به نام خدا

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

اصول علم ربات

تمرین سری چهارم بخش پیاده سازی

نام و نام خانوادگی	مهدی رحمانی
شماره دانشجویی	۹۷۳۱۷۰۱
تاریخ ارسال گزارش	۱۴۰۲/۰۴/۰۳

فهرست گزارش سوالات

بخش صفرم (آماده سازی Workspace)

ابتدا یک Work space برای این پروژه میسازیم و آن را initialize میکنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project4$ mkdir -p hw4_ws/src
mahdi@mahdi:~/Desktop/Robotics/project4$ cd hw4_ws/src/
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$ catkin_init_workspace
Creating symlink "/home/mahdi/Desktop/Robotics/project4/hw4_ws/src/CMakeLists.txt" pointing to "/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ catkin_make
Base path: /home/mahdi/Desktop/Robotics/project4/hw4_ws
Source space: /home/mahdi/Desktop/Robotics/project4/hw4_ws/src
Build space: /home/mahdi/Desktop/Robotics/project4/hw4_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project4/hw4_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project4/hw4_ws/install
####
#### Running command: "cmake /home/mahdi/Desktop/Robotics/project4/hw4_ws/src -DCATKIN_DEVEL_PREFIX=/home/mahdi/Desktop/Robotics/project4/hw4_ws/devel -DC
ahdi/Desktop/Robotics/project4/hw4_ws/install -G Unix Makefiles" in "/home/mahdi/Desktop/Robotics/project4/hw4_ws/build"
####
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
```

برای گام اول ما طبق توضیحات دستورکار کد بخش‌های پایه‌ای نودها برای ما قرار داده شده است و کافی است آن را در کنار پکیج‌های دیگر خود در Workspace کلون کنیم.

- git clone -b main https://github.com/AmirInt/turtlebot3_object_tracker.git

```
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ cd src
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$ git clone -b main https://github.com/AmirInt/turtlebot3_object_tracker.git
Cloning into 'turtlebot3_object_tracker'...
remote: Enumerating objects: 92, done.
remote: Counting objects: 100% (92/92), done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 92 (delta 35), reused 82 (delta 27), pack-reused 0
Unpacking objects: 100% (92/92), 35.53 MiB | 1.79 MiB/s, done.
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$
```

همچنین در این پروژه به ربات turtlebot3 و شبیه‌ساز gazebo و rviz نیاز می‌شود. پس برای این منظور لازم است تا پکیج‌های زیر را نیز در فولدر src مربوط به workspaceمان کلون کنیم.

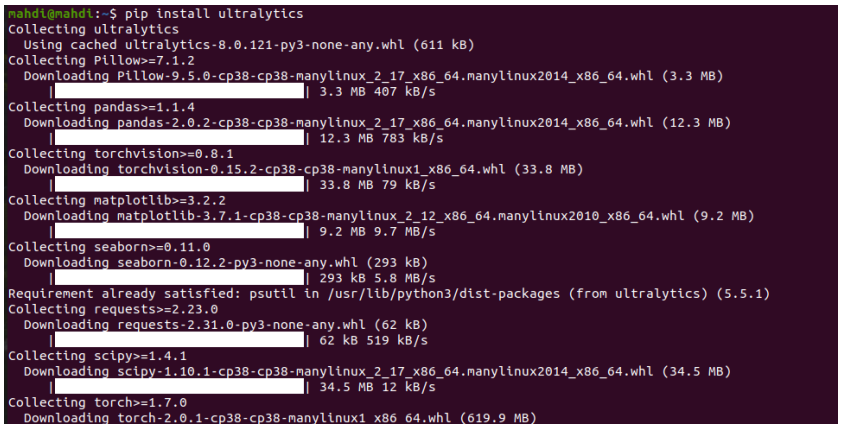
- git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
- git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
- git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git

```
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
Cloning into 'turtlebot3_simulations'...
remote: Enumerating objects: 3160, done.
remote: Counting objects: 100% (721/721), done.
remote: Compressing objects: 100% (145/145), done.
remote: Total 3160 (delta 628), reused 576 (delta 576), pack-reused 2439
Receiving objects: 100% (3160/3160), 15.40 MiB | 1.07 MiB/s, done.
Resolving deltas: 100% (1861/1861), done.
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
Cloning into 'turtlebot3'...
remote: Enumerating objects: 6485, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6485 (delta 0), reused 3 (delta 0), pack-reused 6481
Receiving objects: 100% (6485/6485), 119.95 MiB | 1.61 MiB/s, done.
Resolving deltas: 100% (4020/4020), done.
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
Cloning into 'turtlebot3_msgs'...
remote: Enumerating objects: 409, done.
remote: Counting objects: 100% (167/167), done.
remote: Compressing objects: 100% (54/54), done.
remote: Total 409 (delta 69), reused 151 (delta 59), pack-reused 242
Receiving objects: 100% (409/409), 90.31 KiB | 739.00 KiB/s, done.
Resolving deltas: 100% (170/170), done.
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$
```

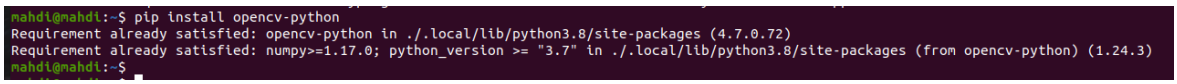
لازم است کتابخانه‌های زیر را نصب کنیم:

- numpy
- ultralytics
- opencv-python

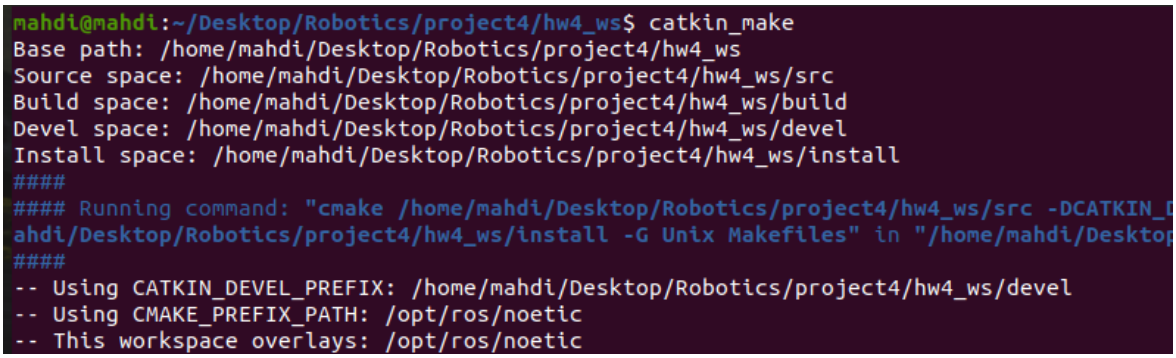
numpy را من از قبل نصب داشتم. لازم است تا ultralytics را نصب کنیم که حجم نسبتا زیادی هم دارد.



سپس لازم است open-CV را هم نصب کنیم:



پس از نصب تمامی Requirementها به دایرکتوری root مربوط به workspaceمان میرویم و catkin_make را اجرا میکنیم.



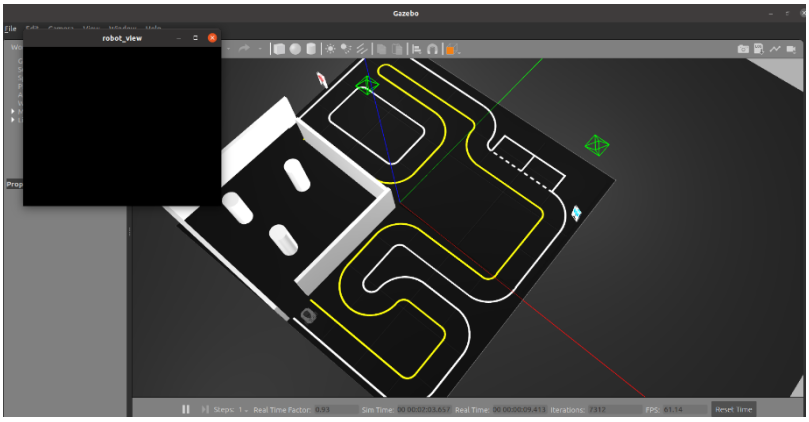
سپس برای شروع میتوانید برنامه را اجرا کرده ولی خب طبیعتا چون پیاده‌سازی نکرده ایم و همچنین تاپیکی را subscribe نکردیم خروجی خاصی نداریم.

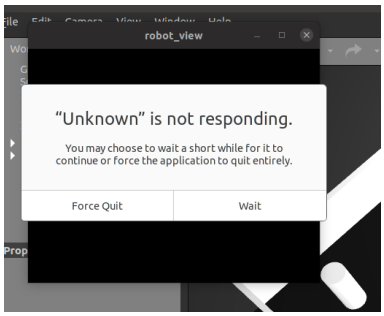
برای این منظور در همان دایرکتوری root مربوط به workspace رفته و کاندهای زیر را اجرا کنید:

- .devel/setup.bash
- roslaunch turtlebot3_object_tracker turtlebot3_object_tracker.launch

