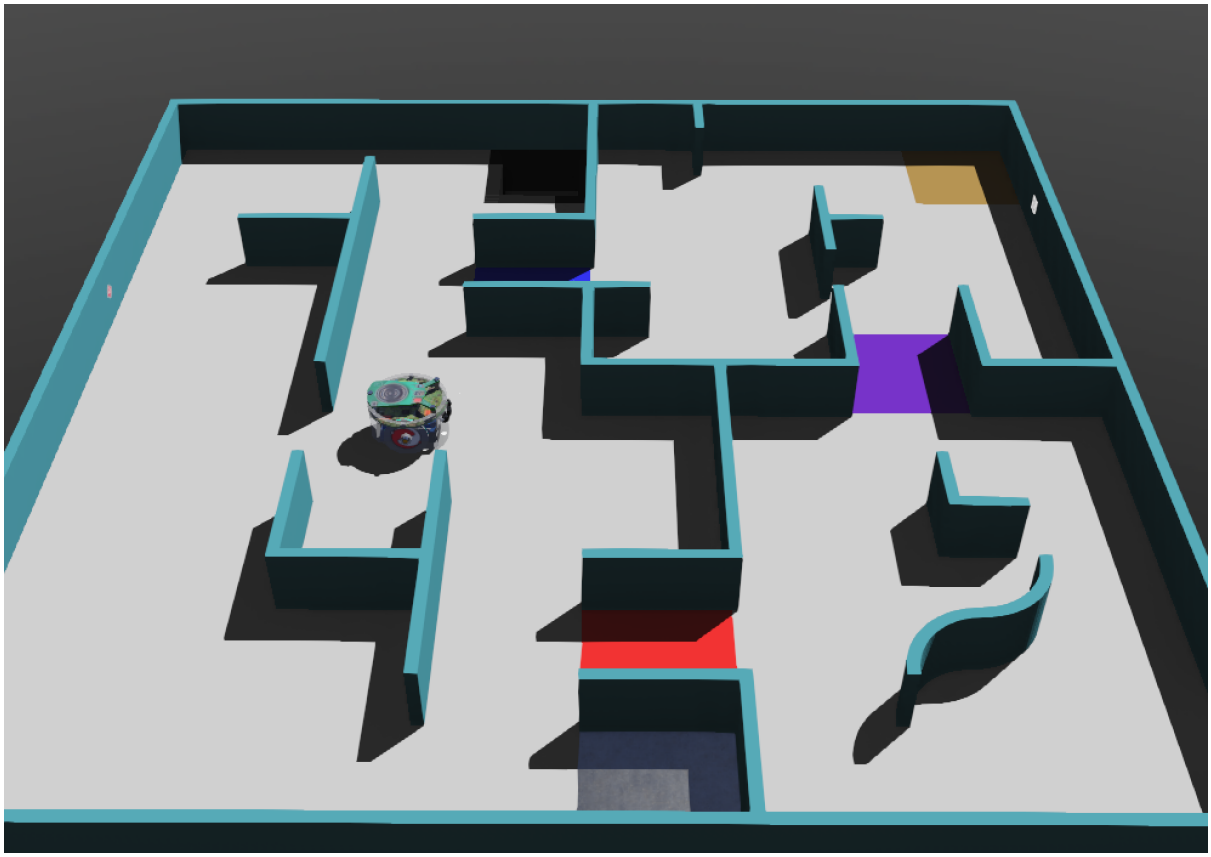


Rescue Maze Simulacija - Upute

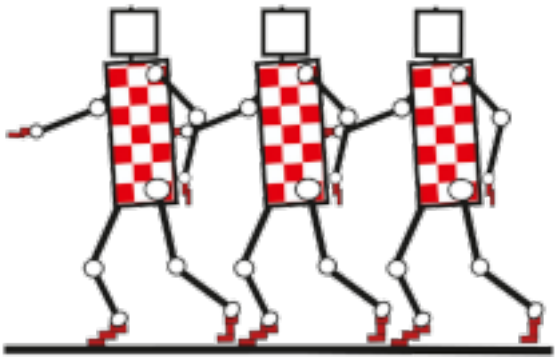
Martin Unetić

velj 01, 2022

Sadržaj

RoboCup Croatia https://robocupcroatia.com/

Hrvatski robotički savez https://hrobos.hr/



1 Uvod

1.1 Erebus Rescue Maze Simulator

Erebus je simulacija RoboCup Rescue Maze arene u kojoj natjecatelji moraju razviti aplikaciju za kontrolu robota koji će istražiti arenu koristeći dopuštene senzore, naći žrtve, te kreirati mapu arene. Pri tome robot mora izbjegavati prepreke i zamke.

