



Department of
Computer Engineering

به نام خدا



Amirkabir University of Technology
(Tehran Polytechnic)

دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر
أصول علم ربات

تمرين سري دوم

نام و نام خانوادگی	مهندی رحمانی
شماره دانشجویی	۹۷۳۱۷۰۱
تاریخ ارسال گزارش	۱۴۰۲/۰۲/۰۵

فهرست گزارش سوالات

۳ گام اول – به کار گیری سرویس و تعیین مقصد در empty map
۱۶ گام اول – نتایج
۲۰ گام دوم – تشخیص مانع به کمک لایدار
۲۹ گام دوم – نتایج
۳۰ گام سوم – کار با Pointcloud
۳۷ گام سوم – نتایج

گام اول - به کار گیری سرویس و تعیین مقصد در empty map

ابتدا لازم است تا یک work space برای گام اول بسازیم و آن را initialize کنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages$ mkdir -p hw2_ws/src
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages$ cd hw2_ws/src/
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages$ catkin_init_workspace
Creating symlink "/home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/CMakeLists.txt" pointing to "/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages$ catkin_make
Base path: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws
Source space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src
Build space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/install
#####
#### Running command: "cmake /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src -DCATKIN_DEVEL_PREFIX=/home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/devel -DCMAKE_INSTALL_PREFIX=/home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/install -G Unix Makefiles" in "/home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_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
```

حال لازم است که یک پکیج مثلا با نام get_destination در فolder سورس بسازیم. همچنین

dependency های لازم را که ممکن است در نوشتن نودها و کد زنی نیاز شود به آن میدهیم:

- catkin_create_pkg get_destination rospy std_msgs sensor_msgs nav_msgs

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$ catkin_create_pkg get_destination rospy std_msgs sensor_msgs nav_msgs
Created file get_destination/package.xml
Created file get_destination/CMakeLists.txt
Created folder get_destination/src
Successfully created files in /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/get_destination. Please adjust the values in package.xml.
Mahdi@Mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$
```

چون قرار است از turtlebot و شبیه ساز gazebo استفاده کنیم لازم است ابتدا از لینک های زیر یک

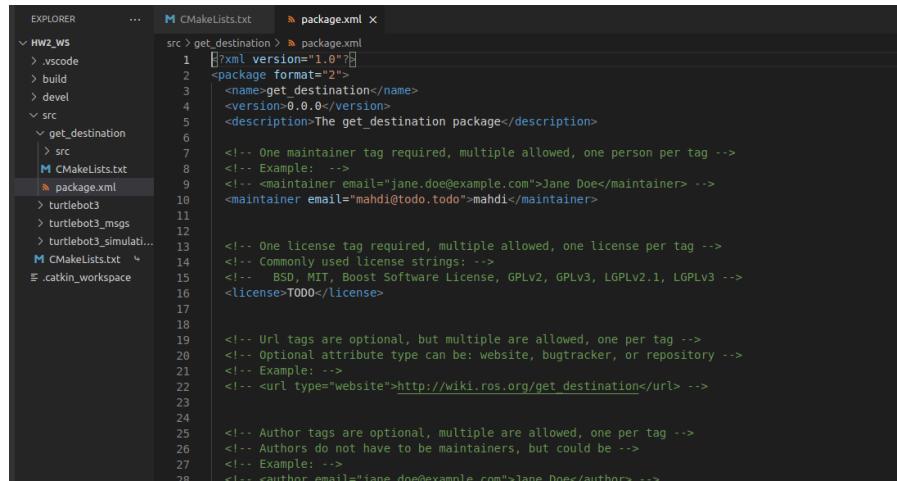
سری پکیج ها را در فolder src خودمان دانلود کنیم:

- 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

پس از دانلود اگر ls بزنیم لیست پکیج های داخل فolder سورس را خواهیم دید:

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_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% (144/144), done.
remote: Total 3160 (delta 629), reused 577 (delta 577), pack-reused 2439
Receiving objects: 100% (3160/3160), 15.40 MiB | 3.09 MiB/s, done.
Resolving deltas: 100% (1859/1859), done.
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
Cloning into 'turtlebot3'...
remote: Enumerating objects: 6481, done.
remote: Total 6481 (delta 0), reused 0 (delta 0), pack-reused 6481
Receiving objects: 100% (6481/6481), 119.95 MiB | 4.39 MiB/s, done.
Resolving deltas: 100% (4020/4020), done.
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_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 | 790.00 KiB/s, done.
Resolving deltas: 100% (170/170), done.
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$ ls
CMakeLists.txt  get_destination  turtlebot3  turtlebot3_msgs  turtlebot3_simulations
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$
```

حال باید سرویس مورد نظر را درون پکیج get_destination ایجاد کنیم. برای این کار کل ورک اسپیس را درون vs code باز میکنیم. از دورن پکیج get destination بايد دو فایل pakages.xml و CMakeLists.txt را باز کنیم تا تغییرات لازم را اعمال کنیم:



```

<?xml version="1.0"?>
<package format="2">
  <name>get_destination</name>
  <version>0.0.0</version>
  <description>The get_destination package</description>
  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example: -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="mahdi@todo.todo">mahdi</maintainer>

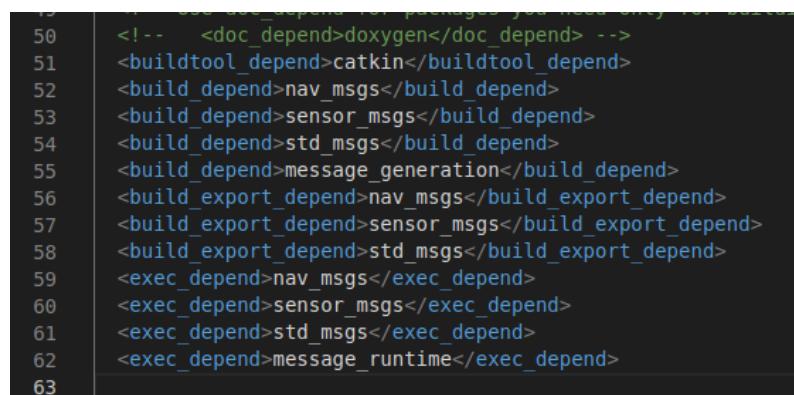
  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!-- BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLV2.1, LGPLV3 -->
  <license>TODO</license>

  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/get_destination</url> -->

  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->

```

۱- در خط ۵۴ اینتر زده و بک build_depend جدید ایجاد کنیم و در خط ۶۲ هم یک exec_depend ایجاد میکنیم:

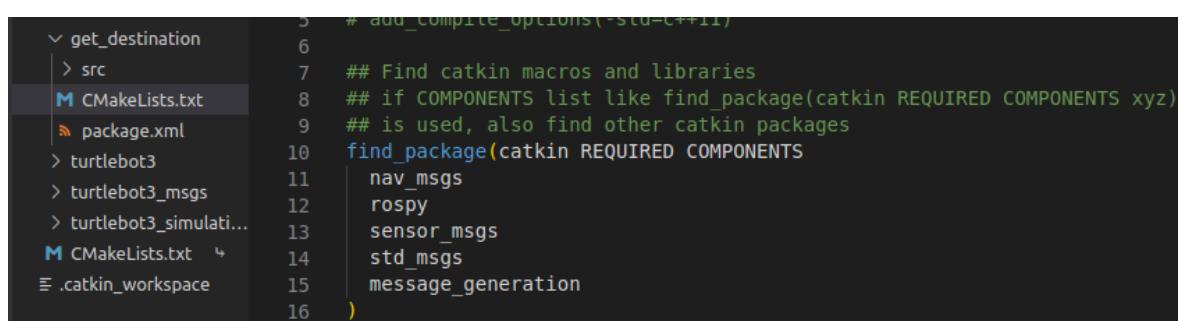


```

<!-- <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>nav_msgs</build_depend>
<build_depend>sensor_msgs</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>message_generation</build_depend>
<build_export_depend>nav_msgs</build_export_depend>
<build_export_depend>sensor_msgs</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>nav_msgs</exec_depend>
<exec_depend>sensor_msgs</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>message_runtime</exec_depend>

```

۲- سپس در خط ۱۵ هم در CMakeLists.txt message_generation را میگذاریم:



```

# add_compile_options(-std=c++11)
## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  nav_msgs
  rospy
  sensor_msgs
  std_msgs
  message_generation
)

```

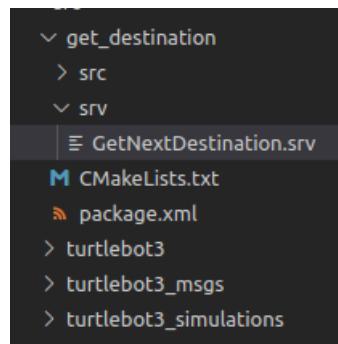
۳- خطوط ۵۹ تا ۶۳ که مربوط به سرویس هست را `uncomment` میکنیم و تغییرات را اعمال میکنیم:

```
57  
58     ## Generate services in the 'srv' folder  
59     add_service_files(  
60         FILES  
61         GetNextDestination.srv  
62     )  
63 
```

۴- خطوط ۷۱ تا ۷۴ هم `uncomment` شوند:

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

۵- برای آنکه کارمان تکمیل شود، در همان پوشه `get_destination` 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.

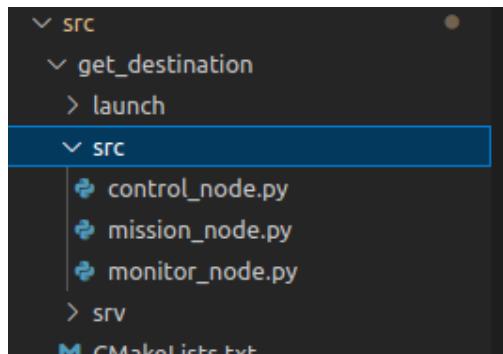
```
CMakeLists.txt package.xml GetNextDestination.srv  
src > get_destination > srv > GetNextDestination.srv  
1   float64 current_x  
2   float64 current_u  
3   ---  
4   float64 next_x  
5   float64 next_y
```

۷- حال به پوشه hw2_ws در ترمینال برگشته و catkin_make میکنیم.

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ catkin_make
Base path: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws
Source space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src
Build space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/install
#####
##### Running command: "cmake /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/devel -DCMAKE_INSTALL_PREFIX=/home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_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++

```

حال به سراغ ساخت نودها میرویم. برای این منظور در فolder src مربوط به get_destination و دو فایل پایتون با نام های control_node.py و mission_node.py میسازیم. همچنین یک نod monitor_node.py نیز برای نمایش مسیر ربات در rviz میسازیم.