```
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ roslaunch turtlebot3_object_tracker turtlebot3_object_tracker.launch
... logging to /home/mahdi/.ros/log/b3169320-113b-11ee-a920-21557b9a3104/roslaunch-mahdi-6663.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.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:42285/

SUMMARY
========

PARAMETERS
 * /follower/robot_description: <?xml version="1....
 * /gazebo/enable_ros_network: True
 * /model: burger
 * /robot_description: <?xml version="1....
 * /rosdistro: noetic
```

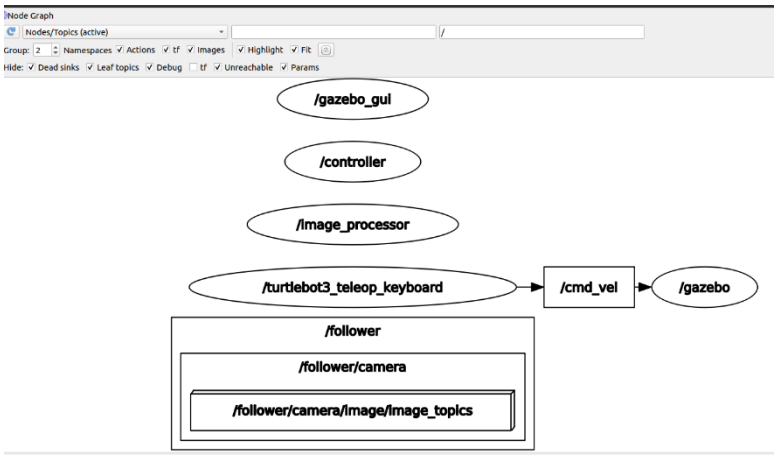
خروجی به صورت زیر خواهد بود و البته ممکنه پیام not responding هم برای شما بیاید.





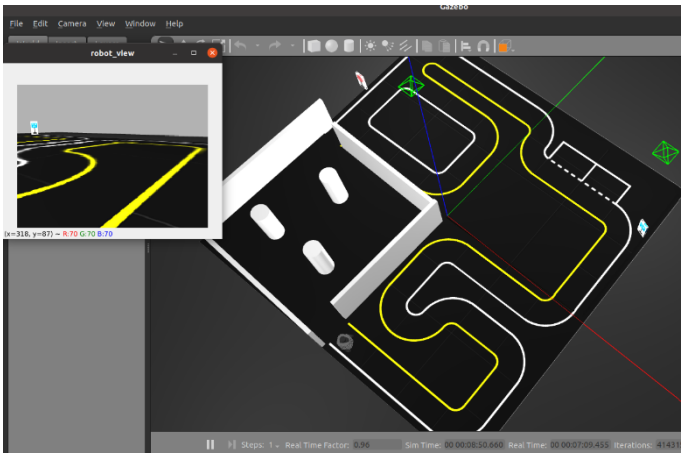
گام اول- توضیحات پیاده‌سازی

در این قسمت در ابتدا لازم است تا از طریق rqt graph بتوانیم تاپیک مناسب را برای subscribe کردن بیابیم. اگر قبل از این که چیزی پیاده سازی کنیم مراجعه کنیم با تصویر زیر مواجه میشویم:

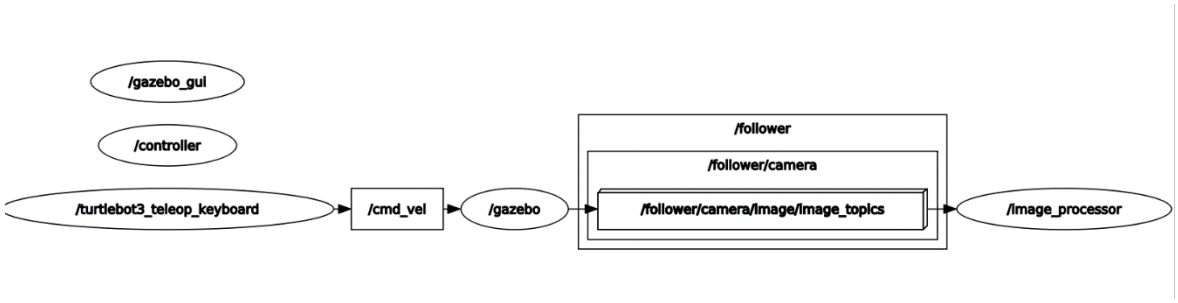


سپس اگر subscribe را به صورت زیر بنویسیم میتوانیم تصویر دوربین را مشاهده کنیم:

- self.camera_subscriber = rospy.Subscriber("/follower/camera/image",Image,self.camera_listener)



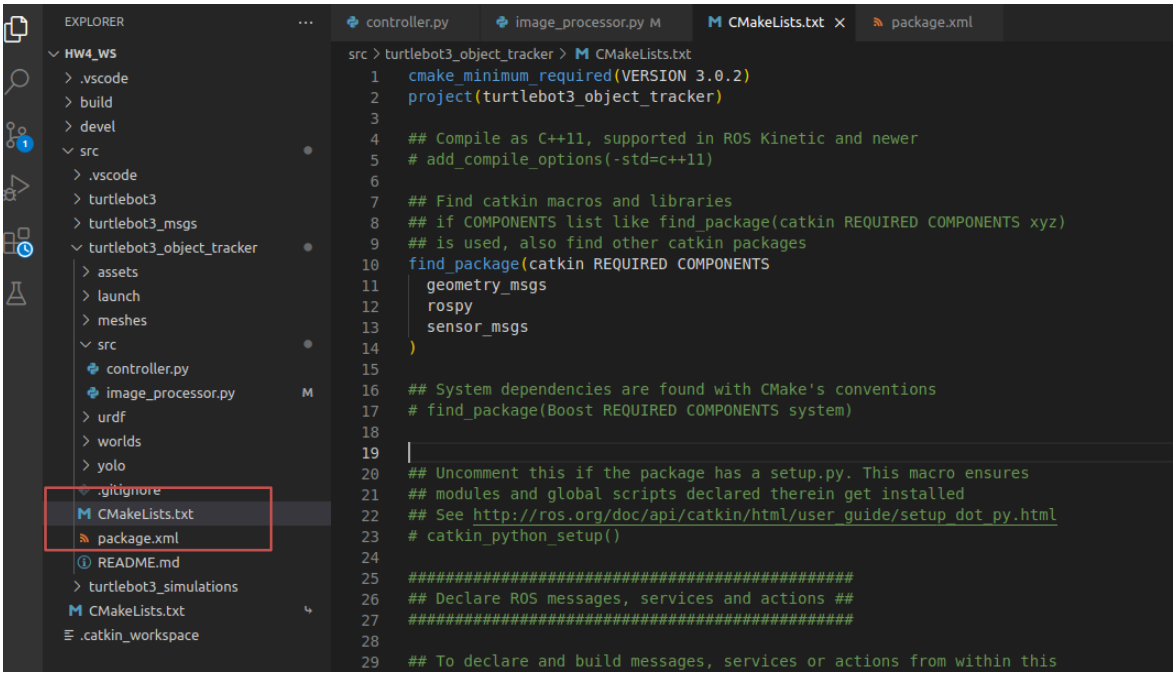
گراف نیز به صورت زیر خواهد شد:



سپس لازم است تا مدل را به متغیر self.model تخصیص دهیم. برای این منظور باتوجه به مسیر ذخیره شدن مدل خواهیم داشت:

- self.model: YOLO = YOLO('../yolo/yolov8n.pt')

حال لازم است تا یک سرویس برای detection بسازیم. برای این منظور ابتدا فایل‌های CMakeLists.txt و package.xml را بازکرده تا تغییرات لازم را در آن‌ها بدهیم.



۱- ابتدا در package.xml به سراغ ۵۵ و ۶۱ میرویم و به ترتیب در این خطوط موارد زیر را اضافه میکنیم:

- <build_depend>message_generation</build_depend>
- <exec_depend>message_runtime</exec_depend>

```
<!--   <test_depend>gtest</test_depend> -->
<!-- Use doc_depend for packages you need only for building documentation: -->
<!--   <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>geometry_msgs</build_depend>
<build_depend>rospy</build_depend>
<build_depend>sensor_msgs</build_depend>
<build_depend>message_generation</build_depend>
<build_export_depend>geometry_msgs</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>sensor_msgs</build_export_depend>
<exec_depend>geometry_msgs</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>message_runtime</exec_depend>
```

۲- سپس در CMakeLists ابتدا باید در خط ۱۴ عبارت message_generation را اضافه نماییم:

```
4   ## Compile as C++11, supported in ROS Kinetic and newer
5   # add_compile_options(-std=c++11)
6
7   ## Find catkin macros and libraries
8   ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9   ## is used, also find other catkin packages
10  find_package(catkin REQUIRED COMPONENTS
11    geometry_msgs
12    rospy
13    sensor_msgs
14    message_generation
15  )
16
```

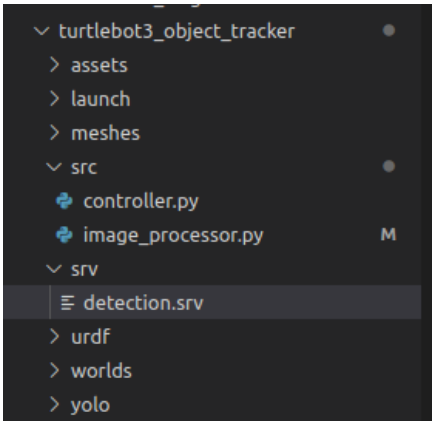
۳- سپس به خط ۵۸ رفته و باید add_service_files را uncomment کنیم و نام سرویس مورد نظر خود را اضافه نماییم:

```
56
57  ## Generate services in the 'srv' folder
58  add_service_files(
59    FILES
60    detection.srv
61  )
62
```

۴- سپس خطوط ۷۱ تا ۷۵ که مربوط به generate_messages هست را uncomment میکنیم:

```
70  ## Generate added messages and services with any dependencies listed here
71  generate_messages(
72    DEPENDENCIES
73    geometry_msgs
74    sensor_msgs
75  )
76
```

۵- برای آنکه کارمان تکمیل شود، در همان پوشه turtlebot3_object_tracker یک پوشه به نام srv درست میکنیم و بعد داخل آن یک فایل با نام اونی که در استپ ۳ مشخص کردیم میذاریم .



۶- توی فایل فوق لازمه که ورودی ها و خروجی های سرویس را بگوییم. بین ورودی ها و خروجی ها با --- جدا میشود. اگرم ورودی نداره خط اول همون --- میشود.

src > turtlebot3_object_tracker > srv > detection.srv

```
string label
---
float64 xc
float64 yc
float64 width
float64 height
float64 image_x
float64 image_y
```

۷- حال به پوشه hw4_ws در ترمینال برگشته و catkin_make میکنیم.