Erebus je razvijen na Webots platformi, profesionalnom simulatoru za robote koji ima široku primjenu u industriji, školstvu i znanosti. Tvrtka Cyberbotics razvija Webots od 1998. godine kao svoj glavni proizvod. Na službenim stranicama dostupna je opsežna dokumentacija kao i brojni primjeri.

Erebus je razvijen za RoboCup Junior 2021, kada je predstavljen.

1.2 Korisni linkovi

Webots https://cyberbotics.com/doc/guide/index

Erebus https://erebus.rcj.cloud/docs/

2 Pravila

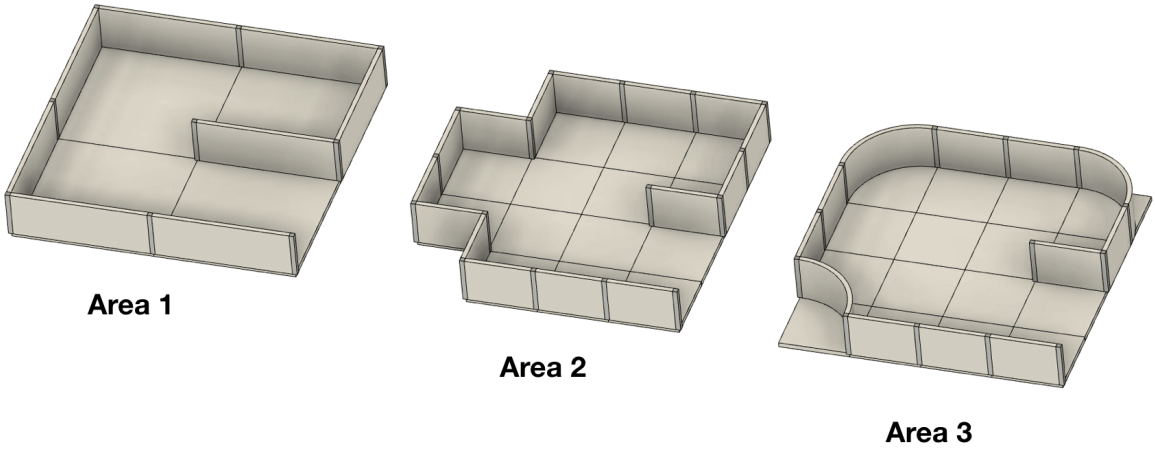
Službena pravila dostupna su na stranicama RoboCup Junior[1].

2.1 Arena

Arena se sastoji od polja veličine 12x12cm, te zidova debljine 1cm i visine 6cm.

Zone

Arena može biti podijeljena na tri zone.



U zoni 1:

- zidovi mogu biti samo na rubovima polja

U zoni 2:

- polja se dijele na četvrtine veličine 6x6cm
- zidovi mogu biti na rubovima čevrtina polja
- prolazi moraju biti širine cijelog polja

U zoni 3:

- polja se dijele na četvrtine veličine 6x6cm
- zidovi mogu biti na rubovima čevrtina polja
- prolazi moraju biti širine cijelog polja
- zidovi mogu biti zakrivljeni polumjera 6cm

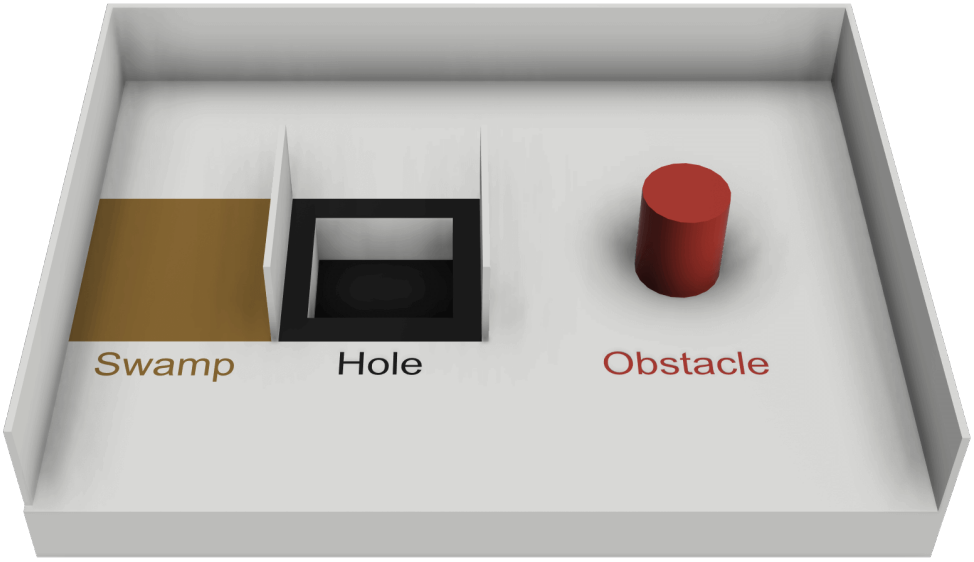
Zone su odijeljene prolazom određene boje:

- plava između zone 1 i 2
- ljubičasta između zona 1 i 3
- crvena između zona 2 i 3

[1] https://cdn.robocup.org/junior/wp/2021/06/2021_RescueSimulation_Rules_final02.pdf

Prepreke

Arena može sadržavati močvare, zamke i prepreke.



Močvare su polja smeđe boje na kojima je kretanje robota usporeno.

Zamke su rupe sa crnim rubom u koje robot upada te se vraća na početnu poziciju ili kontrolnu točku.

Prepreke mogu biti bilo koje boje ili oblika, moraju biti barem 8cm udaljene od zidova.

Žrtve i opasnosti

Žrtve su označene slovima na zidu.

- H - ozlijeđena žrtva
- S - stabilna žrtva
- U - neozlijeđena žrtva

Opasnosti su označene znakovima na zidu.

- zapaljivo (F)
- otrov (P)
- nagrizajuće (C)

- organski peroksid (O)

Lack of Progress

Kada se dogodi da rogot ne napreduje, biti će premješten na početak ili na posljednju kontrolnu točku.

- kada upadne u zamku (automatski)
- kada se 20 sekundi ne pomiče (automatski)
- kada se robot pomiče ali ponavlja iste kretnje (ručno)
- kada kontrolna aplikacija sama zatraži

Mapiranje

Kao dodatno bodovanje robot tijekom obilaska arene može kreirati mapu arene. Nakon obilaska arene robot mapu dostavlja nadzornoj aplikaciji koja ju uspoređuje sa ispravnom.

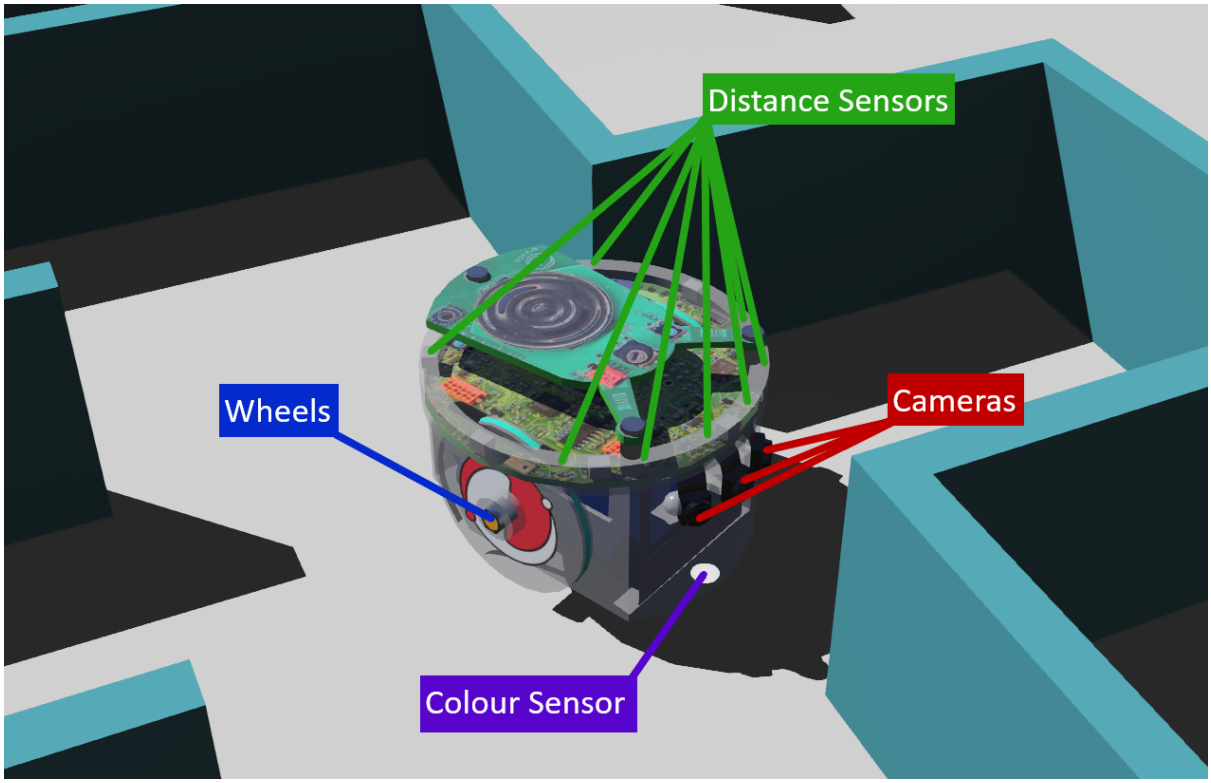
Povratak

Robot koji se po zarvšetku obilaska arene vrati na početno polje dobija dodatne bodove.

2.2 Robot

Osnovni robot

Osnovni robot ima sljedeće senzore: - osam senzora udaljenosti - tri kamere - color senzor - gyroskop - akcelerometar - gps



Modoficirani roboti

Za modifikacije robota koristi se Erebus Robot Customisation 2021:

https://robot.erebus.rcj.cloud/

Kada se koristi modificirani robot, prije simulacije potrebno je učitati JSON datoteku sa opisom robota.

Robot može koristiti: - 8x senzor udaljenosti - 3x kamere - 1x color senzor - 1x gyroskop - 1x akcelerometar - 1x gps - 1x lidar

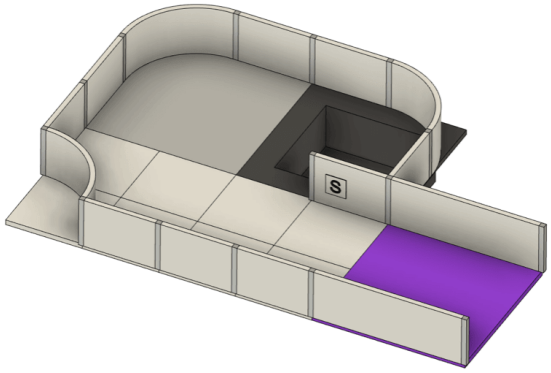
Svaki senzor ima određenu cijenu, ukupna cijena je ograničena.

2.3 Mapa

Robot koji tijekom obilaska arene napravi mapu arene može osvojiti dodatne bodove. Robot mapu dostavlja nadzornoj aplikaciji na kraju obilaska arene.

Svaka četvrtina polja, kao i zidovi oko nje, predstavljeni su jednom ćelijom u tablici.

Element	Oznaka
zidovi	1
zamka	2
močvara	3
kontrolna točka	4
početna točka	5
prolaz zone 1 i 2	6
prolaz zone 1 i 3	7
prolaz zone 2 i 3	8
ozlijeđena žrtva	H
stabilna žrtva	S
neozlijeđena žrtva	U
zapaljivo	F
otrov	P
nagrizajuće	C
organski peroksid	O
ostalo	0



$$\rightleftarrows \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 0 & 4 & 0 & 2 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 4 & 0 & 4 & 0 & 2 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & S & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 8 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 8 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

3 Instalacija

Platforma se sastoji od tri komponente: - Webots - Erebus - Python

Erebus je razvijen na Webots verziji R2021a, prilikom instalacije potrebno je paziti na verziju, na novijim verzijama ne radi ispravno.

U trenutku pisanja dokumenta aktualna verzija Erebus je v21.2.2

3.1 Zahtjevi

- OS-Linux: Ubuntu 18.04 LTS, 20.04 LTS 64bit
- OS-Windows: Win 10, 8.1 64 bit
- OS-Mac: macOS 10.14 ‘Mojave’ or newer (not tested on Big Sur)
- CPU: Dual Core 2 GHz
- RAM: 2 GB
- Nvidia 512MB or AMD OpenGL 512MB

3.2 Windows

Instalacija

- instalirati Python 3.9 64bit[2]
- instalirati Webots R2021a[3]
- preuzeti i raspakirati Erebus v21.2.2[4]

Pokretanje

- pokrenuti Webots
- otvoriti datuteku Erebus-v21_2_2/game/worlds/world1.wbt

Problemi

- ne instalira numpy, termcolor, requests
- ne učitava kontroler
- spora simulacija

3.3 Mac

TODO

3.4 Linux

Testirano sa Ubuntu 20.04 LTS 64bit

Instalacija

Instalirati Python 3

```
sudo apt-get install python3 python3-pip
sudo apt-get install python3-numpy python3-termcolor python3-requests
```

Instalirati Webots R2021a

```
wget https://github.com/cyberbotics/webots/releases/download/R2021a/webots_2021a_
↪amd64.deb

sudo apt install ./webots_2021a_amd64.deb
```

Preuzeti i raspakirati Erebus v21.2.2

```
wget https://gitlab.com/api/v4/projects/22054848/packages/generic/erebus/v21.2.2/
↪Erebus-v21_2_2.zip

unzip ./Erebus-v21_2_2.zip
```

[2] https://www.python.org/ftp/python/3.9.4/python-3.9.4-amd64.exe
[3] https://github.com/cyberbotics/webots/releases/download/R2021a/webots-R2021a_setup.exe
[4] https://gitlab.com/api/v4/projects/22054848/packages/generic/erebus/v21.2.2/Erebus-v21_2_2.zip

Pokretanje

```
webots Erebus-v21_2_2/game/worlds/world1.wbt
```

Problemi

TODO

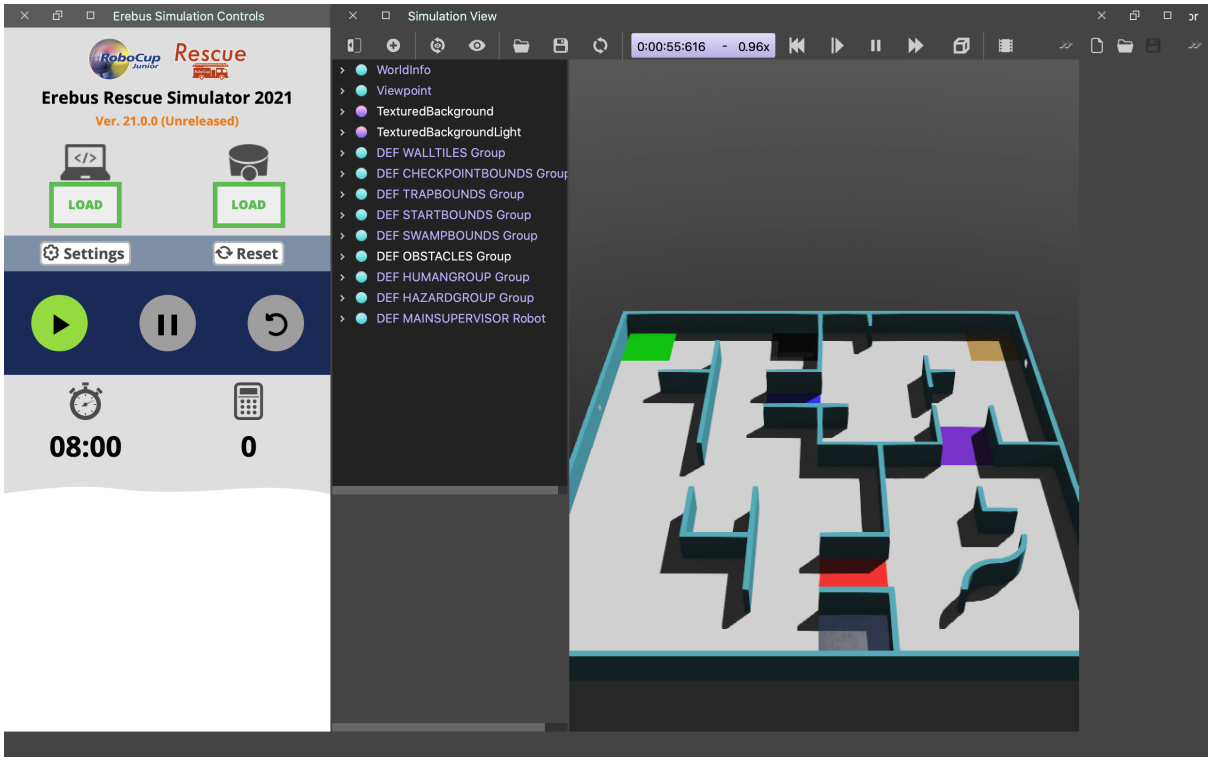
- spora simulacija

4 Primjeri

4.1 Prvi koraci

Kako otvoriti simulaciju i učitati primjer kontrolera.

Pokrenite Webots i učitajte datoteku Erebus-v21_2_2/game/worlds/world1.wbt



S lijeve strane ekrana nalaze se kontrole za Erebus simulaciju:

- u gornjem dijelu su dvije tipke LOAD, prva je učitavanje kontrolera za robota, druga je učitavanje opisa robota ako koristimo modificiranog robota
- u sredini su tipke za pokretanje simulacije, prekid simulacije i za vraćanje robota na početak (lack of progres)
- u donjem dijelu je status simulacije, uključujući vrijeme, bodove i log

Pritisnite prvu tipku LOAD i učitajte primjer kontrolera Erebus-v21_2_2/player_controllers/ExamplePlayerController_updated.py

Kada se kontroler ili opis robota učitaju, tipka mijenja boju iz zelene u narančastu.

Pokrenite simulaciju pritiskom na tipku za početak (prva tipka u srednjem dijelo kontrola).

Kada pokrenete simulaciju, robot se postavlja na početno polje, te se počinje kretati arenom.