حال در گام بعد به سراغ نوشتن کد مربوط به هریک از نودها میرویم. ابتدا کد مربوط به mission را مینویسیم. کد آن به صورت زیر است. در آن یک تابع get_next_dest را داریم که به عنوان call_back مینویسیم. در این متده با توجه به اینکه میخواهیم مقدار پردازش کمینه باشد. ابتدا x را در برای سرویس ما میبایشد. در این متده با توجه به اینکه میخواهیم مقدار پردازش کمینه باشد. ابتدا x را در بازه ۱۰-۱۰ به طور رندوم تولید میکنیم و برای y با توجه به اینکه استیت بعدی باید حداقل ۵ متری این یکی باشد پس باید در بیرون از یک دایره با شعاع ۵ متری آن باشد. روش دیگر این بود که دائما عدد رندوم تولید کنیم به نحوی که فاصله بیشتر از ۵ شود ولی خب ممکن است بدشانس باشیم و خیلی طولانی شود.

```

#!/usr/bin/python3

import rospy
from get_destination.srv import GetNextDestination, GetNextDestinationResponse
import random
import math

class Mission():

    def get_next_dest(self, current_loc):
        cur_x = current_loc.current_x
        cur_y = current_loc.current_y

        rospy.loginfo(f"Service : NEW CALL: {cur_x, cur_y}")

        next_x = random.uniform(-10,10)
        if abs(next_x - cur_x)>=5:
            next_y = random.uniform(-10,10)
        else:
            y_dist = (25 - (next_x - cur_x)**2)**0.5
            valid_min_y1 = cur_y + y_dist
            valid_min_y2 = cur_y - y_dist
            if valid_min_y1 >=10:
                next_y = random.uniform(-10, valid_min_y2)
            elif valid_min_y2 <= -10:
                next_y = random.uniform(valid_min_y1, 10)
            else:
                next_y = random.choice([random.uniform(-10, valid_min_y2),
                                         random.uniform(valid_min_y1, 10)])
        result = GetNextDestinationResponse()
        result.next_x = next_x
        result.next_y = next_y

        return result

def listener():

    rospy.init_node('mission', anonymous=True)
    mi = Mission()
    s = rospy.Service('get_destination', GetNextDestination, mi.get_next_dest)

    rospy.spin()

if __name__ == '__main__':
    listener()

```

کد بعدی مربوط به کنترلر میباشد. در این قسمت، در تابع `init` خود نود را `initialize` کرده و سپس این نود را به عنوان `client` برای سرویس معرفی میکنیم و در گام بعد این نود باید سرعت خطی و زاویه ای را برای ربات در قالب `twist` پابلیش کند. همچنین در این قسمت پارامترهای مختلف را معرفی میکنیم و `linear_speed` را کاربر میتواند به عنوان ورودی از طریق `param` در لانچ فایل بدهد. سپس یک تابع `get_pose` و `get_heading` داریم که به ترتیب زاویه `yaw` ربات را به درجه و دیگری مکان `X` و `Y` ربات را بر حسب متر با خواندن مقادیر از تاپیک `odom` برمیگرداند.

در تابع `ran` که کل کار را در بر دارد، ابتدا لازم است تا یک پیام ریکوئست `bsaziyem` که شامل مکان فعلی ربات است و آن را برای سرویس بفرستیم و در جواب آن به ما `next_x` و `next_y` را بدهد. در گام بعد با توجه به مکان فعلی و مکان بعدی باید زاویه مناسب را بیابیم که به آن جهت بچرخیم و رو بروی آن قرار بگیریم. برای این منظور باید دائما هدینگ ربات را بگیریم تا ببینیم در جهت مناسب قرار گرفتیم یا نه. سپس لازم است تا مستقیم به سمت هدف حرکت کرده. برای اینکه ببینیم به مکان موردنظر رسیدیم یا نه میتوان فاصله نقطه فعلی تا هدف را حساب کرده و هدف را رسیدن به آن فاصله بگذاریم. سپس به مرور با جلو رفتن ربات دائما مکان ربات را گرفته و فاصله اش را تا نقطه مبدا حساب میکنیم.

لازم به ذکر است وقتی `Twist()` را پابلیش میکنیم همه مقادیرش + است و لذا ربات می ایستد. دذر انتهای هر ایتریشن هم اror که فاصله نقطه هدف تا نقطه ای هست که به آن رسیدیم را حساب میکنیم و به یک لیست اضافه مینماییم. در پایان کار بین اعضای لیست میانگین گرفته و به عنوان ارور نهایی اعلام میکنیم. کد آن در ادامه آمده است.

```
#!/usr/bin/python3

import rospy
import tf
import math
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from get_destination.srv import GetNextDestination, GetNextDestinationRequest

class Controller():
    def __init__(self) -> None:
        rospy.init_node("controller" , anonymous=False)
        self.calc_client = rospy.ServiceProxy('get_destination' , GetNextDestination)
        self.cmd_publisher = rospy.Publisher('/cmd_vel' , Twist , queue_size=10)

        self.linear_speed = rospy.get_param("/controller/linear_speed") # m/s
```

```

        self.angular_speed = 0.08
        self.epsilon = 0.17
        self.current_x = 0
        self.current_y = 0
        self.yaw = 90

# heading of the robot
def get_heading(self):
    # waiting for the most recent message from topic /odom
    msg = rospy.wait_for_message("/odom", Odometry)

    orientation = msg.pose.pose.orientation

    # convert quaternion to odom
    roll, pitch, yaw = tf.transformations.euler_from_quaternion((
        orientation.x, orientation.y, orientation.z, orientation.w
    ))

    return math.degrees(yaw)

# position of robot
def get_pose(self):
    # waiting for the most recent message from topic /odom
    msg = rospy.wait_for_message("/odom", Odometry)

    position = msg.pose.pose.position

    return position.x, position.y

def run(self):
    error_array = []
    counter = 0
    while not rospy.is_shutdown():
        counter += 1
        if counter == 6:
            break
        # first we should send our current location to service for
        # getting next destination
        req = GetNextDestinationRequest()
        req.current_x = self.current_x
        req.current_y = self.current_y
        rospy.loginfo(f"Client : current pose: {req.current_x, req.current_y}")
        resp = self.calc_client(req)
        rospy.loginfo(f"Client : Goal pose : {resp.next_x, resp.next_y}")

        # according to the next destination we should calculate our desired angle
        # for finding desired angle we should find next location relative robot
position

```

```

        new_next_x = resp.next_x - self.current_x
        new_next_y = resp.next_y - self.current_y
        sin = new_next_y/math.sqrt(new_next_x**2+new_next_y**2)
        cos = new_next_x/math.sqrt(new_next_x**2+new_next_y**2)
        sin_inv = math.degrees(math.asin(sin))

        if cos >= 0:
            desired_angle = sin_inv
        elif cos <= 0 and sin >=0:
            desired_angle = 180 - sin_inv
        else:
            desired_angle = -180 - sin_inv

        rospy.loginfo(f"Client: desired angle : {desired_angle}")

        # our robot start to rotate
        twist = Twist()

        if 90<=desired_angle<=180 or -180<=desired_angle<=-90:
            twist.angular.z = self.angular_speed
        else:
            twist.angular.z = - self.angular_speed

        while abs(self.yaw - desired_angle) >= self.epsilon:
            self.yaw = self.get_heading()
            self.cmd_publisher.publish(twist)

        # our robot reached to its desired angle and now should be stopped
        self.cmd_publisher.publish(Twist())

        rospy.sleep(1)

        # now we should go straight to reach our destination
        twist = Twist()
        twist.linear.x = self.linear_speed
        self.cmd_publisher.publish(twist)
        desired_distance = math.sqrt((self.current_x-resp.next_x)**2 +
        (self.current_y-resp.next_y)**2)
        prev_x = self.current_x
        prev_y = self.current_y
        walked_distance = math.sqrt((self.current_x-prev_x)**2 + (self.current_y-
        prev_y)**2)

        while abs(desired_distance-walked_distance) >= self.epsilon:
            self.current_x, self.current_y = self.get_pose()
            walked_distance = math.sqrt((self.current_x-prev_x)**2 + (self.current_y-
            prev_y)**2)

```

```

        self.cmd_publisher.publish(Twist())

        # calculate error and append to list
        error_array.append(math.sqrt((self.current_x-
resp.next_x)**2+(self.current_y-resp.next_y)**2))

        rospy.sleep(1)

        rospy.loginfo(f"TOTAL ERROR (in meter): {sum(error_array)/len(error_array)}")

if __name__ == "__main__":
    controller = Controller()

    controller.run()

```

در monitor کد پایتون مربوط به رسم مسیری که ربات آن را طی میکند میباشد که از تاپیک path استفاده میکند. مکان هایی که میروند را در قالب یک آرایه یا لیست ذخیره میکنیم و در rviz آن را نشان میدهیم.

```

#!/usr/bin/python3

import rospy
from nav_msgs.msg import Odometry, Path
from geometry_msgs.msg import PoseStamped
class PathMonitor:

    def __init__(self) -> None:

        rospy.init_node("monitor" , anonymous=False)

        self.path = Path()
        self.odom_subscriber = rospy.Subscriber("/odom" , Odometry , callback=self.odom_callback)
        self.path_publisher = rospy.Publisher("/path" , Path , queue_size=10)

    def odom_callback(self, msg : Odometry):
        self.path.header = msg.header
        pose = PoseStamped()
        pose.header = msg.header
        pose.pose = msg.pose.pose
        self.path.poses.append(pose)
        self.path_publisher.publish(self.path)

if __name__ == "__main__":

```

```
path_monitor = PathMonitor()

rospy.spin()
```

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

```
chmod +x src/*.py
```

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/get_destination$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/get_destination$
```

سپس به سراغ نوشتن لانچ فایل میرویم. در همین پکیج get_destination لازم است تا یک فolder ایجاد کنیم و داخل آن یک get_destination.launch ایجاد کنیم. لانچ فایل به صورت زیر است:

```
<launch>

    <arg name="linear_speed" default="0.2"/>

    <include file="$(find turtlebot3_gazebo)/launch/my_empty_world.launch">
        <arg name="x_pos" value="0.0"/>
        <arg name="y_pos" value="0.0"/>
        <arg name="z_pos" value="0.0"/>
        <arg name="yaw" value="1.57075"/>
    </include>

    <node pkg="get_destination" type="mission_node.py" name="mission" output="screen"></node>

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

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

    <include file="$(find turtlebot_rviz_launchers)/launch/view_robot.launch"/>

</launch>
```

ابتدا یک لانچ فایل دیگر را include میکنیم و میگوییم در آن آرگومان های ورودی مثل مکان اولیه ربات و زاویه اولیه چه باشند . اصلا در کدام نقشه (که اینجا empty_world بود) را بالا بیاورد. و ربات و gazebo را آماده کند.

سپس mission_node.py را لازم است بالا بیاوریم. در نود کنترلر پارامتر linear_speed که میتواند توسط کاربر وارد شود را مینویسیم. در نهایت لازم است نود monitor که برای رسم مسیر لازم بود بالا بیاوریم و همچنین لانج فایل مربوط به rviz را هم صدا بزنیم.

برای rviz لازم است که مانند پکیج‌هایی که در ابتدای کار در ورک اسپیس نصب کردیم، این پکیج را نیز به کمک دستور زیر نصب کنیم:

- git clone -b indigo https://github.com/turtlebot/turtlebot_interactions.git

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
- rosrun get_destination get_destination.launch linear_speed:=0.4

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ rosrun get_destination get_destination.launch linear_speed:=0.4
... logging to /home/mahdi/.ros/log/2c3eff22-e08a-11ed-a292-675131c19bcf/rosrun-mahdi-4634.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.
started rosrun server http://mahdi:33335/

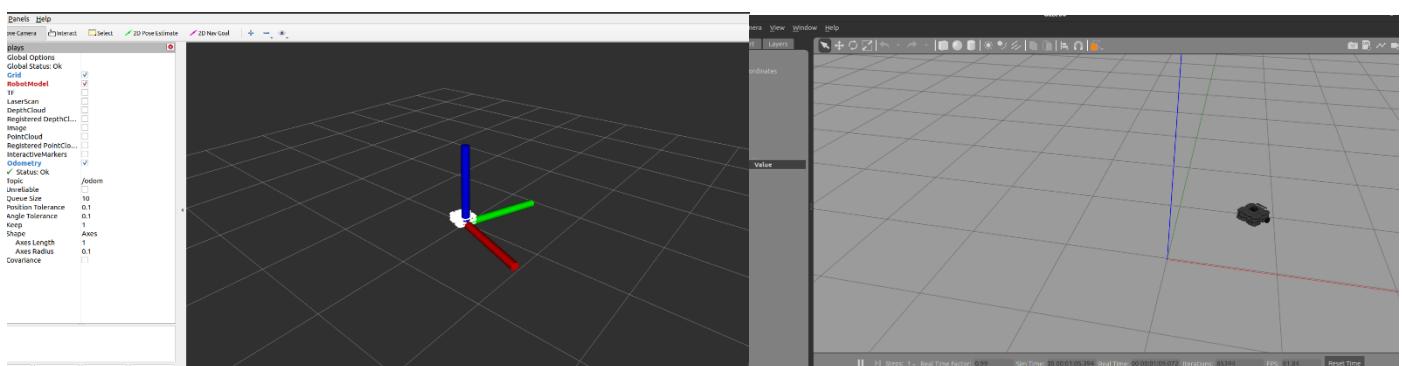
```

