به نام خدا

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

اصول علم ربات

تمرین سری اول

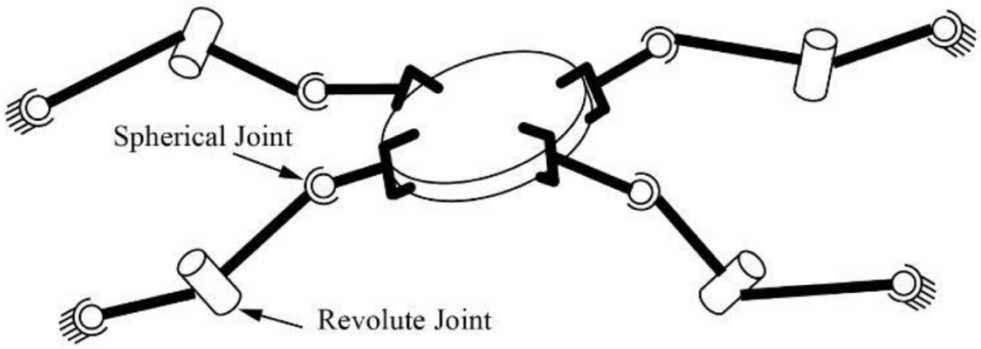
نام و نام خانوادگی	سارا تاجرنیا
شماره دانشجویی	۹۸۳۱۰۱۶
تاریخ ارسال گزارش	۱۴۰۱ /۱ /۶

فهرست گزارش سوالات

بخش عملی

سوال 1 – درجه آزادی ربات

۱. ربات زیر از ۴ بازوی SRS تشکیل شده است که یک دیسک را نگه داشته‌اند. درجه آزادی آن را به دست آورید. (راهنمایی: چهار مفصل واقع شده در چهار انتهای ربات به زمین متصل شده‌اند.) (بارم : ۵ امتیاز)



- Joint = 12 : تعداد joint ها 12 تاست که 4 تای آنها از نوع Revolute joint هستند که هر یک 1 درجه آزادی دارند و 8 تای دیگر از نوع Spherical joint هستند که هر یک 3 درجه آزادی دارند.
- Link = 10 : یکی برای بدنه ربات یکی برای زمین و 8 تا برای پایه های (link های چسبیده به ربات را در نظر نمیگیریم زبرا مستقل از ربات نیستند.)
- M = 6 : ربات به صورت 3 بعدی حرکت میکند و ماکسیمم فضا را میتواند در بر گیرد.

با توجه به فرمول درجه آزادی داریم:

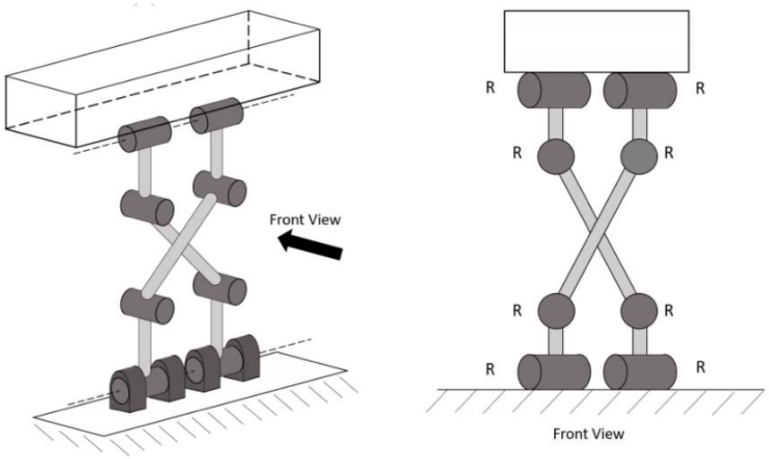
$$\text{dof} = \underbrace{m(N-1)}_{\text{rigid body freedoms}} - \underbrace{\sum_{i=1}^{J} c_i}_{\text{joint constraints}} = m(N-1) - \sum_{i=1}^{J}(m - f_i) = m(N-1-J) + \sum_{i=1}^{J} f_i.$$

➔ Dof = 6*(10 – 1- 12) + (8*3) + (4*1) = -18 + 24 + 4 = 10

مقدار درجه آزادی این ربات برابر 10 است.

سوال ۲ – درجه آزادی ربات

۲. برای ربات زیر درجه آزادی را به دست آورید. (تصویر، ربات را از دو نما نشان می دهد)(بارم : ۵ امتیاز)



- Joint = 8 : که همه ی آنها از نوع Revolute joint هستند که هر یک 1 درجه آزادی.
- Link = 8 : یکی برای بدنه ربات یکی برای زمین و 6 تا برای پایه ها.
- M = 6 : ربات به صورت 3 بعدی حرکت میکند و ماکسیمم فضا را میتواند در بر گیرد.