4.2 Kretanje

Osnove

Kao jedinicu mjere, API koristi radijane.

```
kut_u_radijanima = kut_u_stupnjevima * PI / 180
kut_u_stupnjevima = kut_u_radijanima * 180 / PI
```

Kako biste koristili motore, potrebno je:

- kreirati objekt za svaki motor koji koristimo
- postaviti položaj motora na beskonačno, kako motori ne bi stali
- postaviti željenu kutnu brzinu motora u radijanima u sekundi

```
left_motor = robot.getDevice('wheel1 motor')
left_motor.setPosition(float('inf'))
left_motor.setVelocity(5.0)
```

Praćenje položaja motora

Motori imaju senzore koji prate položaj motora. Za korišenje je potrebno:

- kreirati objekt za svaki motor koji pratimo
- uključiti senzor, parametar je broj milisekundi između dva očitanja, parametar kora biti višekratnik osnovnog vremena osvježavanja simulacije
- očitati vrijednost u radijanima

```
left_encoder = left_motor.getPositionSensor()
left_encoder.enable(TIMESTEP)
value = left_encoder.getValue()
```

Primjer

Primjer kontrolera koje će pomaknuti robota za jedan okret kotača i stati.

```
from controller import Robot

robot = Robot()
TIMESTEP = int(robot.getBasicTimeStep())

left_motor = robot.getDevice("wheel1 motor")
right_motor = robot.getDevice("wheel2 motor")
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))

left_encoder = left_motor.getPositionSensor()
right_encoder = right_motor.getPositionSensor()
left_encoder.enable(TIMESTEP)
right_encoder.enable(TIMESTEP)

left_motor.setVelocity(5.0)
right_motor.setVelocity(5.0)

while robot.step(TIMESTEP) != -1:
    if left_encoder.getValue() > 6.28:
        break
```

(continues on next page)

(nastavak sa prethodne stranice)

```
left_motor.setVelocity(0.0)
right_motor.setVelocity(0.0)
```

4.3 Senzori

Svim senzorima zajedničko je da se prije optrebe moraju uključiti, pri čemu se određuje vrijeme osvježavanja senzora. Vrijeme osvježavanja senzora mora biti višekratnik osnovnog vremena osvježavanja simulcije.

Kako bi koristili senzore, potrebno je:

- kreirati objekt za svaki senzor koji koristimo
- uključiti senzor, parametar je broj milisekundi između dva očitanja
- očitati vrijednost senzora

Senzor udaljenosti

Senzor udaljenosti mjeri udaljenost u rasponu od 0 do 0.8 metara. Vrijednost koju vraćaju je udaljenost to prve prepreke.

```
from controller import Robot

robot = Robot()
TIMESTEP = int(robot.getBasicTimeStep())

distance_sensor = robot.getDevice("ps0")
distance_sensor.enable(TIMESTEP)

while robot.step(TIMESTEP) != -1:
    distance = distance_sensor.getValue()
    print("Distance: " + str(distance))
```

GPS

Lokacijski senzor, ili GPS, vraća trenutni položaj robota u areni. Vrijednost koju vraća je položaj u tri dimenzije u metrima.

Napomena: U Erebusu osi X i Z predstavljaju vodoravnu ravninu, dok je Y okomica.

```
from controller import Robot

robot = Robot()
TIMESTEP = int(robot.getBasicTimeStep())

gps = robot.getDevice("gps")
gps.enable(TIMESTEP)

while robot.step(TIMESTEP) != -1:
    location = gps.getValues()
    x = location[0]
    y = location[1]
    z = location[2]
    print("x: " + str(x) + " y: " + str(y) + " z: " + str(z))
```

Kamera

Kamera vraća sliku kao trodimenzionalnu matricu, koja predstavlja dvodimenzionalnu sliku sastavljenu od piksela, a svaki priksel ima tri komponente, crvenu, zelenu i plavu. Vrijednost komponenata je između 0 i 255, pri čemu je (0, 0, 0) crno, a (255, 255, 255) bijelo.

Erebus kamere koristi na dva načina:

- kao senzor boje (slika veličine 1x1 piksel)
- kao kameru (slika veličine 52x39 piksela)

Sliku dohvaćamo metodom getImage:

```
camera = robot.getDevice('camera')
camera.enable(TIMESTEP)

image = camera.getImage()
```

Veličinu slike koju kamera vraća možemo provjeriti metodama getWidth i getHeight:

```
width = camera.get_width()
height = camera.get_height()
```

Komponente pojedinoh piksela na slici možemo dohvatiti metodama imageGetRed, imageGetGreen i imageGetBlue:

```
red = camera.imageGetRed(image, width, x, y)
green = camera.imageGetGreen(image, width, x, y)
blue = camera.imageGetBlue(image, width, x, y)
```

Senzor boje je kamera veličine slike 1x1 piksel:

```
from controller import Robot, Camera

robot = Robot()
TIMESTEP = int(robot.getBasicTimeStep())

color_sensor = robot.getDevice("colour_sensor")
color_sensor.enable(timestep)

while robot.step(timestep) != -1:

    image = color_sensor.getImage()

    r = colorSensor.imageGetRed(image, 1, 0, 0)
    g = colorSensor.imageGetGreen(image, 1, 0, 0)
    b = colorSensor.imageGetBlue(image, 1, 0, 0)

    print("r: " + str(r) + " g: " + str(g) + " b: " + str(b))
```

LIDAR

TODO

4.4 Komunikacija

Za komunikaciju između nadzorne aplikacije i našeg robota koriste se eiter i receiver objekti.

Emiter koristimo kako bismo:

- prijavili položaj žrtava
- prijavili položaj opasnosti
- prijavili kraj obilaska
- tražili povratak na početak ili kontrolnu točku

Receiver koristimo kako bismo znali kada nas je nadzorna aplikacija vratila na početak ili kontrolnu točku.

Receiver, kao i senzore, potrebno je uključiti, dok je emiter uključen automatski.

```
from controller import Robot

robot = Robot()
TIMESTEP = int(robot.getBasicTimeStep())

receiver = robot.getDevice("receiver")
receiver.enable(TIMESTEP)

emitter = robot.getDevice("emitter")

while robot.step(timestep) != -1:
    pass
```

Prijava žrtava

Bodovi se osvajaju prijavljivanjem žrtava i opasnosti.

```
from controller import Robot
import struct

robot = Robot()
TIMESTEP = int(robot.getBasicTimeStep())

camera = robot.getDevice('camera')
camera.enable(TIMESTEP)

gps = robot.getDevice("gps")
gps.enable(TIMESTEP)

emitter = robot.getDevice("emitter")

def check_victim():
    # do some magic with camera
    return 'H'

def report_victim(victim_type):
    # report victim to supervisor
    # data format:
    # - for bytes of robot x position in cm
    # - for bytes of robot z position in cm
    # - one byte victim type (H, S, U, T, F, P, C, O)
    # robot should stop for one second before reporting victims
    x = int(gps.getValues()[0] * 100)
    z = int(gps.getValues()[2] * 100)
    message = struct.pack("i i c", x, z, victim_type.encode())
```

(continues on next page)

(nastavak sa prethodne stranice)

```
    emitter.send(message)

while robot.step(TIMESTEP) != -1:
    victim_type = check_victim()
    if victim_type is not None:
        report_victim(victim_type)
```

Kraj obilaska

Kraj obilaska prijavljuje se slanjem znaka „E” nadzornoj aplikaciji.

```
emitter.send(b'E')
```

Lack of Progress

Receiver možemo koristiti kako bi kontroler znao da ga je nadzorna aplikacija vratila na početak.

```
if receiver.getQueueLength() > 0:
    data = receiver.getData()
    tup = struct.unpack('c', data)
    if tup[0].decode("utf-8") == 'L':
        print("Detected Lack of Progress!")
    receiver.nextPacket()
```

Mapiranje

TODO