SUMMARY

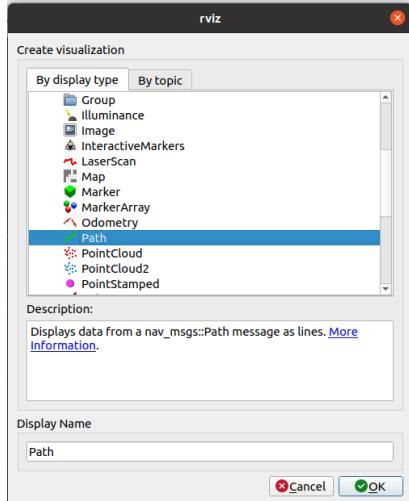
```
=====
PARAMETERS
  * /controller/linear_speed: 0.2
  * /gazebo/enable_ros_network: True
  * /robot_description: <?xml version="1....
  * /rosdistro: noetic
  * /rosversion: 1.15.15
  * /use_sim_time: True

NODES
  /
    controller (get_destination/control_node.py)
    gazebo (gazebo_ros/gzserver)
    gazebo_gui (gazebo_ros/gzclient)
    mission (get_destination/mission_node.py)
    monitor (get_destination/monitor_node.py)
    rviz (rviz/rviz)
    spawn_urdf (gazebo_ros/spawn_model)
```

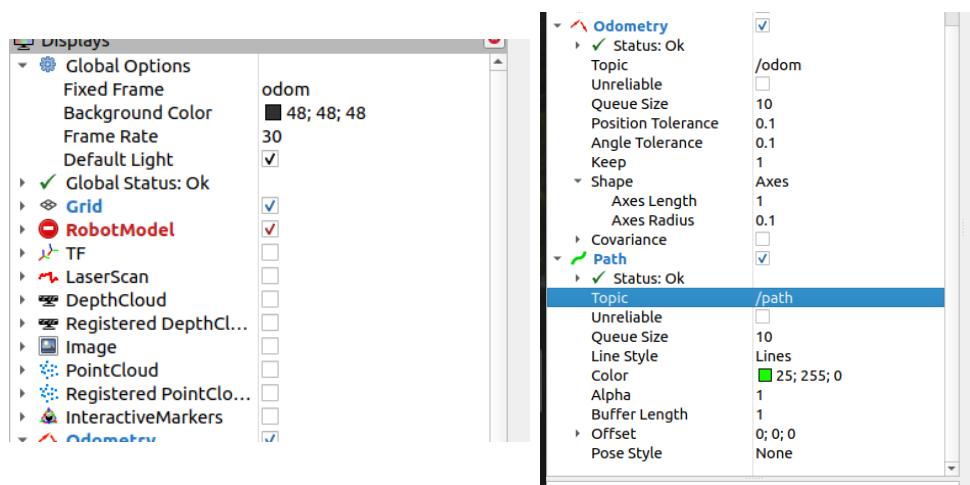
در تصویر سمت راست نمونه‌ای از rviz و در چپ نمونه‌ای از تصویر در gazebo را میبینید.



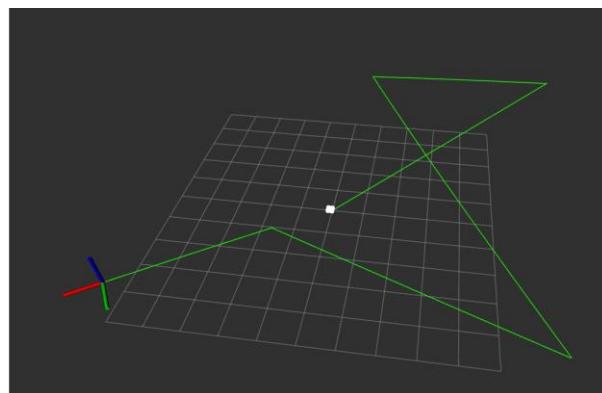
سپس باید در rviz تاپیک مد نظرمان را معرفی کنیم که path را رسم کند. برای این کار باید ابتدا path را اضافه کنیم:



سپس تنظیمات را به صورت زیر تنظیم کنیم:



در نهایت شکلی زیر را مشاهده خواهیم کرد و خطای مورد نظر پس از ۵ ایتریشن در ترمینال نمایش داده میشود:



برای مثال برای سرعت خطی ۰.۴ متر بر ثانیه خطای برابر ۰.۵ متر به طور میانگین میباشد:

```
[spawn_urdf-6] process has finished cleanly
log file: /home/mahdi/.ros/log/2c3eff22-e08a-11ed-a292-675131c19bcf/spawn_urdf-6*.log
context mismatch in svga_surface_destroy
context mismatch in svga_surface_destroy
[INFO] [1682112086.622072, 120.444000]: Client : current pose: (7.702108884819241, 9.322501312706734)
[INFO] [1682112086.624806, 120.447000]: Service : NEW CALL: (7.702108884819241, 9.322501312706734)
[INFO] [1682112086.626073, 120.448000]: Client : Goal pose : (0.03464052401505491, 8.506429012614266)
[INFO] [1682112086.626948, 120.449000]: Client: desired angle : -173.92470494304666
[INFO] [1682112151.215398, 184.448000]: Client : current pose: (0.21991050590007083, 8.881303573321535)
[INFO] [1682112151.217776, 184.450000]: Service : NEW CALL: (0.21991050590007083, 8.881303573321535)
[INFO] [1682112151.218797, 184.451000]: Client : Goal pose : (6.298148735781524, -4.785266674286365)
[INFO] [1682112151.219526, 184.452000]: Client: desired angle : -66.02278450418933
[INFO] [1682112247.589534, 280.017000]: Client : current pose: (6.7013587523467155, -4.3554567363482946)
[INFO] [1682112247.592484, 280.020000]: Service : NEW CALL: (6.7013587523467155, -4.3554567363482946)
[INFO] [1682112247.593595, 280.021000]: Client : Goal pose : (-1.3320122870171183, 0.06271281533481066)
[INFO] [1682112247.594359, 280.022000]: Client: desired angle : 151.1902166600912
[INFO] [1682112334.096781, 365.778000]: Client : current pose: (-1.6242123524299368, -1.058533603540513)
[INFO] [1682112334.099380, 365.780000]: Service : NEW CALL: (-1.6242123524299368, -1.058533603540513)
[INFO] [1682112334.100496, 365.781000]: Client : Goal pose : (-8.061470038375207, -5.1295159266465635)
[INFO] [1682112334.101203, 365.782000]: Client: desired angle : -147.69035444862072
[INFO] [1682112387.021368, 417.680000]: TOTAL ERROR (in meter): 0.5591287785048014
[controller-3] process has finished cleanly
log file: /home/mahdi/.ros/log/2c3eff22-e08a-11ed-a292-675131c19bcf/controller-3*.log
```

گام اول - نتایج

۱- **ویدیوی کارکرد ربات برای هر سرعت خطی:** در فolder مربوطه ۳ فیلم که هر یک مربوط به یکی از سرعت‌های ۰.۲ و ۰.۴ و ۰.۸ متر بر ثانیه است، قرار گرفته است. دقت شود به علت خواسته سوال که باید حداقل ۵ ایترشن نمایش داده شود و همچنین برای بالا بردن دقت، که مجبور به کم کردن سرعت زاویه ای شدیم، زمان فیلم‌ها با عرض پوزش بیشتر شد.

۲- **نمای مسیر ربات در شبیه ساز Rviz به ازای هر سرعت خطی:**

برای این منظور در ترمینال در ورودی لازم است سرعت خطی را بدهیم. با توجه به آنکه در دستور کار گفته شده بود حداقل ۵ بار فرآیند یافتن مقصد جدید ادامه یابد، من هم برای طولانی تر نشدن ویدیوها دقیقا همان ۵ بار را ست کردم. ابتدا لازم است که ترمینال را سورس کنیم و نوع ربات را اکسپورت کنیم:

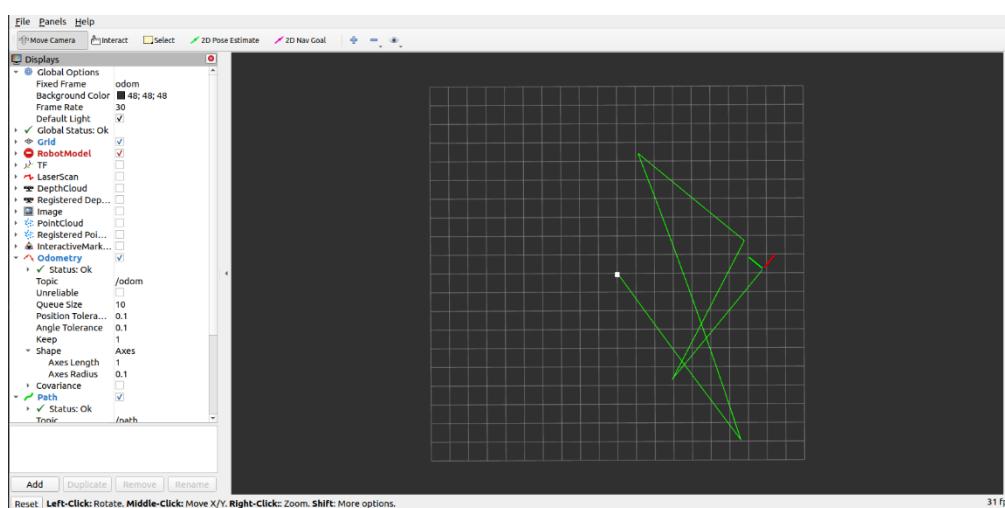
```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ . devel/setup.bash  
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ export TURTLEBOT3_MODEL=waffle  
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ █
```

• برای سرعت ۰.۲ m/s

ابتدا دستور زیر را وارد میکنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ roslaunch get_destination get_destination.launch linear_speed:=0.2  
... logging to /home/mahdi/.ros/log/cab33948-e1b6-11ed-ab6b-279cef036b5f/roslaunch-mahdi-161756.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.  
started roslaunch server http://mahdi:40919/  
  
SUMMARY  
=====
```

در نهایت نمای مسیر ربات در Rviz پس از ۵ iteration به صورت زیر خواهد بود:

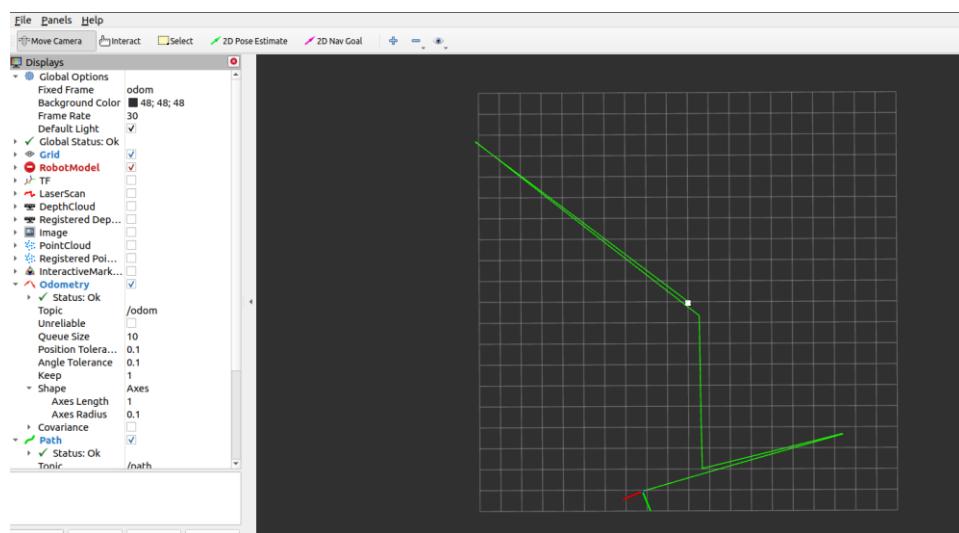


• برای سرعت : 0.4 m/s

ابتدا دستور زیر را وارد میکنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ roslaunch get_destination get_destination.launch linear_speed:=0.4
... logging to /home/mahdi/.ros/log/392d569e-e1ed-ab6b-279cef036b5f/roslauch-mahdi-358332.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.
```