```
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ catkin_make
Base path: /home/mahdi/Desktop/Robotics/project4/hw4_ws
Source space: /home/mahdi/Desktop/Robotics/project4/hw4_ws/src
Build space: /home/mahdi/Desktop/Robotics/project4/hw4_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project4/hw4_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project4/hw4_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/mahdi/Desktop/Robotics/project4/hw4_ws/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/mahdi/Desktop/Robotics/project4/hw4_ws/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/noetic
-- This workspace overlays: /opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/mahdi/Desktop/Robotics/project4/hw4_ws/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/googletest': gtests will be built
```

حال لازم است سرویس مربوط به human detection را در image_processor پیاده سازی کنیم.

ابتدا خط زیر را در init اضافه میکنیم:

- self.human_detection_server = rospy.Service('detection', detection, self.service_callback)

```
        # TODO: Setup your "human detection" service
        self.human_detection_server = rospy.Service('detection', detection, self.service_callback)

        self.update_view()
```

همچنین لازم است تا متدی که قرار است سرور سرویس اجرا کند یعنی service_callback را پیاده کنیم. ابتدا باید image_np را به مدل بدهیم و self.results را آپدیت کنیم. سپس یک detectionResponse میسازیم و باید فیلدهای جوابی که قرار است به کلاینت دهیم را پر کنیم. حال باید باکس‌هایی که داخل result هست را بیابیم و از بین آن‌ها یکی مربوط به person است البته آن هم به شرط اینکه person در کادر باشد. در این صورت مشخصات آن باکس را در قالب پیام پاسخ میگذاریم و برای کلاینت ارسال میکنیم. چنانچه person که همان label ماست به عنوان label باکسی حضور نداشت باید یک مقداری

برای فیلدها قرار دهیم که گیرنده با دیدن آن‌ها متوجه شود person داخل کادر نبوده. برای این منظور من مقدار ۱۰۰۰ قرار دادم که باتوجه به سایز تصویر غیرقابل دسترس است.

```
52
53      def service_callback(self, input):
54
55          self.results = self.model(self.image_np)
56          target_result = self.results[0]
57
58          service_output = detectionResponse()
59          service_output.image_x, service_output.image_y = self.image_res[1]/2, self.image_res[0]/2
60
61          for box in target_result.boxes:
62              id = box.cls
63
64              if self.model.names[int(id)] == input.label:
65                  # get box coordinates in (xc, yc, w, h) format
66                  # (box center) -> (xc,yc)
67                  # (width and height of box) -> (w,h)
68                  box_coordinates = box.xywh
69                  service_output.xc, service_output.yc, service_output.width, service_output.height = box_coordinates.tolist()[0]
70                  return service_output
71
72          service_output.xc, service_output.yc, service_output.width, service_output.height = 1000, 1000, 1000, 1000
73          return service_output
```

سپس لازم است تغییرات اندکی در update_view ایجاد کنیم که بتواند باکس‌ها را به کمک annotator دور objectها ترسیم کند و label آن‌ها را نیز قرار دهد.

```
87
88                  # TODO: You can use an "Annotator" to draw object bounding boxes on frame
89                  for res in self.results:
90                      annotator = Annotator(frame)
91                      boxes = res.boxes
92                      for box in boxes:
93                          # get box coordinates in (x1, y1, x2, y2) format
94                          # (top left corner) -> (x1,y1)
95                          # (bottom right corner) -> (x2,y2)
96                          box_coordinates = box.xyxy[0]
97                          object_id = box.cls
98                          annotator.box_label(box_coordinates, self.model.names[int(object_id)])
```

دقت شود برای یافتن ابعاد باکس به کمک مختصات ها دو روش است یکی box.xyxy که مختصات دو گوشه‌ی سمت چپ بالا و سمت راست پایین را میدهد. دیگری box.xywh که مختصات نقطه مرکزس باکس و عرض و ارتفاع آن را میدهد.

کد مربوط به این قسمت در صفحه بعد کامل آمده است:

```
#!/usr/bin/python3

# Python
import copy
# Object detection
import cv2
import numpy as np
from ultralytics import YOLO
from ultralytics.yolo.utils.plotting import Annotator
from ultralytics.yolo.engine.results import Results
import torch
# ROS
import rospy
from sensor_msgs.msg import Image
# detection service
from turtlebot3_object_tracker.srv import detection, detectionResponse

class ImageProcessor:
    def __init__(self) -> None:
        # Image message
        self.image_msg = Image()
        self.image_res = 240, 320, 3 # Camera resolution: height, width
        self.image_np = np.zeros(self.image_res) # The numpy array to pour the image data into
        # TODO: Subscribe on your robot's camera topic
        # NOTE: Make sure you use the provided listener for this subscription
        self.camera_subscriber =
rospy.Subscriber("/follower/camera/image",Image,self.camera_listener)
        # TODO: Instantiate your YOLO object detector/classifier model
        self.model: YOLO = YOLO('../yolo/yolov8n.pt')
        # TODO: You need to update results each time you call your model
        self.results: Results = None
        self.cv2_frame_size = 400, 320
        cv2.namedWindow("robot_view", cv2.WINDOW_NORMAL)
        cv2.resizeWindow("robot_view", *self.cv2_frame_size)
        # TODO: Setup your "human detection" service
        self.human_detection_server = rospy.Service('detection', detection, self.service_callback)
        self.update_view()
    def camera_listener(self, msg: Image):
        self.image_msg.data = copy.deepcopy(msg.data)
    def service_callback(self, input):
        self.results = self.model(self.image_np)
        target_result = self.results[0]
        service_output = detectionResponse()
        service_output.image_x, service_output.image_y = self.image_res[1]/2, self.image_res[0]/2
        for box in target_result.boxes:
            id = box.cls
            if self.model.names[int(id)] == input.label:
                # get box coordinates in (xc, yc, w, h) format
```

```
                # (box center) -> (xc,yc)
                # (width and height of box) -> (w,h)
                box_coordinates = box.xywh
                service_output.xc, service_output.yc, service_output.width, service_output.height =
box_coordinates.tolist()[0]
                return service_output
        service_output.xc, service_output.yc, service_output.width, service_output.height = 1000,
1000, 1000, 1000
        return service_output
    def update_view(self):
        try:
            while not rospy.is_shutdown():
                if len(self.image_msg.data) == 0: # If there is no image data
                    continue
                # Convert binary image data to numpy array
                self.image_np = np.frombuffer(self.image_msg.data, dtype=np.uint8)
                self.image_np = self.image_np.reshape(self.image_res)
                self.results = self.model(self.image_np)
                frame = copy.deepcopy(self.image_np)
                # TODO: You can use an "Annotator" to draw object bounding boxes on frame
                for res in self.results:
                    annotator = Annotator(frame)
                    boxes = res.boxes
                    for box in boxes:
                        # get box coordinates in (x1, y1, x2, y2) format
                        # (top left corner) -> (x1,y1)
                        # (bottom right corner) -> (x2,y2)
                        box_coordinates = box.xyxy[0]
                        object_id = box.cls
                        annotator.box_label(box_coordinates,
self.model.names[int(object_id)])
                cv2.imshow("robot_view", cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))
                cv2.waitKey(1)
        except rospy.exceptions.ROSInterruptException:
            pass


if __name__ == "__main__":
    rospy.init_node("image_processor", anonymous=True)
    rospy.on_shutdown(cv2.destroyAllWindows)
    image_processor = ImageProcessor()
    rospy.spin()
```

در قسمت کنترلر لازم است تا در تابع init هم service proxy را تعریف کنیم و هم نود را به عنوان پابلیشر cmd_vel برای follower تعریف کنیم.

```
19
20      # TODO: Create a service proxy for your human detection service
21      rospy.wait_for_service('detection')
22      self.calc_client = rospy.ServiceProxy('detection', detection)
23
24      # TODO: Create a publisher for your robot "cmd_vel"
25      self.cmd_publisher = rospy.Publisher('/follower/cmd_vel' , Twist , queue_size=10)
```

سپس در قسمت ران لازم است بگوییم که درخواستی برای human detection service بفرستد و چنانچه person در کادر بود و اطلاعات آن را برگرداند سرعت زاویه‌ای را به کمک کنترلر محاسبه کنیم.

برای محاسبه خطایی که باید در kp که اینجا angular_vel_coef تعریف شده است، در آن ضرب شود میایم فاصله x را برای وسط باکس تا وسط تصویر میابیم و سپس سرعت زاویه‌ای را محاسبه کرده و پابلیش میکنیم.

همچنین اگر انسان در تصویر نبود من اینگونه تعریف کردم که ربات freeze شود و حرکت نکند.

```
27      def run(self) -> None:
28          try:
29              while not rospy.is_shutdown():
30                  # TODO: Call your service, ride your robot
31                  req = detectionRequest()
32                  req.label = 'person'
33
34                  resp = self.calc_client(req)
35
36                  if resp.xc == 1000:
37
38                      self.cmd_publisher.publish(self.freeze)
39
40                  else:
41
42                      person_x, person_y = resp.xc, resp.yc
43                      image_center_x, image_center_y = resp.image_x, resp.image_y
44
45                      error = person_x-image_center_x
46                      angular_velocity = self.angular_vel_coef*(error)
47
48                      self.move.angular.z = -angular_velocity
49                      self.cmd_publisher.publish(self.move)
```

درنهایت کد کامل آن در ادامه آمده است:

```
#!/usr/bin/python3
# ROS
import rospy
from geometry_msgs.msg import Twist
# detection service
from turtlebot3_object_tracker.srv import detection, detectionRequest
```

```python
class Controller:

    def __init__(self) -> None:
        # Use these Twists to control your robot
        self.move = Twist()
        self.move.linear.x = 0.1
        self.freeze = Twist()
        # The "p" parameter for your p-controller, TODO: you need to tune this
        self.angular_vel_coef = 0.002
        # TODO: Create a service proxy for your human detection service
        rospy.wait_for_service('detection')
        self.calc_client = rospy.ServiceProxy('detection', detection)
        # TODO: Create a publisher for your robot "cmd_vel"
        self.cmd_publisher = rospy.Publisher('/follower/cmd_vel' , Twist , queue_size=10)

    def run(self) -> None:
        try:
            while not rospy.is_shutdown():
                # TODO: Call your service, ride your robot
                req = detectionRequest()
                req.label = 'person'
                resp = self.calc_client(req)
                if resp.xc == 1000:
                    self.cmd_publisher.publish(self.freeze)
                else:
                    person_x, person_y = resp.xc, resp.yc
                    image_center_x, image_center_y = resp.image_x, resp.image_y

                    error = person_x-image_center_x
                    angular_velocity = self.angular_vel_coef*(error)

                    self.move.angular.z = -angular_velocity
                    self.cmd_publisher.publish(self.move)

        except rospy.exceptions.ROSInterruptException:
            pass

if __name__ == "__main__":
    rospy.init_node("controller", anonymous=True)

    controller = Controller()
    controller.run()
```

گام اول- اجرا و نتایج

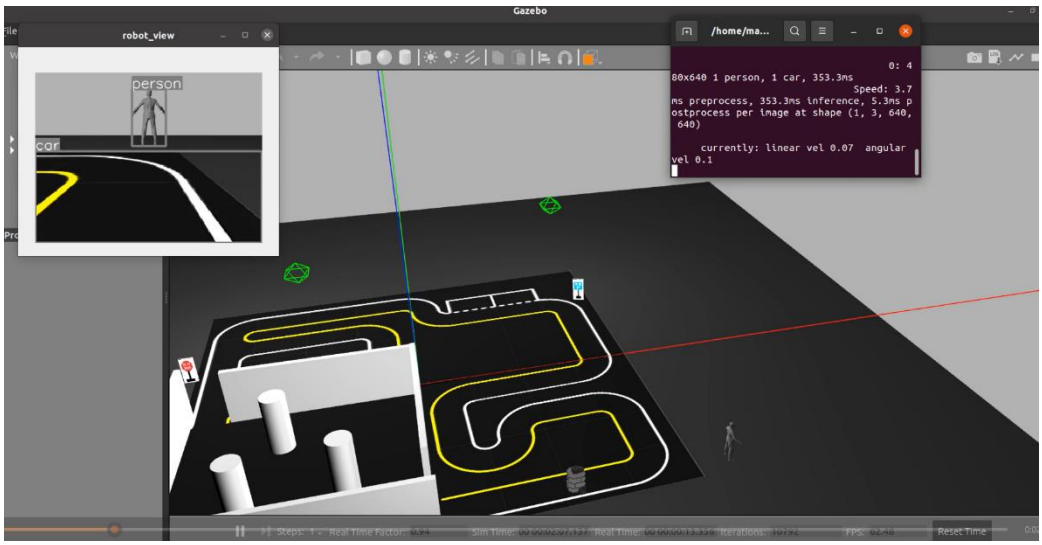
در نهایت برای اجرا دستورات زیر را وارد کرده:

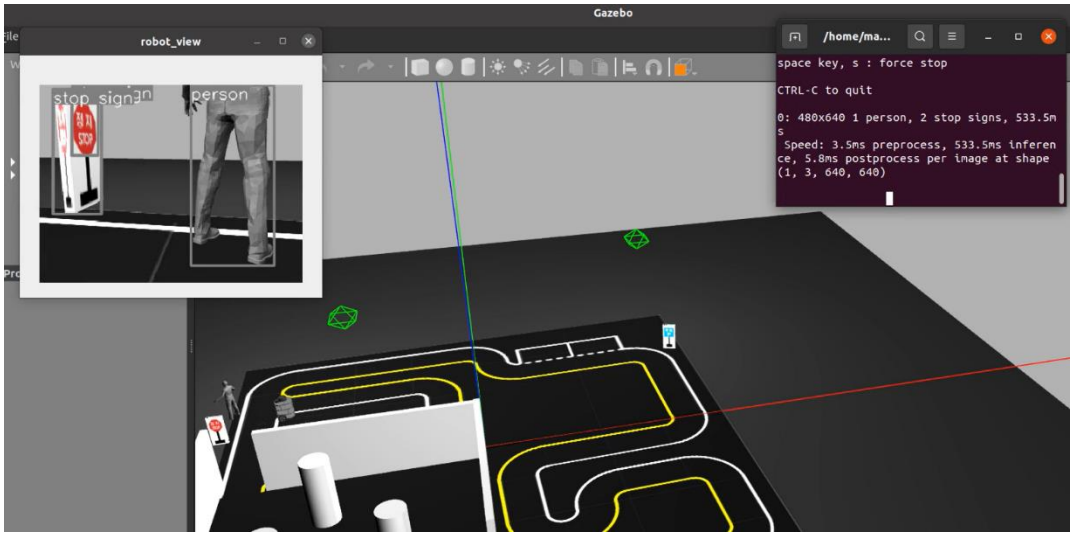
- . devel/setup.bash
- roslaunch turtlebot3_object_tracker turtlebot3_object_tracker.launch

سپس ربات باتوجه به سرعت خطی خود شروع به حرکت کرده و با توجه به فاصله‌اش تا person میزان سرعت زاویه‌ای خود را تنظیم میکند.

میتوان به کمک کلیدهای wو d و a و s و x شخص را هدایت کرد و پیش برد تا ربات هم آن را دنبال کند. فیلمی جهت صحت و درستی اجرا قرار داده شده است.

در زیر نمایی از دنبال کردن انسان توسط ربات را در Gazebo و robot view مشاهده میکنید:





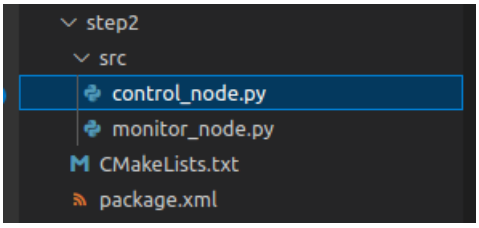
گام دوم(امتیازی)-توضیحات پیاده سازی

برای این قسمت یک پکیج جداگانه درست میکنیم به نام step2 و میکنیم و dependencyهایی که ممکن است به کار بیایند را هم اضافه کنیم.

- catkin_create_pkg step2 rospy std_msgs sensor_msgs nav_msgs

```
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ cd src
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$ catkin_create_pkg step2 rospy std_msgs sensor_msgs nav_msgs
Created file step2/package.xml
Created file step2/CMakeLists.txt
Created folder step2/src
Successfully created files in /home/mahdi/Desktop/Robotics/project4/hw4_ws/src/step2. Please adjust the values in package.xml.
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$
```

حال به سراغ ساخت نودها میرویم. برای این منظور در فولدر src مربوط به step2 میرویم و دوفایل پایتون با نام های control_node.py و monitor_node.py میسازیم. نودmonitor_node.py نیز برای نمایش مسیر ربات در rviz به کار گرفته خواهد شد.



حال در گام بعد به سراغ نوشتن کد مربوط به هریک از نودها میرویم. کد مربوط به نود monitor_node مانند تمرین‌های قبل میباشد. در monitor کد پایتون مربوط به رسم مسیری که ربات آن را طی میکند میباشد که از تاپیک path استفاده میکند. مکان هایی که میرود را در قالب یک آرایه یا لیست ذخیره میکنیم و در rviz آن را نشان میدهیم.

حال به کد نود control_node میپردازیم. در این جا یک متغیر direction داریم که باتوجه به متغیر follow_type مقدارش میتواند ۱ یا ۱- باشد. چنانچه ربات دنبالگر راست باشد این ضریب منفی ۱ است و به عنوان ضریبی در حاصل جمع P+I+D ضرب میشود. همچنین در این جا یک تابع distance_from_wall داریم که با توجه به اسکن‌های لیزر لایدار میتواند فاصله تا دیوار را برگرداند. دقت شود اگر دیوار در سمت راست باشد باید بین مقادیر در range بین ۱۸۰ تا ۳۶۰ مینیمم بگیریم و اگر سمت چپ باشد بین ۰ تا ۱۸۰ را بررسی کنیم. باتوجه به اینکه فاصله مطلوب تا دیوار را ۱.۵ گرفتیم بنابراین اختلاف فاصله تا دیوار و این مقدار ۱.۵ را باید سعی کنیم که ۰ کنیم و PID در این جهت تلاش میکند. برای tune کردن PID ابتدا ترم P زیاد و D کم بود و باعث میشد که ربات به دیوار برخورد کند.

در نهایت با سعی و خطا به مقدارهایی که دلخواه است رسیدیم. دقت شود از مقادیر wall follower در تمرین قبلی هم ایده گرفتم. همچنین توضیحات کافی با کامنت داخل کد داده شده و کل کد هم در ادامه قرار داده شده است.