با توجه به فرمول درجه آزادی داریم:

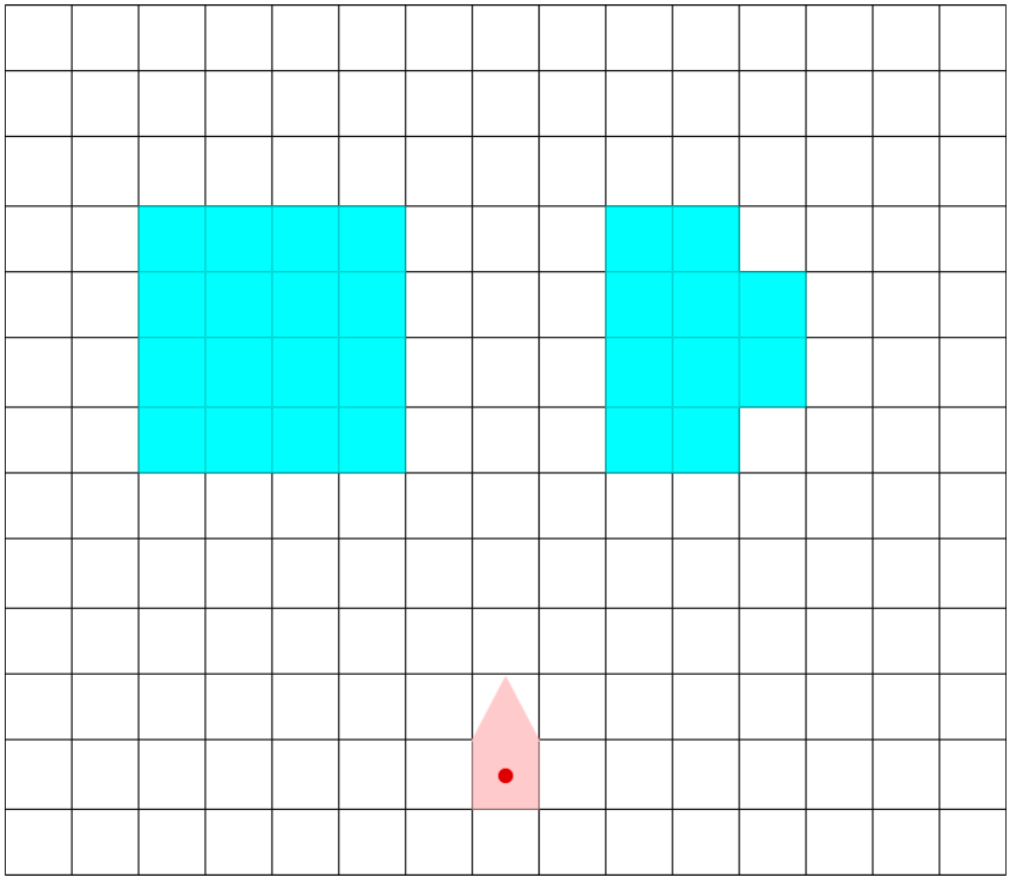
$$\text{dof} = \underbrace{m(N-1)}_{\text{rigid body freedoms}} - \underbrace{\sum_{i=1}^{J} c_i}_{\text{joint constraints}} = m(N-1) - \sum_{i=1}^{J}(m - f_i) = m(N-1-J) + \sum_{i=1}^{J} f_i.$$

➔ Dof = 6*(8 – 1- 8) + (8*1) = -6 + 8 = 2

مقدار درجه آزادی این ربات برابر 2 است.

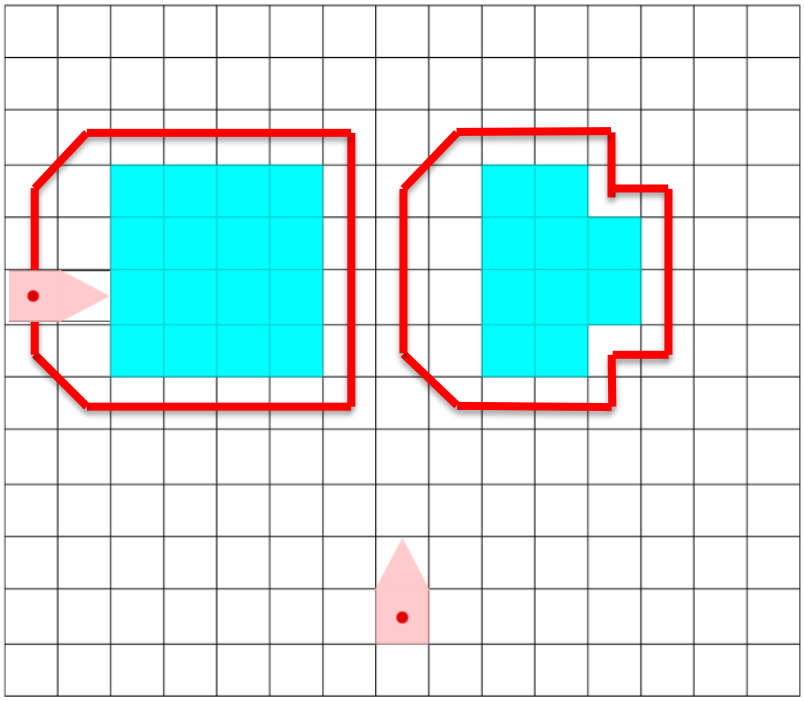
سوال 3 – محاسبه C_space

۳. برای ربات مشخص شده در تصویر زیر، C-space را مشخص کنید. (فرض کنید که ربات تنها امکان دوران ۹۰ درجه را دارد) (بارم: ۱۰ امتیاز)