در نهایت نمای مسیر ربات در Rviz پس از ۵ iteration به صورت زیر خواهد بود:

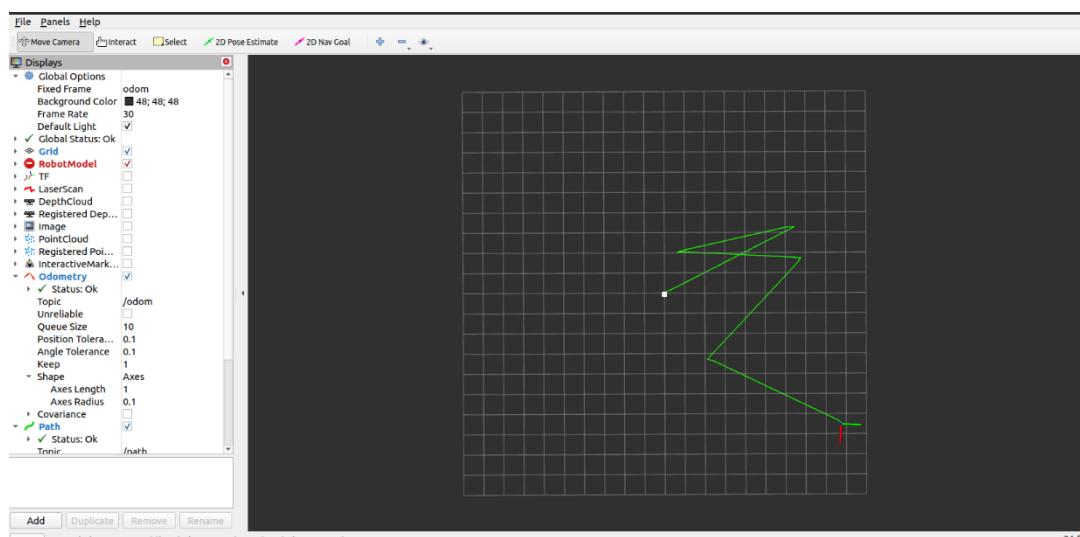


• برای سرعت : 0.8 m/s

ابتدا دستور زیر را وارد میکنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ roslaunch get_destination get_destination.launch linear_speed:=0.8
... logging to /home/mahdi/.ros/log/089ec290-e1d5-ab6b-279cef036b5f/roslauch-mahdi-383486.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.
```

در نهایت نمای مسیر ربات در Rviz پس از ۵ iteration به صورت زیر خواهد بود:



۳- خطای انحراف از مقصد به ازای هر سرعت خطی. برای این کار می توانید پس از ایست ربات فاصله اش را تامقصد کنونی به دست آورده و درنهایت میانگین فواصل را گزارش دهید.

طبق راهکار ارائه شده در کد نیز به این صورت عمل کردیم که پس از پیمودن مسافت مدنظر، فاصله مقصد مورد نظر را از مقصدی که فعلاً به آن رسیدیم، میابیم. در انتهای هر ایتریشن با توجه به توضیحات ارور را حساب کردیم و به یک لیست آن را اضافه کردیم. در نهایت جمع ارورها را بر تعدادشان تقسیم کرده تا میانگین خطای خطا به دست آید.

```
107     # calculate error and append to list
108     error_array.append(math.sqrt((self.current_x-resp.next_x)**2+(self.current_y-resp.next_y)**2))
109
110     rospy.sleep(1)
111
112     rospy.loginfo("TOTAL ERROR (in meter): %s", sum(error_array)/len(error_array))
113
114
```

• برای سرعت 0.2 m/s

همانطور که مشاهده میکنید برای این حالت میانگین خطای خطا برابر 0.68 m میباشد.

```
[ INFO] [1682241075.795729606, 0.136000000]: DiffDrive(ns = //): Advertise odom on odom
[spawn_urdf-4] process has finished cleanly
log file: /home/mahdi/.ros/log/cab33948-e1b6-11ed-ab6b-279cef036b5f/spawn_urdf-4*.log
context mismatch in svga_surface_destroy
context mismatch in svga_surface_destroy
[INFO] [1682241159.733163, 81.478000]: Client : current pose: (-8.915592939008288, -6.5823663698770645)
[INFO] [1682241159.739989, 81.484000]: Service : NEW CALL: (-8.915592939008288, -6.5823663698770645)
[INFO] [1682241159.741847, 81.486000]: Client : Goal pose : (6.858567932357602, -2.2308379218260015)
[INFO] [1682241159.743437, 81.487000]: Client: desired angle : 15.422282136927956

[INFO] [1682241275.197592, 192.543000]: Client : current pose: (6.33945588024934, -1.1659330250754705)
[INFO] [1682241275.203145, 192.549000]: Service : NEW CALL: (6.33945588024934, -1.1659330250754705)
[INFO] [1682241275.207071, 192.552000]: Client : Goal pose : (2.1907522023853225, -7.262549332370135)
[INFO] [1682241275.209295, 192.554000]: Client: desired angle : -124.23500639081001

[INFO] [1682241357.071352, 270.879000]: Client : current pose: (1.7889967525326658, -6.7396011883967235)
[INFO] [1682241357.079567, 270.886000]: Service : NEW CALL: (1.7889967525326658, -6.7396011883967235)
[INFO] [1682241357.082516, 270.887000]: Client : Goal pose : (-5.896469038446077, -2.99536349915728)
[INFO] [1682241357.084699, 270.889000]: Client: desired angle : 154.0253866865704

[INFO] [1682241453.254663, 362.373000]: Client : current pose: (-5.669228770583805, -2.9374627918982057)
[INFO] [1682241453.258860, 362.376000]: Service : NEW CALL: (-5.669228770583805, -2.9374627918982057)
[INFO] [1682241453.261199, 362.379000]: Client : Goal pose : (0.5028136158375567, -7.750100469184571)
[INFO] [1682241453.262375, 362.380000]: Client: desired angle : -37.94525265179532

[INFO] [1682241528.420588, 433.779000]: TOTAL ERROR (in meter): 0.6381606742134585
[controller-6] process has finished cleanly
log file: /home/mahdi/.ros/log/cab33948-e1b6-11ed-ab6b-279cef036b5f/controller-6*.log
```

• برای سرعت 0.4 m/s

همانطور که مشاهده میکنید برای این حالت میانگین خطابرابر 1.3 m میباشد. که نسبت به قبل بیشتر شد.

```
[spawn_urdf-4] process has finished cleanly
log file: /home/mahdi/.ros/log/392d569e-e1d3-11ed-ab6b-279cef036b5f/spawn_urdf-4*.log
context mismatch in svga_surface_destroy
context mismatch in svga_surface_destroy
[INFO] [1682253333.726377, 45.294000]: Client : current pose: (7.605210894456801, 10.064089831819354)
[INFO] [1682253333.731787, 45.298000]: Service : NEW CALL: (7.605210894456801, 10.064089831819354)
[INFO] [1682253333.733911, 45.300000]: Client : Goal pose : (0.24279872801000124, -1.2247536569601145)
[INFO] [1682253333.735322, 45.302000]: Client: desired angle : -123.11177971384026

[INFO] [1682253416.081761, 124.726000]: Client : current pose: (-0.5953338132599394, -0.4368790986317234)
[INFO] [1682253416.088893, 124.732000]: Service : NEW CALL: (-0.5953338132599394, -0.4368790986317234)
[INFO] [1682253416.092463, 124.734000]: Client : Goal pose : (-8.062011055461633, -0.10361174064574996)
[INFO] [1682253416.094251, 124.737000]: Client: desired angle : 177.42368384204676

[INFO] [1682253511.867890, 216.489000]: Client : current pose: (-7.852946811750873, -0.6505274431517092)
[INFO] [1682253511.872091, 216.494000]: Service : NEW CALL: (-7.852946811750873, -0.6505274431517092)
[INFO] [1682253511.874034, 216.496000]: Client : Goal pose : (-6.269769147938752, 2.6524731178135985)
[INFO] [1682253511.875377, 216.497000]: Client: desired angle : -88.36180553458755

[INFO] [1682253595.401703, 296.661000]: Client : current pose: (-6.383003881813028, -7.16987312254785)
[INFO] [1682253595.408951, 296.666000]: Service : NEW CALL: (-6.383003881813028, -7.16987312254785)
[INFO] [1682253595.410735, 296.668000]: Client : Goal pose : (-6.269769147938752, 2.6524731178135985)
[INFO] [1682253595.415194, 296.672000]: Client: desired angle : 89.3391576631728

[INFO] [1682253663.141639, 361.492000]: TOTAL ERROR (in meter): 1.3061005457169994
[controller-6] process has finished cleanly
log file: /home/mahdi/.ros/log/392d569e-e1d3-11ed-ab6b-279cef036b5f/controller-6*.log
```

• برای سرعت 0.8 m/s

همانطور که مشاهده میکنید برای این حالت میانگین خطابرابر 1.54 m میباشد. که نسبت به قبل بیشتر شد.

```
log file: /home/mahdi/.ros/log/ea49499a-e1d0-11ed-ab6b-279cef036b5f/spawn_urdf-4*.log
context mismatch in svga_surface_destroy
context mismatch in svga_surface_destroy
[INFO] [1682252343.394069, 46.475000]: Client : current pose: (3.066119867520944, -6.054887762456496)
[INFO] [1682252343.399542, 46.477000]: Service : NEW CALL: (3.066119867520944, -6.054887762456496)
[INFO] [1682252343.402490, 46.482000]: Client : Goal pose : (2.9853944502500447, -1.0079070474683816)
[INFO] [1682252343.404613, 46.483000]: Client: desired angle : 90.91635605555679

[INFO] [1682252388.973994, 90.572000]: Client : current pose: (2.182863883934378, -1.2105242135771603)
[INFO] [1682252388.979997, 90.575000]: Service : NEW CALL: (2.182863883934378, -1.2105242135771603)
[INFO] [1682252388.981920, 90.575000]: Client : Goal pose : (4.004476954752274, -6.007300291629765)
[INFO] [1682252388.984047, 90.581000]: Client: desired angle : -69.20533662328639

[INFO] [1682252450.311110, 149.439000]: Client : current pose: (1.771448822634434, -6.1855798413512)
[INFO] [1682252450.316513, 149.444000]: Service : NEW CALL: (1.771448822634434, -6.1855798413512)
[INFO] [1682252450.318863, 149.447000]: Client : Goal pose : (-3.897900169953663, -3.7548035491049925)
[INFO] [1682252450.321634, 149.448000]: Client: desired angle : 156.79239995275503

[INFO] [1682252510.097218, 206.934000]: Client : current pose: (-2.9450231739948407, -2.4367553738054712)
[INFO] [1682252510.101434, 206.939000]: Service : NEW CALL: (-2.9450231739948407, -2.4367553738054712)
[INFO] [1682252510.103651, 206.942000]: Client : Goal pose : (-4.377772482253672, -8.954453386316791)
[INFO] [1682252510.105529, 206.943000]: Client: desired angle : -102.39782363721967

[INFO] [1682252554.792892, 249.807000]: TOTAL ERROR (in meter): 1.540194839706074
[controller-6] process has finished cleanly
log file: /home/mahdi/.ros/log/ea49499a-e1d0-11ed-ab6b-279cef036b5f/controller-6*.log
```

همانطور که دیده میشود با زیاد شدن سرعت میزان نیز افزایش میابد چراکه میزان لغرض بیشتر شده و دقت odometry پایین میاید. البته در مقادیر فوق، به علت رندم بودن مقصدها، میزان مسافت طی شده هم تاثیر دارد و ممکن است در یک اجرا نقاط مقصد نزدیک به هم باشند ولی به هر حال آنچه مشهود است با افزایش سرعت خطی دقت کاهش میابد. اگر در ویدیو هم نگاه شود حتی در هنگام شروع به حرکت خطی، ربات یک دریفتی دارد و هر دوچرخ به یک اندازه نمیچرخد که دقیقاً روی مسیر تعیین شده بروند.

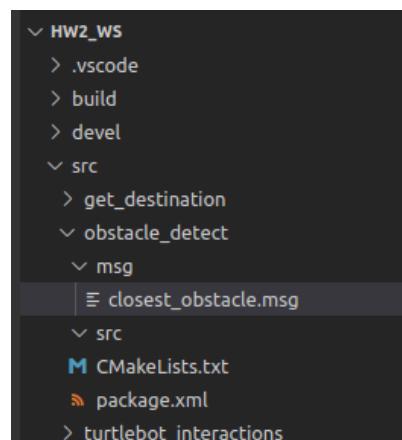
گام دوم - تشخیص مانع به کمک لایدار

حال لازم است که یک پکیج مثلا با نام get_destination در فolder سورس بسازیم. همچنین dependency های لازم را که ممکن است در نوشتن نودها و کد زنی نیاز شود به آن میدهیم:

- catkin_create_pkg obstacle_detect rospy std_msgs sensor_msgs nav_msgs

```
mahdi@mahdi:~$ cd Desktop/  
mahdi@mahdi:~/Desktop$ cd Robotics/project2/HW2_9731701/packages/hw2_ws/  
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ cd src/  
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$ catkin_create_pkg obstacle_detect rospy std_msgs sensor_msgs nav_msgs  
Created file obstacle_detect/package.xml  
Created file obstacle_detect/CMakeLists.txt  
Created folder obstacle_detect/src  
Successfully created files in /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/obstacle_detect. Please adjust the values in package.xml.  
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$
```

حال در گام بعد در مسیر ورک اسپیس میایم و vs code را باز میکنیم. لازم است در پوشه obstacle_detect ایجاد کنیم. سپس در آن باید custom message را ایجاد نماییم.



همچنین محتوای داخل این پیام به صورت زیر میباشد:

```
closest_obstacle.msg X  
src > obstacle_detect > msg > closest_obstacle.msg  
1 float64 distnace  
2 float64 direction  
3 |
```

حال لازم است تا تنظیمات مربوط به custom message را انجام دهیم. برای این منظور CMakeLists.txt و همچنین package.xml را باز میکنیم. برای این منظور میتوانید به قسمت beginner tutorials سطح در wiki.ros مراجعه میکنیم و قسمت CreatingMsgAndSrv (لینک) میرویم. براساس آن قرار است پیش

برویم.

اولین تغییر در package.xml میباشد. دو خطی که در خطوط ۴۰ و ۴۶ هستند را uncomment کنید.

```
31 <!-- The *depend tags are used to specify dependencies -->
32 <!-- Dependencies can be catkin packages or system dependencies -->
33 <!-- Examples: -->
34 <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
35 <!--   <depend>roscpp</depend> -->
36 <!--   Note that this is equivalent to the following: -->
37 <!--     <build_depend>roscpp</build_depend> -->
38 <!--     <exec_depend>roscpp</exec_depend> -->
39 <!-- Use build_depend for packages you need at compile time: -->
40 <build_depend>message_generation</build_depend>
41 <!-- Use build_export_depend for packages you need in order to build against this package: -->
42 <!--   <build_export_depend>message_generation</build_export_depend> -->
43 <!-- Use buildtool_depend for build tool packages: -->
44 <!--   <buildtool_depend>catkin</buildtool_depend> -->
45 <!-- Use exec_depend for packages you need at runtime: -->
46 <exec_depend>message_runtime</exec_depend>
47 <!-- Use test_depend for packages you need only for testing: -->
48 <!--   <test_depend>gtest</test_depend> -->
49 <!-- Use doc_depend for packages you need only for building documentation: -->
50 <!--   <doc_depend>doxygen</doc_depend> -->
```

حال باید به سراغ تغییرات داخل CMakeLists برویم. یکی اینکه در خط ۱۰ باید message_generation را اضافه نماییم.

```
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   nav_msgs
12   rospy
13   sensor_msgs
14   std_msgs
15   message_generation
16 )
17 |
```

تغییر دیگر در خط ۱۰۷ میباشد که باید uncomment نماییم و بقیه موارد را پاک کرده و CATKIN_DEPENDS را جلوی message_runtime اضافه نماییم.

```
98 #####
99 ## catkin specific configuration ##
100 #####
101 ## The catkin_package macro generates cmake config files for your package
102 ## Declare things to be passed to dependent projects
103 ## INCLUDE_DIRS: uncomment this if your package contains header files
104 ## LIBRARIES: libraries you create in this project that dependent projects also need
105 ## CATKIN_DEPENDS: catkin_packages dependent projects also need
106 ## DEPENDS: system dependencies of this project that dependent projects also need
107 catkin_package(
108   CATKIN_DEPENDS message_runtime
109 )
110 |
```

سپس به دنبال `closest_obstacle.msg` و آن را `uncomment` میگردیم و میکنیم و `add_message_files` را به جای `Message2.msg` و `Message1.msg` میگذاریم.

```
50
51     ## Generate messages in the 'msg' folder
52     add_message_files(
53         FILES
54         closest_obstacle.msg
55     )
56
```

حال باید `comment` را از `generate_message` در بیاریم.

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

کار دیگری که باید بکنیم اینه که در خط ۱۱۵ اون `include` را هم `uncomment` کنیم

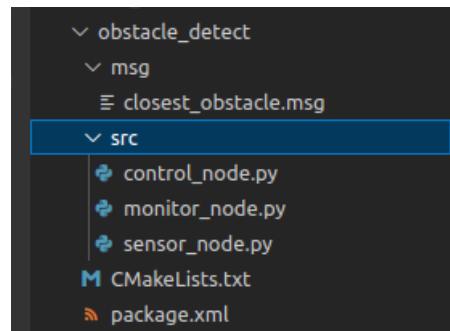
```
113
114     ## Specify additional locations of header files
115     ## Your package locations should be listed before other locations
116     include_directories(
117         include
118         ${catkin_INCLUDE_DIRS}
119     )
```

حال به علت تغییراتی که در این دو فایل اعمال کردیم لازم است در `catkin_make` work space یک بار را بزنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ catkin_make
Base path: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws
Source space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src
Build space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/install
#####
##### Running command: "cmake /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/
2/HW2_9731701/packages/hw2_ws/devel -DCMAKE_INSTALL_PREFIX=/home/mahdi/Desktop/Robotics/project
/home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/build"
#####
-- Using CATKIN_DEVEL_PREFIX: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws
-- 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 "
```

حال یک بار هم `vscode` را بسته و دوباره باز میکنیم. حال میتوان `import` را انجام داد. ممکن است هنوز مسیج ساخته شده را در غیر اینصورت نشناشد.

خب حال به سراغ ساخت نودها میرویم. در فایل src لازم است ۳ نود ایجاد کنیم. یکی sensor_node.py که باید با خواندن laser scan کمترین فاصله از موانع را پیدا کرده و در تاپیک مناسب برگرداند. یکی که وظیفه کنترل ربات اعم از چرخش مناسب و حرکت روبه جلو را دارد. و یک نود control_node.py که برای نمایش مسیر ربات در Rviz لازم میباشد.



حال به سراغ نوشتن کد در sensor_node.py میرویم:

```
#!/usr/bin/python3

import rospy
from obstacle_detect.msg import closest_obstacle
from sensor_msgs.msg import LaserScan

class Sensor():

    def __init__(self) -> None:
        rospy.init_node('sensor', anonymous=True)
        self.pub = rospy.Publisher('ClosestObstacle', closest_obstacle, queue_size=10)
        self.laser_subscriber = rospy.Subscriber("/scan", LaserScan,
callback=self.laser_callback)
        self.rate = rospy.Rate(0.2) #Hz
        self.closest_msg = closest_obstacle()

    def laser_callback(self, msg: LaserScan):
        min_dist = min(msg.ranges)
        direction = msg.ranges.index(min_dist)
        self.closest_msg.distance = min_dist
        self.closest_msg.direction = direction

    def run(self):

        while not rospy.is_shutdown():
            rospy.loginfo(f' closest obstacle {self.closest_msg}')
            self.pub.publish(self.closest_msg)
            self.rate.sleep()
```

```

if __name__ == '__main__':
    my_sensor = Sensor()
    my_sensor.run()

```

کار نود فوق این است مقدار minimum فاصله ای که از آرایه ranges از پیام scan دریافت میکند را به همراه ایندکس این فاصله در این آرایه که نشان دهنده همان زاویه مورد نظر است را به ترتیب در فیلدهای publish و direction جاگذاری کند و distance کند.

حال به سراغ نود control_node.py میرویم:

```

#!/usr/bin/python3

import rospy
import tf
import math
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from obstacle_detect.msg import closest_obstacle

class Controller():
    def __init__(self) -> None:
        rospy.init_node("controller" , anonymous=False)
        rospy.Subscriber('ClosestObstacle', closest_obstacle, self.closest_callback)
        self.cmd_publisher = rospy.Publisher('/cmd_vel' , Twist , queue_size=10)

        self.linear_speed = 0.2 # m/s
        self.angular_speed = 0.2

        self.closest_obstacle = closest_obstacle()

    def closest_callback(self, data):
        self.closest_obstacle = data

    # heading of the robot
    def get_heading(self):
        # waiting for the most recent message from topic /odom
        msg = rospy.wait_for_message("/odom" , Odometry)

        orientation = msg.pose.pose.orientation

        # convert quaternion to odom
        roll, pitch, yaw = tf.transformations.euler_from_quaternion((
            orientation.x ,orientation.y ,orientation.z ,orientation.w

```

```

        )))

    return math.degrees(yaw)

def move(self, mode, direction=0):

    twist = Twist()
    self.cmd_publisher.publish(twist)
    rospy.sleep(1)

    if mode == "rotate":
        first_heading = self.get_heading()
        if direction <= 180:
            while abs(self.get_heading()-first_heading) < (180-direction):
                twist.angular.z = -self.angular_speed
                self.cmd_publisher.publish(twist)
        else:
            while abs(self.get_heading()-first_heading) < (direction-180):
                twist.angular.z = -self.angular_speed
                self.cmd_publisher.publish(twist)

    elif mode == "go_straight":
        twist.linear.x = self.linear_speed
        self.cmd_publisher.publish(twist)

    else:
        rospy.loginfo('the mode is not correct')

def run(self):

    sensor_is_ready = False
    self.move("go_straight")

    while not rospy.is_shutdown():

        dist, direction = self.closest_obstacle.distnace,
        self.closest_obstacle.direction

        if dist == 0.0 and direction == 0.0 and not sensor_is_ready:
            continue
        sensor_is_ready = True

        if dist < 2 and not (120 < direction < 240):
            rospy.loginfo(f'controller : {dist, direction}')
            self.move("rotate", direction)
            self.move("go_straight")

```

```

if __name__ == "__main__":
    controller = Controller()
    controller.run()

```

در نود فوق وظیفه داریم، که پیام‌های publish شده از نود sensor را دائمًا بخوانیم چنانچه در زاویه دیدمان (که من برای مثال یک زاویه دید ۲۴۰ درجه تعریف کردم) مانعی بود که فاصله ارسال شده از سنسور بینگر این بود که کمتر از ۲ متری است باید بچرخیم و پشتمان به آن بشود. این زاویه دید برای آن تعریف شده، چون ما زمانی که به مانع میرسیدیم و میچرخیدیم و پشتمان به مانع میشد دوباره برای ما پیامی میامد که مانعی در فاصله کمتر از دو متری پیدا شده و در پشت شما است و دوباره میچرخیدیم و در یک لوپ می‌افتدیم. میشد با یک بیت flag و تعریف کردن state هم آن را هندل کرد اما باز کمی جای کار داشت چراکه ما دقیقا در دو متری مانع نمی‌ایستادیم و کمی خطأ داریم و مثلا در ۱.۸ m آن می‌ایستیم برای مثال. پس در اون حالت هم چنانچه کمی حرکت میکرد باز میچرخید چون در دو متری آن یک مانع یافت میشد. همچنین هر direction ای که به ما بدهد به اندازه ۱۸۰ – direction باید چرخش کنیم که پشتمان به آن مانع شود. موارد مربوط به چرخش و حرکت خطی درتابع move هندل شده است.

نود monitor_node.py مانند قسمت قبل میباشد و بنابراین از آوردن مجدد آن خودداری کردم.

حال تمامی فایل‌های پایتون را به صورت زیر executable میکنیم.

```

mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/obstacle_detect$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/obstacle_detect$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/obstacle_detect/src$ ls
control_node.py  monitor_node.py  sensor_node.py
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/obstacle_detect/src$ 

```

در گام بعدی به سراغ نوشتن لانج فایل میرویم. طبق خواسته سوال مکان اولیه ربات ۰, ۰ و با زاویه ۰.۷۵ رادیان میباشد که درون لانج فایل این مقادیر را اعمال میکنیم.

ابتدا برای راحتی detect_obstacles.world را در مسیر زیر کمی میکنیم:

/hw2_ws/src/turtlebot3_simulations/turtlebot3_gazebo/worlds

حال لازم است تا در دایرکتوری زیر یک launch file پایه برای بالا آوردن ربات در نقشه بنویسیم.

/hw2_ws/src/turtlebot3_simulations/turtlebot3_gazebo/launch\$

ابتدا یک کپی از my_empty_world.launch که آن را در گام یک خودمان از روی ساختیم، تهیه میکنیم. سپس اسم آن را عوض میکنیم به turtlebot3_emoty_world.launch و آن را باز میکنیم و به صورت زیر تغیر میدهیم:



```

1 <launch>
2   <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
3   <arg name="x_pos" default="0.0"/>
4   <arg name="y_pos" default="0.0"/>
5   <arg name="z_pos" default="0.0"/>
6   <arg name="yaw" default="0.0"/>
7
8   <Include file="$(find gazebo_ros)/launch/empty_world.launch">
9     <arg name="world_name" value="${find turtlebot3_gazebo}/worlds/detect_obstacles.world"/>
10    <arg name="paused" value="false"/>
11    <arg name="use_sim_time" value="true"/>
12    <arg name="gui" value="true"/>
13    <arg name="headless" value="false"/>
14    <arg name="debug" value="false"/>
15  </Include>
16
17  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
18
19  <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" />
20 </launch>

```

حال به سراغ launch فایل اصلی که داخل پکیج detect_obstacle ایجاد کردیم، میرویم.



```

<launch>

  <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_detect_obstacles.launch">
    <arg name="x_pos" value="0.0"/>
    <arg name="y_pos" value="0.0"/>
    <arg name="z_pos" value="0.0"/>
    <arg name="yaw" value="0.75"/>
  </include>

  <node pkg="obstacle_detect" type="sensor_node.py" name="mission" output="screen"></node>

  <node pkg="obstacle_detect" type="control_node.py" name="controller" output="screen"></node>

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

  <include file="$(find turtlebot_rviz_launchers)/launch/view_robot.launch"/>

</launch>

```

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 obstacle_detect obstacle_detect.launch`

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ roslaunch obstacle_detect obstacle_detect.launch
... logging to /home/mahdi/.ros/log/b54535ba-e11c-11ed-8e89-dd05e5a3fe14/roslaunch-mahdi-45781.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.
started roslaunch server http://mahdi:37523/

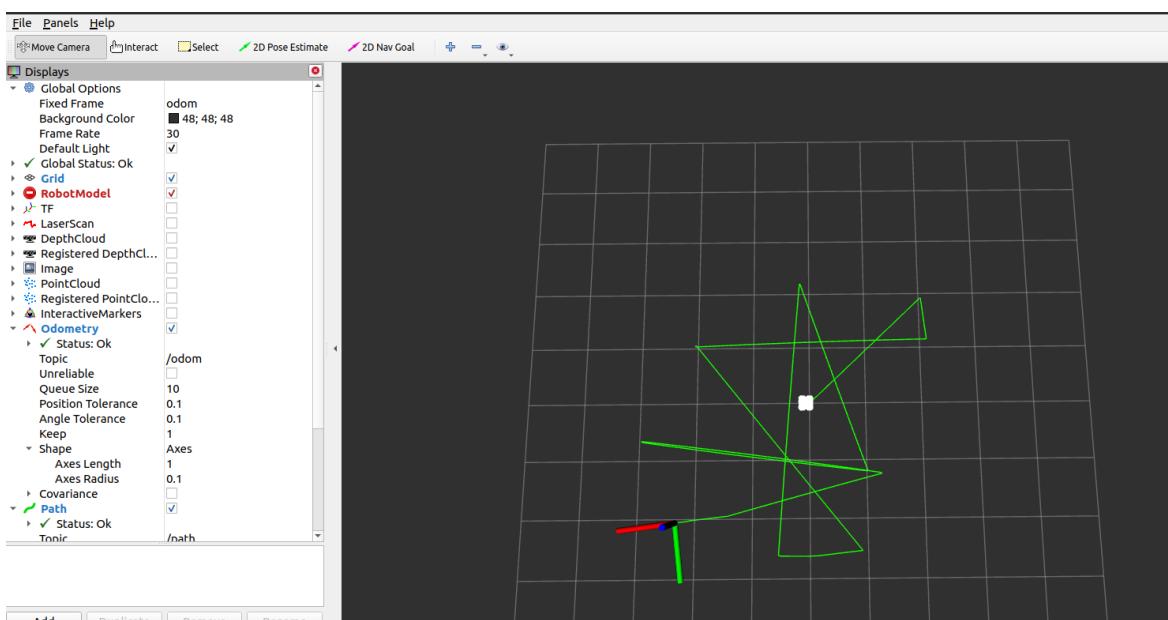
SUMMARY
=====

PARAMETERS
  * /gazebo/enable_ros_network: True
  * /robot_description: <?xml version="1....
  * /rosdistro: noetic
  * /rosversion: 1.15.15
  * /use_sim_time: True

NODES
/
  controller (obstacle_detect/control_node.py)
  gazebo (gazebo_ros/gzserver)
  gazebo_gui (gazebo_ros/gzclient)
  mission (obstacle_detect/sensor_node.py)
  monitor (obstacle_detect/monitor_node.py)
  rviz (rviz/rviz)
  spawn_urdf (gazebo_ros/spawn_model)

auto-starting new master
process[master]: started with pid [45791]
```

برای مثال نمونه‌ای از مسیری که ربات پیموده در تصویر زیر قابل مشاهده میباشد.



گام دوم - نتایج

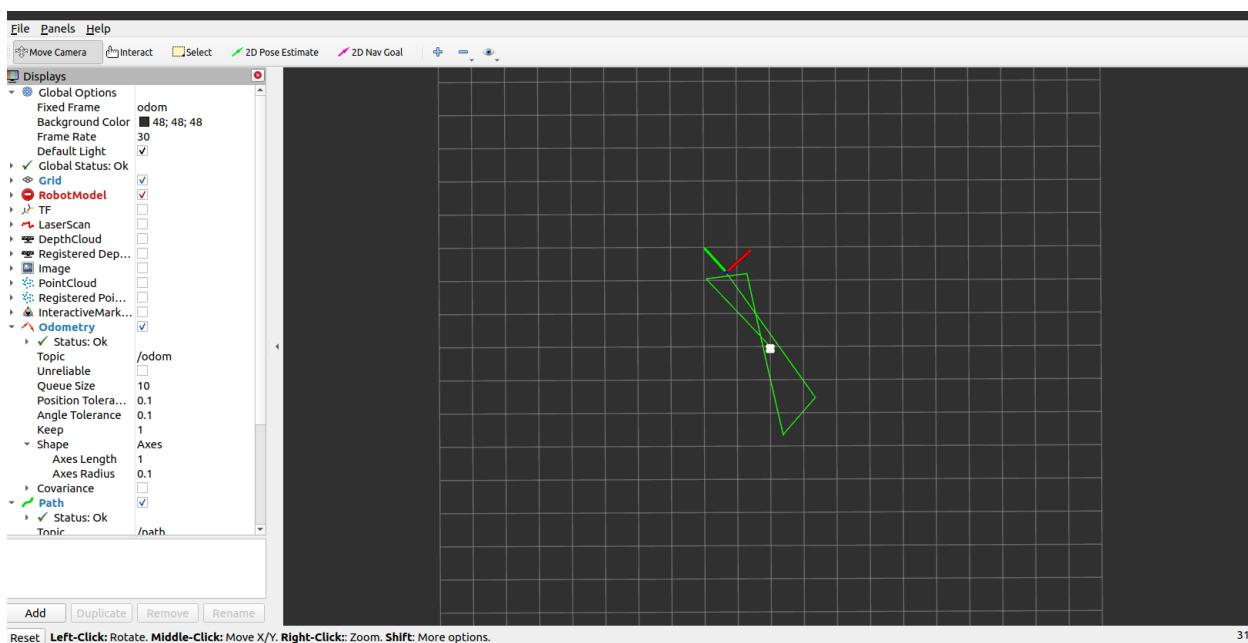
۱- ویدیوی کارکرد ربات:

ویدیوی مذکور ضبط شده و با فرمت گفته شده در یک پوشه ارسال شده است.

دقت شود که برای آنکه کمی سرعت بالاتر رود که زمان فیلم کم شود، سرعت خطی و زاویه ای را کمی زیاد کردم که باعث میشود دقت پایین تر بیاید و برای مثلا ۲ متر را کمی رد میکند و سپس متوجه میشود با اینکه در کد شرط را روی کمتر از ۲ متر گذاشته بودم.

۲- نمای مسیر ربات در شبیه ساز Rviz

نمونه ای از آن را در صفحه قبل میتوانید مشاهده کنید. نمونه دیگری که نمونه ضبط شده در ویدیو هست هم در زیر آمده است.



گام سوم - کار با Pointcloud

بخش اول:

پکیجی به نام pcd_view بسازید.

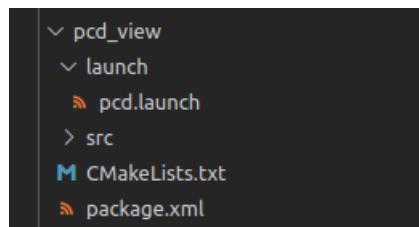
- catkin_create_pkg pcd_view rospy std_msgs sensor_msgs

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ cd src/  
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$ catkin_create_pkg pcd_view rospy std_msgs sensor_msgs  
Created file pcd_view/package.xml  
Created file pcd_view/CMakeLists.txt  
Created folder pcd_view/src  
Successfully created files in /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/pcd_view. Please adjust the values in package.xml.  
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$
```

یک launch file در آن گذاشته و turtlebot3_stage_4.launch های launch file و turtlebot3_gazebo_rviz.launch include ا به وش زیر در فایل خود نماید.

```
<include file="$(find turtlebot3_gazebo)/launch/turtlebot3_stage_4.launch"/>
```

نام لانچ فایل را pcd.launch گذاشتم:



محتوای لانچ فایل، به صورت زیر میباشد:

```
pcd.launch > pcd_view > launch > pcd.launch
  1   <launch>
  2     <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_stage_4.launch"/>
  3     <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_gazebo_rviz.launch"/>
  4
  5   </launch>
  6
  7
  8
```

حال یايد در ترمینال يوبي و در work space دستور catkin_make اجرا کنيم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ catkin_make
Base path: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws
Source space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src
Build space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/install
```

سپس باید ترمینال را سورس کنیم و مدل ربات را اضافه کنیم و سپس به کمک دستور roslaunch لانچ فایلی که نوشتم را اجرا کنیم.

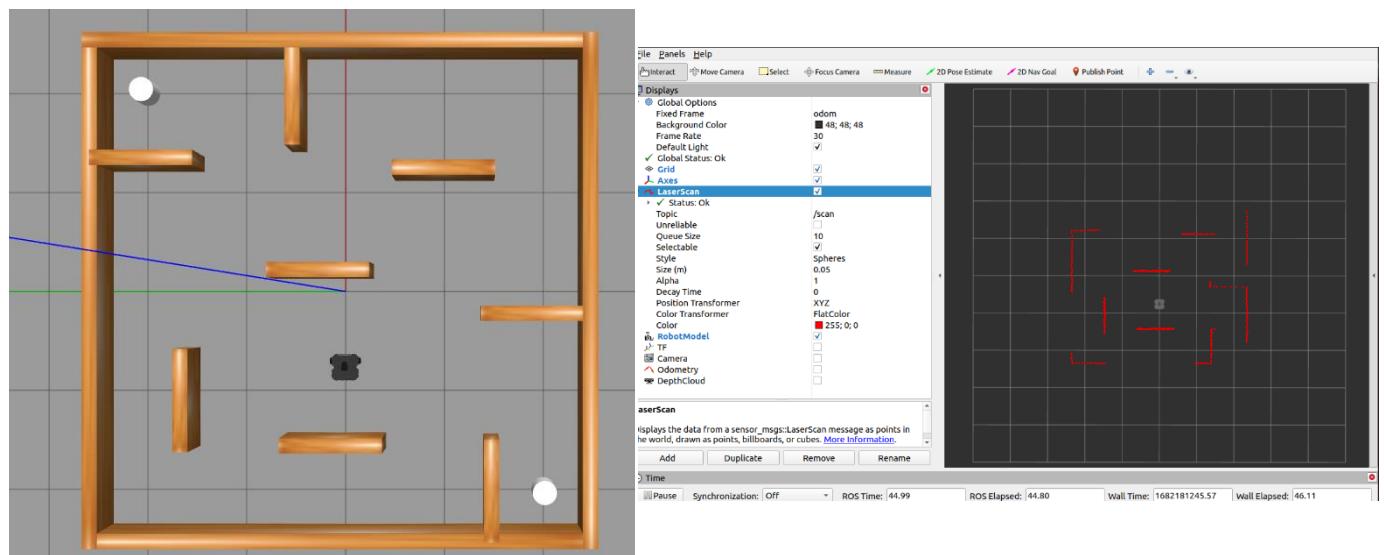
```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ rosrun pcd_view pcd.launch
... logging to /home/mahdi/.ros/log/a3b91ea2-e125-11ed-8e89-dd05e5a3fe14/rosrun-mahdi-73601.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 rosrun server http://mahdi:45617

SUMMARY
=====
PARAMETERS
* /gazebo/enable_ros_network: True
* /robot_description: <?xml version="1....
* /robot_state_publisher/publish_frequency: 50.0
* /robot_state_publisher/tf_prefix:
* /rosdistro: noetic
* /rosversion: 1.15.15
* /use_sim_time: True

NODES
/
```

با این دستور RViz و Gazebo می شوند.



برای نمای بهتر می توانید از منوی سمت چپ، Points Style را به LaserScan مربوط به تعیین دهید.
اگر برای شما هم ، مثل من  در صورت انجام این کار terminate شد میتوانید مثلا بذارید روی rviz و اندازه نقاط را مثلا 0.05m بذارید. (برای این موضوع سرچ کردم ظاهرا یک باگ sphere میباشد و نقاط به شدت بزرگ میشوند و برنامه terminate میشود) میتوانید [لینک ۱](#) و [لینک ۲](#) را برای این مشکل بخوانید.

سپس در یک ترمینال دیگر turtlebot3_teleop.launch را از پکج turtlebot3_teleop_key.launch ران کنید و ربات را با کیبورد کنترل نمایید.

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_973i701/packages/hw2_ws$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
... logging to /home/mahdi/.ros/log/2a2dc182-e12a-11ed-8e89-dd05e5a3fe14/roslaunch-mahdi-84203.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 roslaunsh server http://mahdi:37537/

SUMMARY
=====

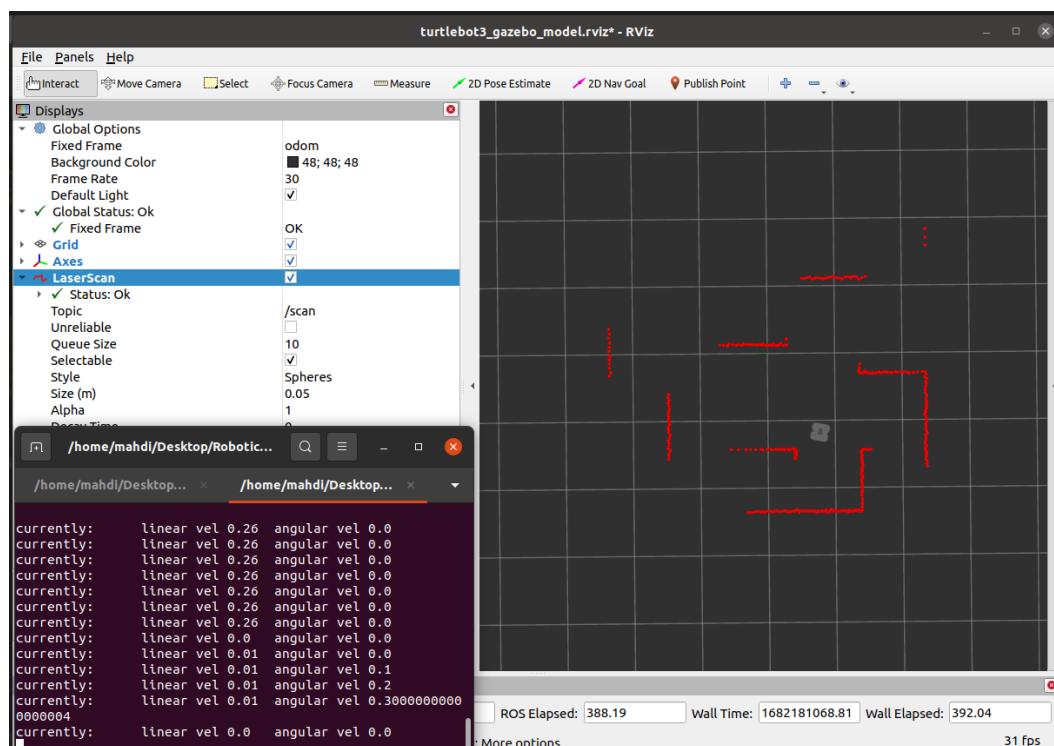
PARAMETERS
  * /model: waffle
  * /rosdistro: noetic
  * /rosversion: 1.15.15

NODES
/
  turtlebot3_teleop_keyboard (turtlebot3_teleop/turtlebot3_teleop_key)

ROS_MASTER_URI=http://localhost:11311

process[turtlebot3_teleop_keyboard-1]: started with pid [84223]

Control Your TurtleBot3!
-----
Moving around:
      W
      a   s   d
      X
```



بخش دوم:

در این بخش میتوان به [لينك](#) مراجعه کرده و روال کار را متوجه شد. همچنین ویدیوی یوتوب در این [لينك](#) نیز مراحل کار را گفته است.

در این بخش از داده‌ی PointCloud به LaserScan می‌رسیم.

برای این کار به سرویس laser_assembler از پکج laser_scan_assembler نیاز داریم. این سرویس داده‌های مربوط به LaserScan را جمع آوری کرده و مجموع آنها را در فرمت یک پیامPointCloud2 از پکج sensor_msgs باز می‌گرداند. دستور اجرای نود این سرویس را به انتهای launch file خود اضافه کنید.

دقت کنید که این نود دو پارامتر دریافت می‌کند:

۱ - max_scans : که تعداد بیشینه اسکن‌ها را در یک پیامPointCloud2 مشخص می‌کند. مقدار آنرا " ۴۰۰ " بگذارید.

۲ - Fixed_frame : که فریم publish کردن پیام‌ها مشخص می‌کند. برای کاربرد ما این مقدار را " odom " ست کنید

همچنین ما به یک نود برای فرستادن درخواست به سرویس بالا، دریافت و publish کردن پیغام laser_assembler نیاز داریم. برای نوشتن این نود از AssembleScan2 از پکج PointCloud2 استفاده نمایید.

با توجه به توضیحات ابتدای فایل پایتون با نام laser_to_pcd.py در قسمت src مربوط به پکیج pcd_view ایجاد می‌کنیم. سپس کد زیر را در آن مینویسیم:

```
#!/usr/bin/python3

import rospy
from laser_assembler.srv import AssembleScans2
from sensor_msgs.msg import PointCloud2

class PointCloud():
    def __init__(self) -> None:
        rospy.init_node("laser_to_pcd")
        rospy.wait_for_service("assemble_scans2")
        self.assemble_scans = rospy.ServiceProxy('assemble_scans2', AssembleScans2)
        self.pub = rospy.Publisher ("/laser_pointcloud", PointCloud2, queue_size=1)

    def run(self):
```

```

while not rospy.is_shutdown():
    try:
        resp = self.assemble_scans(rospy.Time(0,0), rospy.get_rostime())
        #rospy.loginfo("Got cloud with %u points" % len(resp.cloud.data))
        self.pub.publish(resp.cloud)

    except rospy.ServiceException as e:
        rospy.loginfo("Service call failed: %s" %e)

    rospy.sleep(1)

if __name__ == '__main__':
    pcd = PointCloud()
    pcd.run()

```

حال به سراغ لانچ فایل میرویم و تغییرات زیر را اعمال میکنیم. دو تا include اول که توضیحاتش در بخش ۱ داده شد برای بالا آوردن ربات در rviz و gazebo میباشد.

```

<launch>

<include file="$(find turtlebot3_gazebo)/launch/turtlebot3_stage_4.launch"/>

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

<node pkg="laser_assembler" type="laser_scan_assembler" name="my_assembler">
    <param name="max_scans" type="int" value="400" />
    <param name="fixed_frame" type="string" value="odom" />
</node>

<node pkg="pcd_view" type ="laser_to_pcd.py" name="laser_to_pcd" output="screen"/>

</launch>

```

حال ابتدا لازم است که فایل پایتون مربوطه را executable کنیم:

```

mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src$ cd pcd_view/
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/pcd_view$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/pcd_view$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/pcd_view/src$ ls
laser_to_pcd.py
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws/src/pcd_view/src$ 

```

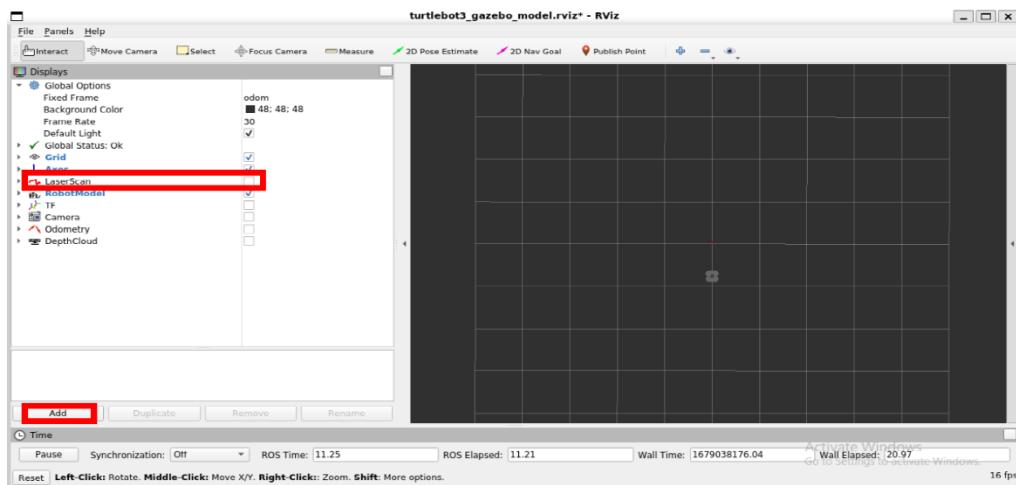
سپس در work space رفته و catkin_make است تا در این ترمینال سورس کنیم و ربات را معرفی نماییم و سپس لانچ فایل را اجرا نماییم:

```
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project2/HW2_9731701/packages/hw2_ws$ rosrun pcd_view pcd.launch
... logging to /home/mahdi/.ros/log/60c50a7a-e15c-11ed-8e89-dd05e5a3fe14/rosrun.launch-mahdi-152233.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.

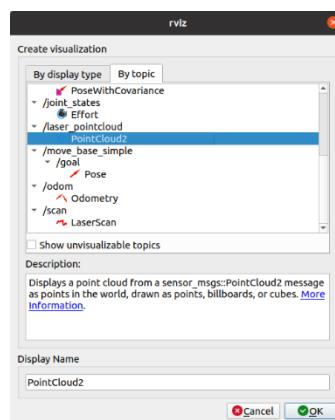
xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:34515/

SUMMARY
=====
PARAMETERS
  * /gazebo/enable_ros_network: True
  * /my_assembler/_fixed_frame: odom
  * /my_assembler/_max_scans: 400
  * /robot_description: <?xml version="1.....
  * /robot_state_publisher/_publish_frequency: 50.0
  * /robot_state_publisher/_tf_prefix:
  * /rosdistro: noetic
  * /rosversion: 1.15.15
  * /use_sim_time: True
```

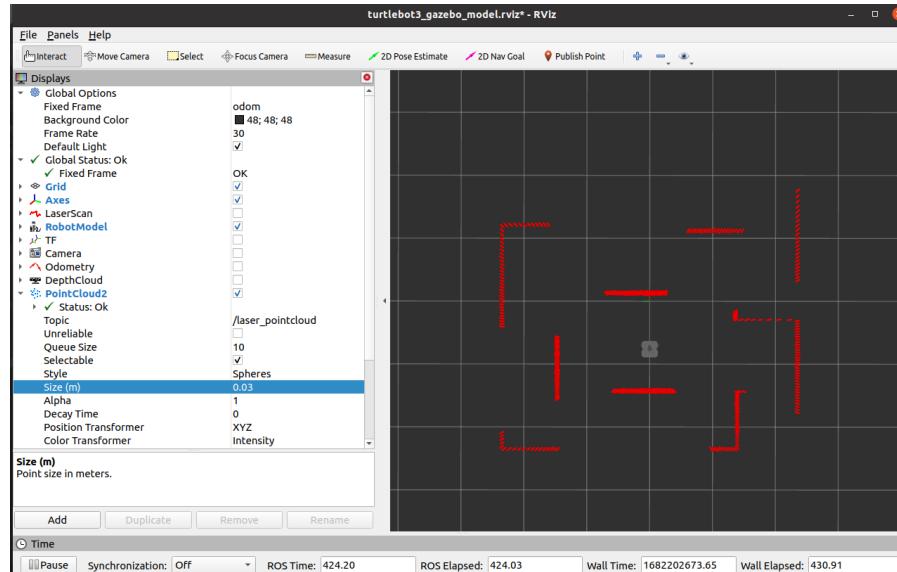
در منوی سمت چپ با Uncheck LaserScan ، آن را Disable نمایید. سپس گزینه Add را انتخاب کنید.



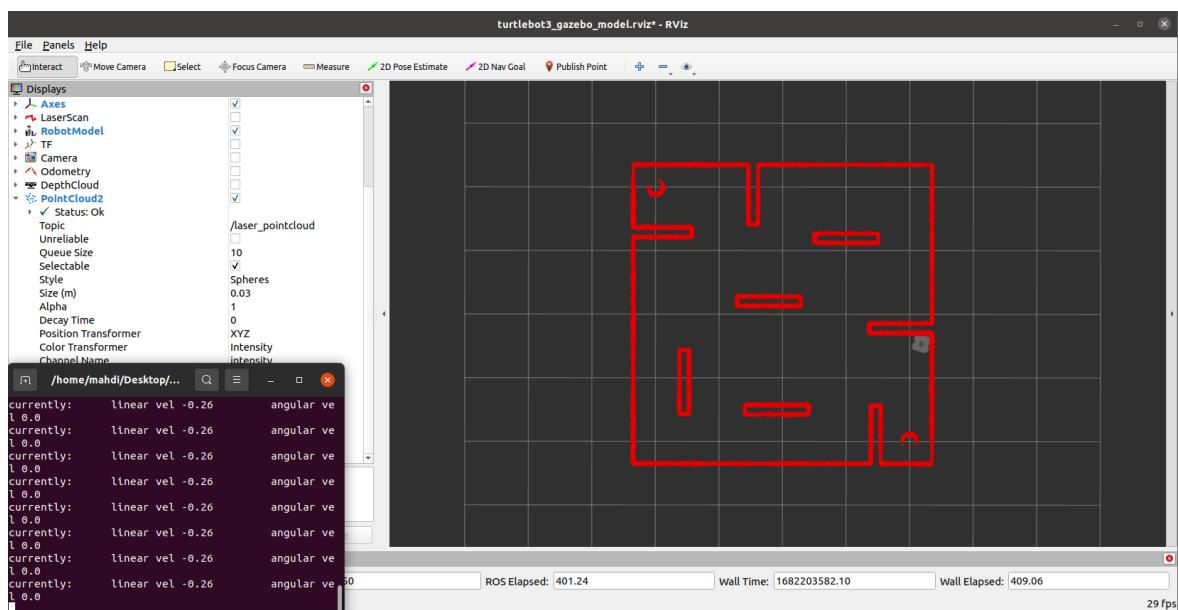
در منوی باز شده، در بخش PointCloud2 By Topic بر روی کلیک نمایید.



پس از این کار RViz داده های دریافتی از تاپیک مربوط به PointCloud2 را نمایش می دهد.



اکنون در یک ترمینال دیگر turtlebot3_teleop را از پکج turtlebot3_teleop_key.launch ران کنید و ربات را با کیبورد کنترل نمایید. با حرکت ربات می توانید نقشه ای از محیط بسازید و کامل نمایید. البته چون کمی کنترل ربات سخت بود و کار نقشه برداری به کندی انجام میشد مجبور شدم مقدار max_scans را در لانچ فایل افزایش دهم و برابر ۲۰۰۰ قرار دهم.

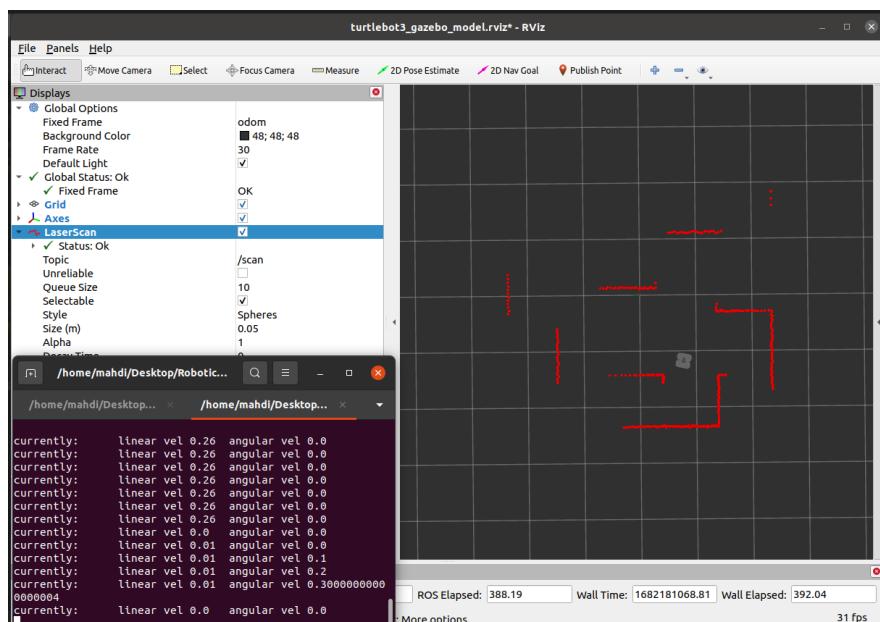


گام سوم - نتایج

بخش اول:

۱- تصویری از کارکرد ربات:

همانطور که در شکل زیر مشاهده میکنید تصویری از کارکرد ربات را نشان میدهد که در آن به کمک teleop ربات را جا به جا کردیم و خروجی اسکن لیزر را در Rviz نشان میدهد.



۲- دلیل لرزش نقاط داده در شبیه ساز چیست؟ دلیل خود را در گزارش تمرین بیاورید.

یکی از علت های آن که در کلاس هم توضیح داده شد، خطای موجود در اسکن کردن محیط میباشد. برای مثال زمانی که یک تابش لیزر به یک نقطه از محیط میخورد و ما فاصله آن را پیدا میکنیم و منتظر با آن در Rviz یک نقطه نشان میدهیم، لزومی ندارد که اگر مجدد پرتو لیزر به همان نقطه بخورد باز در Rviz دقیقا همان نقطه را نشان دهیم و ممکن است کمی جلوتر یا عقب تر نشان دهیم که به دلیل نرخ بالای اسکن کردن از محیط به نظر میرسه که نقاط داده در حال لرزش هستند. پس در حقیقت نویزی که در سنسور وجود دارد میتواند باعث این موضوع شود.

حال اگر ربات در حال حرکت باشد و سنسور هم همراه با آن در حال حرکت هست و این باعث میشود این اثر لرزش کمی بیشتر هم بشود.

نکته دیگر بحث پردازش این دیتاهای ROS میباشد که ممکن است در هنگام تبدیل نقاط به فریم لوکال لیزر اسکنر، خطایی اندک وجود داشته باشد و حتی بحث های پردازش سخت افزاری و زمان پراسس موجب پدیده jittering شود.

همچنین برخی نقاط ممکن است بازتابشان در لحظاتی به درستی به سنسور نرسد و آن نقطه متناظر به درستی در یک لحظه تشخیص داده نشود و در لحظه بعد مجدد در RVIZ نمایش داده شود که این هم خب باز میتواند تاثیر گذار باشد.

در این [لينك](#) نیز در این رابطه توضیحاتی دادند.

بخش دوم:

۱- **ویدیوی کارکرد ربات:** ویدیو ضبط شده و در فolder مربوطه قرار داده شد.

۲- **تصویر نهایی ربات:**

در زیر میتوانید تصویر نهایی از نقشه برداری را مشاهده کنید. (چون کمی کنترل ربات سخت بود و کار نقشه برداری به کندی انجام میشد مجبور شدم مقدار max_scans را در لانچ فایل افزایش دهم و برابر ۲۰۰۰ قرار دهم که تعداد اسکن های بیشتری ذخیره شود و تا رسیدن به پایان کار قبلی ها پاک نشوند).

