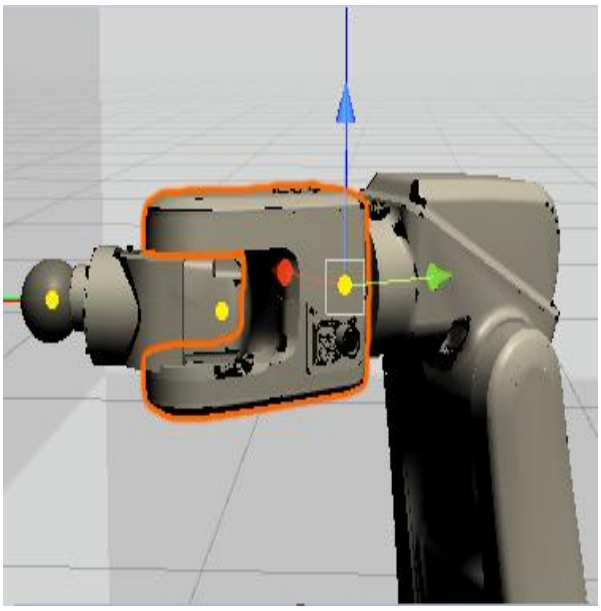
Četvrti zglob želimo rotirati tako da se njegova lokalna orijentacija po z osi poklopi s vektorom normale dobivene točkama alata, pozicije četvrtog zgloba i pozicije cilja. Na Slika 5.8 ta normala je označena plavom linijom, a točke koje tvore ravninu žutom bojom. Lokalna z os je označena plavom strelicom.

Peti zglob rotiramo tako jednostavno oko osi tako da se poklopi vektor od četvrtog zgloba do alata s vektorom smjera zida. Rezultanti izgled robota je prikazan na Slika 5.9.

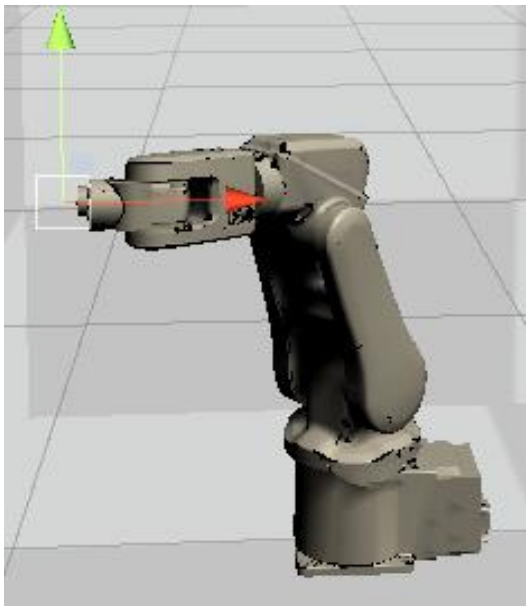
Slika 5.6 Robot doseže cilj



Slika 5.7. Ključne točke za orijentaciju



Slika 5.8. Orijentacija četvrtog zgloba



Slika 5.9. Robot pravilno orijentira alat

5.3. Generiranje zidova

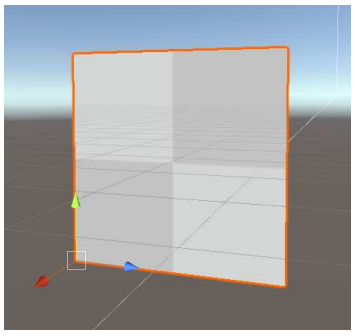
Zidove sam predstavio kao dvodimenzionalne plohe. Svaki zid posjeduje komponente `MeshFilter` i `MeshRender` koje su prije opisane. Također posjeduje komponentu `MeshCollider` koja služi za registraciju kolizija i skripte Paintable koju ću objasniti u narednom poglavlju i skriptu `DeformMesh` koja izobličuje zid na početku simulacije.

`DeformMesh` kao parametar uzima vektor smjera deformacije i temeljem njega i funkcije Perlinovog šuma (*eng. Perlin noise function*) pomiče vrhove objekta. Operacije koje se računaju za svaki vrh objekta obično se izvršavaju na grafičkoj kartici putem skripte sjenčara (*eng. shader*) čime bi bila mnogo optimiziranija, ali onda procesor ne bi raspolagao podacima pozicija vrhova pa se sudari ne bi mogli pravilno računati.

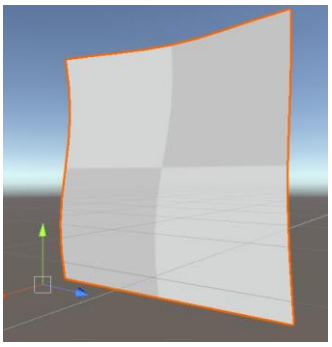
Funkcija Perlinovog šuma je funkcija pseudo slučajnih brojeva (Slika 5.11) sa svojstvom da su joj rezultati prirodni poredani tj. da razlike u kratkom periodu vremena nisu velike [17]. Za razliku od uobičajnih funkcija pseudo slučajnih bojeve (Slika 5.10). Za zid je korištena dvodimenzionalna funkcija Perlinovog šuma (Slika 5.13).

Slika 5.10 Funkcija slučajnih brojeva (preuzeto iz [17])

Slika 5.11. Perlinova funkcija slučajnih brojeva(preuzeto iz [17])



Slika 5.12. Ravan zid



Slika 5.13. Izobličen zid

5.4. Bojanje zidova

Za prikaz boje objekta najjčešće se koriste teksture. U teksturu se pospremaju podaci o boji pojedinog vrha objekta. Teksture se mapiraju na objekte uv mapama koje određuju koji piksel na dvodimenzionalnoj teksturi odgovara kojemu vrhu trodimenzionalnog objekta [5].

Svaki objekt koji se može bojati mora na sebi imati skriptu `Paintable`, mora imati nešto veći broj vrhova npr. zidovi koji su dvodimenzionalne plohe se mogu prikazati sa samo 4 vrha ali za precizno bojanje je potrebno mnogo više. Također tekstura koja je definirana u sjenčaru objekta ne smije biti kompresirana, po mogućnosti zapisana u formatu RGBA32, ne smije generirati mipmape i mora imati omogućenu funkcionalnost čitanja i pisanja (sve te mogućnosti de mogu podesiti prilikom unošenja (*eng. import*) teksture u Unity).

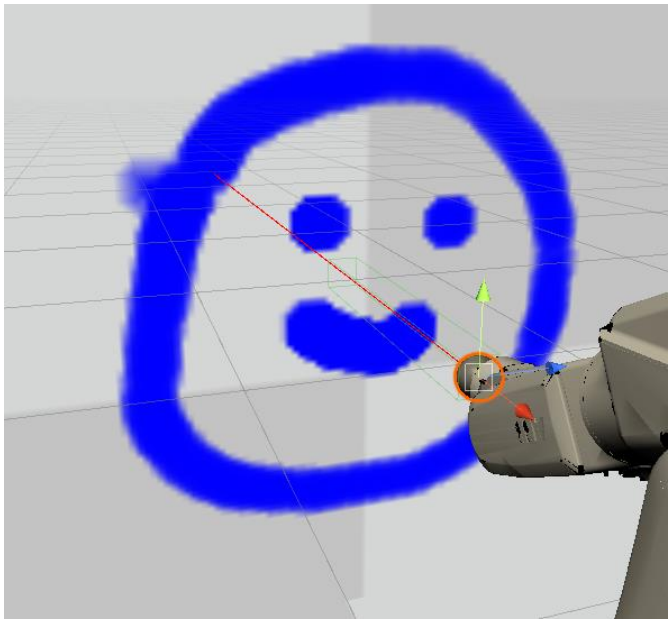
Skripta `Paintable` na početku simulacije stvara kopiju teksture objekta koju onda postavlja kao aktivnu teksturu u materijalu. Kopija je nužna jer bi sve promjene nad teksturom bile trajne i svi objekti koji dijele teksturu bi se bojali istovremeno. Prilikom završetka simulacije kopirana tekstura se briše.

Funkcionalnost crtanja po teksturi se šalje zraku od vrha alata prema zidu. Glavna metoda je Unityjeva `Physics.Raycast` kojom se ispituje pogađa li zraka neki objekt u sceni. Zraka je definirana kao vektor smjera s početnom točkom, u implementaciji je to vektor orijentacije zida od vrha alata. Ako pogađa metoda vraća istinu i sve podatke vezane uz pogodak zapisuje u varijablu hit tipa `RaycastHit`. Jedan od podataka koji se zapisuju su dvodimenzionalne uv koordinate teksture. Te koordinate su zapisane kao decimalni brojevi između 0 i 1 pa ih je potrebno skalirati s veličinom teksture. Metoda `PaintWithRadius` samo mijenja boje piksela u zadanom polumjeru (Kod 3).

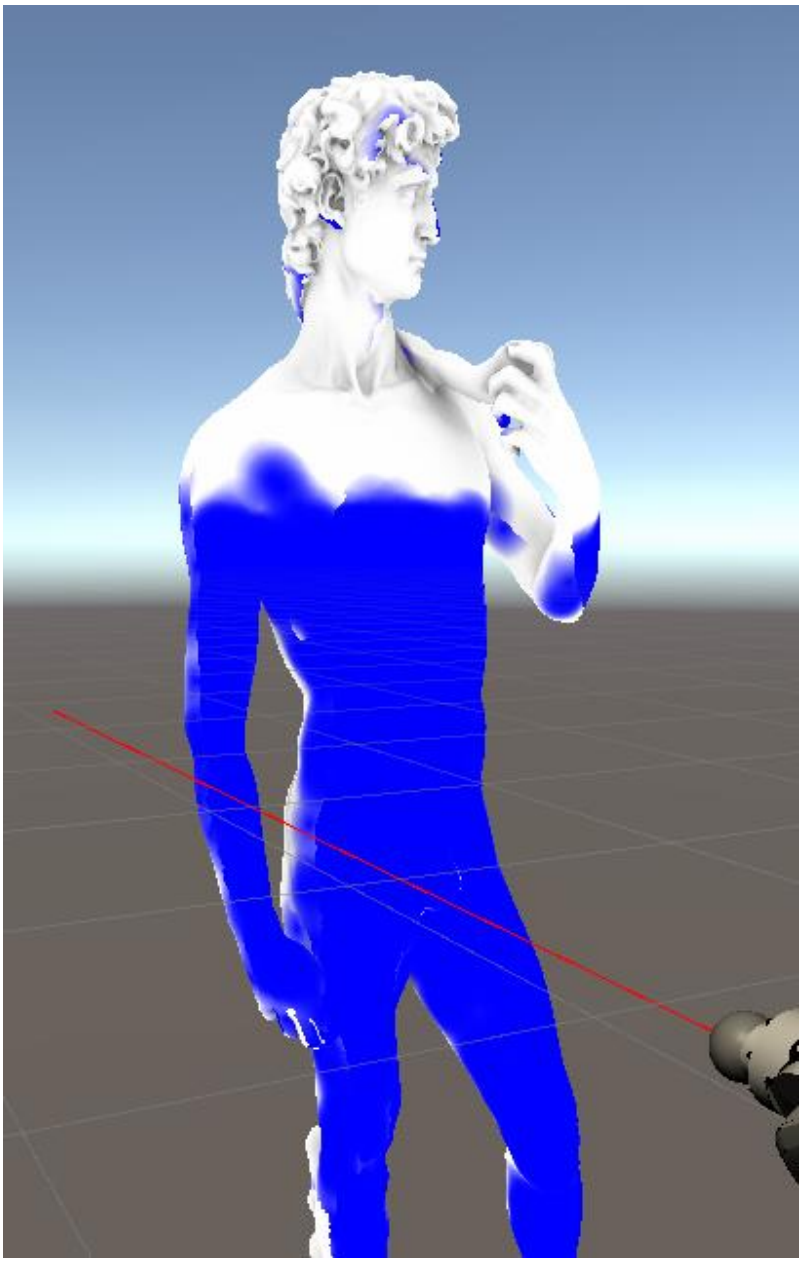
```
public void PaintOnTexture(Vector3 direction, Effector effector, int radius)
{
    radius = (int)(radius * radiusModifier);

    Ray ray = new Ray(effector.transform.position, direction);

    if (Physics.Raycast(ray, out RaycastHit hit))
    {
        Vector2 uvHit = hit.textureCoord;
        uvHit.x *= _workingTexture.width;
        uvHit.y *= _workingTexture.height;

        PaintWithRadius(uvHit, radius);
    }
}
```

Kod 3. Bojanje teksture

Ova metoda radi dobro u praksi. Na jednostavnim objektima radi bez ikakve mane (Slika 5.14) ali na složenijim objektima sa složenijim uv mapama može doći do manjih pogrešaka (Slika 5.15). Na složenijem modelu vidimo da su pobojani i dijelovi kose iako robot ne može doseći iznad razine prsiju. Pogreška se javlja zbog raspodjele uv mape tj. pikseli čija se boja koristi kao boja kose su susjedi pikselima tijela tako da prilikom bojanja uđu u krug piksela čiju boju mijenjamo. Rješenje za taj problem bi moglo biti pucanje više zraka za bojanje takvih da svaka zraka odgovara samo jednom pikselu teksture.



Slika 5.14. Bojanje po jednostavnom objektu.



Slika 5.15. Bojanje po složenijem objektu

5.5. Pomicanje baze

Kao je doseg robota mali naspram veličini prostorije robota je potrebno pomicati kada mu ciljna točka izađe izvan dohvata. Implementacija se sastoji od dvije bitne funkcije. Prva je funkcija za pomak baze robota, a druga je funkcija koja se brine za održavanje udaljenosti između robota i zidova.

Pomicanje baze se dvija svaki put kada alat nakon izvođenja funkcije inverzne kinematike ne može doseći ciljnu točku. Funkcija kao parametar dobiva vektor smjera od alata prema ciljnoj točki. Pomicanje je ograničeno na os paralelnu sa zidom. Pomak lijevo ili desno određujemo predznakom skalarnog umnoška ulaznog parametra s vektorom smjera desnog zida. Funkcija `MaintainDistanceToWall` osigurava sigurnu udaljenost robota od zida (Kod 4).

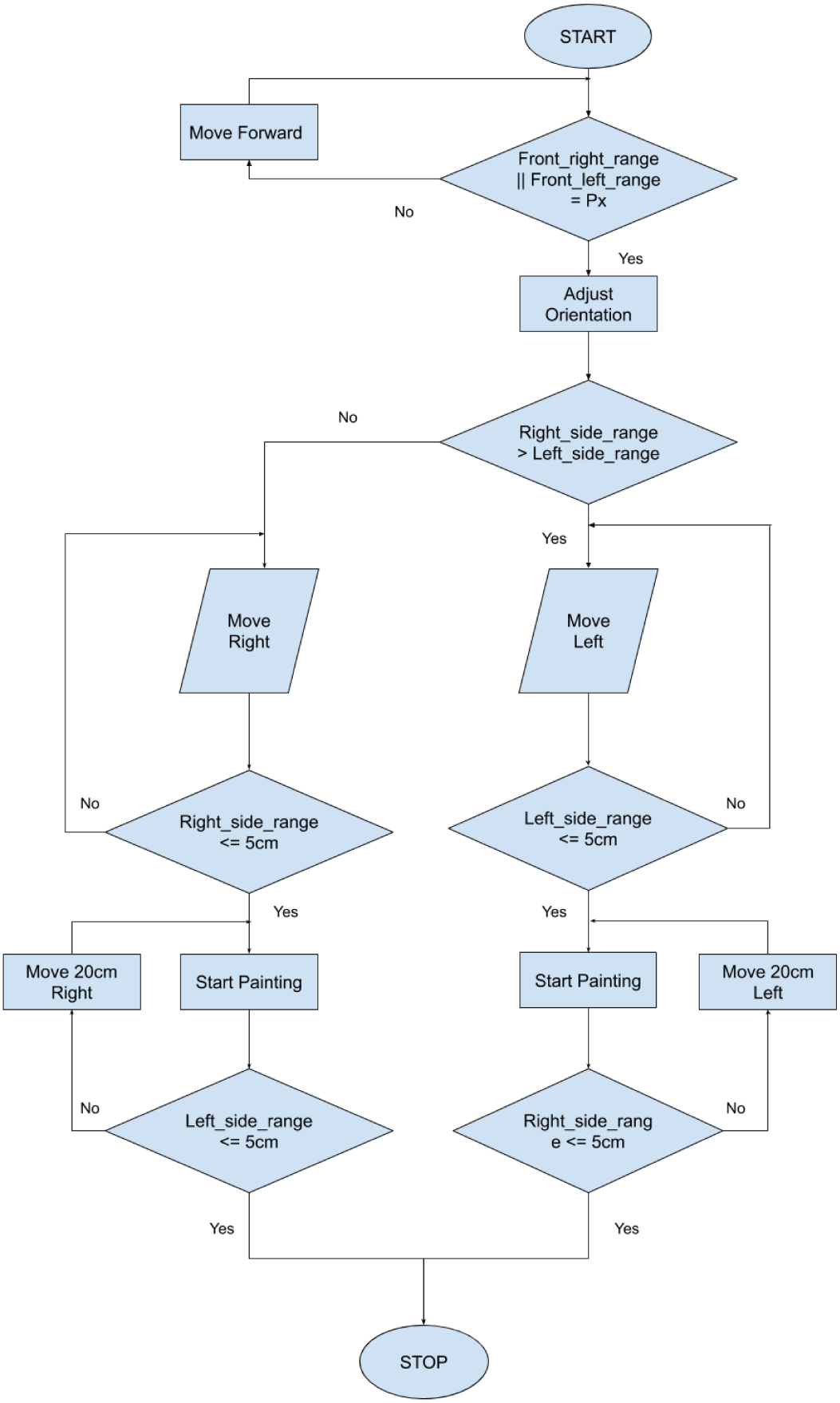
```
private void Move(Vector3 direction)
{
    float distance = 0.1f;

    Vector3 wallDirection = Target.Instance.GetDistanceFromWallRightVector().normalized;
    Vector3 moveDirection;
    direction.y = 0;

    if (Vector3.Dot(direction, wallDirection) > 0) moveDirection = wallDirection;
    else moveDirection = wallDirection * -1;

    moveDirection = moveDirection.normalized * distance;
    moveDirection = MaintainDistanceToWall(moveDirection);
    this.transform.position = moveDirection;
}
```

Kod 4. Pomicanje baze robota

5.6. Planiranje putanje alata

Cilj simulacije je bojanje prostorije. Problem bojanja prostorije razdjeljujemo u potproblem bojanja jednog zida i rotacija robotske ruke prema sljedećem zidu.

Plan puta za premazivanje jednog zida je pronalaženje i pozicioniranje alata u jedan od kuteva zida. Uzmimo slučaj da se pomaknemo u gornji lijevi kut. Od njega se pomičemo pravocrtno kontinuirano do poda. Slijedi pomak u desno za promjer spreja pa pomak prema pravocrtno do stropa i jedan završni pomak u desno da bi odradili jedan ciklus koji onda ponavljamo do desnog zida (Slika 5.16).



Slika 5.16. Algoritam bojanja jednog zida (adaptivno iz [6])

Ovaj algoritam je implementiran u skripti `OneWallPath` s razlikom da uvijek forsira gornji lijevi kut za početni. Algoritam koristi Unityjev sustav korutina koje omogućuju pauziranje odvijanja funkcije do nekog kasnijeg trenutka npr. ciljnu točku pomičemo samo za mali dio puta unutar jednog osvježavanja ekrana.

Pomicanje alata prema gornjem lijevom kutu pronalazi ciljnu poziciju i linearno interpolira alat prema njoj (Kod 5). Ova metoda se može pozvati kao korutina jer vraća enumerator i ključnom riječi yield staje s izvršavanjem dok se ne izvrši metoda `LerpTargetToDestination` koja je također korutina koja svako osvježavanje ekrana pomakne ciljnu točku prema destinaciji uz održavanje minimalne udaljenosti od zida (Slika 5.19 i Slika 5.20). Mana ovog koda je što poziciju kuta računam pomoću udaljenosti od ciljne točke. Idealno bi robot sa svojim senzorima trebao prepoznati gdje se nalazi kut.

```
public IEnumerator MoveTargetToUpperLeftWallCorner()
{
    Vector3 ceilingDistanceVector = target.GetDistanceFromWallAboveVector();
    Vector3 distanceLeftVector = target.GetDistanceFromWallLeftVector();
    Vector3 movement = ceilingDistanceVector + distanceLeftVector;

    Vector3 destination = target.transform.position + movement;
    yield return LerpTargetToDestination(destination);
}
```

Kod 5. Pomicanje alata u gornji lijevi kut zida

Kod za bojanje zida jednog zida nakon pozicioniranja u gornji lijevi kut se sastoji od pomicanja alata prema podu, pomicanja desno i pomicanja prema stropu. Funkcija `DonePainting` vraća istinu kada je udaljenost između alata robata i desnog zida dovoljno malena (Kod 6).

```
public IEnumerator PaintCoroutine()
{
    do{
        yield return PaintTillFloor();
        yield return PaintRightForRadius();
        yield return PaintTillCeiling();
        yield return PaintRightForRadius();
    } while (DonePainting());
    yield return PaintTillFloor();
}
```

Kod 6. Bojanje jednog zida

Rotacija prema sljedećem zidu je pojednostavljena tako da se robot rotira u desno za 90 stupnjeva i teleportira u novi položaj. Moguća je dorada tako da se pauzira pomak ciljne točke i onda izvrši linearna interpolacija robota do novog položaja (Kod 7).

```
private void MoveAfterRotation()
{
    Target target = Target.Instance;
    Vector3 defaultDistanceFromWall = target.GetDistanceFromWallRightVector().normalized *
                                      (Parameters.LEFT_WALL_DISTANCE_AFTER_ROTATION - Parameters.EFFECTOR_WIDTH)
                                    + target.RotationDirection *
                                      (Parameters.FRONT_WALL_DISTANCE_AFTER_ROTATION - target.distanceToMaintain);
    Vector3 destination = target.transform.position + defaultDistanceFromWall;
    destination.y = 0;
    this.transform.position = destination;
    //StartCoroutine(LerpBaseToDestination(destination));
    CCD_IK.Instance.ResolveIK();
}
```

Kod 7. Pomicanje baze robota nakon rotacije

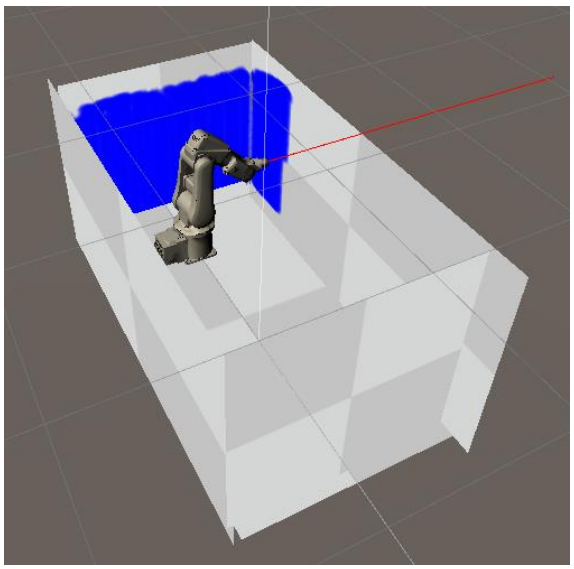
5.7. Rezultati

Rezultat rada je robot koji je u stanju kontinuirano bojati pravokutnu prostoriju sa zakrivljenim zidovima (Slika 5.23). Pri čemu održava konstantnu udaljenosti između zida i alata (Slika 5.19 i Slika 5.20) i pravilno se rotira prema sljedećem zidu (Slika 5.17) pri čemu zahtjeva da postoji zid s desne strane robota. Pri planiranju puta pazi se na minimizaciju dvostrukih prijelaza tako da se funkcija bojanja isključi tijekom repozicioniranja u gornji lijevi kut i udaljenost pomicanja alata prema desno se određuje radiusom spreja.

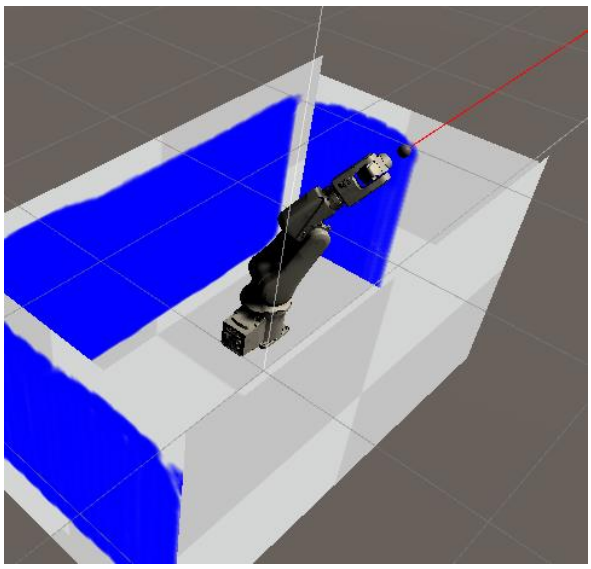
Robotska ruka se pokreće svaki puta kada se ciljna točka pomakne. To kao posljedicu ima nemogućnosti vertikalnog dosega robota do točke (Slika 5.17). Rezultat je neravnina na gornjem i donjem rubu pobojanog djela.

Problem koji se javlja zbog mane u Unityevom sjenčaru za prikaz teksture na plohi je prikaz plohe koja bi trebala biti iza druge ispred nje (Slika 5.18 neobojeni dio zida se nalazi na dijelu plohe lijevog zida koja je iza plohe prednjeg zida). Problem se rješava pomicanjem kamere.

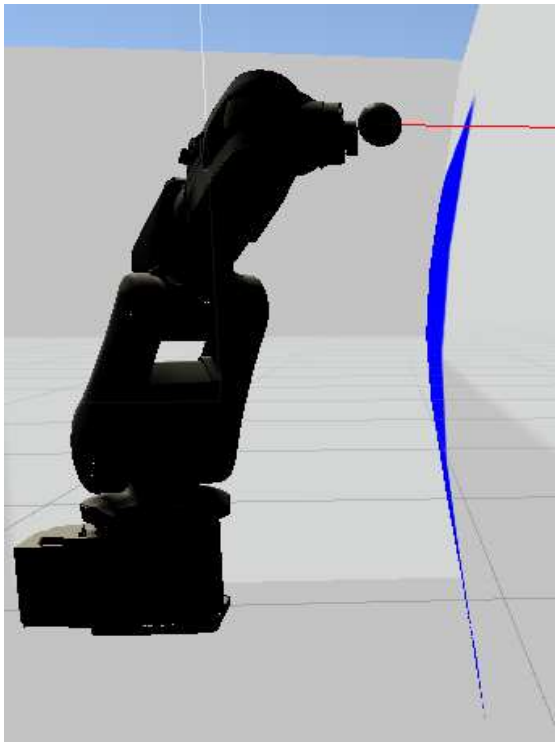
Ograničenja algoritma planiranja putanje alata dolaze do izražaja kada su neravnine na zidovima jako izražene. Može de dogoditi da robot krivo zaključi da je dosegao pod ili strop prostorije (Slika 5.21). Također isti problem se može javiti i prilikom prelaska na novi zid ako su rubovi jako ispupčani (Slika 5.22).



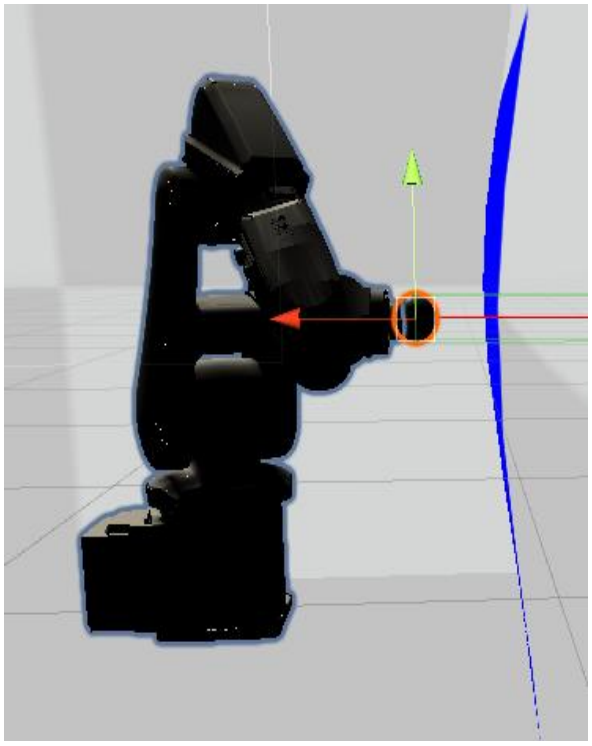
Slika 5.17. Početak bojanja male prostorije



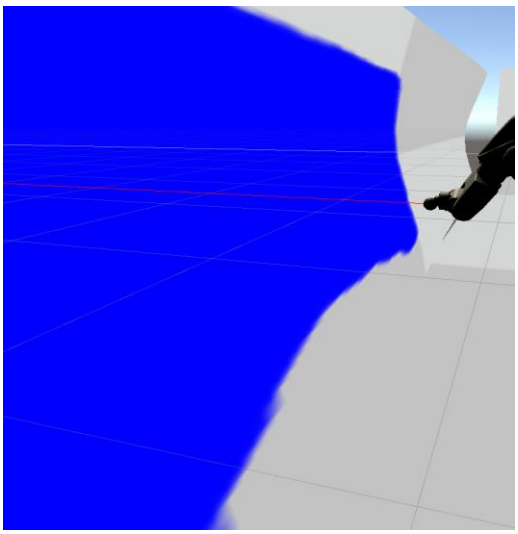
Slika 5.18. Krivo iscrtavanje zida



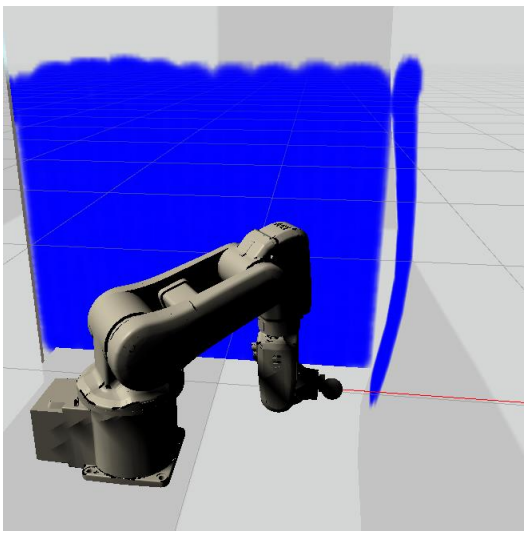
Slika 5.19. Održavanje udaljenosti od zida



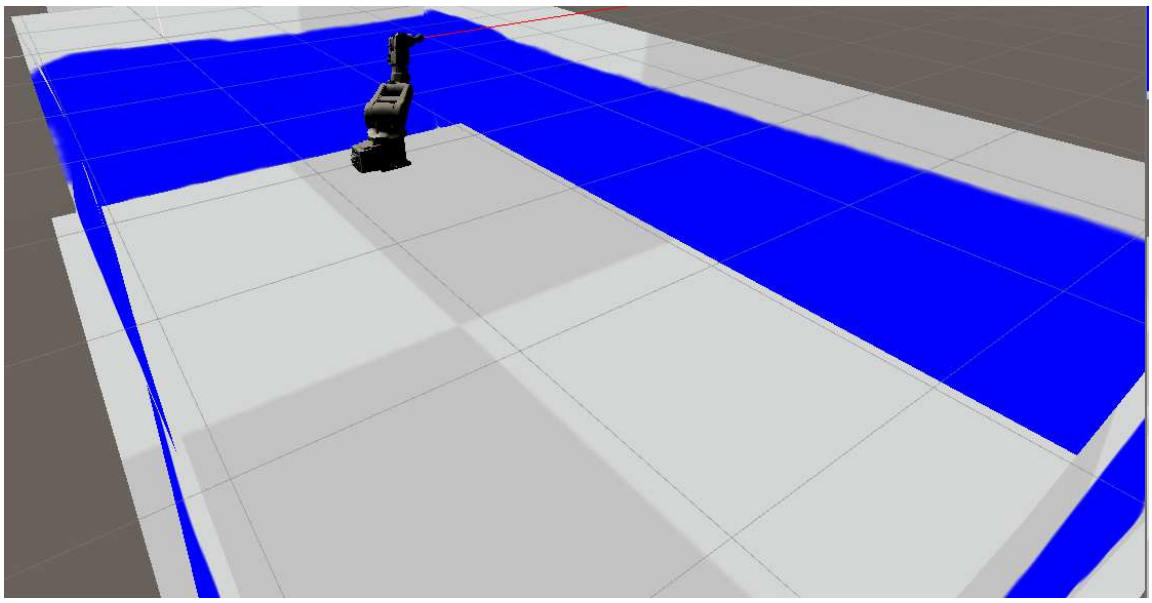
Slika 5.20. Održavanje udaljenosti od zida sredina



Slika 5.21. Loša detekcija tla



Slika 5.22. Loš detekcija ruba



Slika 5.23 Obojana velika prostorija sa zakrivljenim zidovima

Zaključak

U ovom radu smo proučavali temelje robotike i njene ključne probleme i algoritme. Također smo implementirali simulaciju robota za premazivanje zidova. Opisali smo tri algoritma inverzne kinematike od kojih svaki uzima drugačiji pristup za rješenje istog problema. Dotakli smo se i problema pronalaženja putanje alata robota gdje smo opisali gibanje manipulatora od točke do točke i gibanje manipulatora kontinuirano po putanji.

Slijedio je opis implementacije robota u sustavu za razvoj digitalnih igara Unity. Za implementaciju je bilo potrebno definirati strukturu robota u hijerarhijskom grafu scene, implementirati sustav za rješavanje problema inverzne kinematike uz ograničenja, generirati prostoriju koju robot treba premazati, vizualizirati bojanje zidova, definirati pomicanje baze robota za dohvaćanje ciljnih točaka van svoga originalnog dosega i isplanirati cjelokupnu putanju alata.

Budući razvoj bi uključivao mogućnost prebojavanja složenijih prostorija koje ne moraju biti pravokutne ili mogu sadržavati više od četiri zida, metodu prepoznavanja objekata na zidu i njihovo zaobilaženje i implementacija sustava dinamike manipulatora koje bi robota pomicali fizičkim silama umjesto promjenama pozicije i rotacije u trodimenzionalnom prostoru.

Literatura

[1] B. Novaković, *Robotika,* Leksikografski zavod Miroslav Krleža, Poveznica: https://tehnika.lzmk.hr/robotika/; pristupljeno 15. svibnja 2020.

[2] *Jeff Linnell, Robots are hard, game engines are not: why we built our own simulator using Unity,* Formant, Poveznica: https://formant.io/news-and-blog/2019/10/09/analytics/robots-are-hard-game-engines-are-not-why-we-built-our-own-simulator-using-unity/; pristupljeno 15. Svibnja 2020.

[3] Zdenko Kovačić, Stjepan Bogdan, Vesna Krajči. *Osnove robotike,* 2. izdanje, Zagreb: Graphis 2018.

[4] Andreas Aristidou, *Forward And Backward Reaching Inverse Kinematics*, University of Cyprus, Poveznica: http://www.andreasaristidou.com/FABRIK.html; pristupljeno 16. svibnja 2020.

[5] Shahriar Shahrabi, *Mesh Texture painting in Unity Using Shaders*, Medium, Poveznica: https://medium.com/@shahriyarshahrabi/mesh-texture-painting-in-unity-using-shaders-8eb7fc31221c; pristupljeno 16. svibnja 2020.

[6] Mohamed Abdellatif, *System Design Considerations for Autonomous Wall Painting Robot,* International Journal of Engineering Research & Technology (IJERT), Poveznica: https://www.ijert.org/research/system-design-considerations-for-autonomous-wall-painting-robot-IJERTV2IS100556.pdf; pristupljeno 16. svibnja 2020.

[7] Baoyong Zhang, *Path Planing for Spray Pinting Robot of Workpiece Surfaces,* Hindawi, Poveznica: https://www.hindawi.com/journals/mpe/2013/659457/; pristupljeno 16. svibnja 2020.

[8] Donald L. Pieper, *The kinematics of manipulators under computer control*, Standford University, Department of Mechanical Engineering, 24. Lipanja 1968.

[9] Luis Bermudez, *Overview of Jacobian IK*, Medium, Poveznica: https://medium.com/unity3danimation/overview-of-jacobian-ik-a33939639ab2; pristupljeno 15. Lipnja 2020

[10] Sammuel R. Buss, *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods,* University of California, Department of Mathematics, San Diego, 7. Listopad 2009

[11] Li-Chun Tommy Wang and Chin Cheng Chen, *A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulation*, IEEE on robotics and automation, Vol 7., Kolovoz 1991.

[12] Ryan Juckett, *Cyclic Coordinate Descent in 2d*, Poveznica : http://www.ryanjuckett.com/programming/cyclic-coordinate-descent-in-2d/; Pristupljeno 16. Lipnja 2020

[13] Andreas Aristidou and Joan Lasenby, *FABRIK: A fast, iterative solver for the Inverse Kinematics problem,* Cambridge, 15. svibnja 2011.

[14] ABB, IRB 120, https://new.abb.com/products/robotics/industrial-robots/irb-120; Pristupljeno 18. Lipnja 2020

[15] Thomas Varhegyi, Martin Melik-Merkumians, Michael Steinegger, Georg Halmetschlager-Funkel and Georg Schitter, *A Visual Setvoing Approach for a Six Degrees-of-Freedom Industrial Robot by RGB-D Sensing,* Beč, Konferencija OAGM & ARW, listopad 2017.

[16] Unity documentation, Rotation and Orientation in Unity, https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html; Pristupljeno 20. Lipnja 2020

[17] Khan Academy, *Perlin noise*, https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise; Pristupljeno 22.Lipnja 2020

Sažetak

Prilagodljivo upravljanje virtualnim robotom korištenjem sustava Unity na primjeru premazivanja zidova

Robotika je široko područje na granici strojarstva i računarstva. Proučili smo osnovne pojmove robotike, tri algoritma inverzne kinematike i problem planiranja putanje alata. Implementacija robota za premazivanje zidova ostvarena je u programskom sustavu Unity i sadrži implementaciju algoritma cikličkog koordinatnog spusta za rješavanje problema inverzne kinematike, generiranje zidova korištenjem funkcije Perlinovog šuma, bojanje zidova manipulacijom tekstura trodimenzionalnog objekta i algoritam za planiranje putanje alata s održavanjem udaljenosti od zidova. Rezultat je implementacija koja može efikasno premazati cjelokupnu prostoriju s izobličenim zidovima.

Ključne riječi: robotika, inverzna kinematika, putanja alata

Summary

Adaptable control of virtual robot in Unity for a wall painting task

Robotics is a wide scientific area on the border of mechanical engineering and computing. We studied the basic concepts of robotics, three algorithms of inverse kinematics and the tool oath planing problem. The implementation of the wall painting robot was realisedin the Unity game engine and includes the implementation of a cyclic coordinate descent algorithm to solve the problem of inverse kinematics, a wall generation algorithm using the Perlin noise function, painting wall by manipulating a three-dimensional objects texture and a tool path planing algorithm whitch maintains the distance between the tool and the wall. The resulting implementation can effectively paint an entire room with distorted walls.

Keywords: robotics, inverse kinematics, tool path planing