SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6511

PRILAGODLJIVO UPRAVLJANJE VIRTUALNIM ROBOTOM KORIŠTENJEM SUSTAVA UNITY NA PRIMJERU SLAGANJA DASAKA

Karlo Čulo

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6511

PRILAGODLJIVO UPRAVLJANJE VIRTUALNIM ROBOTOM KORIŠTENJEM SUSTAVA UNITY NA PRIMJERU SLAGANJA DASAKA

Karlo Čulo

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6511

Pristupnik:	**Karlo Čulo (0036500146)**
Studij:	Računarstvo
Modul:	Programsko inženjerstvo i informacijski sustavi
Mentor:	doc. dr. sc. Mirko Sužnjević

Zadatak: **Prilagodljivo upravljanje virtualnim robotom korištenjem sustava Unity na primjeru slaganja dasaka**

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 dasaka te ih slagati u organizirane cjeline. U procesu slaganja potrebno je detektirati veličinu dasaka kao i poprečne daščice koje omogućuju sušenje. Prikažite funkcionalnost na primjeru dinamički skupine dasaka na kojima će biti stogovi dasaka različitih karakteristika kao i različite veličine i učestalosti poprečnih daščica. Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Rok za predaju rada: 12. lipnja 2020.

Zahvala

Zahvaljujem svom mentoru doc. Dr. sc. Mirku Sužnjeviću na sjajnoj motivaciji, povjerenju i konstruktivnim kritikama. Zahvaljujem kolegi Nikoli Vugdeliji na korisnim savjetima i uputama.

Sadržaj

Uvod

Čovjek je još od svojih početaka koristio visoku inteligenciju da pretvori obične predmete u oruđa koja bi mu olakšavala svakidašnjicu, obrađivao i prerađivao razne materijale kako bi unaprijedio ista. Tijekom povijesti oruđa su postajala sve složenija i samim time puno više olakšavala čovjekov život.

Danas, tehnologija je napredovala u tolikoj mjeri da čovjek ne mora koristiti oruđa niti obavljati razne poslove jer je za to stvorio robote. Roboti su svakim danom više prisutni u našim životima i sve su napredniji i sofisticiraniji. Razvoj programske potpore za robota vrlo je složen proces, zahtjeva mnogo pažnje i testiranja jer i malene greške mogu biti vrlo skupe te mogu znatno usporiti čitav proces. Testiranje se vrši na pravom robotu, što znači da samo jedan član može testirati svoj rad na jednom robotu, a veći broj robota za testiranje je ujedno i veći trošak što nije obilježje inženjerskog pristupa rješavanja problema.

Upravo taj problem možemo riješiti stvaranjem virtualnog robota u nekom od razvojnih okruženja kao što je Unity. Razvijanje robota u takvom okruženju zahtjeva daleko manje resursa, dok u isto vrijeme daje vrlo precizne rezultate. Uz Unity i inverznu kinematiku gore navedeni problemi rješavaju se znatno lakše te uz uporabu određenih sučelja odrađeni posao s relativnom lakoćom se prenosi u stvarne robote. Unity nam omogućuje vrlo preciznu izradu objekata, simuliranje fizičkih pojava kao što su gravitacija i trenje, praćenje raznih parametara te automatizaciju putem skripti pisanih u programskom jeziku C.

Primjena potonjeg odrađena je u ovom završnom radu kroz automatizacija robota IRB 120 tvrtke ABB. Zadatak robota je da prepozna daske zadanih dimenzija te ih posloži u organiziranu hrpu, to čini tako da skenira svaku dasku pa ovisno o svojstvima daske premjesti istu na drugo mjesto. Kretnje robota uglavnom su izvedene implementacijom algoritma inverzne kinematike, točnije FABRIK algoritmom (eng. Forward and Backward Reaching Inverse Kinematics). Spomenuti algoritam funkcionira tako da se krajnja komponenta robota pomiče kroz prostor, a ostale komponente ju prate od prolazeći kroz petlje algoritma koji je u drugom poglavlju preciznije opisan.

Cilj ovog rada jest prikazati i dokazati prethodno navedene tvrdnje. Kroz Unity te primjenu algoritma inverzne kinematike spomenuti robot automatski se kreće kroz prostor, skenira objekte i pamti značajne položaje, odlučuje kako organizirati daske na temelju zadanih

parametara pri pokretanju procesa. Što sve točno radi robot i način na koji to radi opširnije su opisani u sljedećim poglavljima.

1. Tehnologije i alati

Razvijanje ovog rada izvršeno je u pogonskom sklopu za igre Unity. Skripte su pisane u programskom jeziku C#. Za modeliranje 3D objekata korišten je alat Blender.

1.1. Unity

Unity je višeplatformski pogonski sklop za igre kojeg je razvila dansko-američka tvrtka Unity Technologies. Izbačen je 2005. godine isključivo za Mac-OS X (operativni sustav tvrtke Apple), no danas je dostupan na više od 25 platformi.

Unity je poprilično intuitivan i jednostavan za korištenje, omogućuje povuci i ispusti (eng. drag and drop) funkcionalnost te ima ugrađenu trgovinu imovina (eng. Asset store) na kojoj korisnici mogu izmjenjivati razne sadržaje, od objekata preko tekstura pa sve do skripta. Preuzimanje imovine može biti besplatno ili se može plaćati ovisno od toga što je vlasnik odlučio. Jedna od glavnih imovina, s kojom većina ljudi krene u razvoj u Unityju zove se Essentials te nudi mnoštvo sadržaja koji je sasvim dovoljan za razvoj jednostavne video igre.

Na internetskim stranicama Unityja nalazi se dokumentacija ([5]) koja je vrlo pregledna, poprilično precizno opisana, a najvažnije od svega ima prikazane primjere primjene raznih svojstava Unityja i programskog jezika C#.

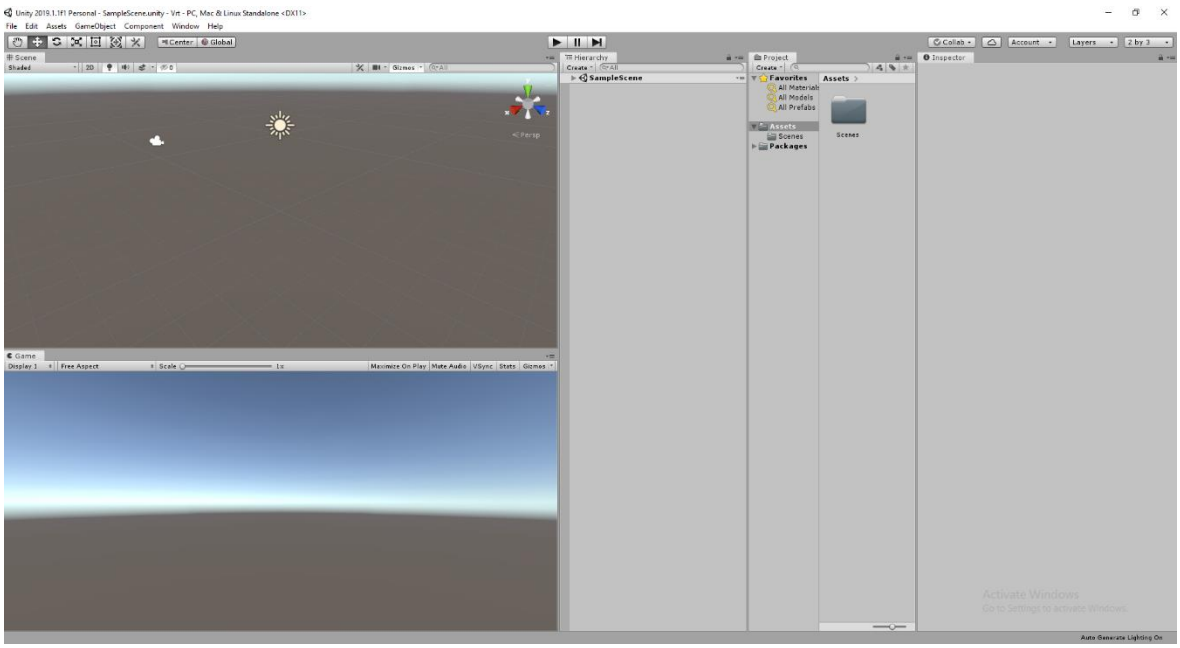
Veliki broj korisnika Unityja na internetskim forumima dijeli probleme i rješenja s kojima se susreću. Takvi forumi ([6]) od velike su koristi svakome, ali pogotovo onima koji se tek prvi puta susreću sa ovim alatom. Također na platformi YouTube postoji mnogo edukacijskih videa od kojih su neki korišteni pri izradi ovog rada.

Statistički podatci 2018. godine govore da je približno pola mobilnih igara na tržištu razvijeno u Unityju i preko 60 posto sadržaja proširene stvarnosti i virtualne stvarnosti što mnogo govori o veličini i proširenosti ovog pogonskog sklopa.

Može se koristiti za razvoj trodimenzionalnih i dvodimenzionalnih video igara, virtualne stvarnosti i proširene stvarnosti i mnogočega drugog. Iako je primarno razvijen za razvoj video igara, mnogo druge industrije ga koriste kao na primjer filmska industrija, arhitektura i inženjerstvo.

U filmskoj industriji našao je široku primjenu pri izradi animacija. Izrada animiranih filmova ili njihovih dijelova u Unityju daje brojne mogućnosti, od kreiranja kompleksnih efekata do simuliranja kompleksnih animacija kretanja i simulacija raznih fizičkih pojava kao što su gravitacijska sila i elastičnost.

Posljednjih godina, Unity je alat koji tvrtke sve češće koriste u svrhu simulacije kontrole robota. Neke tvrtke, poput tvrtke Formant, otišle su i korak dalje te napravile svoj vlastiti simulator za robotiku.



Slika 1.1 Korisničko sučelje Unityja

1.2. C#

C Sharp je programski jezik opće namjene kojeg je razvio Microsoft kao dio .NET inicijative 2000. godine. Radi se o programskom jeziku za višestruke paradigme, obuhvaća

funkcionalno, objektno orijentirano, opće programiranje i druge. Ciljevi dizajna samog jezika su da bude jednostavan, moderan, opće namjene i objektno orijentiran.

Njegova implementacija stoga podržava provjere jakog tipa (eng. Strong type checking), otkrivanje pokušaja korištenja neinicijaliziranih varijabli te automatski sakupljač smeća (eng. Garbage collector).

Ime je dobio po glazbenoj notaciji „#“ koja označava povišene note. Shodno imenovanju jezika C++ kojem sufiks ++ označava inkrement u odnosu na programski jezik C, znak „#“ je ligatura četiri znaka „+“ što je inkrement programskog jezika C++.

1.3. Blender

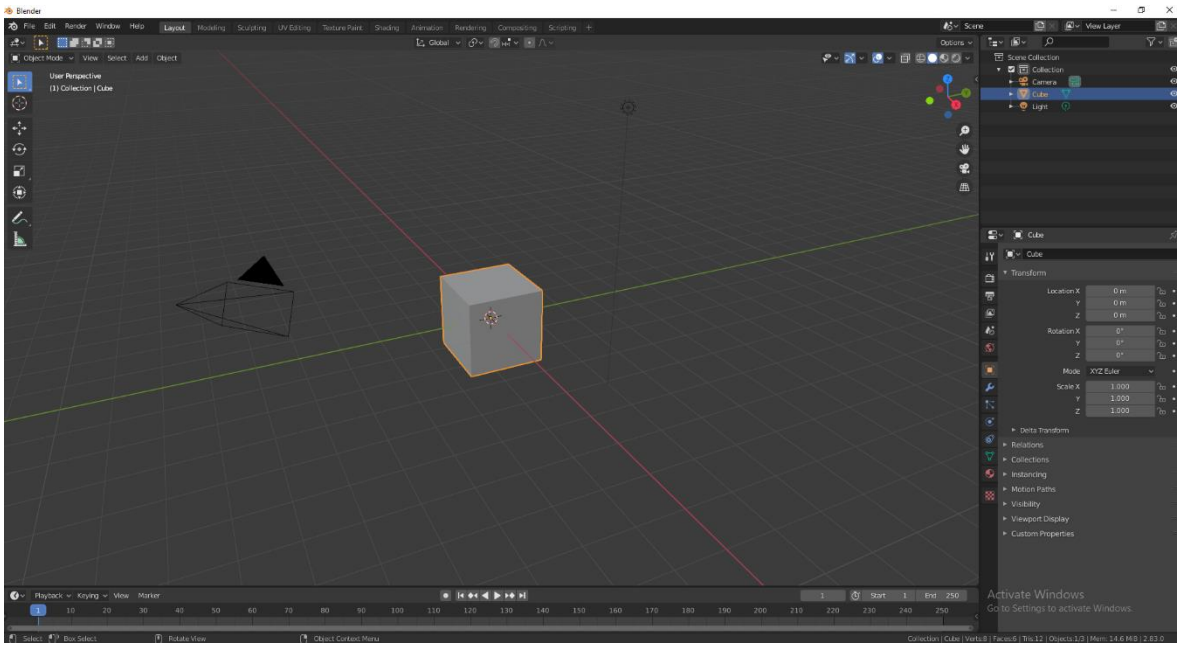
Blender je besplatni softver otvorenog koda za 3D računalnu grafiku. Nizozemski animacijski studio NeoGeo započeo je razvoj Blendera kao svoje interne aplikacije. Dio dizajna i mogućnosti Blendera preuzeti su iz aplikacije Traces koja je također bila razvijena isključivo za NeoGeo krajem 80-ih.

Prva potpuna verzija objavljena je u siječnju 1995. godine. Otkako je Blender postao otvorenog koda, mnogo se razvio i došlo je do značajnog refaktoriranja izvornog koda te je omogućen razvoj za razne druge funkcionalnosti koje su kasnije razvijene. Alat nudi 3D modeliranje, UV mapiranje, izradu tekstura, simulaciju fluida i dima, simulaciju čestica, oblikovanje kiparenjem, uređivanje videa. Neke od ovih funkcionalnosti potaknute su na Google-ovom Summer of Code programu u kojem Blender sudjeluje od 2005. godine.

Od 2002. Blender je besplatan za korištenje.

Blender je alat široke primjene od kojih su neke izrada animiranih filmova, vizualnih efekata, umjetnost, modeliranje 3D objekata za 3D printere te izrada video igara.

Od 2005. godine, svakih jednu do dvije godine, Blender Foundation objavljuje nove kreativne projekte kako bi potakli inovaciju u Blenderu. Prvi projekt, Elephants Dream, doživio je veliki uspjeh. Radi se o animiranom filmu koji su napravili neki od najistaknutijih Blender umjetnika te programera koristeći isključive besplatne alate.



Slika 1.2Korisničko sučelje Blender-a

2. Specifikacija zahtjeva

Zadatak ovog rada je izrada automatskog robota koji će moći detektirati različite daske sa početne gomile dasaka koja nije organizirane te ih zatim slagati u organizirane cjeline, pritom pazeći na raspored poprečnih daščica koje su između svakog reda normalnih dasaka i služe za sušenje.

Model robota na koji je iskorišten za realizaciju ovog rada jest IRB 120 tvrtke ABB ([4]). Naime, radi se o robotu sa šest zglobova od kojih su neki torzijski dok su drugi rotacijski. Budući daje gomila dasaka četverokutnog oblika i nema potrebe za kružnim kretnjama, u implementaciji su korišteni samo rotacijski zglobovi.

Kako bi se ovaj zadatak uspješno izvršio potrebno je riješiti sljedeće probleme.

Prvi problem jest kretanje zglobova robota tako da te kretnje zorno simuliraju kretnje istog robota u stvarnom svijetu.

Sljedeći problem jest pozicioniranje hvataljke u prostoru za podizanje i spuštanje dasaka.

Treći jest problem kretnje robota kroz prostor od platforme do platforme.

Posljednji i ujedno najopsežniji jest automatizacija cijelog procesa, donošenje odluka kroz ciklični proces premještanja dasaka.

Detaljan opis rješavanja ovih problema sadržan je u četvrtom poglavlju ovog rada, dok je u sljedećem poglavlju opisan jedna od najvažnijih značajki u razvoju programske potpore za robote a to je inverzna kinematika.

3. Inverzna kinematika

3.1. Općenito

Izrada realističnih i vjerodostojnih pokreta dugo je predstavljao problem u poljima robotike i računalne grafike. Inverzna kinematika je matematički proces izračuna varijabli zglobova tijela kako bi se krajnja točka promatranog tijela postavila u željenu poziciju u odnosu na početnu i njome se rješava potonji problem.

Često znamo parametre zglobova tijela sa kojim radimo, njihove pozicije i rotacije kao i kraj kinematičkog lanca pa izračun pozicija zglobova korištenjem trigonometrijskih formula, proces zvan *forward kinematics*, nije problem. Problem nastaje kada radimo istu operaciju unatrag i tu nam je inverzna kinematika od velike koristi.

Inverzna kinematika također se koristi i za rekonstrukciju pokreta objekta nastalih prije pokreta koji su nam poznati iz nekih izvora kao što je video tog pokreta ili čak pokrete kamere iz videa viđenog iz perspektive te kamere.

Primjenu nalazi u ponajviše u robotici i računalnoj animaciji. Rješenja inverzne kinematike dijele se u dva dijela, analitička i numerička.

3.1.1. Analitička rješenja

Analitičko rješenje inverzne kinematike su izrazi zatvorene forme koji kao ulaz primaju poziciju krajnjeg člana kinematičkog lanca, a za izlaz daju pozicije zglobova. Ovaj pristup može biti znatno brži od rješenja numeričkog tipa, ali tu naravno postoji jedan problem.

Problem je što analitičko rješenje inverzne kinematike postoji, ali ne uvijek, tj. ne za svaki tip robota. Primjer jednog takvog robota na kojeg se može primijeniti analitičko rješenje jest

robot koji se koristi u ovom projektu, sastoji se od 6 zglobova od kojih su tri rotacijska, a ostala tri torzijski zglobovi.

S druge strane robot sa sedam zglobova jest primjer robota kod kojeg njegovi stupnjevi slobode nadilaze stupnjeve slobode krajnjeg člana što znači da postoji beskonačno mnogo rješenja inverzne kinematike i samim time ne postoji analitičko rješenje.

3.1.2. Numerička rješenja

Postoje mnoge metode modeliranja i rješavanja inverzne kinematike. Najfleksibilnije su upravo one koje se oslanjaju na iterativnu optimizaciju pronalaska približnog rješenja, zbog poteškoća invertiranja jednadžbe *forward kinematics*-a. Glavna ideja ovih metoda je da tu jednadžbu modeliraju koristeći razvoj u Taylorov red što je najčešće jednostavnije za invertirat i riješiti.

Tu su nam prikladne za promatranje heurističke metode koje se oslanjaju na jednostavne, iterativne operacije kako bi postepeno došli do aproksimacije rješenja. Jedna od njihovih prednosti je ta što nisu računalno zahtjevni, vraćaju konačnu poziciju poprilično brzo, te često podržavaju ograničenja zglobova. Jedna od takvih metoda je *Forward and Backward Reaching Inverse Kinematics* koja je detaljno opisana u sljedećem poglavlju.

3.2. FABRIK (Forward and Backward Reaching Inverse Kinematics)

FABRIK ([3]) je inačica inverzne kinematike koja koristi točke (zglobove) i linije (kosti) kroz interativni algoritam kako bi se riješio problem. Problem dijeli u dvije faze, prvo je korak naprijed, a zatim korak nazad dok se pritom u svakom koraku pridržava rotacijskih ograničenja zglobova i njegovih orijentacija ponovno pozicionirajući i preusmjeravajući zglobove i kosti tijela.

Glavne prednosti opisanog algoritma su njegova jednostavnost, lakoća implementacije na razne skeletalne konfiguracije, mogućnost direktne optimizacije i mogućnost rada sa modelima koji imaju višestruke krajnje točke. Navedene odlike čine ga povoljnim za primjenu u sustavima u stvarnom vremenu.

U ovom poglavlju opisan je FABRIK algoritam na primjeru implementacije korištene u ovom radu.

Prije samog početka primjene algoritma, moramo zadati parametre koji su duljina lanca, broj iteracija, maksimalno dopušteno odstupanje, metu i pritku.

Duljina lanca jest broj tipa *int* koji označava duljinu kinematičkog lanca, točnije broj kostiju između zglobova samog lanca što bi značilo ako je zadana duljina lanca n onda se promatrano tijelo sastoji od n+1 zglobova na koje se algoritam primjenjuje i izračunava.

Broj iteracija određuje kroz koliko će iteracija proći algoritam pri jednom pozivu funkcije izračuna prije nego što se izračunate pozicije primjene na zadane zglobove.

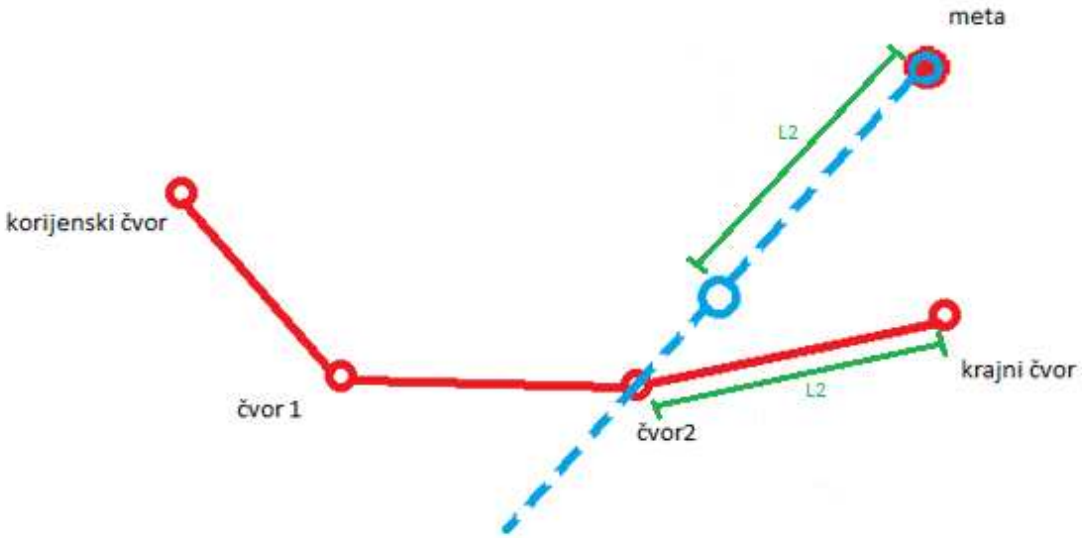
Maksimalno dopušteno odstupanje označava broj od kojeg udaljenost, od krajnjeg člana kinematičkog lanca do mete, mora biti manja da bi se proces izračuna prekinuo, tj. kako se ne bi nepotrebno prolazilo kroz sve iteracije algoritma ako su zglobovi već na željenoj poziciji.

Meta je objekt na čiju poziciju se pokušava postaviti krajnji čvor kinematičkog lanca provedbom FABRIK algoritma.

Pritka je objekt prema kojem zglobovi teže u smislu da se od mnogih krajnjih položaja koje zglobovi mogu zauzet uzmu oni koji su najbliži ravnini pritke.

Algoritan započinje inicijalizacijom varijabli od kojih su najbitnije pozicije zglobova, posebno korijenskog zgloba koji se ne pomiče, razmaci između zglobova, početne rotacije koje služe kao referentne informacije za daljnji razvoj pozicija i rotacija.

Zatim ulazimo u iterativni dio algoritma gdje se prvo pozicije svakog zgloba, osim korijenskog zgloba koji zadržava svoju poziciju jer njega ne želimo pomicati, postepeno pomaknu prema svom prethodnom zglobu u smjeru mete. Potom ulazimo u petlju koja se izvršava broj puta koji je jednak zadanom broju iteracija koji je prethodno spomenut. Pritom se krajnji član kinematičkog lanca postavlja na poziciju mete, a ostali zglobovi ga slijede za taj pomak, svaki ovisi o poziciji prethodnoga. Izračun svake sljedeće pozicije, dakle osim krajnjeg zgloba koji je na poziciji mete, računa se tako da se pozicija n postavi na poziciju koja odgovara vektorskom zbroju pozicije n i normali između nove pozicije čvora n i pozicije n-1 pomnožene sa udaljenošću između dva zgloba na n-toj i (n-1)-oj poziciji, ta udaljenost je naravno konstantna i na početku procesa spremljena u podatkovno polje. Pri svemu tome naravno preskačemo korijenski zglob, on zadržava svoju poziciju. Na slici ispod prikazana je ilustracija prethodno opisanog procesa, točnije samo prva dva koraka. Crvenom bojom označeni su zglobovi na svojim početnim pozicijama, a plavom bojom označeni su zglobovi na svojim novim pozicijama. Isprekidana plava linija je prikaz vektorskog zbroja nove pozicije i normale razlike između nove pozicije krajnjeg zgloba i trenutne pozicije njegovog prethodnog zgloba. To je *Backward* dio iz naziva FABRIK.



Slika 3.1 FABRIK prvi korak

Nakon toga odrađujemo reverznu stvar od prethodno opisane. Ovoga puta krećemo od zgloba pozicioniranog nakon korijenskog. Redom pomičemo svaki zglob prema korijenskom po vektorskom zbroju pozicije zgloba n-1 i normale između nove pozicije čvora n i pozicije n-1 pomnožene sa udaljenošću između dva zgloba na n-toj i (n-1)-oj poziciji. Shodno prethodnom tekstu o *Backward* dijelu algoritma, ovaj reverzni dio ima i reverzno ime, *Forward* dio.

Po završetku izračunate pozicije se primjenjuju na zglobove i uz to računaju se rotacije zglobova u odnosu na početne rotacije koje su spremljene pri inicijalizaciji i postavljaju se na zglobove.

4. Implementacija

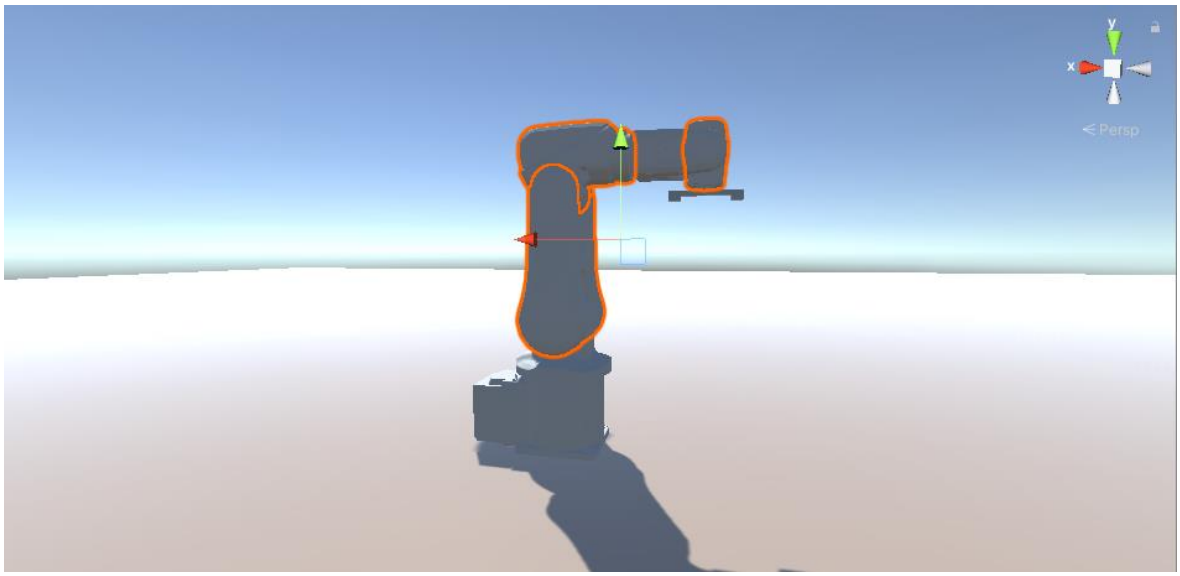
U ovom poglavlju posebno su opisana rješenja problema navedenih u specifikaciji. Svaki od tih problema u glavnini je riješen izradom skripti napisanih u programskom jeziku C#.

4.1. Kretnje zglobova robota

Kretnje zglobova robota, izračun njihovih pozicija, obavljen je primjenom algoritma FABRIK. Izvedba toga sastoji se od nekoliko stavki.

Prvo, odlučeno je da će za potrebe izvršavanja ovog zadatka biti potrebni samo rotacijski zglobovi. Zglobovi koji su odabrani redom su nazvani, počevši od krajnjeg zgloba, zadanim nazivljem tvrtke ABB ([4]), IRB120_3-58_IRC5_Link5_rev0, IRB120_3-58_IRC5_Link3_rev0_2, IRB120_3-58_IRC5_Link2_rev0_2_3. Prikaz ovih zglobova nalazi se na slici (Slika 4.1 Zglobovi robota) gdje su narančastom bojom označeni obrubi navedenih zglobova. Na objekt krajnjeg zgloba postavljena je skripta FABRIK algoritma kao njegova komponenta. Algoritam je opisan u prethodnom poglavlju stoga nije potrebno ponavljat opis načina njegovog rada, no navest će se nekoliko pojedinosti koje su specifične za izvršavanje ovog zadatka. Budući da se ne koriste svi zglobove robota, algoritam je kroz inicijalizaciju zglobova objekta morao nekako prepoznati isključivo one na koje je potrebno primijeniti algoritam.

Rješenje toga je izvedeno na način da je svakom od prethodno navedenim zglobovima kao roditelj postavljen prazan objekt, što u kontekstu Unityja označava objekt koji u sebi ima samo *Tranform ([5])* komponentu koja sadrži informacije o poziciji, rotaciji i skali objekta, te tim objektima postavio oznaku „Joint“ koju algoritam u skripti prepoznaje te uzima samo te objekte za izračun njihovih pozicija, dok ostali zglobovi prate te promjene jer su zglobovi hijerarhijski posloženi na način da je svaki zglob dijete svom zglobu prethodniku.



Slika 4.1 Zglobovi robota

Druga stavka jest ta da su zadane toče objekta zglobova bile u nepovoljnim položajima za izračun algoritma što je riješeno tako da su gore spomenutim praznim objektima prije nego što su postavljeni za roditelje zglobovima, ručno podešene pozicije točki okreta tako da vjerno reprezentiraju točke okreta u stvarnosti. U Unityju, referente točke objekata djece imaju svoju poziciju, rotaciju i skalu postavljene lokalno u odnosu na poziciju, rotaciju i skalu svojih roditelja.

Na posljetku, hvataljka robota za daske, kako bi algoritam ispravno radio, morala je biti pozicionirana tako da njena točka okreta bude usklađena sa točkom okreta krajnjeg zgloba. Problem nastaje kod sakupljanja daski koje je riješeno tako da u skripti dasci koju je potrebno pokupiti za roditelja postavim objekt hvataljke. Naime kako je točka okreta hvataljke na poziciji točke okreta krajnjeg zgloba, daska bi se pri hvatanju postavila u poziciju da prolazi kroz dijelove robota i to nije bilo vjerodostojno.

Kako bi se to ispravilo napravljen je još jedan prazan objekt i pozicioniran na poziciju na kojoj će daske pri hvatanju biti u pogodnom položaju i vjerno simulirati stvarnu situaciju. Taj objekt je naravno postavljen kao dijete objektu hvataljke što je polučilo željene rezultate.

4.2. Pozicioniranje hvataljke

Hvataljka o kojoj je pisano u posljednjem dijelu prethodnog odlomka, kao meta krajnjeg zgloba FABRIK algoritma, morala je imati neki način kretanja kako bi promjenom pozicije demonstrirala rad samog algoritma. Potrebno je eksplicitno istaknuti kao je hvataljka, budući da je dijete krajnjeg čvora, dijete cijeloga robota te promjene pozicije i rotacije hvataljke prate promjene pozicije i rotacije robota.

Izvedba kretnje hvataljke obrađena je pisanjem skripte.

Prvi dio skripte jest i njen najvažniji dio, a to je kod za skeniranje dasaka. Implementacija skeniranja izvršena je korištenjem *Raycasta ([5])*. *Raycast* je operacija u sklopu Unityjeve knjižnice naziva „UnityEngine.UI“. Što ona radi jest da iz objekta kojem skripta pripada šalje zraku koja se potom odbija od objekata i vraća informacije o pogođenom objektu.

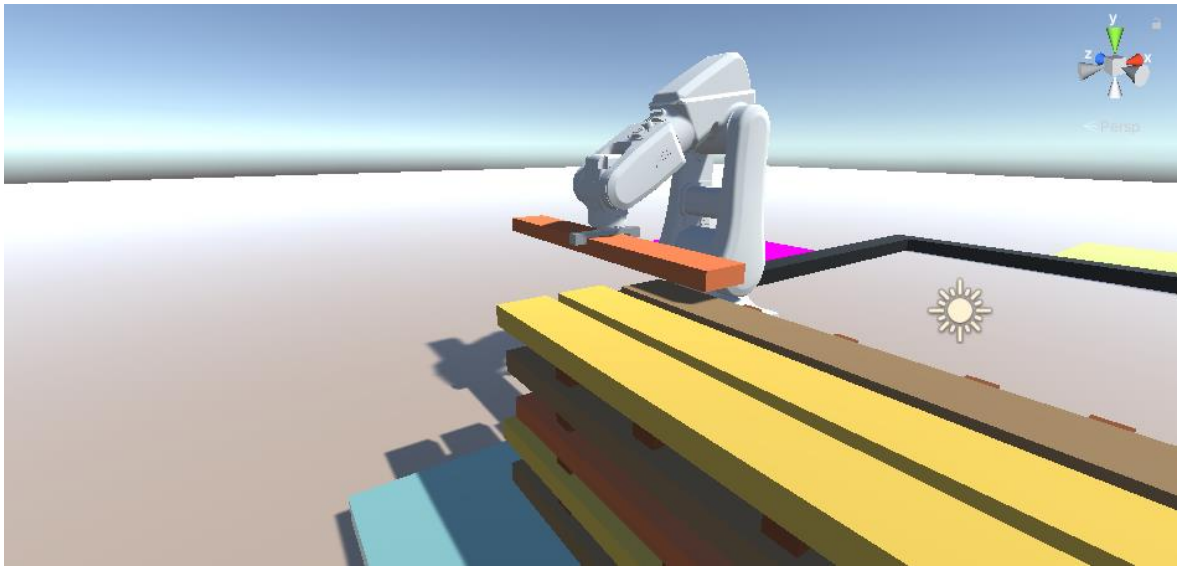
Nakon što je objekt skeniran, na temelju povratnih informacija zrake, u kodu se provjerava je li pogođeni objekt daska te ako jest provjerava se njegova visina u odnosu na tlo pa ako je visina objekta veća nego prethodno najvišeg, zapamtila bi se pozicija tog objekta i spremila u lokalnu varijablu koja je javnog dosega kako bi druge skripte, koje su u sljedećim poglavljima opisane, mogle njima pristupiti i koristiti ih za izračun.

Zatim slijedi dio skripte za kretanje koji se sastoji od nekolicine stanja u kojima se hvataljka može nalaziti. Stanje hvataljke druge skripte mogu dohvatiti preko javne metode koja vraća varijablu tipa *string* u kojoj je spremljeno stanje u trenutku poziva funkcije.

Najbitnija stanja su stanje početka skeniranja kojim se započinje skeniranje gomile dasaka što se izvađa tako da hvataljka prelazi uzduž i poprijeko gomile dasaka i time se *Raycast* odbija od svih površinskih dasaka i sprema potrebne informacije. Najvažnije informacije su veličina daske koja je spremljena u *Scale* varijabli skeniranog objekta i oznaka tog objekta. Daske su izrađene od primitivnog Unityjevog objekta tipa *Cube ([5])* koji je kocka sa skalama svih ravnina (x, y i z) jednakim jedan. Izmjenom iznosa skala za svaku koordinatnu os dobiven je objekt koji reprezentira dasku. Radi jednostavnosti izvedbe i urednosti, različitim daskama, osim boje, razlikuje se samo *z* komponenta skale koja predstavlja njenu duljinu. Oznake koje su bitne su „Daska“ i „Poprečna“, prva označava obične daske dok druga označava poprečne daščice. Pomicanje uzduž vrši skripta za kretnju cijelog robota koja je kasnije opisana.

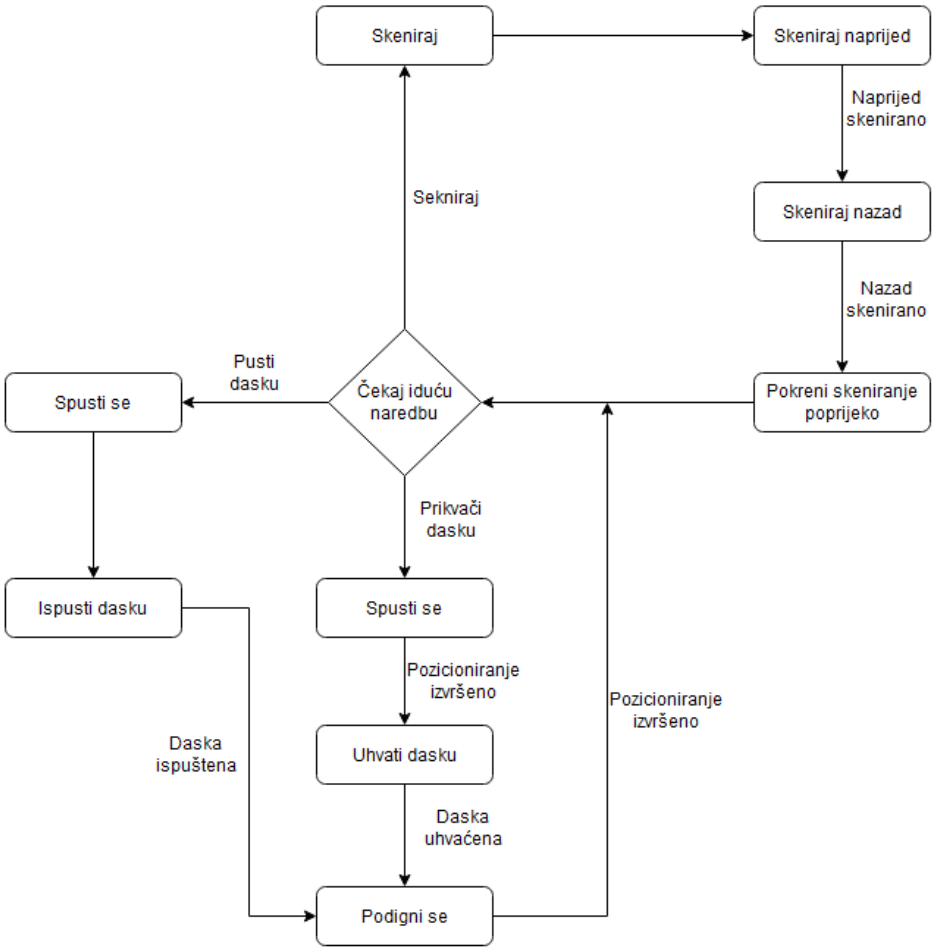
Glatke kretnje izvedene su tako da se prilikom svakog ulaska u stanje koje ima kretnju provjerava je li udaljenost pozicije hvataljke i odredišne pozicije različita od nula, ako nije hvataljka se pomiče za dio puta prema odredištu. Postepena promjena pozicije riješena je preko funkcije *MoveTowards ([5])* objekta tipa *Vector3* koja je zadana funkcija Unityja. Ona kao ulazne parametre prima redom poziciju objekta koji se pomiče, poziciju odredišta te stopu za koju će se pomicati na tom putu, a vrača poziciju objekta uvećana za zadanu stopu prema poziciji odredišta. To je moguće jer su stanja unutar *Update* metode koju Unity poziva jednom po okviru. Nakon što hvataljka dođe u poziciju i udaljenost iznosi nula, ona prelazi u sljedeće definirano stanje ili obavlja što je zadano.

Potom dolazi stanje hvatanja daske u kojem hvataljka putuje prema poziciji zadanog objekta koju joj je odredio stroj stanja koji će se kasnije u ovom radu detaljnije opisati. Hvataljka mijenja poziciju na način koji je naveden pri opisu stanja skeniranja. Kada je hvataljka stigla na odredišnu poziciju ona hvata objekt koji se nalazi na toj poziciji. Kako bi hvataljka uhvatila dasku i nakon toga daska pratila promjene pozicije i rotacije hvataljke, objekt točke rotacije hvataljke postavljen je za roditelja objektu daske i na taj način simulirano hvatanje daske u stvarnom svijetu. Hvataljka nakon što je uhvatila zadanu dasku, vraća se u položaj u kojem je bila prije nego što je krenula, koji je zapamćen u skripti netom prije početka spuštanja. Prijenos uhvaćene daske prikazan je na slici (Slika 4.2 Podignuta daska) koja je snimljena u zaustavljenom trenutku prijenosa uhvaćene daske prema njenoj odredišnoj platformi.



Slika 4.2 Podignuta daska

Na vrlo sličan način radi i stanje otpuštanja daske, samo što nakon dolaska na odredišnu poziciju objektu daske je za roditelja postavljena vrijednost *null* što rezultira simuliranjem otpuštanja daske iz hvataljke u stvarnom svijetu.

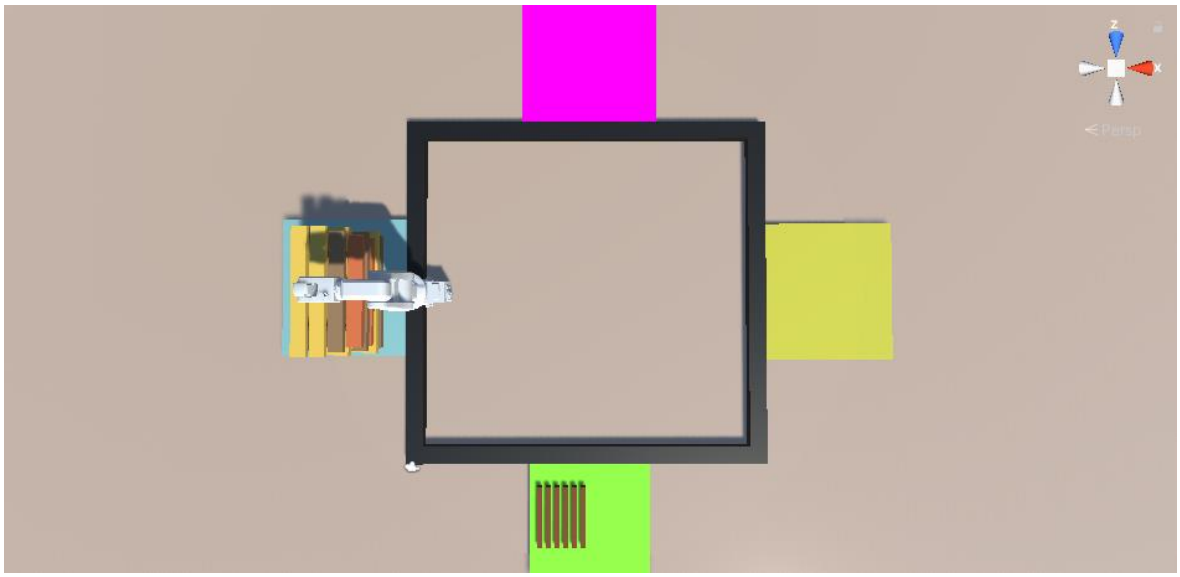


Slika 4.3 Dijagram toka kretanja hvataljke

4.3. Kretanje robota kroz prostor

Robot korišten za ovu simulaciju nije mnogo veći od dasaka, a posebice ne od gomile dasaka, stoga ne može dosegnuti sve potrebne pozicije ako stoji na jednom mjestu, tj. ako mu je baza fiksirana u prostoru.

Rješenje tog problema izveden je izradom trase koja simulira tračnice po kojima se robot kreće kako bi došao od platforme do platforme. Tlocrt scene prikazan je slici (Slika 4.4 Tlocrt). Na slici vidimo crni pravokutnik koji predstavlja netom spomenute tračnice-



Slika 4.4 Tlocrt

Uz tračnice nalaze se platforme za odlaganje dasaka. Na lijevoj platformi plave boje postavljena je početna, neorganizirana gomila dasaka. Daske u spomenutoj gomili su bez rotacije i odvojene jedna od druge. Gornja, ružičasta platforma predviđena je za odlaganje dasaka koje ne odgovaraju ulaznim parametrima koje je zadao korisnik. S desne strane nalazi se žuta platforma na koju se, kao i na prethodnu platformu, organizirano slažu daske, samo na ovu platformu odlažu se daske koje zadovoljavaju ulazne parametre. Na donjoj strani nalazi se platforma zelene boje koja služi za odlaganje poprečnih daščica za sušenje koje su višak ili pak za uzimanje poprečnih daščica za sušenje ako ih nedostaje. Naime, učestalost poprečnih daščica na početnoj, neorganiziranoj platformi je varijabilna pa se tako može

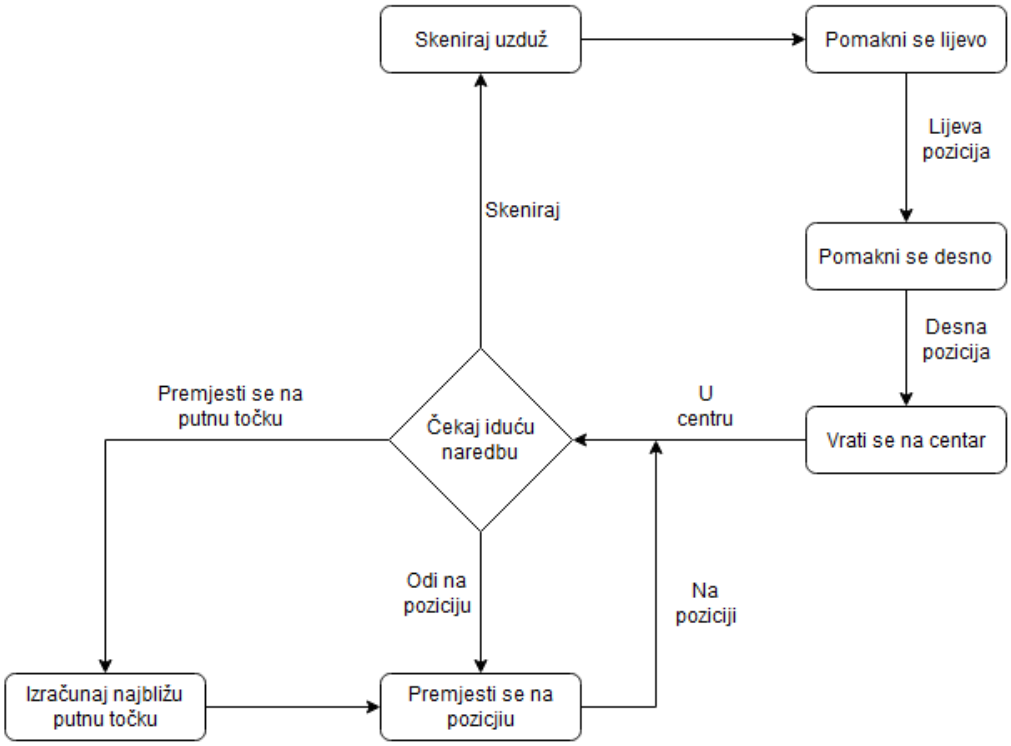
dogoditi da pri organiziranju dasaka na za to predviđene platforme može faliti poprečnih daščica, koje se slažu po dvije između svakog reda organiziranih dasaka, ili ih može biti viška.

Sada kada je pozadina scene poznata, fokus se prebacuje na skriptu za kretanje robota i njen način rada. Robot u sebi sadrži skriptu koja je namijenjena za kretanje čitavog robota po tračnicama. Skripta funkcionira na isti način kao i skripta za hvataljku, sa, naravno, nekim svojim varijantama izvedbe.

Na početku opisa skripte važno je napomenuti da skripta kao početne parametre prima pozicije praznih objekata postavljenih na tračnicama na značajnim mjestima. Radi se o objektima *Waypoint* koji sadrže informacije o pozicijama na tračnicama, te služe za određivanje kretnje robota po tračnicama. Kako je napisano, postavljeni su na značajne pozicije, što su zapravo pozicije koje se najčešće koriste pa je stoga praktičnije i efikasnije pamtiti te učestale pozicije nego ih konstantno proračunavati. Radi se o pozicijama na kutovima tračnica koje su bitne za rotaciju robota. Ti objekti su označeni oznakom *Rotate* kako bi skripta mogla odrediti kada zarotirati robota tako da njegove kretnje simuliraju stvarne. Potom su bitne pozicije ispred svake platforme jer u glavnini robot se kreće između njih. Sljedeće su pozicije između kojih robot ide kako bi uzdužno skenirao gomilu dasaka, koje su naravno konstantne zbog statične veličine platforme.

Sada još preostaje navesti položaje za skupljanje poprečnih daščica. Prilikom skeniranja gomile, ukoliko je sljedeća daska koju je potrebno odnesti poprečna daščica, njena pozicija se pamti kao što je opisano u prethodnom poglavlju i njene informacije su javno dostupne pa tako skripta za kretnju robota dohvaća poziciju poprečne daščice. Iz poznate pozicije poprečne daščice i poznate pozicije najbliže putne točke izračunava se sljedeći položaj na koji bi se robot trebao premjestit. Nakon što je položaj poznat, robot putuje do izračunate točke te postaje spreman za podizanje poprečne daščice, što znači da se hvataljka sada može pozicionirati na mjesto daske i sakupit je.

Glavna značajka ove skripte jesu metode javnog dosega koje služe za generalizirano kretanje robota. Jedna od njih je primjerice metoda *MoveToWaypoint* koja prima kao ulazni parametar putnu točku do koje se robot treba pomaknuti. Metoda prvo poziva privatnu funkciju za izračun najbliže putne točke u odnosu na trenutni položaj robota. Nakon toga robot putuje tračnicama do zadane putne točke te pri dolasku na krajnji položaj javlja da je stigao.

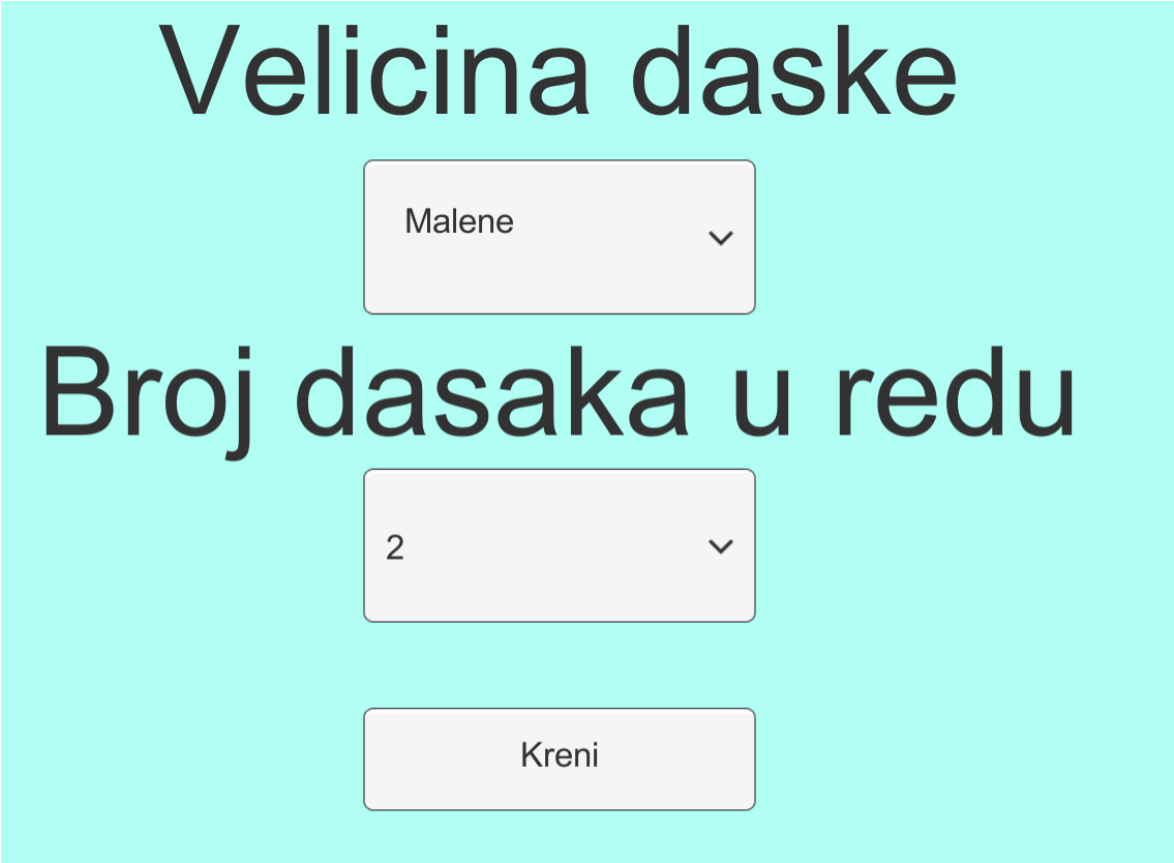


Slika 4.5 Dijagram toka kretnje robota

4.4. Stroj stanja

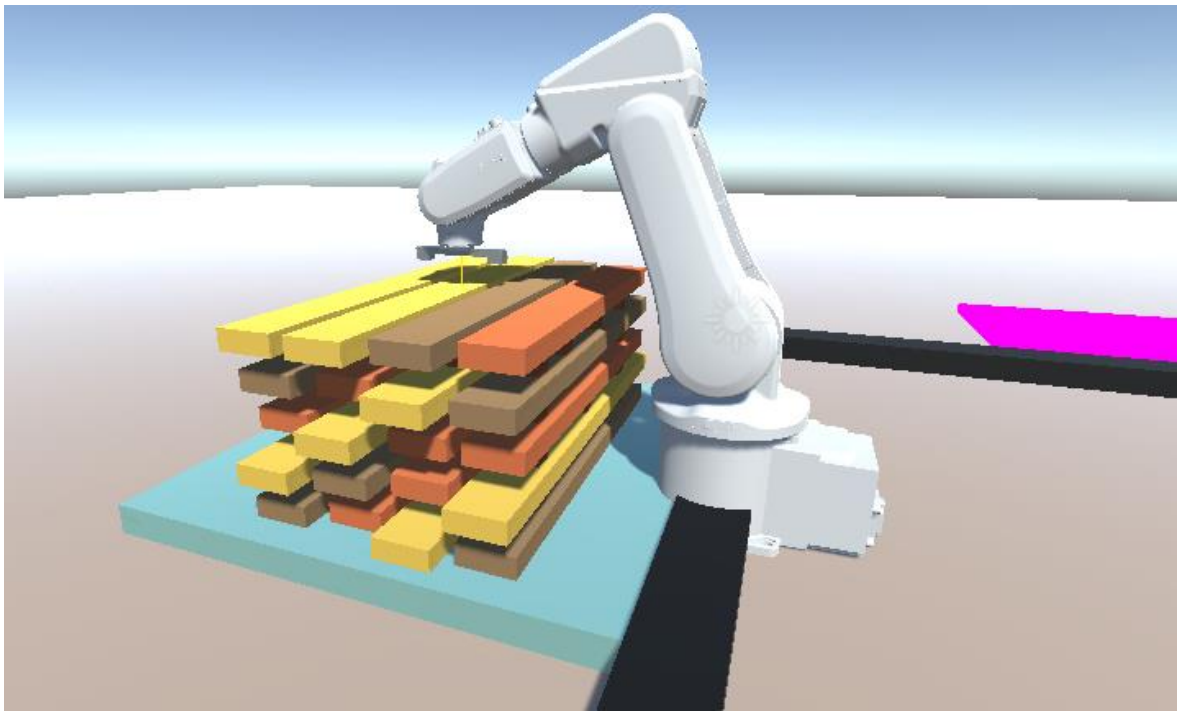
Stroj stanja koji je prethodno nekoliko puta spomenut realiziraj je preko skripte koja je dio praznog objekta naziva „GameManager“, mogli bismo reći srce cijelog procesa jer preko njega se sve odvija. Naziv skripte je „StateMachine“.

Cijeli proces započinje pritiskom na gumb „Kreni“ na čiji pritisak se poziva funkcija inicijalizacije koja je dio skripte StateMachine. Pri inicijalizaciji prvo se zapamte vrijednosti padajućih izbornika iz kojih korisnik bira koje će se daske slagati na posebnu platformu te koliko će dasaka biti posloženo po jednom redu (Slika 4.6 Početni izbornik). Također puni se lista objekata koja sadrži informacije o poprečnim daščice na platformi za poprečne daščice. Postavljaju se položaji za svaku platformu na koje treba doći sljedeća daska, potom se pokreće sam stroj stanja.



Slika 4.6 Početni izbornik

Početno stanje je stanje skeniranja, u tom stanju stroj pokreće hvataljku javljajući njenoj skripti da krene sa skeniranjem uzduž i poprijeko, jer početna pozicija robota je na poziciji za skeniranje ispred početne gomile dasaka (Slika 4.7 Skenrianje gomile).



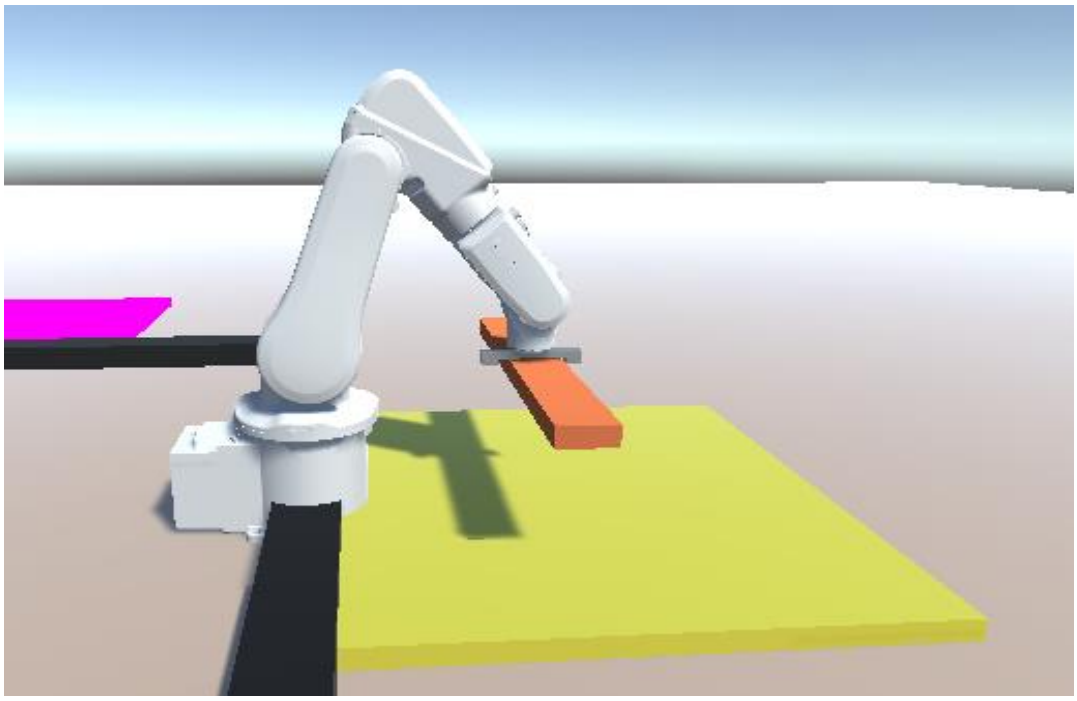
Slika 4.7 Skenrianje gomile

Nakon što su daske skenirane, na način koji je opisan u poglavlju implementiranja kretnje hvataljke, stroj ulazi u stanje kalkulacije.

Stanje kalkulacije je najkompleksnije stanje jer se uzimaju sve informacije dostupne kako bi se odredila sljedeća akcija. Provjerava se koja je daska sljedeća na redu za hvatanje i prijenos, zatim se gleda stanje platforme na koju ta daska treba biti odnesena i na temelju toga se odlučuje što nakon. Prvo imamo slučaj kada je daska zadane veličine, na početku se provjerava koliko je dasaka u posljednjem redu na platformi za daske zadane veličine.

Važno je istaknuti da se broj dasaka u posljednjem redu na platformi pamti i ažurira prilikom svakog odnošenja daske, za svaku platformu, isto kao i broj poprečnih daščica u trenutnom redu te pozicija na koju treba postaviti sljedeću dasku.

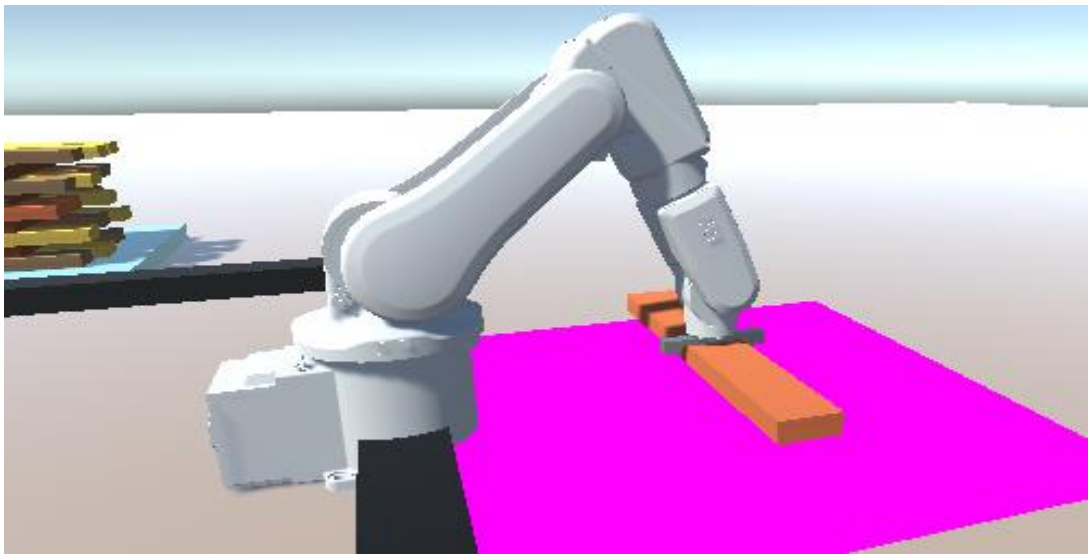
Ako broj dasaka u posljednjem redu manji od zadanog broja dasaka po redu i broj poprečnih daščica u posljednjem redu jednak dva, pokreće se proces hvatanja i podizanja daske te prijenosa na platformu za zadane daske (Slika 4.8 Prijenos daske na platformu za zadane daske). Uz to ažurira se pozicija za postavljanje sljedeće daske na toj platformi, brojač dasaka u zadnjem redu te platforme povećava se za jedan. Nakon toga robot se vraća ispred početne platforme i započinje se cijeli postupak iznova postavljanjem stanja u stanje za skeniranje.



Slika 4.8 Prijenos daske na platformu za zadane daske

Ako je broj dasaka u posljednjem redu jednak zadanom broju dasaka po redu i broj poprečnih daščica u posljednjem redu različit od dva, ažurira se pozicija za postavljanje sljedeće daske na tu platformu te se resetiraju brojači za broj dasaka u posljednjem redu te platforme kao i broj poprečnih daščica. Nakon ažuriranja internih podataka skripte, poziva se metoda koja je zasebni i jednostavniji stroj stanja unutar glavnog stroja stanja. Spomenuta metoda kao ulazni parametar prima odredišnu platformu na koju će odnesti poprečnu daščicu, obavlja posao donošenja poprečne daščice na zadanu platformu korištenjem nekih od dosad opisanih funkcionalnosti kao što su putovanje robota na zadanu poziciju, u ovom slučaju poziciju uz platformu za poprečne daščice i hvatanje i puštanje daščice hvataljkom. Ovaj proces izvodi se dva put jer se postavljaju dvije poprečne daščice na između redova običnih dasaka. Naposlijetku uvjet iz prethodnog slučaja je zadovoljen, a to je taj da su dvije poprečne daščice u posljednjem redu i da je broj dasaka u posljednjem redu manji od zadanog broj dasaka pa se kreće s tim procesom.

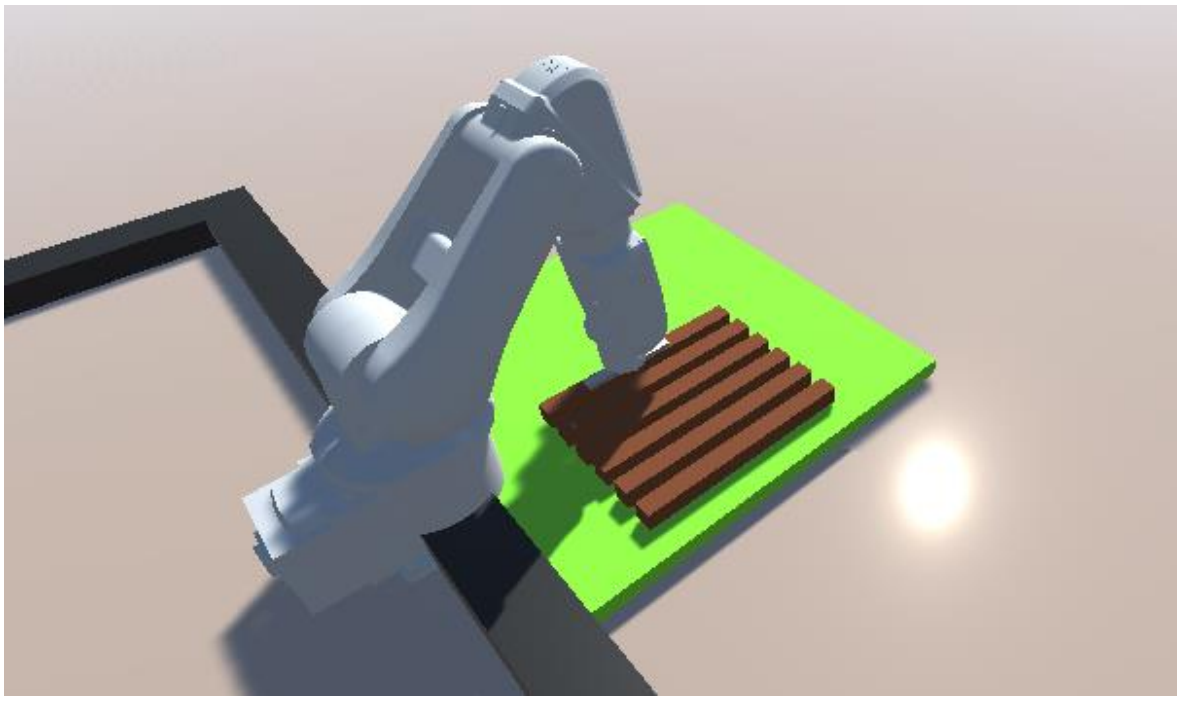
Drugi slučaj je kada je sljedeća daska za premještanje daska koja nije zadane veličine. Postupak je isti kao za daske zdane veličine samo što se u ovom slučaju naravno ažuriraju interne varijable vezane za platformu za daske koje nisu zadane veličine i shodno tome, poprečne daščice nose se na platformu za daske koje nisu zadane veličine (Slika 4.9 Premještanje daske na plaformu za daske koje nisu zadane veličine).



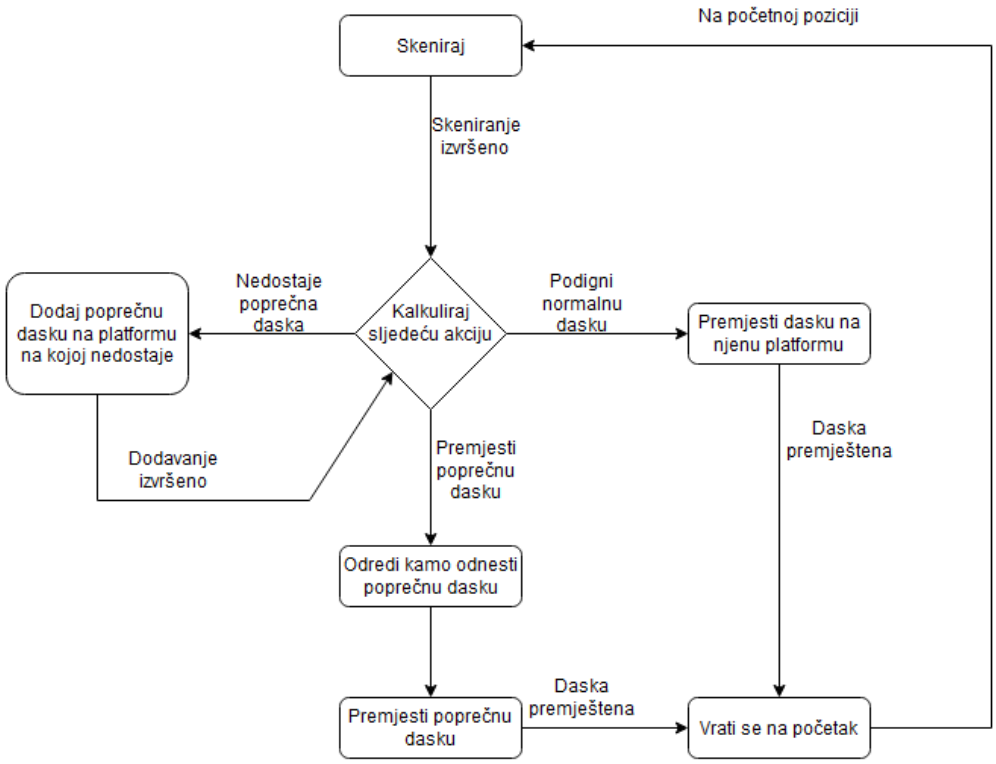
Slika 4.9 Premještanje daske na plaformu za daske koje nisu zadane veličine

Posljednji slučaj je kada je sljedeća daska za premještanje poprečna daščica. Prvo se gleda broj poprečnih daščica u posljednjem redu na platformi za daske zadane veličine i broj dasaka u posljednjem redu zadane veličine. Ako je potrebno, poprečna daščica se prenosi na tu platformu pozivanjem metode koja je u prethodnom koraku opisana. U suprotnom provjerava na isti način stanje na platformi za daske koje nisu zadane veličine te ako je potrebno nosi daščicu na tu platformu na isti način kao i u prethodnom koraku. Ako nije potrebno premještati poprečnu daščicu ni na jednu od platformi za normalne daske, prenosi se na platformu za poprečne daščice.

Prijenos poprečne daščice na platformu za poprečne daščice vrši se pozivanjem nove metode koja je slična metodi za prijenos poprečne daščice na platforme za normalne daske. Ova metoda također je zasebni, manji stroj stanja koji prvo premješta robota na poziciju ispred platforme za poprečne daščice, izračunavajući položaj na temelju putne točke i sljedeće poprečne daščice sadržane u listi koja je opisana na početku ovoga poglavlja. Zatim pokreće hvataljku koja ostavlja daščicu na platformi za poprečne daščice i naposljetku se vraća ispred početne platforme i inicijalizira cijeli proces ispočetka postavljajući stanje glavnog stroja stanja na stanje za skeniranje (Slika 4.10 Prijenos poprečne daščice).



Slika 4.10 Prijenos poprečne daščice



Slika 4.11 Dijagram tijeka stroja stanja

Zaključak

Razvijanjem ovog projekta prikazana je vjerodostojnost virtualnog robota u odnosu na pravog. Simulacija rada robota u virtualnom svijetu vrlo je praktična, razvojni tim može pratiti cijeli proces iz bilo kojeg dijela svijeta sa pristupom internetu što je velika prednost. Uz prethodno navedeno, razne pogreške i moguće nesreće nemaju nikakve posljedice u virtualnom svijetu dok bi u stvarnosti posljedice mogle biti velikih razmjera te koštati mnogo resursa.

Unity je jedan od najkvalitetnijih alata za razvoj projekata ovakvog tipa, a i raznih drugih. Velika količina sadržaja, intuitivnost i jednostavnost korištenja čine ga pogodnim za početnike i za profesionalce. Od svoje prve verzije pa do danas Unity je mnogo napredovao, redovite optimizacije i nadogradnje čine još kvalitetnijim što ukazuje na to da će vrlo vjerojatno još dugo biti jedan od najboljih i da će širina njegove primjene samo rasti.

Literatura

.

[1] Andreas A., Yiorgos C., Joan L., *Extending FABRIK with model constraints*. Wiley Online Library, Poveznica: (https://onlinelibrary.wiley.com), 2015.

[2] DitzelGames. *C# Inverse Kinematics in Unity* 🎓, Poveznica: (https://youtu.be/qqOAzn05fvk), 2019.

[3] EgoMoose, *FABRIK (Inverse kinematics)*, Poveznica: (https://www.youtube.com/watch?v=UNoX65PRehA), 2016.

[4] ABB, *Product specification IRB 120,* Poveznica: (https://library.e.abb.com/public/7139d7f4f2cb4d0da9b7fac6541e91d1/3HAC035960%20PS%20IRB%20120-en.pdf)

[5] Unity, *Scripting API,* Poveznica: https://docs.unity3d.com/ScriptReference/

[6] Unity, *Unityjev forum*, Poveznica: https://answers.unity.com/questions/index.html

[7] Gordan Gledec*, Pravopisni savjeti za izradu studentskih radova v1.4.0*, 2010.

Sažetak

Primjenom FABRIK algoritma za rješavanje problema inverzne kinematike razvijen je projekt u Unityju. Projekt se sastoji od početnog unosa parametara koje korisnik proizvoljno zadaje te na temelju danih parametara robot IRB 120 kreće sa automatskim obavljanjem posla. Robot prvo dolazi do hrpe dasaka te redom skenira dasku po dasku pamteći dimenzije najviše daske, zatim uzima dasku te ovisno u zadanim parametrima slaže dasku na određenu hrpu, na kraju provjerava je li cijeli red dasaka premješten i ako jest skida poprečne daščice za sušenje i postavlja ih na neku od hrpa gdje su potrebne. Skripte za rješavanje algoritma napisane su u programskom jeziku C# isto kao i skripte za kretanje robota.

Ključne riječi: FABRIK, robot, Unity, C#

Summary

A project in Unity was developed by applying the FABRIK algorithm for solving the problem of inverse kinematics. The project consists of inputing the parameters at the beginning and then starting the robot IRB 120 which takes given parameters and starts his automatic work. The robot firstly arrives infront of heap of planks and than scans them while remembering magnitude of the highest plank, secondly it picks up the plank and depending on given parameters takes the plank on specific heap, lastly it checks if a whole row of planks is relocated and if it is, it relocates transverse drying boards to one of the heaps.

Keywords: FABRIK, robot, Unity, C#

Skraćenice

Ovo poglavlje nije obavezno, ali se može dodati radi preglednosti.

ABB	*Asea Brown Boveri*	tvrka koja je razvila robota IRB 120
FABRIK	*Forward and backward reaching inverse kinematics*	algoritam IK
IK	*Inverse kinematics*	inverzna kinematika

Privitak

Upute za korištenje programske podrške

Prilikom pokretanja projekta na ekranu su prikazana dva padajuća izbornika od kojih prvi određje koje daske će se slagati na novu organiziranu hrpu dok drugi padajući izbornik određuje broj dasaka po redu koje će robot slagati na platformama. Nakon odabira željenih parametara potrebno je pritisnuti gumb Kreni nakon čega će program započeti sa izvođenjem.