```
#!/usr/bin/python3

import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import matplotlib.pyplot as plt

class PIDController():
    def __init__(self):
        rospy.init_node('controller', anonymous=False)
        rospy.on_shutdown(self.on_shutdown)
        self.cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

        # if we want to follow wall on right hand the robot should rotate
        # in the inverse direction of default state. so we multiply -1 coefficient
        # to (P+I+D)
        self.follow_type = rospy.get_param("/controller/follow_type")
        if self.follow_type == "right":
            self.direction = -1
        else:
            self.direction = 1

        # angular velocity PID gains
        self.k_p = 1
        self.k_i = 0.0001
        self.k_d = 30

        self.dt = 0.005
        self.v = 0.06
        self.D = 0.5

        rate = 1/self.dt
        self.r = rospy.Rate(rate)

        self.errs = []

    def distance_from_wall(self):
        '''
        this method give us the distance from wall with the help of scans
        comming from the lidar sensor. if direction is -1 means that we
        want to have on the right hand of robot and ranges [180:] is important to
        be checked and if direction is 1 it is vice versa.
```

```
        '''
        laser_data = rospy.wait_for_message("/scan" , LaserScan)
        if self.direction==1:
            rng = laser_data.ranges[:180]
        else:
            rng = laser_data.ranges[180:]
        d = min(rng)
        return d

    def control(self):
        '''
        this function is the main function of this code. we calculate the
        distance error and try to calculate P, I, D terms. the summation of these
        terms define our angular velocity.
        '''
        d = self.distance_from_wall()
        sum_i_theta = 0
        prev_theta_error = 0

        move_cmd = Twist()
        move_cmd.angular.z = 0
        move_cmd.linear.x = self.v
        while not rospy.is_shutdown():
            self.cmd_vel.publish(move_cmd)
            err = d - self.D
            self.errs.append(err)
            sum_i_theta += err * self.dt

            P = self.k_p * err
            I = self.k_i * sum_i_theta
            D = self.k_d * (err - prev_theta_error)

            move_cmd.angular.z = self.direction * (P + I + D)
            prev_theta_error = err
            move_cmd.linear.x = self.v

            #rospy.loginfo(f"P : {P} I : {I} D : {D}")
            #rospy.loginfo(f"error : {err} speed : {move_cmd.linear.x} theta :
{move_cmd.angular.z}")
            d = self.distance_from_wall()
            self.r.sleep()

    def on_shutdown(self):
        '''
        this method plot error of linear and angular velocity separately.
        '''
        rospy.loginfo("Stopping the robot...")
        self.cmd_vel.publish(Twist())
```

```
        plt.plot(list(range(len(self.errs))),self.errs, label='errs')
        plt.axhline(y=0,color='R')
        plt.draw()
        plt.legend(loc="upper left", frameon=False)
        plt.savefig(f"errs_{self.k_p}_{self.k_d}_{self.k_i}.png")
        plt.show()
        rospy.sleep(1)

if __name__ == '__main__':
    try:
        pidc = PIDController()
        pidc.control()
    except rospy.ROSInterruptException:
        rospy.loginfo("Navigation terminated.")
```

در مرحله بعد باید تمامی کدهای پایتون را executable کنیم. برای این کار لازم است در ترمینال در پکیج step2 کد زیر را اجرا کنیم:

- chmod +x src/*.py

```
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src$ cd step2/
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src/step2$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src/step2$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src/step2/src$ ls
control_node.py  monitor_node.py
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws/src/step2/src$
```

سپس به سراغ نوشتن لانچ فایل میرویم. در همین پکیج step3 لازم است تا یک فولدر launch ایجاد کنیم. قبلا لانچ فایل با نام my_empty_world را که در پوشه لانچ مربوط به turtle_bot بود را include میکردیم. در این جا یک فایل مشابه لانچ فایل با نام empty_world مستقیما همینجا اضافه میکنیم. (این لانچ فایل را میتوانید در لانچ فایل‌های turtlebot پیدا کنید.) اسم این لانچ فایل را turtlebot3_maze.launch میگذاریم. محتوای آن به صورت زیر است. دقت شود که زاویه اضافه شدن ربات به map را هم در اینجا تعریف کردیم. همچنین world_name را هم مشخص کردیم.

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle,
waffle_pi]"/>
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>
  <arg name="yaw" value="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find step2/worlds/maze.world"/>
    <arg name="paused" value="false"/>
```

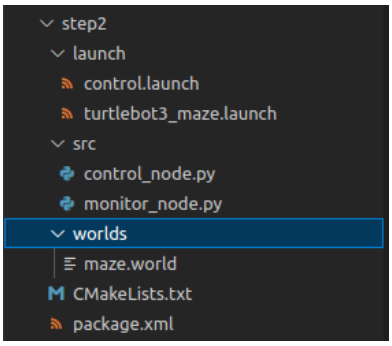
```
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_$(arg
model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -Y $(arg yaw) -param robot_description" />
</launch>
```

همچنین یک control.launch ایجاد کنیم. لانچ فایل به صورت زیر است. ابتدا لانچ فایل قبلی را که نوشتیم include میکنیم و میگوییم در آن آرگومان های ورودی مثل مکان اولیه ربات و زاویه اولیه چه باشند. طبق خواسته سوال در این جا ربات در مکان (۰و۰.۵-) و با زاویه ۰ درجه باید اضافه شود. سپس launch file مربوط به rviz را include میکنیم تا آن را هم برای نمایش مسیر ربات بالا بیاورد.

همچنین نودهای control_node.py و همچنین monitor_node.py را با نام‌های controller و monitor بالا بیاورد. نود monitor که برای رسم مسیر لازم است و آن هم باید بالا بیاید.

```
<launch>

    <arg name="follow_type" default="right"/>

    <include file="$(find step2)/launch/turtlebot3_maze.launch">
        <arg name="x_pos" value="-0.5"/>
        <arg name="y_pos" value="0.0"/>
        <arg name="z_pos" value="0.0"/>
        <arg name="yaw" value="0.0"/>
    </include>

    <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_gazebo_rviz.launch"/>

    <node pkg="step2" type="control_node.py" name="controller" output="screen">
        <param name="follow_type" value="$(arg follow_type)" />
    </node>

    <node pkg="step2" type="monitor_node.py" name="monitor"></node>

</launch>
```

همچنین لازم است یک فولدر به نام worlds در src مربوط به پکیج step3 ایجاد کنیم و فایل square.world را در آنجا اضافه کنیم:



سپس در آخر لازم است تا به دایرکتوری ورک اسپیس برویم و catkin_make را صدا بزنیم. سپس برای استفاده لازم است تا ابتدا سورس کنیم و سپس ربات را اکسپورت کنیم و در نهایت roslaunch را صدا بزنیم:

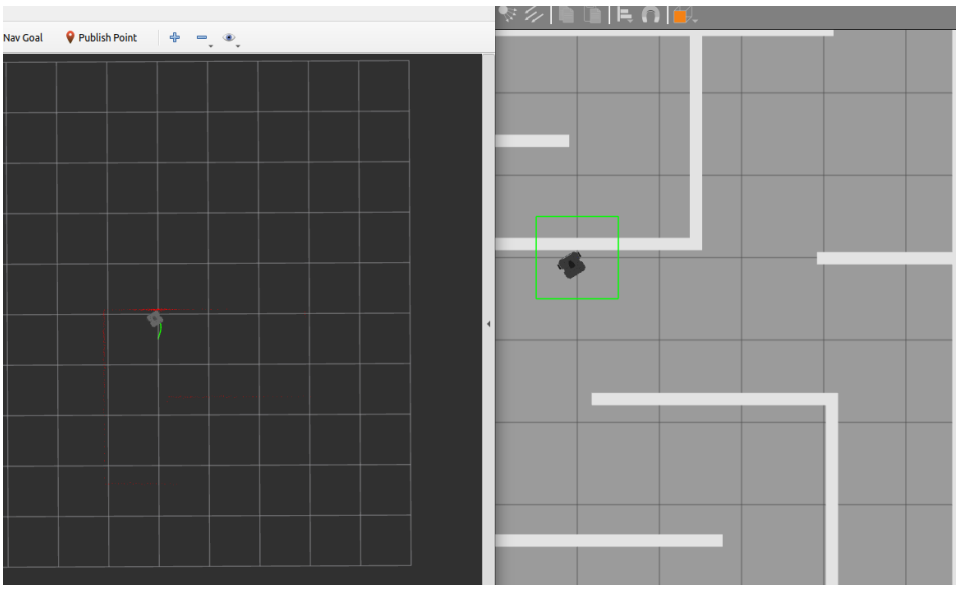
- . devel/setup.bash
- export TURTLEBOT3_MODEL=waffle
- roslaunch step3 control.launch follow_type:=right

```
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ roslaunch step2 control.launch follow_type:=right
... logging to /home/mahdi/.ros/log/505aef56-11fe-11ee-8c1c-5341409bfe14/roslaunch-mahdi-107324.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.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:35027/

SUMMARY
========

PARAMETERS
 * /controller/follow_type: right
 * /gazebo/enable_ros_network: True
 * /robot_description: <?xml version="1....
 * /robot_state_publisher/publish_frequency: 50.0
 * /robot_state_publisher/tf_prefix:
 * /rosdistro: noetic
```

در رابطه با tune کردن PID من ابتدا باتوجه به تمرین قبلی مقادیر زیر را برای gainها انتخاب کردم:

```
# angular velocity PID gains
self.k_p = 0.6
self.k_i = 0.0001
self.k_d = 10
```

سپس نتیجه آن شد که ربات به دیوار برخورد کرد:



باتوجه به اینکه اینجا به دیوار نزدیک هستیم و مستقیم به سمت آن میرویم لازم است که ربات سریعتر به خطایی که از حد آستانه و مطلوب دارد، پاسخ دهد. پس لازم است gain مربوط به P را زیاد کنیم همچنین چون P زیاد شود به همراه خود ممکن است باعث شود Overshoot زیاد شود لازم است تا مقدار gain مربوط به D را هم زیاد کنیم. در نهایت با سعی و خطا به مقادیر زیر رسیدم:

```
# angular velocity PID gains
self.k_p = 1
self.k_i = 0.0001
self.k_d = 30
```

در نهایت میتوانید نتایج در قسمت بعدی مشاهده کنید.

گام دوم(امتیازی) - اجرا و نتایج

اگر دقت شود دو مسیر خروج از maze وجود دارد. چنانچه ربات ما دنبالگر راست باشد و یعنی همیشه دیوار را در سمت راست خود نگه دارد آنگاه از خروجی بالایی خارج شده و چنانچه دنبالگر چپ باشد از خروجی پایینی اجرا شده است.

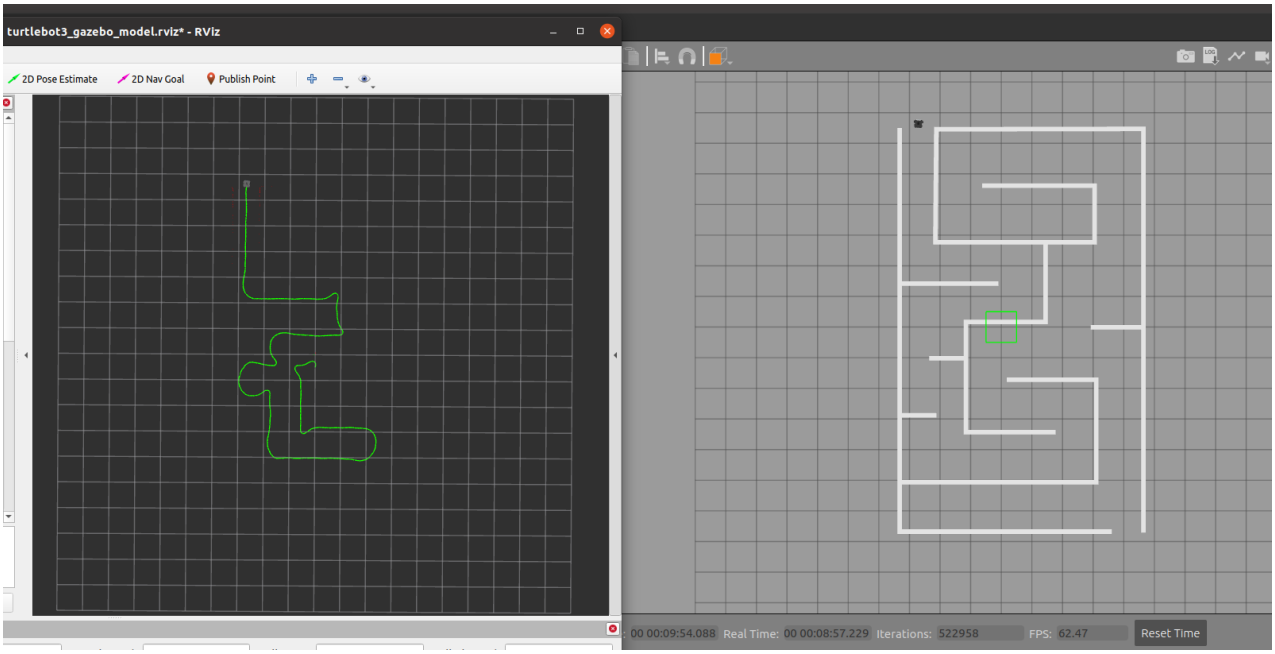
پیاده سازی به گونه‌ای است که در ترمینال میتوان به عنوان ورودی نوع دنبالگر را به کمک left یا right معرفی کرد. حال در ادامه نتیجه اجرا برای هر حالت آمده است. لازم به ذکر است که مقدار follow_type برابر با right میباشد.

- **دنبالگر راست:**

پس از source کردن و export کردن نوع ربات باید دستور زیر را وارد نمایید:

- roslaunch step3 control.launch follow_type:=right

خروجی به صورت زیر خواهد شد:



- **دنبالگر چپ:**

پس از source کردن و export کردن نوع ربات باید دستور زیر را وارد نمایید:

- roslaunch step3 control.launch follow_type:=right

خروجی به صورت زیر خواهد شد: