Računalna simulacija edukacijskog robota mBot

Trstenjak, Nikola

Master's thesis / Diplomski rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:481740

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-22**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Nikola Trstenjak

RAČUNALNA SIMULACIJA EDUKACIJSKOG ROBOTA mBot

Diplomski rad

Voditelj rada:
dr. sc. *Goran Igaly*

Zagreb, 2019.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____ , predsjednik
2. _____ , član
3. _____ , član

Povjerenstvo je rad ocijenilo ocjenom _____ .

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

SADRŽAJ

Uvod 1
Motivacija 3
Korišteni alati 4
O simulaciji 5
Podržane naredbe 5
Kontrola korisnika kroz rad simulacije 7
Datoteka ini 9
Datoteka scene 11
Datoteke s modelima 14
Datoteke s teksturama 16
Tehnički dio 19
Kretanje robota 19
Kretanje kamere 21
Detekcija sudara 22
Ultrazvučni senzor 23
Učitavanje Arduino C naredbi 24
Primjena u nastavi 26
Primjer 1: 26
Primjer 2: 27
Primjer 3: 29
Primjer 4: 31
Daljnji razvoj 33
Zaključak 35
Literatura 36
Sažetak 37
Summary 38
Životopis 39

Uvod

Informatika je školski predmet koji učenicima omogućuje upoznavanje s informacijskom i komunikacijskom tehnologijom koja se u posljednjem stoljeću svakodnevno modernizira i nadograđuje. Nastavni sadržaji pritom učenicima omogućuju stjecanje umijeća uporabe današnjih računala te primjenu različitih vještina. Informatička pismenost je nezaobilazni dio opće pismenosti pojedinca u današnjem vremenu, a sadržajno se nalazi u nastavi informatike. Također se kroz nastavu informatike učenike upoznaje s osnovnim načelima i idejama na kojima počivaju računala odnosno informacijska i komunikacijska tehnologija. Tako dolazi do razvoja sposobnosti za primjenu navedene tehnologije u različitim područjima. Stoga je potrebno nastavne sadržaje kontinuirano nadograđivati te težiti njihovom unaprjeđenju kako bi bili u toku s promjenama. Međusobno prožimanje stečenih znanja i umijeća daje podlogu za buduće cjeloživotno učenje. S obzirom na porast značaja informacijske i komunikacijske tehnologije u svakodnevnom životu, informatika postupno postaje obavezni predmet u osnovnoškolskom obrazovanju.

U skladu s time, posljednjih godina u sklopu nastave javlja se ideja programiranja fizičkih uređaja. Jedan od takvih uređaja je edukacijski robot mBot. Edukacijski robot mBot je uređaj koji na jednostavan način približava robotiku i programiranje osnovnoškolskim učenicima te kao ideja ima puno potencijala u nastavi informatike. Neke od pozitivnih strana mBot-a su jednostavnost upotrebe te vrlo pristupačno sučelje u obliku aplikacije mBlock na bazi programskog jezika Scratch. Relativno visoka cijena robota onemogućuje pojedinačnu nabavu za osobno korištenje robota, ali i masovnu nabavu za škole. Uz to bilo bi vrlo teško organizirati nastavni sat i osigurati prostor za istovremeno korištenje većeg broja robota. Zbog navedenih nedostataka javlja se ideja razvoja računalne simulacije robota koja bi te nedostatke zaobišla, ali očuvala jednostavnost upotrebe. To bi bila aplikacija koja je kompatibilna s radom robota mBot, tako da prima kod namijenjen robotu te nakon toga pruža korisniku vizualnu reprezentaciju rada robota pri čemu bi se razlike u ponašanju robota i simulacije svele na minimum.

Cilj ovog diplomskog rada je početni razvoj takve simulacije koja uz minimalno vrijeme pripreme za rad pruža interaktivnu sliku robota u tri dimenzije te simulira rad robota za definirani kod. Kako je nastava s velikim brojem robota u nastavi vrlo zahtjevna za organizaciju, takva bi simulacija dala priliku za jednostavnije uvođenje edukacijskih robota u nastavu sa smanjenjem troškova te smanjenjem potrebnog vremena pripreme i organizacije same nastave. Kroz diplomski rad rješavaju se problemi kao što su prikazivanje trodimenzionalnog objekta na samom ekranu te interpretiranje ulaznog koda namijenjenog robotu, kretanje samog robota te mnogi drugi problemi implementacije različitih algoritama.

Sastavni dio ovog diplomskog rada čini i programski kod za simulaciju rada robota koji je dostupan na lokaciji https://github.com/ntrsten/mBot-simulator, u datoteci mBot_diplomski.zip, a isti kod je pohranjen i na CD-u koji čini sastavni dio ovog diplomskog rada. Zbog opsežnosti, kod u tiskanoj verziji nije prikazan u cijelosti.

Motivacija

Ključan dio izrade simulacije je zadržavanje jednostavnog sučelja jer je cilj simulacije olakšati sam proces učenja uz pomoć edukacijskog robota, a ne otežati ga. Stoga je bitno osigurati jednostavan način početnog instaliranja i jednostavnost korištenja, ali uz to potrebno je čim više smanjiti zahtjevnost izvršavanja same aplikacije kako ne bi došlo do problema kod pokretanja na starijim računalima. Valja pružiti dovoljnu kontrolu korisniku nad samom aplikacijom čime se osigurava uvid u njezin rad. Simulacija nije izrađena uzimajući u obzir najmanje detalje robota, ni svaku fizičku silu koja se javlja kod kretanja robota, ali je ovisnost o mnogim varijablama svakako moguće dodati kroz daljnji razvoj aplikacije.

Glavni cilj projekta je izrada vizualne reprezentacije osnovnih elemenata robota i okoline te omogućiti jednostavno korištenje simulacije u nastavi uz prethodno opisane elemente.

Elementi koji su izrađeni u projektu:

- Vizualna reprezentacija robota i okoline
- Mogućnost primanja naredbi u obliku Arduino C koda
- Kretanje robota uz pomoć dva motora (lijevi i desni kotač s po jednim motorom)
- Vidljivo mijenjanje pozicije i orijentacije u trodimenzionalnom okruženju kotača robota te tijela robota
- Mogućnost mijenjanja okoline robota (dodavanje zidova)
- Detekcija sudara s dodanim zidovima
- Različiti elementi koji osiguravaju funkcionalnost senzora robota te zaprimanje naredbi (ultrazvučni senzor, infracrveni senzor)
- Očitanje ultrazvučnog senzora (prikazano u kutu ekrana u obliku sedam segmentnog displeja).

Kod razvoja same aplikacije pokušao se smanjiti broj dodanih alata koji bi možda ubrzali sam razvoj, ali ne bi pružali znanje stečeno kroz rješavanje problema koji se javljaju kroz razvoj.

U ovom simulatoru nije bio cilj simulirati najmanje fizičke sile, te su stvari kao trenje i slabljenje baterije zanemareni. Te elemente moguće je dodati u daljnjem razvoju.

Korišteni alati

Za kreiranje trodimenzionalne slike na ekranu korišteno je aplikacijsko programsko sučelje OpenGL. OpenGL je specifikacija kojom se definira rezultat odnosno izlaz određenih naredbi koje se izvršavaju na grafičkim karticama. Same implementacije tih naredbi razvijaju proizvođači grafičkih kartica u obliku upravljačkog programa - *drivera*. Kod razvoja aplikacije pomoću OpenGL-a nije dovoljna sama specifikacija, već je potrebna i biblioteka koja učitava pokazivače na sve OpenGL naredbe. U projektu se koristi GLEW (OpenGL Extension Wrangler). Biblioteka GLEW preuzeta je sa službene stranice (http://glew.sourceforge.net/ s datumom 01.02.2019.). U simulaciji koristi se verzija 2.1.0. koja pruža podršku za OpenGL 4.6., ali se u simulaciji koristi samo OpenGL 3.3. kako bi se osigurala mogućnost rada na što više uređaja.

Uz kreiranje trodimenzionalne slike, potreban nam je i prozor na kojem se ta slika prikazuje. Za kreiranje prozora te kontroliranje pritisaka tipki na tipkovnici koristi se biblioteka SDL2. SDL2 je biblioteka koja na različitim operacijskim sustavima uz mnogo drugih stvari pruža mogućnost kreiranja prozora, dohvaćanje korisničkog unosa te upravljanje fontovima. Biblioteka SDL2 preuzeta je sa službene stranice (https://www.libsdl.org/download-2.0.php s datumom 01.02.2019.). U simulaciji koristi se verzija 2.0.9. Dio za upravljanje fontovima je u samostalnoj biblioteci SDL_ttf (True Type Font) koju je također moguće preuzeti na službenoj stranici (https://www.libsdl.org/projects/SDL_ttf/ s datumom 01.02.2019.).

Kod kreiranja trodimenzionalne reprezentacije i prikazivanja iste na dvodimenzionalnom ekranu te kod kretanja modela u trodimenzionalnom svijetu javlja se puno linearne algebre. Kod transformiranja matrica i vektora koristi se biblioteka OpenGL Mathematics (GLM). Biblioteka GLM preuzeta je s web stranice (https://github.com/g-truc/glm/tags s datumom 01.02.2019.). U simulaciji koristi se verzija 0.9.9.3.

Navedeni alati jedini su vanjski alati korišteni u razvoju aplikacije. Za pokretanje aplikacije potrebno je uz izvršnu datoteku priložiti potrebne biblioteke dinamičkih veza (.dll datoteke). Konačna verzija aplikacije dolazi pripremljena za rad te zapakirana u mapu i ne zahtijeva nikakvu instalaciju.

O simulaciji

Kompletna aplikacija dolazi u sažetoj mapi (.zip) u kojoj se nalazi mapa „resources“ i izvršna datoteka (.exe datoteka) te popratne biblioteke dinamičkih veza (.dll datoteke) potrebne za izvršavanje aplikacije.

Uz izvršnu datoteku moguće je priložiti tekstualnu datoteku koja sadrži argumente i postavke simulacije (.ini datoteka) te ako postoji takva datoteka, potrebno je simulaciju pokrenuti s tom datotekom kao argumentom aplikacije. To je postignuto na jednostavan način, samo kreiranjem datoteke (.bat datoteke na operacijskom sustavu Windows) koja sadrži skriptu kojom se pokreće aplikacija s jednim argumentom.

Uz to, svi potrebni resursi (datoteke koje opisuju 3D objekte (.obj datoteke), datoteke s teksturama za dane objekte u bitmap (.bmp) formatu, datoteka s fontom, tekstualna datoteka koja opisuje *level*, tj. scenu te dvije tekstualne datoteke s kodom za *shader* programe koji se izvode na grafičkoj kartici se nalaze u popratnoj mapi „resources“ uz izvršnu datoteku.

Programski kod za robota mBot u jeziku Arduino C prosljeđuje se simulaciji putem međuspremnika operacijskog sustava. Netom prije pokretanja simulacije dovoljno je iz aplikacije mBlock iskopirati kompletan kod prikazan u „Arduino mode“ načinu uređivanja koda (kombinacije tipki Ctrl+A za označavanje cijelog programskog koda te Ctrl+C za kopiranje označenog koda u međuspremnik). U nastavku je napisano koje su naredbe iz mBlock blokovskog sučelja podržane za korištenje u simulatoru.

Svrha ovakve organizacije dokumenata i datoteka je mogućnost prilagođavanja rada simulatora. Što sve je prilagodljivo putem .ini datoteke, opisano je u nastavku. Isto tako, što točno sadrži određeni tip datoteke i koja je sama struktura tih datoteka također slijedi u nastavku. Ako prilagođavanje aplikacije nije prioritet, postoji mogućnost brzog i jednostavnog pokretanja simulacije. Nakon kopiranja Arduino C koda u međuspremnik, dovoljno je pokrenuti izvršnu datoteku sa ili bez popratne .ini datoteke.

Podržane naredbe

Cilj simulatora je simuliranje kretanja robota za određeni skup naredbi. Simulacija te naredbe prima putem međuspremnika, odnosno dovoljno je samo iskopirati Arduino kod iz aplikacije mBlock. Aplikacija mBlock za programiranje samog robota omogućuje korištenje brojnih naredbi. Trenutno su u simulatoru implementirane neke od njih, ali kroz daljnji razvoj postoji mogućnost dodavanja podrške i za ostale naredbe.

Trenutno podržane naredbe su:

Naredbom mBot Program zadaje se početna točka za program kojim se upravlja radom mBota.

Postavljanje brzine pojedinog motora na određenu brzinu između -255 i 255. Pozitivna vrijednost znači okretanje motora prema naprijed, a negativna vrijednost prema natrag. Vrijednost nula znači mirovanje motora.

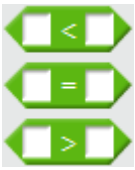
Kretanje robota u jednom od četiri osnovna smjera određenom brzinom.

Postavljanje pauze na izvršavanje programa.

Ponavljanje koda zauvijek. Za razliku od programa kojima je cilj pomoću nekog algoritma riješiti određeni problem i kod kojih je nužan uvjet da program završi rad u nekom vremenu, kod programiranja fizičkih objekata je uvijek prisutna ova beskonačna petlja koja omogućuje trajno reagiranje uređaja na događaje iz vanjskog svijeta.

Uvjetni izrazi koji mogu koristiti dolje navedene naredbe i logičke operatore.

Ako se koristi naredba koja provjerava stanje pritisnutih tipki na infracrvenom upravljaču, svaka tipka se interpretira na tipkovnici računala na kojem se simulacija pokreće. Na primjer ako je kao na slici dan uvjet s tipkom „A“, taj dio koda se izvršava dok se pritisne tipka „A“ na računalu.



Čitanje ultrazvučnog senzora moguće je koristiti uz sva tri operatora uspoređivanja veličina.

Kontrola korisnika kroz rad simulacije

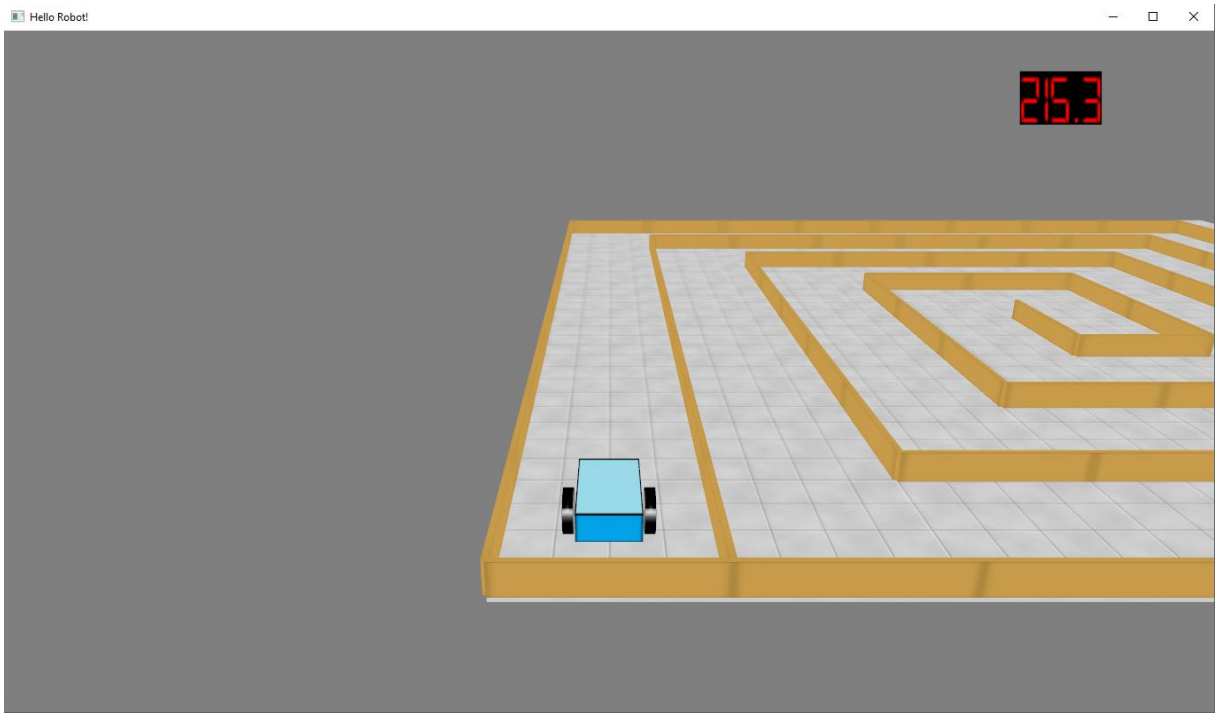
Kroz rad same simulacije poželjno je da korisnik ima kontrolu nad njezinim radom.

Izvršavanje koda moguće je inicijalno odgoditi do pritiska tipke Enter. Ako korisnik želi tu funkcionalnost, dovoljno je u datoteci .ini promijeniti vrijednost argumenta „waitOnStart“.

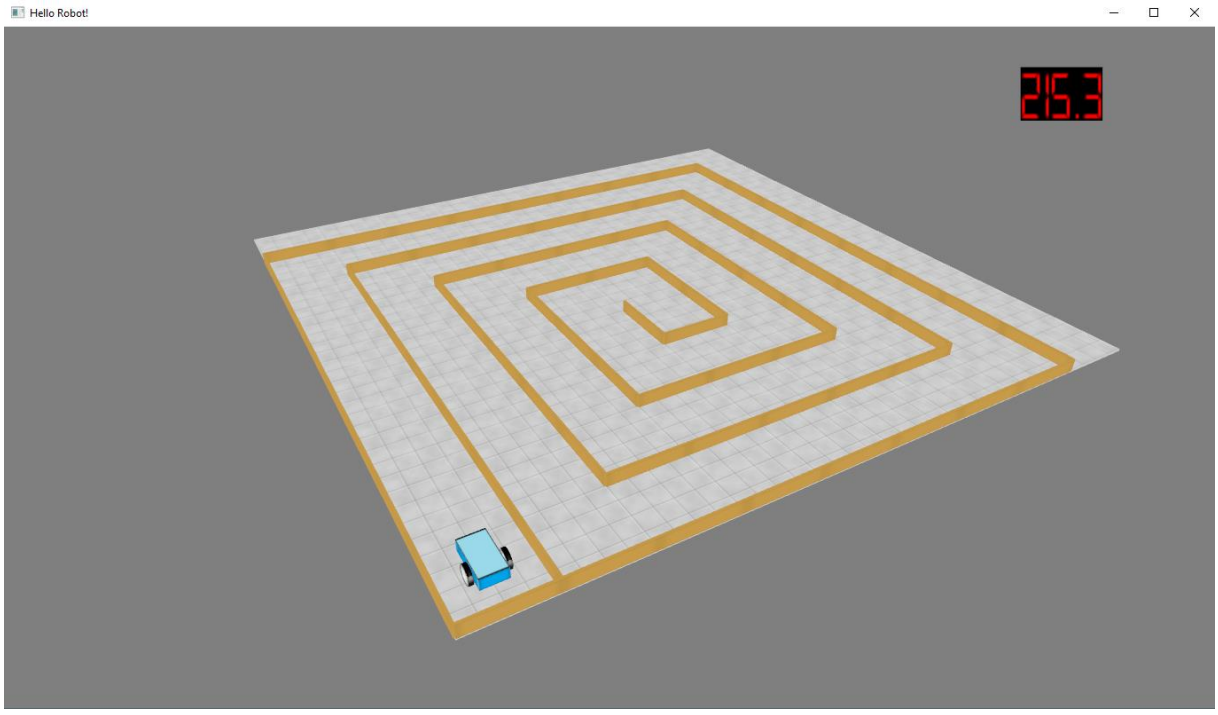
Uz to korisnik ima mogućnost upravljanja kamerom (pogledom) u simulaciji. U simulaciji su implementirane tri vrste pogleda. Na početku je postavljen pogled iz trećeg lica koji je uvijek direktno iznad robota (Slika 1). U tom pogledu korisnik može koristiti funkcionalnost približavanja i udaljavanja pogleda vrtnjom kotačića na mišu. Uz to postoji pogled koji nije vezan uz kretanje robota te u tom pogledu korisnik ima kompletnu kontrolu nad kretanjem kamere (Slika 2). Kut gledanja mijenja se kretanjem miša računala. Lokacija kamere mijenja se tipkama „I“ (naprijed), „K“ (nazad), „J“ (lijevo), „D“ (desno), „U“ (dolje) te „O“ (gore). Te tipke su korištene za kretanje kamera jer su direkcijske tipke na tipkovnici rezervirane za simuliranje infracrvenog upravljača. Isto su tako tipke „A“ – „F“ rezervirane, što onemogućuje korištenje klasičnog WASD kretanja kamere. Treći pogled je pogled „iz prvog lica“, tj. kamera je postavljena na robot (Slika 3). U trećem pogledu, korisnik ne može upravljati kamerom već ona uvijek gleda u smjeru u kojem gleda robot.

Mijenjanje pogleda u simulaciji ostvaruje se pritiskom tipke „V“ na tipkovnici računala.

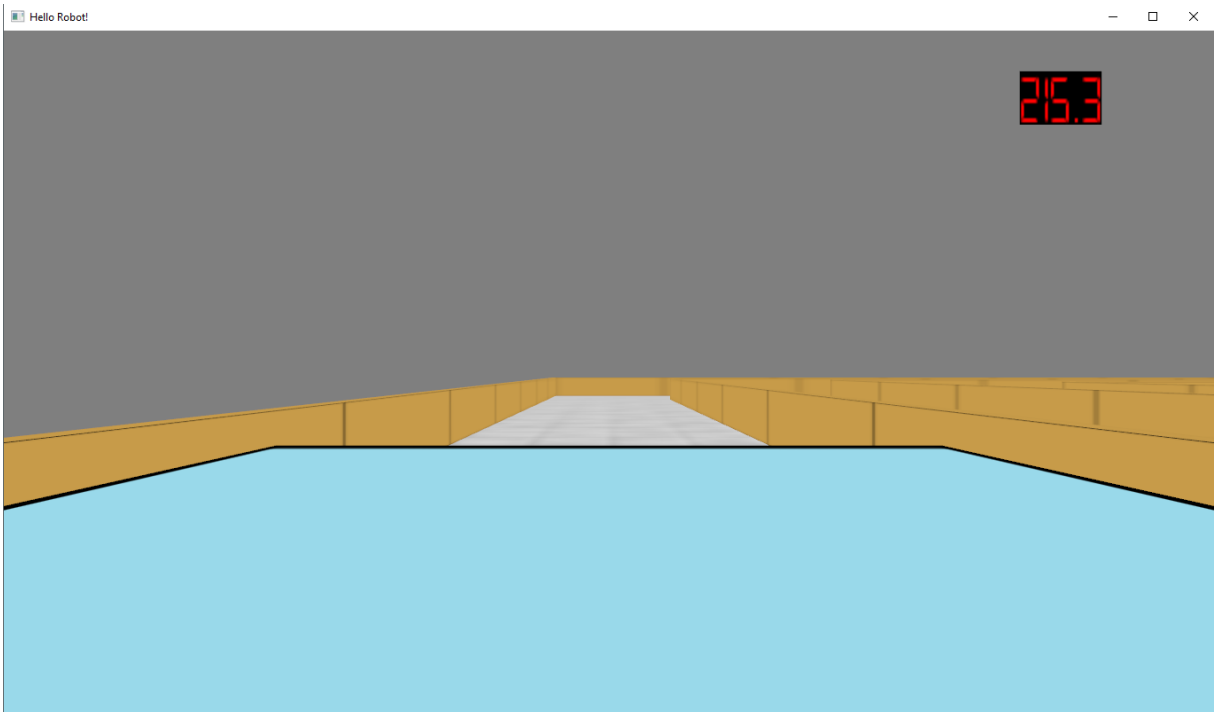
Simulaciju je moguće isključiti u bilo kojem trenutku pritiskom tipke „Esc“.



Slika 1 Prvi pogled; praćenje vozila



Slika 2 Drugi pogled; slobodno kretanje kamere



Slika 3 Treći pogled; pogled na robotu

Datoteka ini

Datoteka s nastavkom .ini, zapravo tekstualna datoteka, koristi se kako bi simulator zaprimio osnovne postavke potrebne za rad. Ako takva datoteka nije proslijeđena simulatoru, koriste se generičke vrijednosti.

U .ini datoteku se zapisuju točni nazivi argumenata (navedeni u zagradama) s lijeve strane znaka jednakosti dok neposredno nakon znaka jednakosti mora pisati vrijednost danog argumenta. Ti argumenti, ne nužno u istom poretku su:

- Širina prikaza simulatora u pikselima (displayWidth); prirodan broj
- Visina prikaza simulatora u pikselima (displayHeight); prirodan broj
- Je li prikaz simulatora preko cijelog zaslona (isFullscreen); boolean varijabla; 1 za da, a 0 za ne
- Ciljana frekvencija prikazivanja slika (targetFramerate); prirodan broj
- Kreće li simulator s izvođenjem kopiranog koda u trenutku pokretanja ili čeka pritisak tipke Enter (waitOnStart); boolean varijabla; 1 za čekanje, 0 za ne
- Putanja do datoteke koja opisuje scenu (levelFile); tekst s putanjom do datoteke bez navodnika
- Veličina jednog polja u sceni u centimetrima (tileSize); decimalan broj

- Visina zidova u sceni u centimetrima (wallHeigth); decimalan broj
- Putanja do .obj datoteke koja opisuje 3D model tijela vozila (carBodyModel); tekst s putanjom do datoteke bez navodnika
- Putanja do .bmp datoteke s teksturom tijela vozila (carBodyTexture); tekst s putanjom do datoteke bez navodnika
- Putanja do .obj datoteke koja opisuje 3D model kotača vozila (carWheelModel); tekst s putanjom do datoteke bez navodnika
- Putanja do .bmp datoteke s teksturom kotača vozila (carWheelTexture); tekst s putanjom do datoteke bez navodnika
- Putanja do .bmp datoteke s teksturom jednog polja poda scene (floorTexture); tekst s putanjom do datoteke bez navodnika
- Putanja do .obj datoteke s teksturom jednog zida scene (wallTexture); tekst s putanjom do datoteke bez navodnika
- Duljina polumjera kotača vozila (robota) u centimetrima (wheelRadius); decimalan broj
- Širina jednog kotača vozila (robota) u centimetrima (wheelWidth); decimalan broj
- Udaljenost od sredine jednog kotača vozila (robota) do sredine drugog kotača u centimetrima (wheelDistance); decimalan broj
- Duljina tijela vozila (robota) u centimetrima (bodyLength); decimalan broj
- Širina tijela vozila (robota) u centimetrima (bodyWidth); decimalan broj
- Visina tijela vozila (robota) u centimetrima (bodyHeight); decimalan broj
- Udaljenost od stražnjeg dijela tijela robota do osovine kotača robota u centimetrima (backBodyAxelOffset); decimalan broj
- Udaljenost od najnižeg dijela tijela robota do osovine kotača robota u centimetrima (bottomBodyAxelOffset); decimalan broj
- Veličina kuta u stupnjevima za koji se kotač okrene u jednoj sekundi uz maksimalnu brzinu okretanja (wheelRotationPerSecond); decimalan broj
- Kut vidnog polja u stupnjevima, ne preporučuje se mijenjati zadanu veličinu od 45 stupnja (fieldOfView); decimalan broj

Slijedi popratni primjer .ini datoteke.

```
displayWidth        =1366
displayHeight       =768
isFullscreen        =0
targetFramerate     =60
waitOnStart         =1
levelFile           =./resources/lvl.lvl
tileSize            =30
```

wallHeight	=5
carBodyModel	=./resources/body.obj
carBodyTexture	=./resources/body.bmp
carWheelModel	=./resources/wheel.obj
carWheelTexture	=./resources/wheel.bmp
floorTexture	=./resources/floor.bmp
wallTexture	=./resources/wall.bmp
wheelRadius	=3.2
wheelWidth	=1.5
wheelDistance	=11.2
bodyLength	=16.7
bodyWidth	=9.0
bodyHeight	=4.0
backBodyAxelOffset	=5.6
bottomBodyAxelOffset	=1.5
wheelRotationPerSecond	=720
fieldOfView	=45

Sve navedene argumente moguće je promijeniti te promatrati ponašanje rada same simulacije.

Datoteka scene

U simulatoru uz vozilo robota prikazuje se i okolna scena. Scena koja okružuje vozilo sastoji se od okomitih zidova te podova raspoređenih u dvodimenzionalno polje. Kako bi se takva scena opisala dovoljno je za pojedino dvodimenzionalno polje definirati s koje strane je okruženo zidovima. Kako susjedna polja mogu imati zajedničke zidove, u simulatoru se definiraju samo donji i lijevi zidovi danog polja. Tako je postignuta mogućnost lakog zapisivanja .lvl datoteke.

Sama datoteka se sastoji od dvodimenzionalnog polja znakova. Znakovi koji se mogu koristiti su ' ', '|', '_' i 'L'. Broj redova znakova i broj pojedinih znakova u svakom retku u datoteci direktno rezultira istim rasporedom u simulatoru.

Ako se na nekom mjestu koristi prazan znak (razmak), na tom mjestu će u simulatoru biti polje koje s donje i s lijeve strane nema zid.

Ako se na nekom mjestu koristi znak vertikalne crte ('|'; AltGr+W), na tom mjestu će u simulatoru biti polje koje s lijeve strane ima zid, a s donje strane nema zid.

Ako se na nekom mjestu koristi znak podcrtavanja ('_'), na tom mjestu će u simulatoru biti polje koje s donje strane ima zid, a s lijeve strane nema zid.

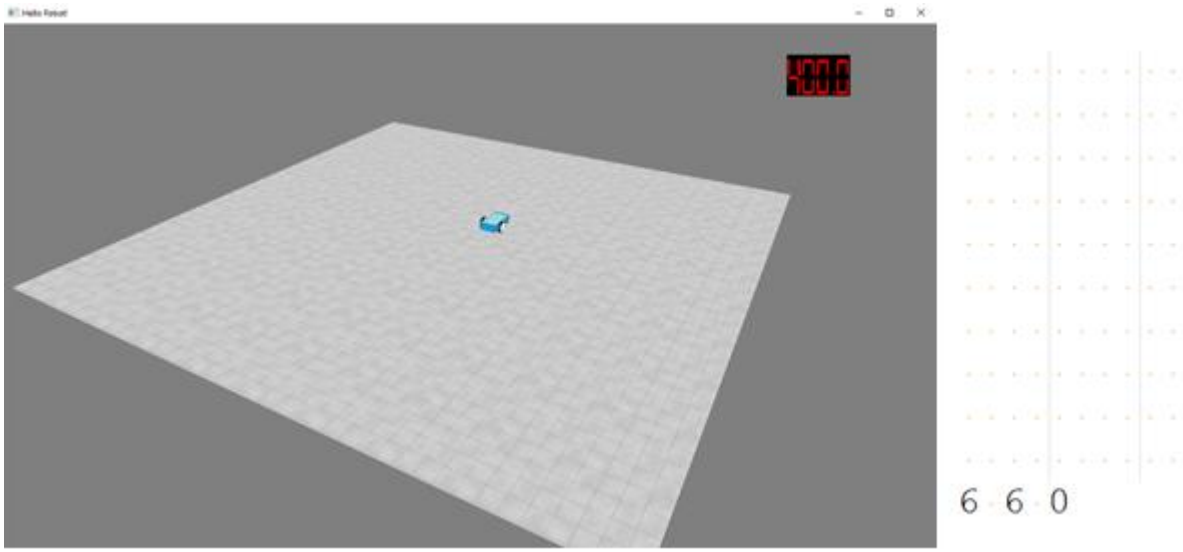
Ako se na nekom mjestu koristi znak velikog slova L ('L'), na tom mjestu će u simulatoru biti polje koje i s donje i s lijeve strane ima zid.

Neposredno nakon polja znakova slijedi novi red s tri cijela broja. Ti brojevi opisuju početnu poziciju robota. Prvi broj definira broj stupca početnog polja robota u priloženoj matrici, a drugi broj definira broj retka početnog polja robota u priloženoj matrici. Ta dva broja nisu ograničena veličinom matrice, te slučaj kad su veći od broja redaka, odnosno stupaca matrice znači samo da je početna pozicija robota izvan naše scene. Isto vrijedi ako su ti brojevi manji od nule. Treći broj je početni kut rotacije robota u stupnjevima. Pozitivan broj je u smjeru suprotnom od kazaljke na satu, a kut od veličine 0 stupnjeva znači da robot gleda prema gore u našoj sceni definiranoj matricom.

Ako korisnik ne želi nikakvu scenu, dovoljno je ostaviti jedan prazan red te nakon toga napisati početnu lokaciju robota.

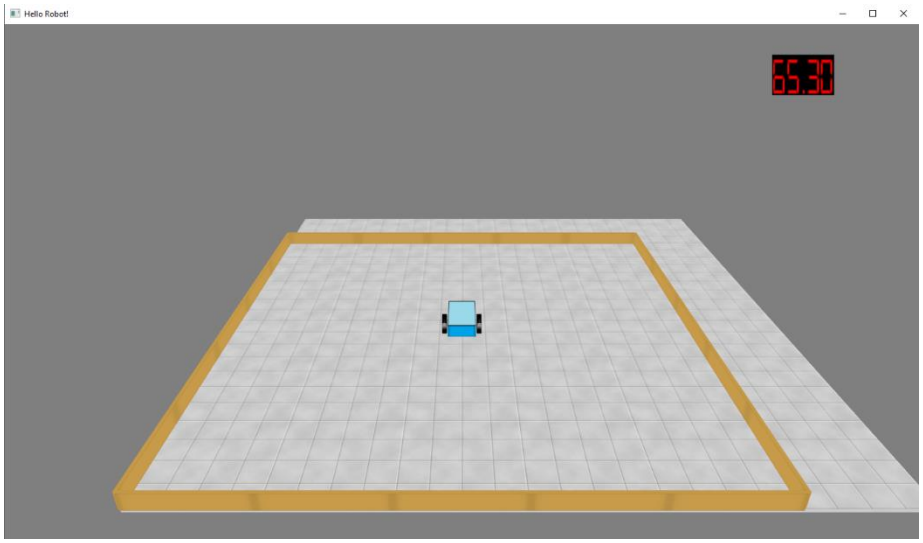
Slijede primjeri scena sa slikama scena uz primjere datoteka iz kojih se te scene generiraju.

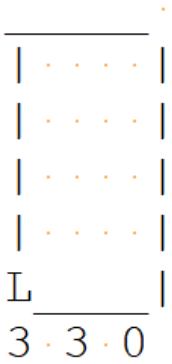
Prazna scena:



Slika 4 Prazna scena uz njezin opis

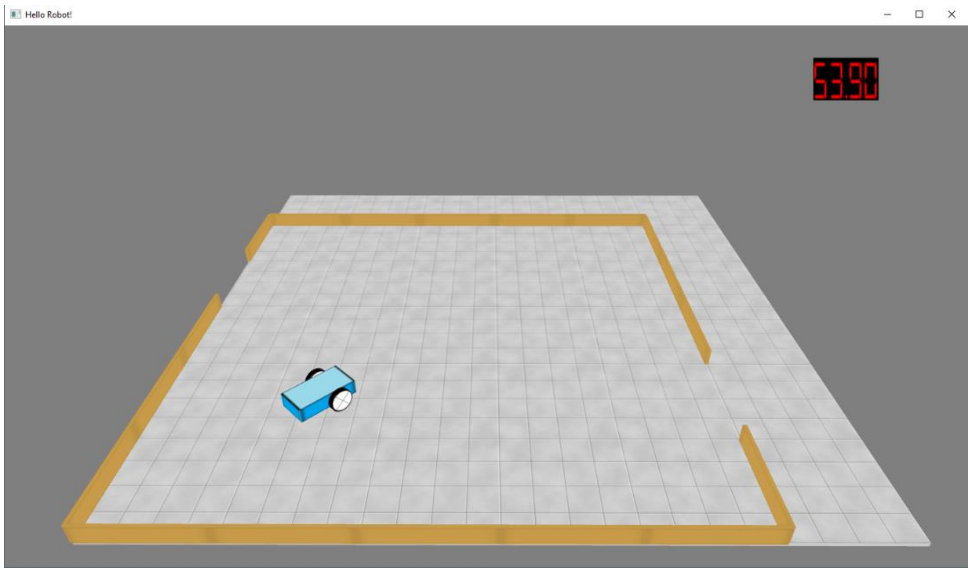
Zidovi oko scene:

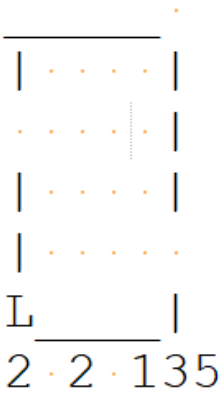




Slika 5 Scena sa zidovima uz njezin opis

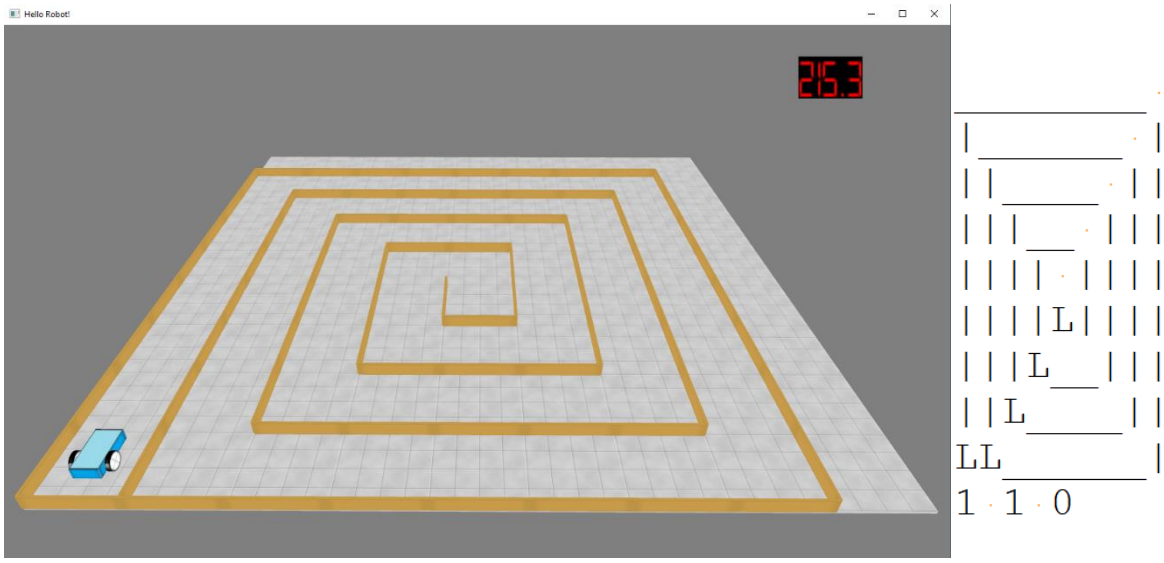
Zidovi s izlazima:





Slika 6 Scena sa vratima u zidovima uz njezin opis

Spiralni zidovi:



Slika 7 Scena sa spiralnim zidovima

Bitno je napomenuti da narančaste točke na slikama s tekstualnim opisom scene nisu znakovi točke '.' već znakovi razmaka, tj. ' ', ali je zbog lakšeg povezivanja teksta i slike ostavljeno vidljivo.

Datoteke s modelima

U našem simulatoru jedno trodimenzionalno tijelo sastoji se od niza trokuta, a svaki trokut od niza vrhova. Svakom vrhu pripisuje se koordinata lokacije u tri dimenzije te koordinata teksture (reprezentirane pomoću .bmp slike). Trodimenzionalna tijela u simulaciji opisuju se pomoću jednostavnih .obj datoteka. Takve datoteke standardizirane su i mogu se otvarati i uređivati pomoću različitih programa za uređivanje trodimenzionalnih slika. Za potrebe ovog programa korišten je program Blender (https://www.blender.org/).

Jednostavna .obj datoteka sastoji se od redova u kojima se zapisuju podaci. Na početku svakog reda zapisuje se indikator tipa podataka (jedan ili nekoliko znakova) u danom redu. Podaci koji nama trebaju za simulaciju su koordinate lokacije vrhova, koordinate tekstura vrhova te raspored tih vrhova u trokute (strane).

Redak u kojem se zapisuju koordinate lokacije jednog vrha trodimenzionalnog modela počinje znakom 'v' (kao *vertex*) te nakon toga slijede razmakom odvojene tri koordinate tog vrha.

Redak u kojem se zapisuju koordinate teksture jednog vrha trodimenzionalnog modela počinje znakovima 'vt' te nakon toga slijede razmakom odvojene dvije koordinate tog vrha. Ovdje se koriste samo dvije koordinate jer se teksture prikazuju pomoću dvodimenzionalnih slika.

Redak u kojem se zapisuju tri vrha koji opisuju jedan trokut (stranu) počinje znakom 'f' (kao *face*) te nakon toga slijede po tri razmakom odvojena skupa po tri indeksa odvojena kosom crtom (indeks lokacije / indeks teksture / indeks normale) prije danih koordinata vrhova. Indeks normale trenutno se ne koristi u simulatoru, kao ni redovi u kojima se zapisuju normale u danom vrhu ('vn'). Efekt refleksije svjetlosti u simulaciji nije implementiran, ali ako se implementira potrebno je koristiti normale u točki modela zapisane u datoteci u redovima koji počinju znakovima 'vn'.

U nastavku slijedi primjer .obj datoteke kojom se opisuje kocka uz detaljniji opis.

```
# Blender v2.79 (sub 0) OBJ File: ''
# www.blender.org
v -1.0 -1.0  1.0
v -1.0  1.0  1.0
v -1.0 -1.0 -1.0
v -1.0  1.0 -1.0
v  1.0 -1.0  1.0
v  1.0  1.0  1.0
v  1.0 -1.0 -1.0
v  1.0  1.0 -1.0
vt  0.25 0.25
vt  0.50 0.50
vt  0.25 0.50
vt  0.75 0.50
vt  0.50 0.75
vt  0.50 1.00
vt  0.25 0.75
vt  0.00 0.75
vt  0.50 0.25
vt  0.25 0.00
vt  0.50 0.00
vt  0.75 0.75
vt  0.25 1.00
vt  0.00 0.50
vn -1.0  0.0  0.0
vn  0.0  0.0 -1.0
```

```
vn  1.0  0.0  0.0
vn  0.0  0.0  1.0
vn  0.0 -1.0  0.0
vn  0.0  1.0  0.0
s off
f 2/1/1 3/2/1 1/3/1
f 4/4/2 7/5/2 3/2/2
f 8/6/3 5/7/3 7/5/3
f 6/8/4 1/3/4 5/7/4
f 7/5/5 1/3/5 3/2/5
f 4/9/6 6/10/6 8/11/6
f 2/1/1 4/9/1 3/2/1
f 4/4/2 8/12/2 7/5/2
f 8/6/3 6/13/3 5/7/3
f 6/8/4 2/14/4 1/3/4
f 7/5/5 5/7/5 1/3/5
f 4/9/6 2/1/6 6/10/6
```

Kako bismo pojasnili način zapisivanja modela, možemo se poslužiti ovim primjerom. Uzmemo li redak „f 2/1/1 3/2/1 1/3/1“, dobijemo opis jednog trokuta (tri razmakom odvojena niza od tri prirodna broja). Prvi od tih nizova je 2/1/1. Kao što smo i prethodno utvrdili prvi od tih brojeva govori nam indeks koordinate prvog vrha koji opisuje taj trokut. Referencira se indeks prethodno navedenih vrhova u istoj datoteci, što znači koordinate prvog vrha prvog trokuta našeg modela zapisane su u drugom redu s početnim slovom 'v' u našoj datoteci. To je redak „v -1.0 1.0 1.0“. Drugi od tih brojeva nam na isti način govori koordinate tekstura našeg vrha. Treći od tih brojeva opisuje nam normalu u danom vrhu, što trenutno ne koristimo u simulaciji. Postupak se ponavlja za svaki od tri vrha u svakom od opisanih trokuta u datoteci.
Kod kreiranja .obj datoteke potrebno je osigurati da je najmanji dio modela trokut, a ne četverokut jer postoji analogan način zapisivanja modela kao skupa četverokuta. Taj način zapisivanja simulator ne podržava.

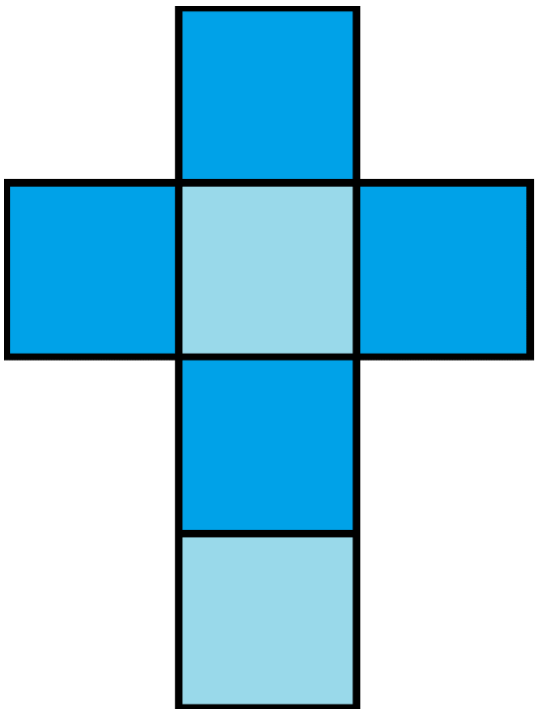
Datoteke s teksturama

Sve teksture koje simulator koristi spremaju se u slikovnom formatu. Jedini podržani slikovni format je BMP (*bitmap image file*) format. Razlog tome je jednostavna struktura zapisa tog formata što omogućuje jednostavnu implementaciju čitanja datoteke u simulatoru.

Sama datoteka sastoji se od zaglavlja zadane veličine (54 bajta). Prva dva bajta u datoteci su vrijednost znaka „B“ te vrijednost znaka „M“ u ASCII kodu. U nastavku zaglavlja zapisuju se informacije o samoj slici, kao što su širina i visina slike u pikselima, te broj bitova korištenih za zapis boje jednog piksela.

Neposredno nakon zaglavlja slijedi tablični zapis boje svakog od piksela u RGBA (4 bajta za svaki piksel) modelu. Ako je alfa vrijednost isključena koristi se RGB model (3 bajta za svaki piksel).

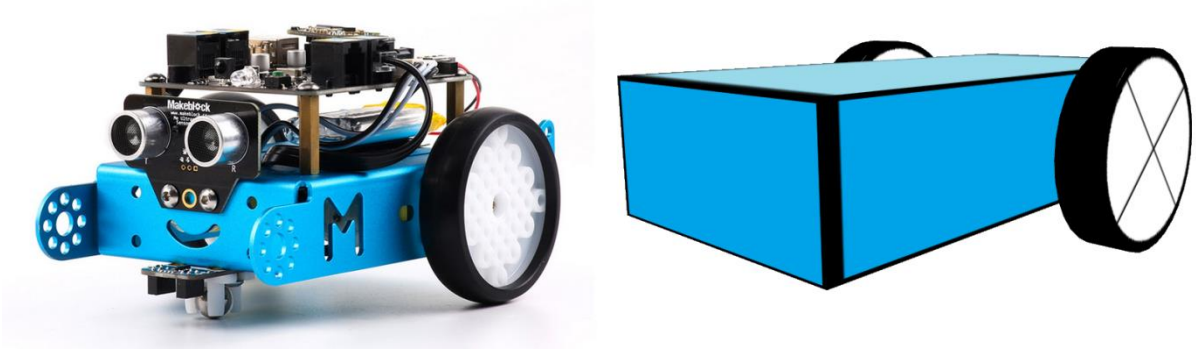
Primjer datoteke s teksturom za tijelo robota (kocka koja se ovisno o veličinama skalira u svaku od dimenzija) je na slici Slika 8.



Slika 8 Tekstura za tijelo robota

Promjene na ovoj datoteci, kao i na svim ostalim datotekama, moguće je promatrati u simulaciji nakon ponovnog pokretanja simulacije.

Na slici prikazani su robot mBot te robot u simulaciji.



Slika 9 Usporedba pravog robota i robota u simulaciji

Tehnički dio

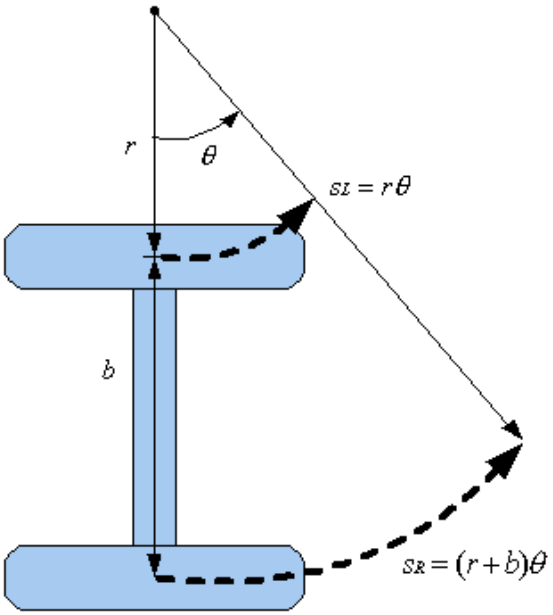
Kretanje robota

Edukacijski robot mBot je vozilo koje ima dva pogonska motora, jedan na lijevom, jedan na desnom kotaču. Pogonski motori ne ovise jedan o drugom te se kotači mogu neovisno okretati. Time se dobiva mogućnost okretanja vozila u cijelosti. Kod rada simulacije, potrebno je kod svakog koraka računati poziciju vozila ovisno o brzini okretanja svakog od kotača. Ako oba kotača imaju istu brzinu vrtnje, robot prati ravnu crtu u smjeru trenutnog zakretnog kuta. Ako kotači imaju suprotne brzine, robot se samo okreće oko točke između kotača (na sredini). Ako jedan kotač ima veću brzinu okretanja od drugog, robot prati krivulju, zakrivljenu prema sporijem kotaču. U simulaciji kretanje robota u trećoj dimenziji nije moguće, pa je računanje pozicije samim time jednostavnije.

Prvi i najjednostavniji slučaj je kad oba kotača imaju istu brzinu vrtnje. Sama promjena pozicije ovisi o trenutnoj orijentaciji vozila i brzini okretanja kotača. Dovoljno je izračunati put koji je prošao jedan od kotača okrećući se određenom brzinom u nekom vremenu. Put koji je prošao kotač dobijemo množenjem veličine kuta za koji se kotač okrenuo s polumjerom kotača i promjenom vremena. Nakon toga uz pomoć trigonometrije izračunava se promjena pozicije robota na pojedinoj osi.

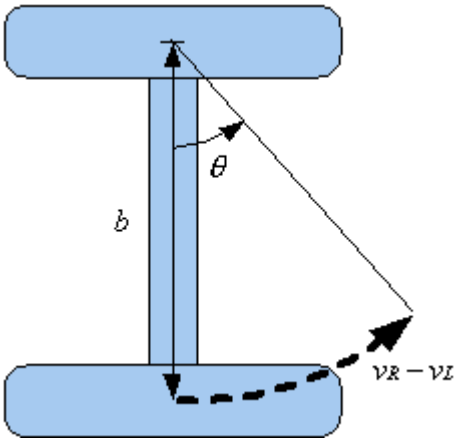
$$PromjenaPozicijePoJednojOsi = PutKotača * sin(ZakretniKutVozila)$$
$$PromjenaPozicijePoDrugojOsi = PutKotača * cos(ZakretniKutVozila)$$
$$PromjenaZakretnogKuta = 0$$

Slučaj kada kotači imaju suprotne brzine okretanja, što se tiče računanja pozicije, pripada slučaju kada kotači imaju različite brzine okretanja. Ta situacija prikazana je slikom 10. U tom slučaju potrebno je izračunati promjenu pozicije robota te promjenu orijentacije robota.



Slika 10 Kretanje robota

Promjena orijentacije neovisna je o referentnoj točki s obzirom na koju se sustav promatra, pa se cijeli sustav može promatrati u okviru relativnom na poziciju lijevog kotača. Tako se dobije situacija prikazana slikom 11.



Slika 11 Kretanje robota s obzirom na referentnu točku

Sada se lako izračuna promjena orijentacije za dane pređene putove svakog od kotača. Polumjer okretanja je udaljenost od sredine jednog kotača do sredine drugog kotača. Prikažimo to u obliku jednadžbe u ovisnosti o vremenu t.

$$\frac{d\theta}{dt} = \frac{v_r - v_l}{b}$$

Integrira li se dana jednakost dobijemo orijentaciju u ovisnosti o vremenu t.

$$\theta(t) = \frac{(v_r - v_l)t}{b} + \theta_0$$

Ova jednakost vrijedi i u apsolutnom okviru promatranja sustava. Znamo da promjena pozicije robota ovisi o brzini točke između (sredina osovine) kotača. Brzina kretanja te točke je zapravo aritmetička sredina brzine kretanja dvaju kotača. Ovdje je vidljivo da ako su brzine jednog i drugog kotača suprotne, robot se zapravo ne kreće, nego samo mijenja orijentaciju. Sada je moguće objediniti dobivene činjenice u formulu promjene pozicije na pojedinoj osi u ovisnosti o vremenu.

$$\frac{dx}{dt} = \left(\frac{v_r + v_l}{2}\right)\cos\big(\theta(t)\big)$$

$$\frac{dy}{dt} = \left(\frac{v_r + v_l}{2}\right)\sin\big(\theta(t)\big)$$

Integriraju li se dane jednakosti uz početne pozicije x_0 te y_0 dobiju se danje jednakosti.

$$x(t) = x_0 + \frac{b(v_r + v_l)}{2(v_r - v_l)}\left(\sin\left(\frac{(v_r - v_l)t}{b} + \theta_0\right) - \sin(\theta_0)\right)$$

$$y(t) = x_0 + \frac{b(v_r + v_l)}{2(v_r - v_l)}\left(\cos\left(\frac{(v_r - v_l)t}{b} + \theta_0\right) - \cos(\theta_0)\right)$$

Kod implementacije u kodu je potrebno pripaziti na slučaj kada su v_r i v_l jednake veličine zbog dijeljenja s nulom.

Kretanje kamere

Kamera je pogled na robot u simulaciji i potrebno je implementirati kretanje kamere kako bi bilo moguće usmjeriti pogled na trenutnu lokaciju robota. Pozicija kamere u simulaciji je zapravo točka u trodimenzionalnom koordinatnom sustavu, a smjer gledanja kamere određuje se uz pomoć kuta nagiba te zakretnog kuta. Te kutove moguće je interpretirati kao kutove za određivanje točke u sfernom koordinatnom sustavu.

Poziciju kamere u slučaju prvog pogleda (kamera prati robot), lako je implementirati. Potrebno je samo u odnosu na orijentaciju robota postaviti kameru iznad i iza robota te postaviti nagib kamere fiksno prema robotu, a orijentacija kamere prati orijentaciju robota. U slučaju trećeg pogleda (kamera na robotu) pozicija kamere ista je kao i pozicija robota te orijentacija kamere prati orijentaciju robota.

Kamera u aplikaciji implementirana je u obliku matrice pogleda kojom se množi svaka od koordinata svake točke svakog modela u aplikaciji. Time se dobiva pomak kompletne

slike u okviru zaslona. Kod određivanja te matrice koristi se biblioteka GLM. Matrica pogleda određuje se funkcijom *lookAt* koja kao argumente prima trenutnu lokaciju kamere u koordinatnom sustavu, vektor smjera orijentiran i usmjeren direktno ispred kamere te vektor smjera orijentiran i usmjeren direktno iznad kamere.

Implementacija kretanja kamere sukladno s unosom korisnika zahtijeva praćenje unosa korisnika. Kretanje miša rezultira okretanjem kamere, a pritisak na određene tipke rezultira kretanjem kamere. Pomoću biblioteke SDL2 moguće je očitati kretanje miša po svakoj od dvije osi. Dobivene veličine interpretiraju se kao promjene veličina kutova u sfernom koordinatnom sustavu. Kako bi se dobio vektor smjera gledanja kamere, potrebno je sferne koordinate pretvoriti u koordinate u Kartezijevom koordinatnom sustavu.

```
fwd.x = cos(glm::radians(Yaw)) * cos(glm::radians(Pitch));

fwd.y = sin(glm::radians(Pitch));

fwd.z = sin(glm::radians(Yaw)) * cos(glm::radians(Pitch));

Front = glm::normalize(fwd);
```

Za normalizaciju vektora te preračunavanje veličina kutova u radijane koristi se biblioteka GLM. Ovim postupkom dobiva se vektor „Front“ koji je orijentiran i usmjeren direktno ispred kamere. Vektor smjera koji pokazuje direktno iznad kamere određuje se uz pomoć vektorskog produkta. Kao referentni vektor koristi se vektor „WorldUp“, vektor orijentiran i usmjeren gore (u svijetu simulacije) u bilo kojem trenutku te se prvotno određuje vektor koji je orijentiran desno („Right“) od pogleda kamere kao vektorski produkt vektora „Front“ i „WorldUp“. Nakon toga vektorskim produktom vektora „Right“ i „Front“ dobije se vektor orijentiran i usmjeren iznad kamere.

```
Right = glm::normalize(glm::cross(Front, WorldUp));

Up    = glm::normalize(glm::cross(Right, Front));
```

Za računanje vektorskog produkta koristi se biblioteka GLM. Nakon toga dovoljno je odrediti matricu pogleda koja se šalje grafičkoj kartici pomoću naredbe *lookAt*.

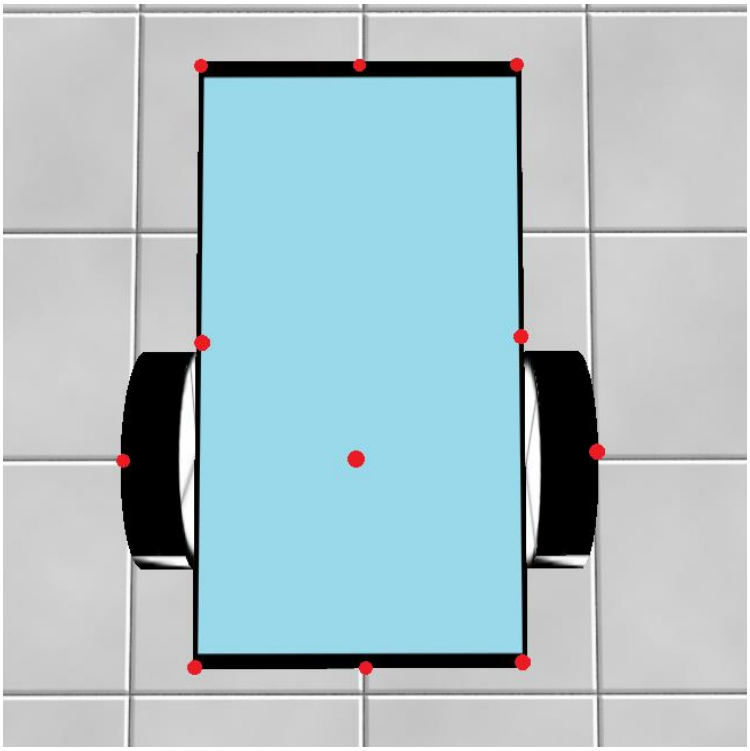
```
glm::lookAt(Position, Position + Front, Up);
```

Detekcija sudara

Kako je u simulaciju moguće dodati zidove kao dio scene, potrebno je implementirati algoritam kojim se provjerava je li se dogodio sudar između robota i nekog od zida. Jedina moguća lokacija zidova je na granici polja scene (scena je podijeljena na polja). Ta

činjenica pojednostavljuje postavljeni problem. Dovoljno je za svaki pomak vozila robota provjeriti je li se neka od kontrolnih točaka pomaknula s jednog polja na drugo te ako je, provjerava se postoji li zid između tih točaka. Ako zid postoji, dogodio se sudar.

Ovaj način detekcije nije savršen jer ovisi o broju kontrolnih točaka. Potrebno je uzeti veliki broj kontrolnih točaka na robotu. Trenutno u simulaciji postoji po jedna točka za svaki od četiri vrha na donjoj strani vozila, po četiri točke na sredini bočnih strana vozila, jedna točka na svakom od kotača te jedna točna na sredini vozila (Slika). Uz to, algoritam ovisi o implementaciji zidova u simulaciji te ako se način implementacije zidova mijenja, potrebno je promijeniti i algoritam detekcije sudara.



Slika 12 kontrolne točke robota

Sa slike 12 vidljivo je koji su nedostaci navedenog algoritma. Sudar se ne provjerava za cijeli model robota već samo za nekoliko određenih točaka. Uz trenutnu implementaciju zidova, implementacija efikasnijeg algoritma za detekciju sudara bila bi vrlo zahtjevna.

Ultrazvučni senzor

U simulaciji implementirana je funkcionalnost očitanja udaljenosti robota od prepreke (zida) s ultrazvučnim senzorom. Algoritam računanja udaljenosti vrlo je sličan algoritmu detekcije sudara te isto kao i algoritam detekcije sudara ovisi o trenutnoj implementaciji

zidova u simulaciji. Algoritam provjerava nalazi li si zid na putu robota na postupno sve većoj udaljenosti od robota. Ako se zid ne nalazi ispred robota, ispisuje se maksimalna vrijednost očitanja senzora (400 cm).

Kako bi korisnik dobio povratnu informaciju, potrebno je ispisati očitanje ultrazvučnog senzora. Za ispisivanje brojki potrebno je iz fonta napraviti teksturu koja se koristi kod crtanja nekog modela. OpenGL ne može rukovati fontovima, pa je potrebno koristiti SDL2_ttf kako bi se iz datoteke fonta (datoteka tipa True Type Font, tj. ttf) dobila tekstura koja se koristi kod crtanja malog sedam segmentnog zaslona (Slika). Crvene brojke u kutu simulatora na crnoj podlozi zapravo se nalaze na pravokutniku u prostoru koji je uvijek na istom mjestu relativno na lokaciju kamere ukoliko se matrica perspektive ne mijenja (s promjenom kuta vidnog polja u .ini datoteci).

Slika 13 Sedam segmentni zaslon

Učitavanje Arduino C naredbi

Kao što je već spomenuto, simulator prima naredbe pomoću međuspremnika računala. Za pristupanje međuspremniku, koristi se biblioteka *windows.h*. Kod čitanja međuspremnika potrebno je samo provjeriti jesu li podaci u međuspremniku u tekstualnom formatu te ako jesu, dohvatiti te podatke uz zaključavanje pristupa međuspremniku (kako za vrijeme čitanja podataka ne bi došlo do promjene podataka).

Nakon kopiranja teksta iz međuspremnika, potrebno je filtrirati kroz tekst i pronaći dijelove u kojima se nalazi kod za robota. Nakon što se tekst filtrira, simulator može koristiti dobiveni kod na način da prolazi po kodu red po red, prepoznaje naredbu te ukoliko je implementirana, reagira na naredbu.

Na sljedećem primjeru moguće je dobiti uvid u proces interpretacije koda koji je korišten u simulatoru.

Spremnik s našim kodom (naredba po naredba u redovima) je spremljen u varijabli „carCode“. Indeks trenutne naredbe je zapisan u varijabli „currentLine“.

Na početku tražimo sadrži li trenutna naredba dio poznate naredbe „motor_9“, a znamo da se „motor_9“ zapisuje u Arduino načinu rada kada se koristi naredba postavljanja brzine motora.

Slika 14 Naredba kretanja u smjeru

```
commandLocation = std::string(carCode[currentLine]).find(std::string("motor_9"));
if(commandLocation != std::string::npos){
```

Ako je naredba prepoznata, nastavlja se sa učitavanjem argumenta.

```
        unsigned int colonLocation =
std::string(carCode[currentLine]).find(std::string(":"));
        unsigned int length = carCode[currentLine].end() - 2 -
(carCode[currentLine].begin() +                colonLocation + 2);
        std::istringstream
        speedString(std::string(carCode[currentLine]).substr(colonLocation + 2, length));
        float speed;
        speedString >> speed;
        if (speed > 255) speed = 255;
        if (speed < -255) speed = -255;
```

Nakon dohvaćanja argumenta, poziva se funkcija koja simulira ponašanje robota.

```
        m_car->SetRWheelSpeed(speed);
        currentLine++;
        return 0;
  }
```

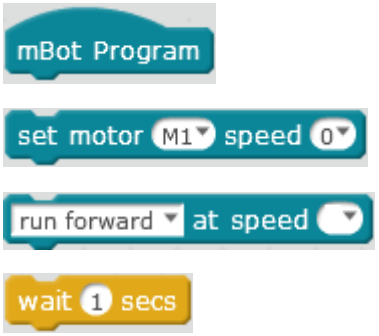
Ovdje je vidljivo da se kod pisanja algoritma pretpostavlja točno definiran način zapisivanja koda u programskom jeziku Arduino C. Kao referenca uzima se aplikacija mBlock 3 i način zapisivanja koda koji se koristi u toj aplikaciji kod Arduino načina rada. Ukoliko se način zapisivanja određenih naredbi promijeni, potrebno je prilagoditi dio algoritma koji upravlja čitanjem te naredbe u simulaciji.

Primjena u nastavi

Svrha samog projekta je korištenje simulacije u nastavi kako bi se povećao interes učenika za učenje programiranja te kako bi samo učenje bilo pristupačnije. U nastavku je opisano nekoliko primjera koje je moguće provesti pomoću simulatora. Ti primjeri nalaze se uz kod simulacije na https://github.com/ntrsten/mBot-simulator.

Primjer 1:

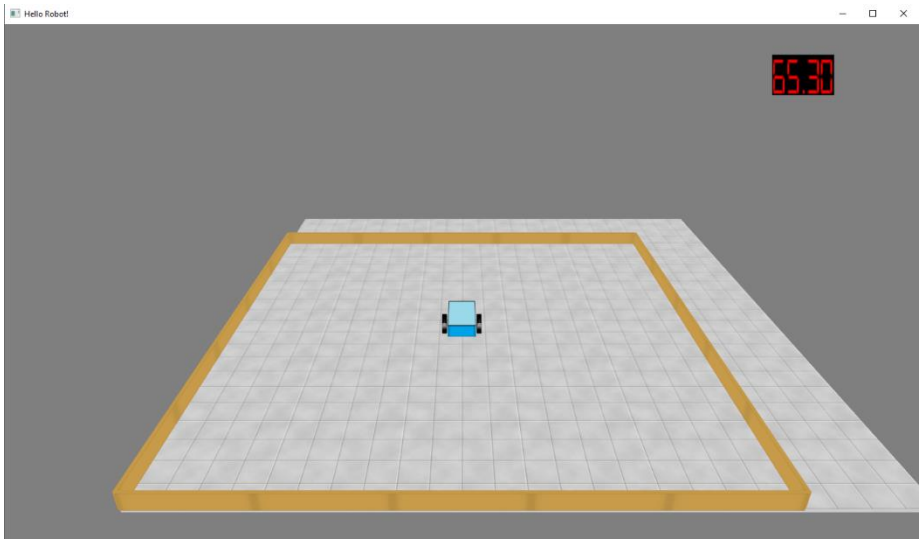
Korištenjem osnovnih naredbi u aplikaciji mBlock, učenici se mogu upoznati s radom simulatora te radom samog robota.

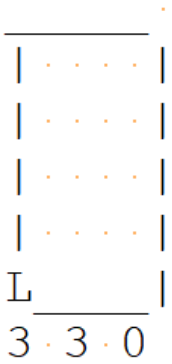


Korištenjem samo ovih naredbi moguće je zadati mnoge zadatke, kao što su promatranje utjecaja promjene varijabli na kretanje robota te opisivanje tih promjena ili pisanje programa tako da robot ima određenu putanju (geometrijski lik, krivulja, isprekidana crta, itd.) ili se okreće za određeni kut (npr. pravi kut).

Ako se koristi scena s zidovima, umjesto prazne scene moguće je promatrati poveznicu između lokacije robota te očitanja na sedam segmentnom zaslonu. Uz to, zidovi bi ograničili prostor za kretanje robota te bi u tom slučaju bilo potrebno planirati putanju

samog robota u odnosu na početnu lokaciju, isto kao i kod programiranja pravog robota.

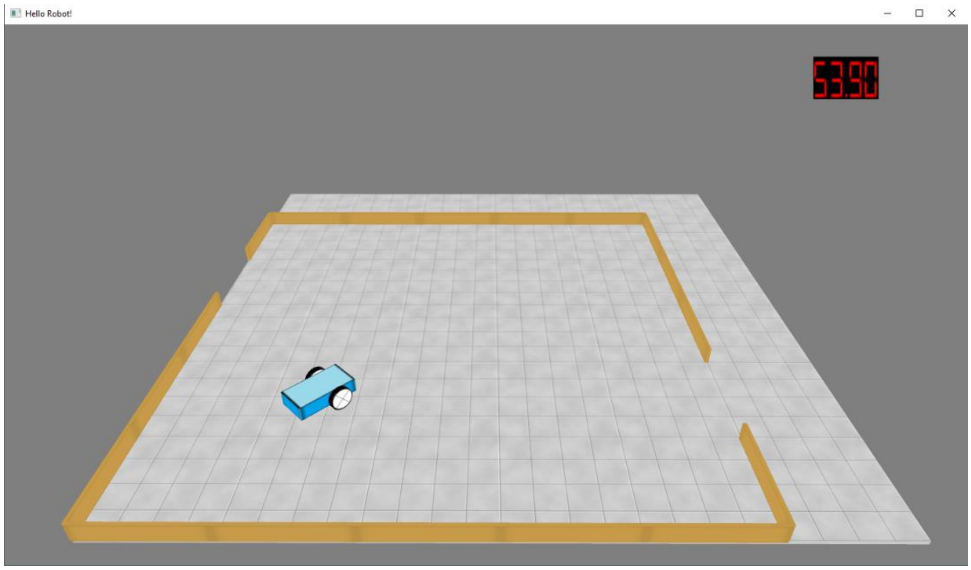


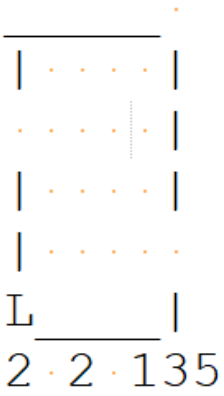


Primjer 2:

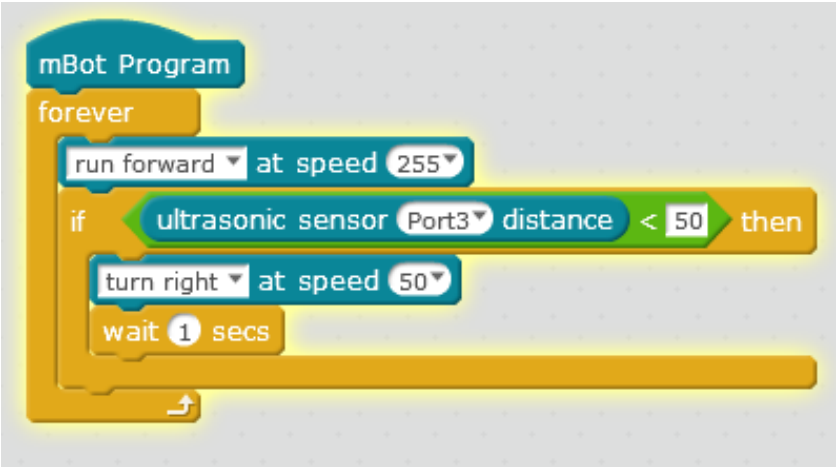
Doda li se korištenje beskonačne petlje te ultrazvučnog senzora, dobije se koncept samostalnog rada robota. Beskonačna petlja daje uvid u sam rad robota, a uz to i poveznicu sa svakim elektronskim uređajem. Bilo kakav elektronski uređaj ima niz naredbi koje se izvršavaju s ponavljanjem i bez prestanka, pa tako i robot mBot (u simulaciji i u pravom svijetu). Ultrazvučni senzor daje priliku za uvođenje koncepta komunikacije robota s vanjskim svijetom. Ovisno o očitanju senzora, moguće je prilagoditi kretanje robota.

Koristi li se scena s vratima u zidovima, moguće je postaviti zadatak implementacije samostalnog rada robota koji bi neovisno o početnoj poziciji pronašao izlaz iz kutije.





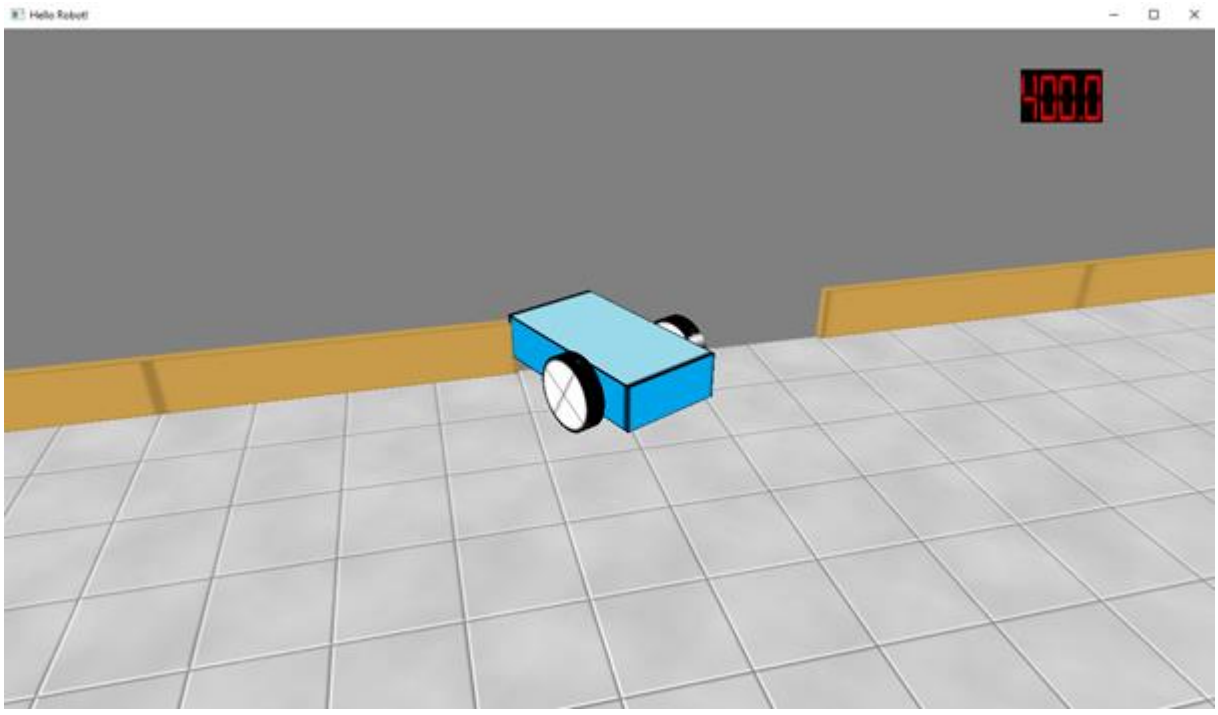
Kod koji se koristi u primjeru:



Ovo nije jedina kombinacija naredbi te je moguće prilagoditi vrijednosti varijabli koje se koriste u kodu.

Kroz ovaj primjer valja obratiti pažnju na nepreciznost senzora te mogućnost zapinjanja robota uz zid. Isto kao i u pravom svijetu, postoji mogućnost da robot ne prima pravilno očitanje udaljenosti od zida jer zid nije direktno ispred robota, već se robot prema zidu

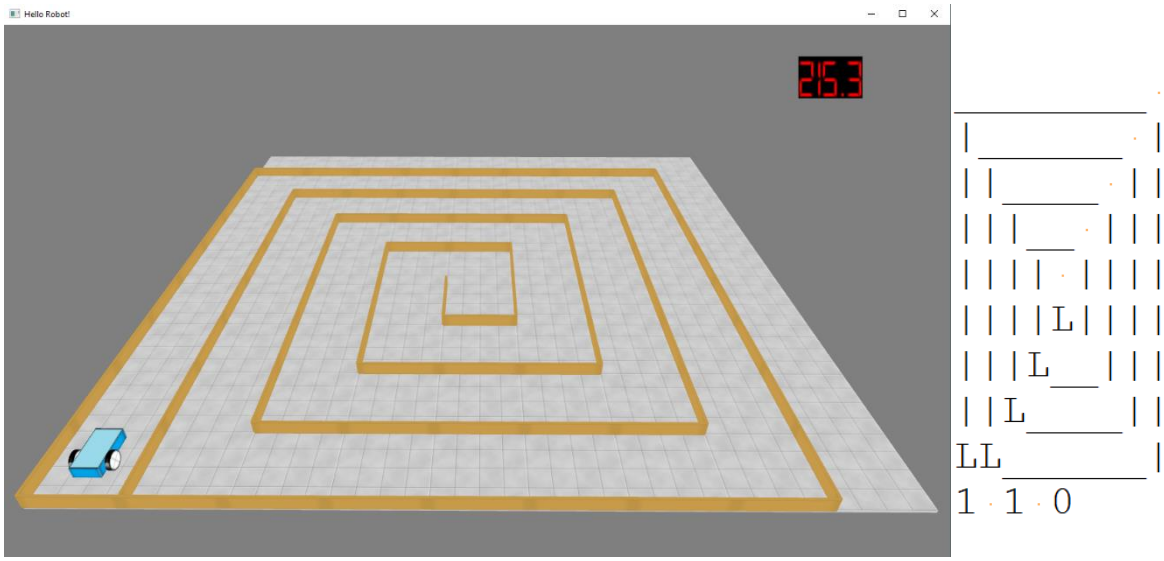
kreće pod određenim kutom. Kako senzori ne pokrivaju cijelu širinu robota, vrlo često se dogodi situacija da senzor ne očita zid koji je ispred kotača robota.



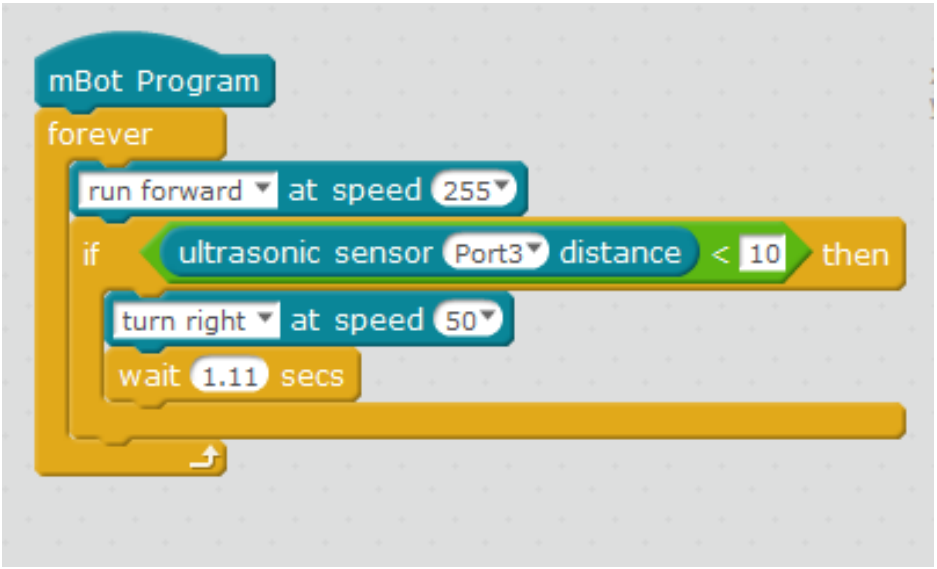
Slika 15 Zapinjanje robota

Primjer 3:

Koristeći iste naredbe, ali scenu zidovima u spirali moguće je zadati zadatak implementacije rada robota koji bi samostalno došao do sredine labirinta.



Kod koji se koristi u primjeru:



Analizom ovog primjera također možemo uočiti problem nepreciznost senzora. Senzor ne može razlikovati slučaj kada je zid s lijeve robota od slučaja kada je zid s desne strane robota. Isto tako ako je kut između putanje robota i zida premali, postoji mogućnost ranije opisanog bočnog zapinjanja robota uz zid.

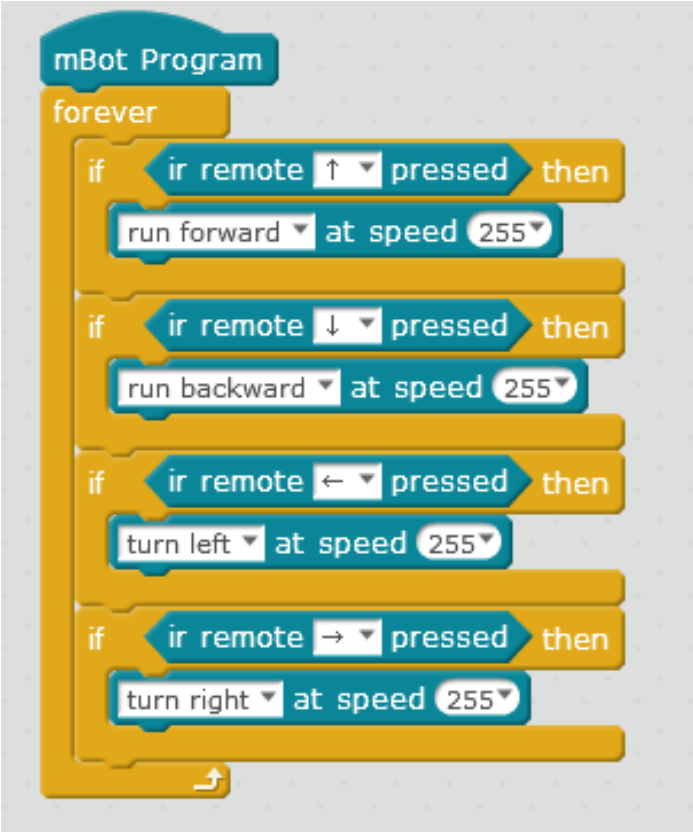
Uz to kroz ovaj primjer može se obratiti pozornost na metodu pokušaja i pogrešaka. Ako se ne znaju točne brzine kretanja kotača te točne dimenzije, vrlo je teško precizno

odrediti točne vrijednosti varijabli brzine okretanja te vremena čekanja (koliko dugo se okreće robot) kako bi se robot okrenuo za pravi kut. Te veličine određuju se samim isprobavanjem različitih vrijednosti te prilagođavanjem istih.

Primjer 4:

U simulatoru postoji mogućnost simuliranja unosa putem daljinskog infracrvenog upravljača.

Pomoću ove funkcionalnosti moguće je implementirati upravljanje robotom infracrvenim upravljačem. U simulatoru infracrveni upravljač zamjenjuje se tipkovnicom na računalu. Neovisno o sceni koristi se sljedeći kod:



Koristeći sličnu strukturu, moguće je mijenjati varijable u naredbama kako bi putanja robota kod skretanja opisivala krivulju čime bi se dobio dojam fluidnijeg kretanja robota (unutarnji kotač kod skretanja bi se vrtio sporije).

Kroz ove primjere moguće je metodom analogije objasniti učenicima da isto kao što je na računalu u međuspremniku u jednom trenutku moguće imati spremljenu samo jednu određenu stvar (u našem slučaju kod robota) tako i kod stvarnog robota mBot postoji samo jedan aktivni kod koji se izvršava.

Daljnji razvoj

Ovaj projekt izrađen je kao osnova za izradu simulacije robota mBot. U ovom projektu postoji puno potencijala za dodatan razvoj i implementaciju mnogih algoritama. Uz to postoji i puno prostora za optimizaciju trenutnog koda. Također, ovaj projekt služi i kao platforma za učenje programiranja (računalna grafika, izrada interaktivnih aplikacija, izrada simulacija).

Sav kod uz sve potrebne biblioteke nalazi se na https://github.com/ntrsten/mBot-simulator. Dovoljno je samo preuzeti kod spreman za kompiliranje. Uz to, na navedenoj adresi se nalazi i radna verzija aplikacije spremna za korištenje.

Kroz daljnji razvoj moguće bi bilo dodati mnoge stvari, neke od kojih su opisane u nastavku.

Dodavanje podrške za više naredbi. Petlja *for* trenutno nije implementirana, no moguće bi bilo jednostavnom rekurzijom (rekurzija zato što je potrebno podržati ugniježđene petlje) implementirati funkcionalnost te petlje. Isto tako valjalo bi dodati podršku za uvjetni izraz *if then else.*

Trenutno simulator ima podršku za po jedan format od određenog tipa datoteka. Ukoliko bi bilo potrebe za širenjem podrške, moguće je dodati vanjsku biblioteku ciljanu za širu upotrebu čitanja podataka iz različitih tipova datoteka.

Kao što je već spomenuto, detekcija sudara trenutno nije optimalna. U budućnosti valjalo bi implementirati algoritam koji kod provjere sudara promatra model u cijelosti. Uz to bi bilo potrebno i prilagoditi način opisivanja scene i zidova, npr. zapisivanje točnih lokacija zidova kako bi sam algoritam i način definiranja zidova bili neovisni (direktna povezanost algoritma s načinom zapisivanja podataka velika je mana algoritma).

Optimizacija rada ultrazvučnog senzora kako bi preciznije prikazao situaciju u stvarnom svijetu. Trenutno očitanje senzora ne ovisi o kutu koji zatvaraju zid i robot, ali u stvarnom svijetu zbog refleksije valova dolazi do nepreciznog očitanja kada je taj kut prevelik.

Moguće je dodati i ovisnost o trenutnom punjenju baterije robota, što bi utjecalo na preciznost i brzinu kretanja robota.

Iz svega navedenog možemo zaključiti da se radi o projektu u kojem je moguće implementirati mnogo različitih algoritama vezanih uz izradu simulacije. S jedne strane, korištenje simulatora omogućuje rad u redovnoj nastavi s većom skupinom učenika, posebno u ranijoj fazi učenja programiranja fizičkih objekata kada se učenici tek susreću s pojmovima motora, senzora i beskonačne petlje. Također, učenici analizom ponašanja

simulatora i robota u stvarnom svijetu otkrivaju različite faktore koji utječu na ponašanje robota. Postupnim svladavanjem više naredbi blokovskog programiranja učenici mogu procijeniti na koji bi se način simulator mogao proširiti tako da simulira rad pojedinih senzora. Zbog toga što se radi o programu otvorenog koda, napredniji učenici mogu i sami pokušati implementirati neku funkciju robota i na taj način stvoriti vlastitog robota.

Zaključak

Svrha ovog diplomskog rada bila je razviti simulator rada edukacijskog robota mBot koji podržava neke osnovne naredbe blokovskog programskog jezika mBlock, a koje su specifične za programiranje robota. Uspostavljen je sustav za opisivanje fizičkih karakteristika i oblika robota putem niza parametara, opisan je način za definiranje scene po kojoj se robot kreće. Implementirana su tri načina praćenja robota čime je učenicima omogućeno praćenje rada robota "iz prvog lica" što doprinosi boljem razumijevanju algoritama kojima se definira ponašanje robota. Sustav je oblikovan tako da ne postavlja prevelike zahtjeve na snagu računala, a također se je vodilo računa i o tome da se simulacija može pokrenuti bez instalacije i bez posebnog podešavanja parametara sustava.

Želja nam je ovaj sustav testirati u stvarnim uvjetima, u školama, te na temelju iskustava korištenje učenika i nastavnika dorađivati sustav tako da još bolje služi svojoj svrsi. Rad na razvoju ovog sustava dobra je vježba za korištenje aplikacijskog programskog sučelja OpenGL. S obzirom da je prva verzija simulatora razvijena i testirana samo na operacijskom sustavu Windows, s eventualnim uključivanjem novih razvojnih programera moguće bi ovaj sustav bilo implementirati i na drugim operacijskim sustavima, prvenstveno na operacijskom sustavu Linux.

Literatura

1. J. Gregory, Game Engine Architecture, Second Edition, A K Peters/CRC Press, 2014.
2. B. Stroustrup, The C++ Programming Language, 4th Edition, Addison-Wesley Professional, 2013.
3. GLEW: The OpenGL Extension Wrangler Library, http://glew.sourceforge.net/ ; s datumom 01.02.2019
4. Simple DirectMedia Layer - SDL version 2.0.9 (stable), https://www.libsdl.org/download-2.0.php ; s datumom 01.02.2019
5. OpenGL Mathematics, https://glm.g-truc.net/0.9.9/index.html ; s datumom 01.02.2019
6. Learn OpenGL, extensive tutorial resource for learning Modern OpenGL, https://learnopengl.com/ ; s datumom 01.02.2019
7. OpenGL - The Industry Standard for High Performance Graphics, https://www.opengl.org/ ; s datumom 01.02.2019
8. docs.gl (Dokumentacija za OpenGL), http://docs.gl/ ; s datumom 01.02.2019
9. A Tutorial and Elementary Trajectory Model for The Differential Steering System, http://rossum.sourceforge.net/papers/DiffSteer/ ; s datumom 01.02.2019.
10. Makeblock: Global STEAM Education Solution Provider, https://www.makeblock.com/ ; s datumom 01.02.2019

Sažetak

Kao dio, te kao cilj ovog diplomskog rada izrađena je računalna simulacija edukacijskog robota mBot. U ovom radu najprije je opisana motivacija same izrade te korišteni alati kod izrade simulacije. Nakon toga detaljno se opisuje simulacija i sve što je implementirano u okviru simulacije te kako se koristi simulacija i svi njezini dijelovi kao što su popratne datoteke. Opisani su i načini prilagođavanja rada simulatora. Uz to, u radu su opisani značajniji problemi s kojima se dolazi u susret kod razvoja simulacije robota. Na primjer određivanje pozicije robota (kao vozila na dva pogonska kotača) kod kretanja, izrada samog sučelja te prikazivanje slike na različite načine, detekcija sudara, itd.

Cilj rada je primjena simulatora u školama. Zato se u okviru rada opisuju načini na koje je moguće primijeniti simulator. Postavljeno je nekoliko primjera te je opisano što se provođenjem tih primjera postiže. Uz to su istaknuti razni problemi koji se javljaju kod programiranja edukacijskog robota mBot, ali i općenito kod programiranja fizičkih objekata.

U sklopu rada postoji platforma za daljnji razvoj simulacije te ideje i planovi za daljnji razvoj.

Summary

As a part of this graduate thesis, a computer simulation of the educational robot mBot was created. In this paper are described the motivation of the development itself and the tools used to develop the simulation. After that, the simulation, everything implemented within the simulation, as well as all its parts, such as the accompanying files are described in detail. Customisation options of the simulation are also described. In addition, the paper describes the more significant problems encountered in the development of robotic simulations. For example, determining the position of a robot (as two-wheel drive vehicles) when moving, creating the interface itself, and displaying images in different ways, collision detection, etc.

The aim of the paper is to apply simulators in schools. Therefore, within the scope of the paper, are described ways in which the simulator can be applied. Several examples have been set out and it is described what is achieved by implementing these examples. In addition, they highlight the various problems that arise when programming the educational robot mBot, but also in general when programming physical objects.

As part of the paper there is a platform for further development of the simulation, as well as ideas and plans for further development.

Životopis

Nikola Trstenjak rođen je 22. srpnja 1993. godine u Čakovcu. Završio je Osnovnu školu Gornji Mihaljevec. Nakon osnovne škole, upisuje prirodoslovno matematički smjer u Gimnaziji Josipa Slavenskog u Čakovcu. Na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu je završio preddiplomski sveučilišni studij Matematika; smjer: nastavnički, te nakon toga upisuje diplomski sveučilišni studij Matematika i informatika; smjer: nastavnički na istom fakultetu.