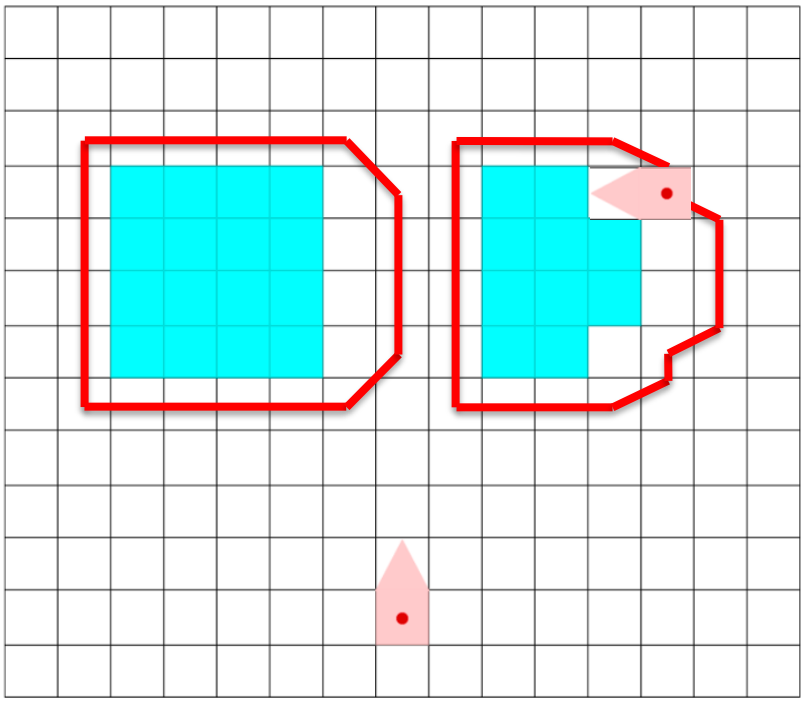


از آنجایی که ربات تنها امکان دوران 90 درجه را دارد پس تنها میتواند به صورت عمود بر سطح موانع درون شکل یا موازی با آنها باشد در نتیجه مرکز ربات (نقطه قرمز) که همیشه در وسط یک خانه است در کمترین حالت ممکن در فاصله 0.5 از مانع قرار دارد. پس میتوان گفت c-space به صورت نشان داده شده است.(این برای حالت چهارم است در شکل های زیر است.)

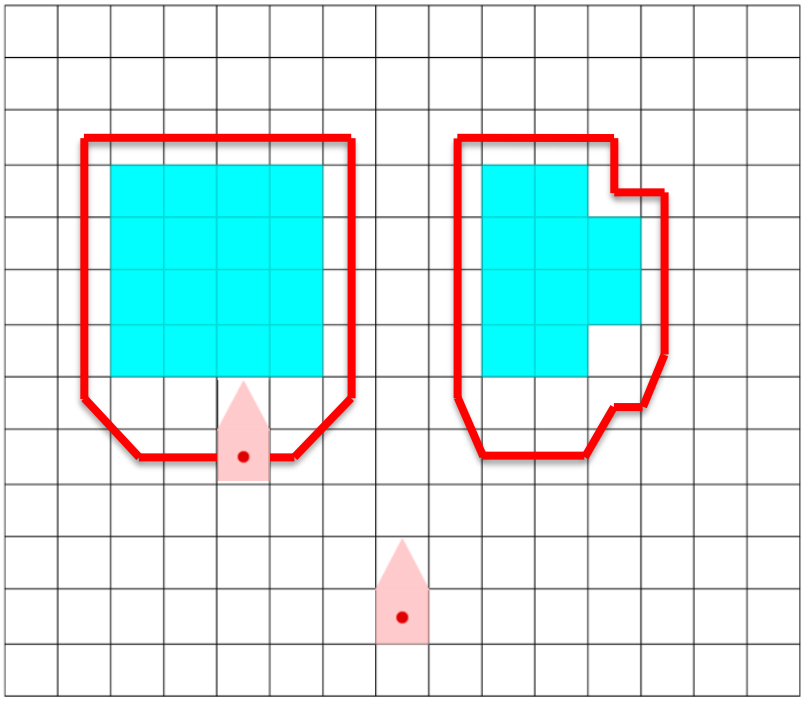
در صورتی که دوران ۹۰ درجه به راست داشته باشد:



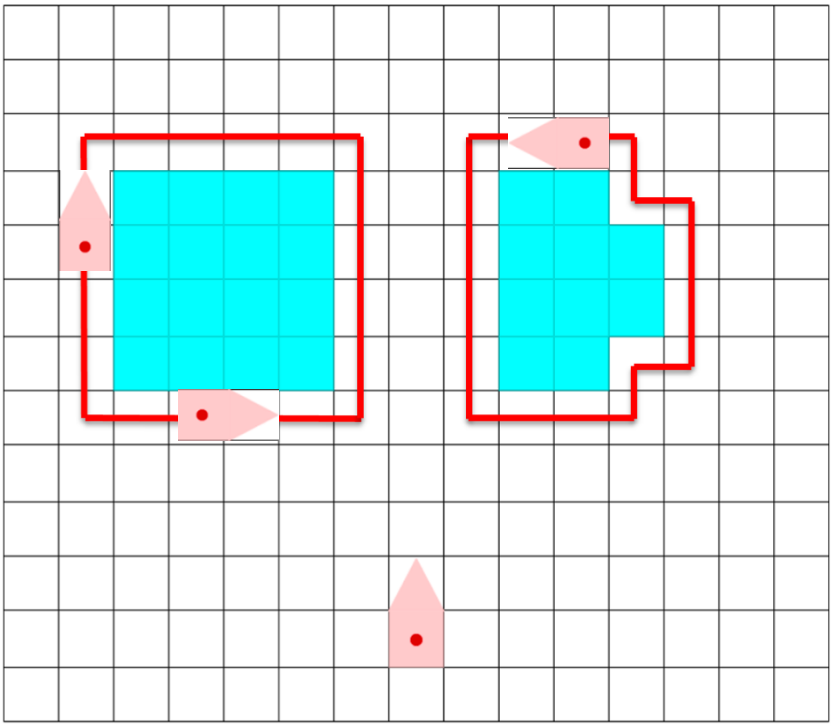
در صورتی که دوران ۹۰ درجه به چپ داشته باشد:



در صورتی که فقط به طور صاف حرکت کند و هر جایی نچرخد:



در صورتی که ربات در هر جا بتواند ۹۰ درجه به هر جهتی بچرخد:



بخش عملی(گام اول)

student_request

گام اول (۵۵ امتیاز) : لازم ا ست که فایل random_student.py را دریافت کرده و با کمک آنها شروع به تولید داده در گره student_request کنید و در تاپیک std_request قرار دهید. دقت کنید همانطور که در ویدیو تدریســـیاری اول توضیح داده شده ا ست، پس از ساخت پکیج مورد نظر برای گام اول، فایل ramdom_student.py را به عنوان یک گره وارد پکیج ساخته شده بکنید و custom message مورد نظر خود را با توجه به مراحلی که در ویدیو توضیح داده شده، بسازید. فیلد های این message را با توجه به کد زده شده در فایل random_student.py تعریف نمایید.

message تعریف شده توسط تاپیک std_request به گره splitter منتقل می‌شود. وظیفه این گره این است که با توجه به فیلد departement، دانشـــجویان را به دو گروه software و hardware تقســـیم کرده و در دو تاپیک software یا hardware آن ها را publish کند.

در انتها نیز گره‌ها با نام‌های software و hardware وظیفه چاپ اطلاعات دریافتی را دارند.

ابتدای کار پس از نصب برنامه، در کنار فایل src یک فایل به عنوان msg میسازیم تا custom message را به صورت زیر در آن ذخیره کنیم.

```
string name
int8 age
string last_name
string departement
```

حال در فایل random_student.py تابع student_request را پیاده سازی میکنیم تا student هایی با تمام مشخصات را به صورت رندوم تولید کند.

```
def student_request():
        '''Node Talker'''
        student = Stu()
        pub = rospy.Publisher('students', Stu, queue_size=10)
        rospy.init_node('random_student', anonymous=True)
        rate = rospy.Rate(1) # 1hz -> 1sec

        while not rospy.is_shutdown():
                '''for i in range(3):'''
                student = randStudent()
                pub.publish(student)
                rate.sleep()

```

splitter

سپس تابع splitter.py را طراحی کرده که در این کلاس دانشجویان را از کلاس random_student میگیرد و بر اساس department شان که hardware است یا hardware آنها را در دو گروه pub_hard و pub_soft،publish میکند.

*splitter.py

```
#!/usr/bin/python3

from rand_stu.msg import Stu
from random import randint, seed, choice
from time import time
import rospy

pub_soft = rospy.Publisher('Software', Stu, queue_size=10)
pub_hard = rospy.Publisher('Hardware', Stu, queue_size=10)


def classify(stu):
        if(stu.departement == "Software"):
                pub_soft.publish(stu)

        if(stu.departement == "Hardware"):
                pub_hard.publish(stu)


def split():
        rospy.init_node('splitter', anonymous=True)
        rospy.Subscriber("students", Stu, classify)
        rospy.loginfo('splitting')
        rospy.spin()

if __name__ == '__main__':
        try:
                split()
        except rospy.ROSInterruptException:
                pass
```

Hardware/software

سپس کلاس های hardware.py و software.py را طراحی میکنیم که در آنها اطلاعات دانشجویانی که departement آنها به کلاس مربوط است را میگیرد و آنها را چاپ میکند.

Hardware:

hardware.py

```
#!/usr/bin/python3

import rospy
from rand_stu.msg import Stu


def printHardwares(stu):
        rospy.loginfo('\ninfo studente of hardware departement: \n name: \t\t\t{}\n last name: \t\t{}\n age: \t\t\t{}\n departement:\t\t{}\
                \n'.format(stu.name, stu.last_name, stu.age, stu.departement))


def hardwares():
        rospy.init_node('hardware', anonymous=True)
        rospy.Subscriber("Hardware", Stu, printHardwares)
        rospy.loginfo('spliting hardware...')
        rospy.spin()

if __name__ == '__main__':
        try:
                hardwares()
        except rospy.ROSInterruptException:
                pass
```

Software:

*software.py
~/Desktop/catkin_ws1/src/rand_stu/src

```
#!/usr/bin/python3

import rospy
from rand_stu.msg import Stu


def printSoftwares(stu):
        rospy.loginfo('\ninfo studente of hardware departement: \n name: \t\t\t{}\n last name: \t\t{}\n age: \t\t\t{}\n departement:\t\t{}\
                \n'.format(stu.name, stu.last_name, stu.age, stu.departement))


def softwares():
        rospy.init_node('software', anonymous=True)
        rospy.Subscriber("Software", Stu, printSoftwares)
        rospy.loginfo('spliting software...')
        rospy.spin()

if __name__ == '__main__':
        try:
                softwares()
        except rospy.ROSInterruptException:
                pass
```

اجرای برنامه

برای اجرای برنامه ابتدا باید master را اجرا کنیم تا گره ها از طریق آن به هم متصل شوند.

```
sara@ubuntu:~/Desktop/catkin_ws1$ . devel/setup.bash
sara@ubuntu:~/Desktop/catkin_ws1$ roscore
... logging to /home/sara/.ros/log/02941b90-ae5d-11ec-89dd-71b668ee30ab/roslaunc
h-ubuntu-10166.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:36869/
ros_comm version 1.15.14


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [10174]
ROS_MASTER_URI=http://ubuntu:11311/
```

سپس هر یک از 4 فایل پایتون هر یک با اسم خود در دستورات زیر در terminal های متفاوت اجرا میکنیم:

Random_student	- 1
Splitter	- 2
Hardware	- 3
Software	- 4

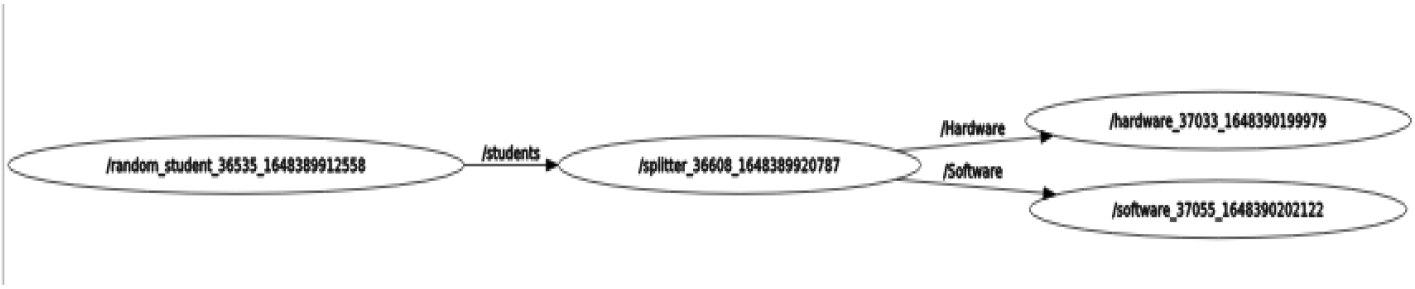
```
sara@ubuntu:~/Desktop/catkin_ws1$ . devel/setup.bash
sara@ubuntu:~/Desktop/catkin_ws1$ rosrun rand_stu random_student.py
```

خروجی اجرای برنامه به صورت زیر است:

```
[INFO] [1648391609.536125]:
nfo studente of hardware departement:
 name:                  Aref
 last name:             Modiri
 age:                   27
 departement:           Software

[INFO] [1648391611.536273]:
nfo studente of hardware departement:
 name:                  Maryam
 last name:             Ansari
 age:                   37
 departement:           Software

[INFO] [1648391615.536299]:
nfo studente of hardware departement:
 name:                  Reza
 last name:             Kabiri
 age:                   39
 departement:           Software
```

```
[INFO] [1648391596.536591]:
info studente of hardware departement:
 name:                  Amir Hosein
 last name:             Hoseini
 age:                   26
 departement:           Hardware

[INFO] [1648391600.536279]:
info studente of hardware departement:
 name:                  Javad
 last name:             Hoseini
 age:                   58
 departement:           Hardware

[INFO] [1648391602.536383]:
info studente of hardware departement:
 name:                  Ramin
 last name:             Hoseini
 age:                   24
 departement:           Hardware
```

گراف که با دستور rqt_graph فراخوانی میشود به شرح زیر است:



گام دوم

گام دوم (۲۵ امتیاز) : در این بخش می‌خواهیم کنترل turtlebot را در Gazebo به دست بگیریم. پیش از شروع این بخش دستورات زیر را اجرا کنید:

```
cd ~/catkin_ws/src/
git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
cd ~/catkin_ws && catkin_make
```

این دستورات، پکیج های مورد نیاز برای کار با turtlebot را نصب می‌کنند.

پس از نصب پکیج‌ها، برای این سناریو موارد زیر باید انجام شوند:

- پکیج مربوط به turtlebot انواع مختلفی از این ربات را با فیزیک متفاوت در اختیار ما قرار می‌دهد. برای این سناریو از ربات waffle استفاده کنید.
- مختصات اولیه ربات را در empty world نقطه‌ی (۳،۲) در نظر بگیرید.
- حال به کمک گره teleoperation حرکت ربات به وسیله کیبورد را بدست بگیرید.
- در آخر سه مورد بالا را یکبار دیگر با ایمپورت کردن محیط funky-maze.world پیاده سازی کنید با این تفاوت که نقطه شروع را می‌توانید به صورت دلخواه غیر از مبدا انتخاب کنید. (تنها تفاوت در محیط ربات اعمال می‌شود که جای empty world از این فایل برای محیط ربات در Gazebo استفاده کنید)

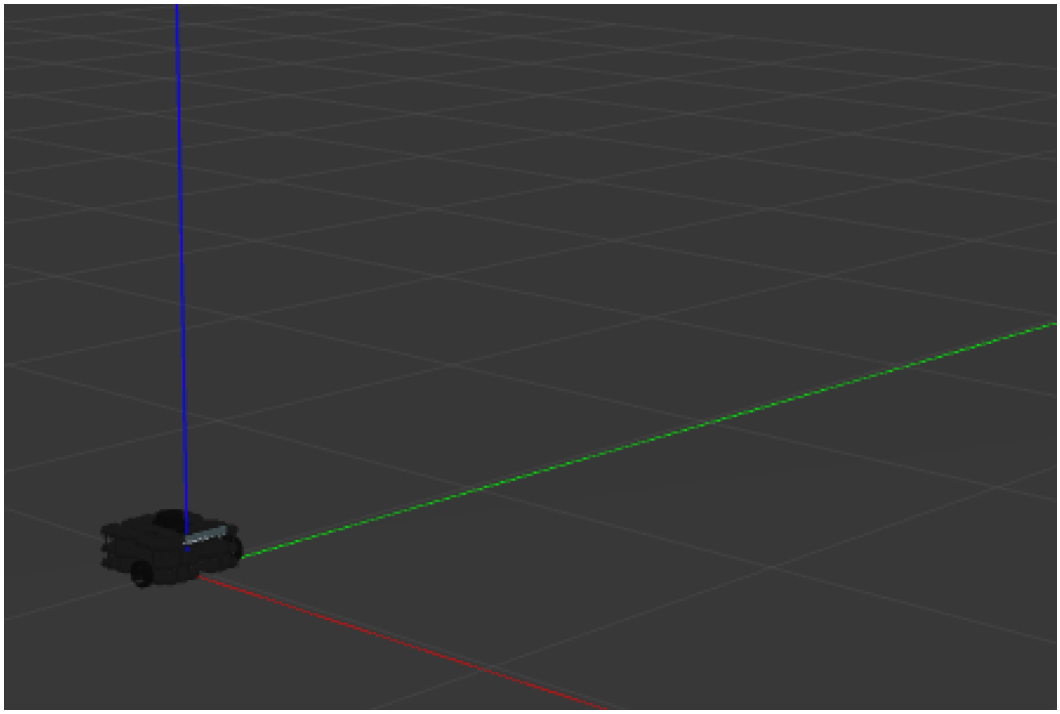
راهنمایی: برای مورد چهارم باید در مسیر turtlebot3_simulations/turtlebot3_gazebo/launch یک فایل launch. مشابه فایل turtlebot3_empty_world.launch ایجاد کنید و به جای empty.world، فایل funky-maze.world را قرار دهید. حال می توانید با دستور roslaunch ، محیط مورد نظر را بالا بیاورید.

ربات waffle

در گام دوم ابتدا git ها را clone میکنیم. حال برای قسمت اول دستورات زیر را فراخوانی میکنیم تا ربات waffle را در نقشه به ما نمایش دهد.

```
sara@ubuntu:~/Desktop/catkin_ws1$ . devel/setup.bash
sara@ubuntu:~/Desktop/catkin_ws1$ roscd
sara@ubuntu:~/Desktop/catkin_ws1/devel$ export TURTLEBOT3_MODEL=waffle
sara@ubuntu:~/Desktop/catkin_ws1/devel$ roslaunch turtlebot3_gazebo turtlebot3_e
mpty_world.launch
```

و در ادامه برنامه gazebo به شکل زیر اجرا میشود.



تغییر مختصات اولیه

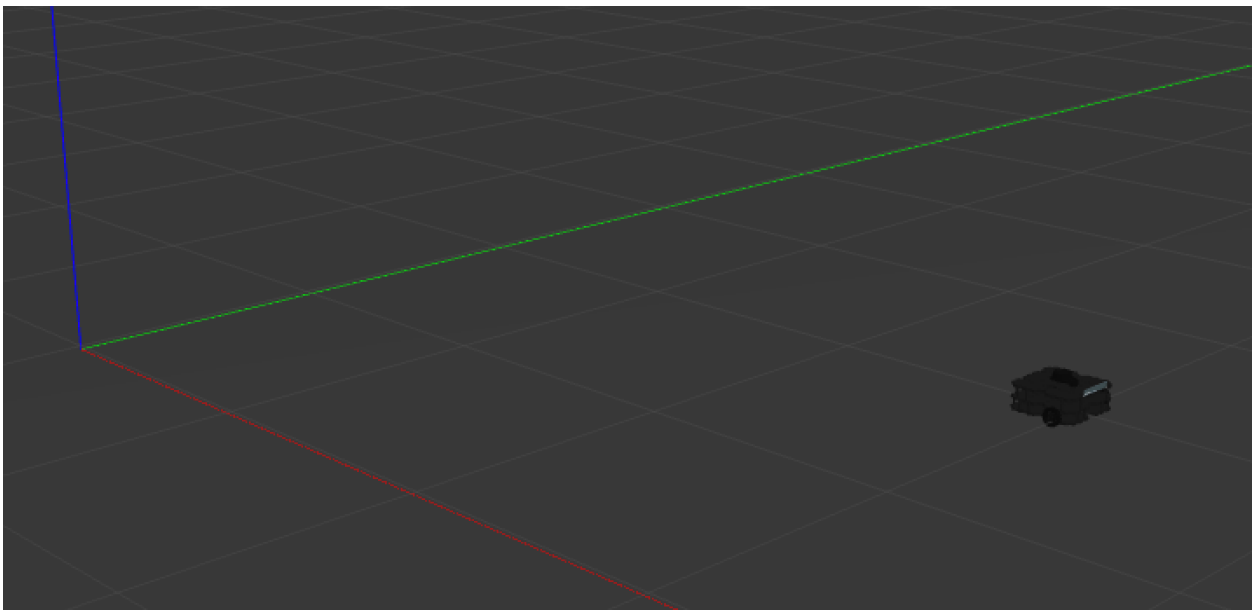
سپس برای قسمت دوم با کمک README و git ها باید فایل
turtlebot3_empty_world.launch را در آدرس زیر تغییر میدادیم تا مختصات اولیه ربات به نقطه (3,2) تغییر پیدا کند.

Catkin_ws1 -> src -> turtlebot3_simulations -> turtlebot3_gazebo -> launch -> turtlebot3_empty_world.launch

نقطه x = 3 و y = 2 را به صورت زیر تغییر میدهیم:

```
1 <launch>
2   <arg name="model" default="$(env TURTLEBO
3   <arg name="x_pos" default="3.0"/>
4   <arg name="y_pos" default="2.0"/>
5   <arg name="z_pos" default="0.0"/>
```

در ادامه برنامه را مثل قسمت اول اجرا میکنیم و برنامه gazebo به شکل زیر اجرا میشود:



کنترل حرکت ربات

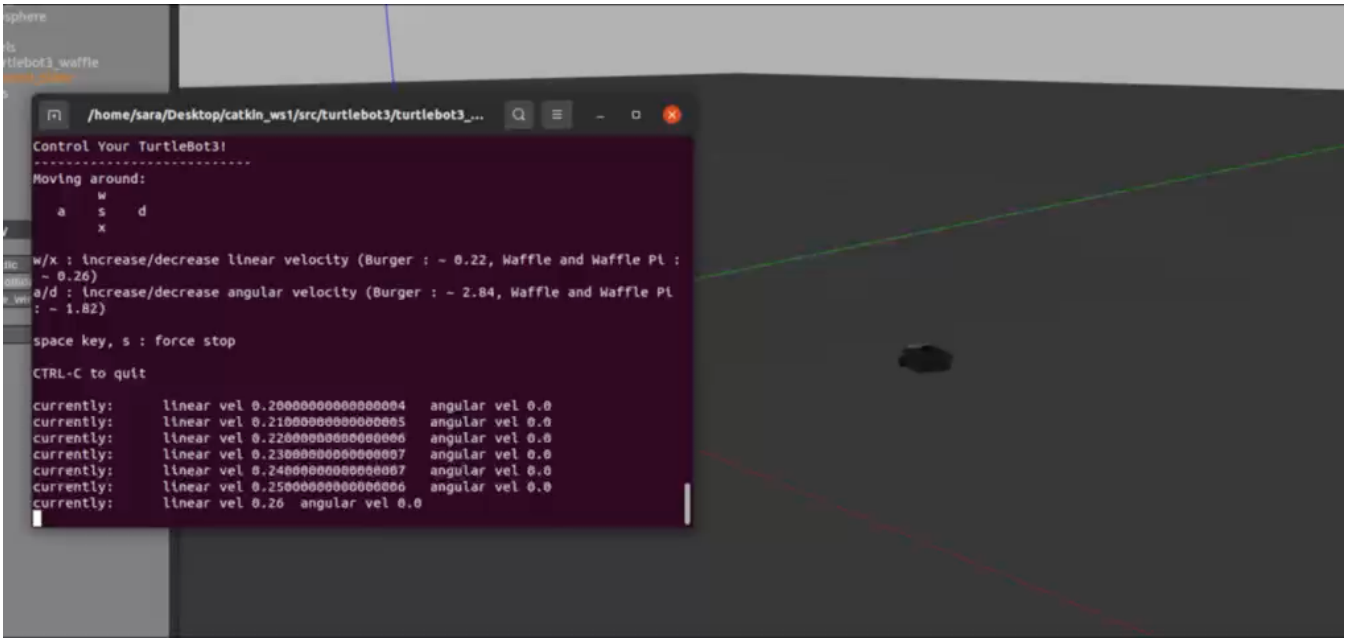
در قسمت سوم برای به دست گرفتن کنترل ربات دستور زیر را اجرا میکنیم:

```
sara@ubuntu:~/Desktop/catkin_ws1$ . devel/setup.bash
sara@ubuntu:~/Desktop/catkin_ws1$ roslaunch turtlebot3_teleop turtlebot3_teleop_
key.launch
```

حرکت ربات هم به شکل زیر انجام میشود:

```
Moving around:
        w
   a    s    d
        x
```

که در فیلم زیر مثالی از آن آورده شده:



محیط funky_maze

برای قسمت چهارم ابتدا فایل funky_maze.launch ضمیمه شده در آدرس زیر ذخیره میکنیم:

Catkin_ws1 -> src -> turtlebot3_simulations -> turtlebot3_gazebo -> world -> funky-maze.world

سپس فایل empty_world که در قسمت اول داشتیم را کپی کرده و با نام funky_maze در همان جا (آدرس زیر) ذخیره میکنیم و اسم فایل را در آن تغییر میدهیم(خط 8).

Catkin_ws1 -> src -> turtlebot3_simulations -> turtlebot3_gazebo -> launch -> turtlebot3_funky_maze_world.launch

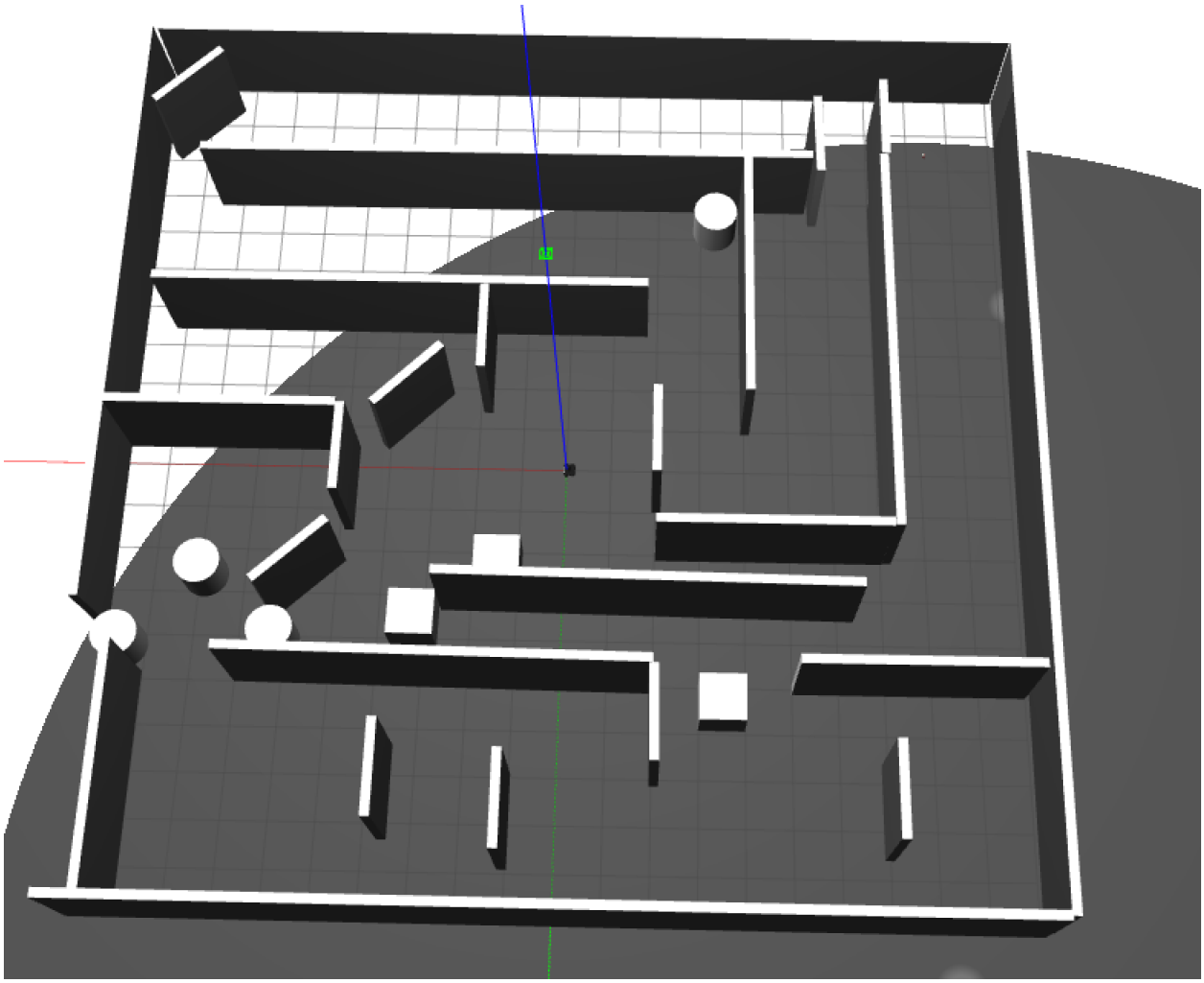
سپس دستور catkin_make را اجرا میکنیم تا تغییرات اعمال شوند.

ربات waffle (funky_maze)

برای اجرای قسمت اول دستورات زیر را اجرا میکنیم:

```
sara@ubuntu:~/Desktop/catkin_ws1/devel$ export TURTLEBOT3_MODEL=waffle
sara@ubuntu:~/Desktop/catkin_ws1/devel$ roslaunch turtlebot3_gazebo turtlebot3_f
unky_maze_world.launch
```

برنامه به صورت زیر اجرا میشود:



تغییر مختصات اولیه (funky_maze)

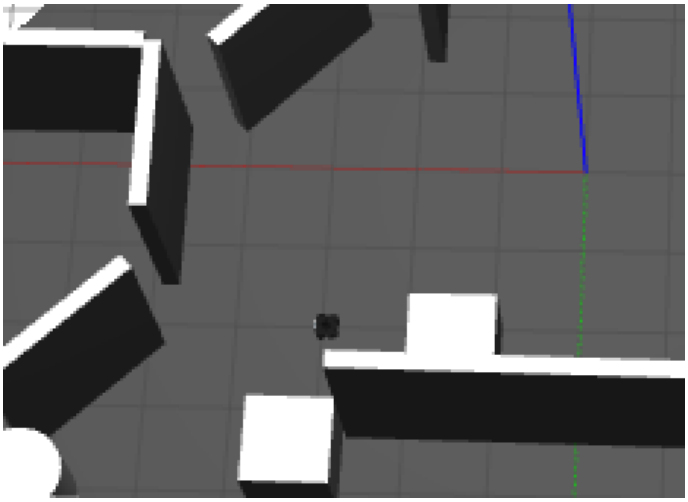
سپس برای قسمت دوم آدرس زیر تغییر میدادیم تا مختصات اولیه ربات به نقطه (3,2) تغییر پیدا کند.

Catkin_ws1 -> src -> turtlebot3_simulations -> turtlebot3_gazebo -> launch -> turtlebot3_funky_maze_world.launch

نقطه x = 3 و y = 2 را به صورت زیر تغییر میدهیم و تغیرات را ذخیره میکنیم:

```
1 <launch>
2   <arg name="model" default="$(env Tl
3   <arg name="x_pos" default="3.0"/>
4   <arg name="y_pos" default="2.0"/>
5   <arg name="z_pos" default="0.0"/>
```

در ادامه برنامه را مثل قسمت اول اجرا میکنیم و برنامه به شکل زیر اجرا میشود:



کنترل حرکت ربات (funky_maze)

در قسمت سوم برای به دست گرفتن کنترل ربات دستور زیر را مثل قبلی اجرا میکنیم:

```
sara@ubuntu:~/Desktop/catkin_ws1$ . devel/setup.bash
sara@ubuntu:~/Desktop/catkin_ws1$ roslaunch turtlebot3_teleop turtlebot3_teleop_key
 launch
```

که در فیلم زیر مثالی از آن آورده شده: