به نام خدا

**دانشگاه صنعتی امیرکبیر**

**دانشکده مهندسی کامپیوتر**

# اصول علم ربات

## پروژه نهایی

| | |
|---|---|
| نام و نام خانوادگی | مهدی رحمانی – هادی ناظمی اسفندیاری |
| شماره دانشجویی | ۹۷۳۱۷۰۱- ۹۷۳۰۰۳۴ |
| تاریخ ارسال گزارش | |

# فهرست گزارش سوالات

## آماده‌سازی محیط ROS

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

```
mahdi@mahdi:~/Desktop/Robotics/project5$ mkdir -p hw5_ws/src
mahdi@mahdi:~/Desktop/Robotics/project5$ cd hw5_ws/src/
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src$ catkin_init_workspace
Creating symlink "/home/mahdi/Desktop/Robotics/project5/hw5_ws/src/CMakeLists.txt" pointing to "/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws$ catkin_make
Base path: /home/mahdi/Desktop/Robotics/project5/hw5_ws
Source space: /home/mahdi/Desktop/Robotics/project5/hw5_ws/src
Build space: /home/mahdi/Desktop/Robotics/project5/hw5_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project5/hw5_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project5/hw5_ws/install
####
#### Running command: "cmake /home/mahdi/Desktop/Robotics/project5/hw5_ws/src -DCATKIN_DEVEL_PREFIX=/home/mahdi/Desktop/Robotics/project5/hw5_ws/devel
ahdi/Desktop/Robotics/project5/hw5_ws/install -G Unix Makefiles" in "/home/mahdi/Desktop/Robotics/project5/hw5_ws/build"
####
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Detecting C compiler ABI info
```

همچنین در این پروژه به ربات turtlebot3 و شبیه‌ساز gazebo 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/project5/hw5_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% (681/681), done.
remote: Compressing objects: 100% (126/126), done.
remote: Total 3160 (delta 596), reused 555 (delta 555), pack-reused 2479
Receiving objects: 100% (3160/3160), 15.40 MiB | 3.21 MiB/s, done.
Resolving deltas: 100% (1852/1852), done.
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_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% (6485/6485), done.
remote: Compressing objects: 100% (2184/2184), done.
remote: Total 6485 (delta 4062), reused 6357 (delta 4029), pack-reused 0
Receiving objects: 100% (6485/6485), 119.94 MiB | 4.00 MiB/s, done.
Resolving deltas: 100% (4062/4062), done.
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_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 | 261.00 KiB/s, done.
Resolving deltas: 100% (170/170), done.
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src$
```

# سناریوی اول

- **آماده‌سازی‌های اولیه گام اول**

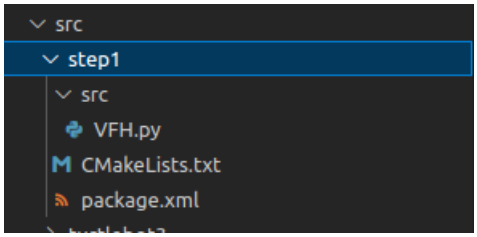
برای این قسمت یک پکیج جداگانه درست میکنیم به نام step1 میکنیم و dependencyهایی که ممکن است به کار بیایند را هم اضافه کنیم.

```
catkin_create_pkg step1 rospy std_msgs sensor_msgs nav_msgs
```

```
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src$ catkin_create_pkg step1 rospy std_msgs sensor_msgs nav_msgs
Created file step1/package.xml
Created file step1/CMakeLists.txt
Created folder step1/src
Successfully created files in /home/mahdi/Desktop/Robotics/project5/hw5_ws/src/step1. Please adjust the values in package.xml.
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src$
```

حال به Work Space برگشته و با زدن دستور . code میتوان این ورک اسپیس را در VS Code باز کرد.

حال به سراغ ساخت نودها میرویم. برای این منظور در فولدر src مربوط به step1 میرویم و یک فایل پایتون با نام VFH.py میسازیم.




حال در گام بعد به سراغ نوشتن کد مربوط نود ساخته شده میرویم.

- **مقادیر و فرمول‌های لازم برای الگوریتم VFH**

ابتدا لازم است یک سری از ثابت‌ها را با توجه به داده‌های مسئله به دست آوریم و بعد به کلیت الگوریتم بپردازیم.

در صورت سوال گفته شده که نقاط شروع و پایان به ترتیب عبارتند از:

- Starting point: (0, 0)
- Goal point: (13, 7)

که ما باید از نقطه شروع حرکت کنیم و با اعمال الگوریتم VFH و چرخش به کمک PID به نقطه نهایی برسیم.

سپس گفته شده که ۵ درجه را باید یک sector تعریف کرد. پس مقدار $\alpha$ و n به صورت زیر است که n برابر با تعداد sectorها میباشد.

- $\alpha = 5°$
- $n = \frac{360}{5} = 72$

همچنین در رابطه $a\text{-}b\ d_{ij}$ گفته شده که مقادیر $a$ و $b$ به صورت زیر میباشند:

- $a = 1$
- $b = 0.25$

همچنین گفته شده که $d$ حداکثر فاصله میشود باید مقدار $a\text{-}bd$ برابر با ۰ شود. بنابراین داریم:

$$\begin{cases} d_{max} = \frac{\sqrt{2}(w_s)}{2} \\ a - b\ d_{max} = 0 \end{cases} \rightarrow 1 - 0.25 \times \frac{\sqrt{2}(w_s)}{2} = 0 \rightarrow w_s = \frac{8}{\sqrt{2}} \quad , d_{max} = 4$$

همچنین در رابطه زیر که برای *smoothing* هست باید مقدار $l$ را برابر با ۱ در نظر بگیریم:

$$h'_k = \frac{h_{k-l}+2h_{k-l+1}+\dots+lh_k+\dots+2h_{k+l-1}+h_{k+1}}{2l+1} \qquad \boxed{l = 2}$$

مقدار *threshold* یا حد آستانه را برابر با یک لیست از آستانه ها میگیریم. این مقدار باید *tune* شود و در حقیقت متناسب با مسئله میباشد. در مسئله توضیح داده شده که اثرات خیلی زیاد یا خیلی کم بودن *threshold* چیست و همچنین گفته شده که باتوجه به*global plane* بیایم و به صورت *adaptive* این مقدار را تعریف کنیم.

همچنین برای $s_{max}$ طبق مقاله داریم:

$$s_{max} = 18$$

همچنین عملکرد خوب *VFH* برای سرعت خطی حدود *0.6 m/s* نیز اوکی است ولی ما آن را در ماکسیمم حالت برابر با *0.5 m/s* میگیریم. در ادامه پارامترهای مربوط به کنترل سرعت را بیان میکنیم:

در این جا یک $v_{max}$ داریم که به صورت زیر تعیین میشود:

$$V_{max} = 0.5\ \frac{m}{s}$$

همچنین یک مقدار $V_{min}$ داریم که در این جا و با ایده گیری از مقاله برابر با :

$$V_{min} = 4\ \frac{cm}{s} = 0.04\ \frac{m}{s}$$

یک مقدار $h_m$ داریم که طبق مقاله به صورت تجربی تعیین میشود و موجب کاهش مناسب سرعت میشود.  این مقدار را در این پروژه برابر با :

$h_m =$

یک مقدار $\Omega_{max}$ نیاز است که ماکسیسم سرعت زاویه‌ای *yaw* ربات میباشد که برابر با :

$\Omega_{max} = 120°/sec$

- **نوشتن کد مروبط به نودها**

در این قسمت دو نود داریم یکی VFH.py و دیگری control.py میباشند.

ابتدا در قسمت init لازم است که نود را initialize کنیم و همچنین باید این نود را به عنوان publisher برای cmd_vel معرفی نماییم و سپس همان مقادیر ثابتی که در قسمت قبل توضیح داده شد را set میکنیم..

سپس باتوجه به مراحل مقاله لازم است تا توابعی برای ساخت هیستوگرام و smooth کردن و سپس یافتن دره‌ها و همچنین یافتن target sector و در نهایت یافتن زاویه مناسب باتوجه به شرایط قرارگیری هدف نسبت به دره‌ها تعریف کرد.

همچنین برای یافتن thresholdهای مناسب لازم است تا یک دید مناسب از مقادیر داشته باشیم که برای این منظور نمودار histogram را رسم نمودیم.

```
#!/usr/bin/python3

import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
from nav_msgs.msg import Odometry
import tf
import math
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.colors import TwoSlopeNorm
import numpy as np
from step1.srv import GetVFHData, GetVFHDataResponse

class VFH():

    def __init__(self):
```

```
        rospy.init_node('VFH', anonymous=False)
        self.cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=5)
        self.VFH_service = rospy.Service('vfh_data', GetVFHData, self.path_planning)
        rospy.loginfo(f"service is created")

        # This varibales are input and is going tobe changed
        # the sevice inputs varibales are goal_x, goal_y and threshold
        self.goal_x = 13
        self.goal_y = 7
        self.threshold = 2.5

        # some varibales related to window and constants of VFH algorithm
        self.alpha = 5                      # angle of each sector
        self.n = 72                         # number of sectors
        self.a = 1                          # <a> parameter in a-bd formula
        self.b = 0.25                       # <b> parameter in a-bd formula
        self.d_max = self.a/self.b          # maximum distance that a-bd is not zero
        self.ws = math.sqrt(2)*self.d_max   # window size (ws = sqrt(2)*dmax)
        self.l = 2                          # l parameter in smoothing formula
        self.s_max = 16                     # boundary for defining wide or narrow valley

        # varibale for saving laser sacn
        self.laser_scan = None

        # some varibales containing results of VFH algorithm at each step
        self.smooth_hist = []
        self.desired_angle = []
        self.all_valleys = []
        self.goal_angle  = 0
        self.sectors_in_threshold = []

        # some varibales of plot
        self.fig = plt.figure()
        self.fig.set_figheight(self.fig.get_figheight() * 1.5)
        self.axes = [self.fig.add_subplot(211), self.fig.add_subplot(212)]
        self.plots = []
        self.vertical_axlines = []
        self.horizontal_axlines = []
        self.flag = False

    # 1) Define some methods as our tools

    def get_laser_scan(self):
        '''
        This method subscribe scan topic and wait for messages if it.
        if a message is arrived we get the laser scan.
        '''
        self.laser_scan = rospy.wait_for_message("/scan", LaserScan)
```

```
    def get_robot_pose(self):
        '''
        get x and y coordinate of position of the robot
        get the yaw angle of robot in world.
        We call it, heading of the robot.
        '''
        # waiting for the most recent message from topic /odom
        msg = rospy.wait_for_message("/odom" , Odometry)

        orientation = msg.pose.pose.orientation
        position = msg.pose.pose.position

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

        return position.x, position.y, yaw


    # 2) create polar histogram
    def create_histogram(self):
        '''
        This method get laser scan as its input and then
        use the related formula for computing h_k:
        1) mij = cij * (a-b dij)
        2) hk = Sigma (mij) for mij in sector k
        '''
        histogram = []
        self.get_laser_scan()

        for i in range(self.n):

            h_k = 0

            for j in range(5*i, 5*(i+1)):
                d = self.laser_scan.ranges[j]
                # if d = inf then 0*inf=nan so we assign 1000 instead of inf
                if d==math.inf:
                    d=10000
                # we should check if the obstacle is inside square window or not
                cij = 0
                if abs(d*math.cos(math.radians(j))) <=(self.ws/2) and
abs(d*math.sin(math.radians(j))) <=(self.ws/2):
                    cij = 1 - (d/self.ws)

                mij = (cij**2)*(self.a - (self.b * d))
```

```
                h_k += mij

            histogram.append(h_k)

        return histogram

    # 3) smoothing histogram
    def get_smooth_hist(self, histogram):
        '''
        This method get histogram as an input and smooth it.
        the formula for smoothing is:
        h_prim_k = (h_(k-2) + 2*h_(k-1) + 2*h_(k) + 2*h_(k+1) + h_(k+2))/ (2*2+1)
        '''
        h = histogram
        smooth_histogram = []
        for i in range(self.n):
            h_prim_k = (h[(i-2)%self.n] + 2*h[(i-1)%self.n] + 2*h[i] + 2*h[(i+1)%self.n] +
h[(i+2)%self.n])/5
            smooth_histogram.append(h_prim_k)

        return smooth_histogram

    # 4) find valleys
    def find_valleys(self ,smooth_histogram):
        # first we should find all sector numbers
        # wich their h_k is less than threshold
        self.sectors_in_threshold = []
        sector_number = 0
        self.all_valleys = []
        for h_k in smooth_histogram:
            if h_k < self.threshold:
                self.sectors_in_threshold.append(sector_number)
                self.all_valleys.append(h_k)
            else:
                self.all_valleys.append(0)
            sector_number += 1

        # Then we should sperate the acceptable sectors into valleys
        # a series of consequent sector number make a valley
        valleys = []
        valley = []
        for sector_number in self.sectors_in_threshold:
            if not valley or sector_number == valley[-1] + 1:
                valley.append(sector_number)
            else:
                valleys.append(valley)
                valley = [sector_number]
        if valley:
```

```
            valleys.append(valley)

        # now we should consider if 0 exists in first valley and 359 exists in
        # last valley, these valleys are not seperate and we should merge them
        if len(valleys) != 0 and 0 in valleys[0] and (self.n-1) in valleys[len(valleys)-1]:
            valleys[len(valleys)-1] = valleys[len(valleys)-1] + valleys[0]
            valleys.pop(0)

        return valleys

    # 5) find the target sector which the robot should reach it
    def find_target_sector(self):

        robot_x, robot_y, yaw = self.get_robot_pose()
        angle_to_goal = math.atan2(self.goal_y - robot_y, self.goal_x - robot_x)

        target_angle = math.degrees(angle_to_goal - yaw)
        if target_angle <0:
            target_angle += 360

        target_sector = round(target_angle / self.alpha)

        return target_sector, target_angle

    # 6)find desired angle for rotation of robot
    def get_desired_angle(self, valleys, targetsector, target_angle):
        edge_points = []
        #rospy.loginfo(f"P_a : {valleys} ")
        for valley in valleys:
            if targetsector in valley:
                # if target sector is inside one of the valleys we can go straight toward it
                desired_theta = target_angle
                return desired_theta
            else:
                # if its not in a valley we should find nearest valley
                # so here we should save edges of each valley
                edge_points.append(valley[0])
                edge_points.append(valley[len(valley)-1])

        # we wanna find nearest edge to target sector
        my_ruler = 0
        closest_edge = 0
        end_of_valley = False
        while(True):
            my_ruler += 1
            left = targetsector - my_ruler
            right = targetsector + my_ruler
            if left >= 0 and left in edge_points:
```

```
                closest_edge = left
                end_of_valley = True
                break
            elif left<0 and self.n+left in edge_points:
                closest_edge = self.n+left
                end_of_valley = True
                break
            elif right in edge_points:
                closest_edge = right
                break

        # also according to closest edge we can find its corresponding valley
        best_valley_index = int(edge_points.index(closest_edge)/2)
        best_valley = valleys[best_valley_index]
        kd = 0
        if len(best_valley) <= self.s_max:
            # narrow valley
            kd = best_valley[int(len(best_valley)/2)]

        else:
            # wide valley
            if end_of_valley:
                new_valley = best_valley[-self.s_max:]
                kd = new_valley[int(len(new_valley)/2)]
            else:
                new_valley = best_valley[:self.s_max]
                kd = new_valley[int(len(new_valley)/2)]

        desired_theta = kd*self.alpha + self.alpha/2

        return desired_theta

    # 6)find desired angle for rotation of robot
    def get_desired_angle2(self, valleys, targetsector, target_angle):
        edge_points = []
        #rospy.loginfo(f"P_a : {valleys} ")
        for valley in valleys:
            if targetsector in valley:
                # if target sector is inside one of the valleys we can go straight toward it
                desired_theta = target_angle
                return desired_theta
            else:
                # if its not in a valley we should find nearest valley
                # so here we should save edges of each valley
                edge_points.append(valley[0])
                edge_points.append(valley[len(valley)-1])

        # we wanna find nearest edge to target sector
```

```
        my_ruler = 0
        closest_edge = 0
        end_of_valley = False
        while(True):
            my_ruler += 1
            left = targetsector - my_ruler
            right = targetsector + my_ruler
            if left >= 0 and left in edge_points:
                closest_edge = left
                end_of_valley = True
                break
            elif left<0 and self.n+left in edge_points:
                closest_edge = self.n+left
                end_of_valley = True
                break
            elif right in edge_points:
                closest_edge = right
                break

        kn = closest_edge
        kf = kn
        # also according to closest edge we can find its corresponding valley
        best_valley_index = int(edge_points.index(closest_edge)/2)
        best_valley = valleys[best_valley_index]
        kd = 0
        if end_of_valley:
            for i in range(1, len(best_valley)-1):
                if (best_valley[-i-1]*self.alpha <90 or best_valley[-i-1]*self.alpha > 270) and
i<self.s_max:
                    kf=kn-i
                else:
                    break
        else:
            for i in range(1, len(best_valley)-1):
                if (best_valley[i]*self.alpha <90 or best_valley[i]*self.alpha > 270) and
i<self.s_max:
                    kf=kn+i
                else:
                    break

        desired_theta = (round((kf+kn)/2)%self.n)*self.alpha
        return desired_theta



    def path_planning(self, goal_details):
```

```
        self.goal_x, self.goal_y, self.threshold = goal_details.goal_x, goal_details.goal_y,
goal_details.threshold
        #rospy.loginfo(f"server : {[self.goal_x, self.goal_y, self.threshold]} ")
        result = GetVFHDataResponse()

        #1) create histogram
        hist = self.create_histogram()

        #2) smooth hist
        self.smooth_hist = self.get_smooth_hist(hist)

        #3) find valleys
        valleys = self.find_valleys(self.smooth_hist)
        if len(valleys) == 0:
            result.desired_angle = 0
            result.h_prim_c = 1000
            return result

        #4) find target sector
        target_sector, target_angle = self.find_target_sector()

        #5) get desired angle
        self.desired_angle = self.get_desired_angle(valleys, target_sector, target_angle)

        #6) create the response of client
        result.desired_angle = self.desired_angle
        #current_sector_to_move = int(self.desired_angle/self.alpha)
        #result.h_prim_c = self.smooth_hist[current_sector_to_move]
        result.h_prim_c = max(max(self.smooth_hist[0:7]), max(self.smooth_hist[65:72]))
        return result

    def draw_hist(self, i):
        if self.smooth_hist and not self.flag:
            bins = [x * 5 for x in range(1, int(360 / self.alpha) + 1)]
            for ax, data in zip(self.axes, [self.smooth_hist, self.all_valleys]):
                ax.cla()
                self.plots.append(ax.bar(bins, height=data, width=-self.alpha, align='edge',
edgecolor='white'))
                ax.set_xticks([0, 90, 180, 270, 360])
                ax.set_xticklabels(['0°', '90°', '180°', '270°', '360°'])
                ax.set_yticks(sorted(list(ax.get_yticks()) + [self.threshold]))
                ax.set_xlabel('angle(degree)')
                ax.set_ylabel('h_prim_k')
                self.horizontal_axlines.append(ax.axhline(y=self.threshold, color='r', linestyle='-
-', linewidth=1, label='Threshold'))
                self.vertical_axlines.append(ax.axvline(x=self.desired_angle, color='k',
linestyle='--', linewidth=1, label='desired angle'))
```

```
                #self.vertical_axlines.append(ax.axvline(x=self.goal_angle, color='g', linestyle='-
-', linewidth=1, label='goal angle'))
                ax.legend()
                ax.set_title('smooth polar histogram' if ax == self.axes[0] else 'all valleys')
            self.fig.canvas.draw()
            self.fig.canvas.flush_events()
            self.flag = True

        elif self.flag:
            for plot, data in zip(self.plots, [self.smooth_hist, self.all_valleys]):
                for i, bar in enumerate(plot):
                    bar.set_height(data[i])
            for axvline, axhline in zip(self.vertical_axlines, self.horizontal_axlines):
                axvline.set_xdata(self.desired_angle)
                axhline.set_ydata(self.threshold)

if __name__ == '__main__':

    vfh = VFH()
    anim = FuncAnimation(vfh.fig, vfh.draw_hist, cache_frame_data=False)
    plt.show(block= True)
    rospy.spin()
```

سپس کد دیگر مربوط به نود کنترل میباشد. در مقاله در رابطه با کنترل سرعت مطالبی آمده است و به کمک آن میتوان سرعت خطی ربات را تنظیم کرد و به کمک PID میتوان کنترل روی سرعت زاویه ای داشت. کد آن نیز در ادامه آمده است.

```
#!/usr/bin/python3

import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import matplotlib.pyplot as plt
import math
from nav_msgs.msg import Odometry
import tf
from step1.srv import GetVFHData, GetVFHDataRequest

class PIDController():


    def __init__(self):
```

```
        rospy.init_node('controller', anonymous=False)
        rospy.wait_for_service('vfh_data') # wait for response from service
        self.calc_client = rospy.ServiceProxy('vfh_data', GetVFHData)

        self.k_i = 0.0
        self.k_p = 0.7
        self.k_d = 0.9

        self.dt = 0.005
        self.D = 0
        rate = 1/self.dt

        self.goals = [(4.5, -0.2), (3.4, 4.46), (2.3, 1.5), (0.5, 1.7), (1.0, 5.43),
                                (1.95, 5.55), (3.9, 7), (5, 5.5), (5.9,5.3), (5.6, 3),
                                (7.6, 3.1),(7.3, 5), (7.6, 6.7), (13, 6.65)]
        self.thresholds = [3, 2.5, 3, 3, 2.5, 3, 3.7, 3.5,3.6, 3.4, 4, 3.5, 3, 3]
        self.goal_counter = 0
        self.epsilon = 0.3
        # speed parameters
        self.V_max = 0.3                    # (m/s)
        self.V_min = 0.05                   # (m/s)
        self.omega_max = math.radians(120)   # (rad/s)
        self.h_m = 10

        self.r = rospy.Rate(rate)
        self.cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=5)
        self.errs = []

    def get_robot_pose(self):
        '''
        get x and y coordinate of position of the robot
        get the yaw angle of robot in world.
        We call it, heading of the robot.
        '''
        # waiting for the most recent message from topic /odom
        msg = rospy.wait_for_message("/odom" , Odometry)

        orientation = msg.pose.pose.orientation
        position = msg.pose.pose.position

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

        return position.x, position.y, yaw

    def update_goal_counter(self):
```

```
        cur_x, cur_y, yaw = self.get_robot_pose()
        dist = math.sqrt((cur_x-self.goals[self.goal_counter][0])**2 + (cur_y-
self.goals[self.goal_counter][1])**2)
        if dist < self.epsilon:
            self.goal_counter += 1
        req = GetVFHDataRequest()
        req.goal_x, req.goal_y = self.goals[self.goal_counter]
        req.threshold = self.thresholds[self.goal_counter]
        resp = self.calc_client(req)
        return resp

    def refine_angle(self, angle):
        final_angle = 0
        if 0<=angle<130:
            final_angle = angle
        elif 230<=angle<360:
            final_angle = angle-360
        else:
            final_angle = 0
        return math.radians(final_angle)

    def control_velocity(self):

        resp = self.update_goal_counter()

        d = self.refine_angle(resp.desired_angle)
        sum_i_theta = 0
        prev_theta_error = 0

        move_cmd = Twist()
        move_cmd.angular.z = 0
        move_cmd.linear.x = self.V_max



        while not rospy.is_shutdown():

            self.cmd_vel.publish(move_cmd)
            resp = self.update_goal_counter()
            d = self.refine_angle(resp.desired_angle)
            #rospy.loginfo(f"desired_angle : {[resp.desired_angle]} ")
            err = d - self.D
            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)

            omega = P + I + D
```

```
            prev_theta_error = err
            #rospy.loginfo(f"omega : {[omega]} ")

            #self.h_m = self.thresholds[self.goal_counter]+2
            h_prim_prim_c = min(self.h_m, resp.h_prim_c)
            V_prim = self.V_max*(1-(h_prim_prim_c/self.h_m))
            V = V_prim*(1-(omega/self.omega_max))+self.V_min

            move_cmd.angular.z = omega
            move_cmd.linear.x = V

            self.r.sleep()


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

- **اجرایی کردن کدهای پایتون**

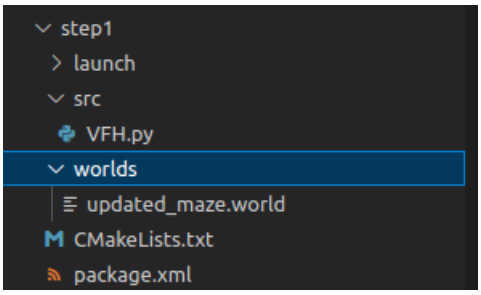
در مرحله بعد باید تمامی کدهای پایتون را executable کنیم. برای این کار لازم است در ترمینال در پکیج step2 کد زیر را اجرا کنیم:

chmod +x src/*.py

```
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src$ cd step1/
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src/step1$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src/step1$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src/step1/src$ ls
VFH.py
mahdi@mahdi:~/Desktop/Robotics/project5/hw5_ws/src/step1/src$
```

- **اضافه کردن updated_maze.world:**

برای این منظور در همین فولدر step1 یک فولدر به نام worlds ایجاد میکنیم و سپس فایل updated_maze.world داده شده را در آنجا اضافه مینماییم.




- **نوشتن لانچ فایل**

سپس به سراغ نوشتن لانچ فایل میرویم. در همین پکیج step1 لازم است تا یک فولدر launch ایجاد کنیم. یک روش این است که یک لانچ فایل با نام my_empty_world را که در پوشه لانچ مربوط به turtle_bot بود ( و درواقع یک کپی از empty_world.launch است) را include کنیم. 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" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find step1)/worlds/updated_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 vfh.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 (0, 0) و با زاویه 0 رادیان باید اضافه شود.

سپس نود VFH.py که ساختیم را باید بالا بیاورد. خروجی آن را هم screen گذاشتیم که خروجی‌های loginfo را در که در کد گذاشتیم در ترمینال نشان دهد.

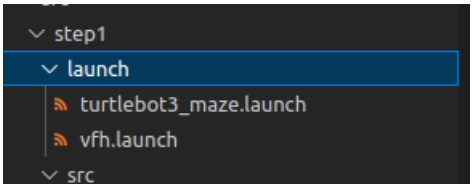
```
<launch>

    <include file="$(find step1)/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>

    <node pkg="step1" type="VFH.py" name="VFH" output="screen"></node>

</launch>
```

در نهایت پوشه مربوطه به صورت زیر در می اید:




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 step1 vfh.launch

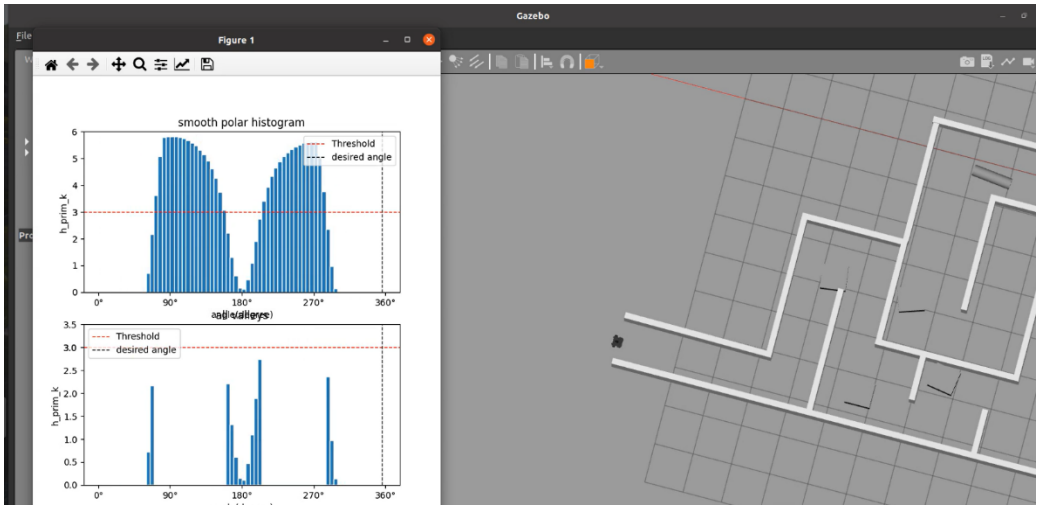
```
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project4/hw4_ws$ roslaunch step2 control.launch follow_type:=right
... logging to /home/mahdi/.ros/log/505aef56-11fe-11ee-8c1c-5341409bfe14/roslaunch-mahdi-107324.log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

در نهایت خواهیم دید با تنظیم کردن درست مقادیر threshold میتوان ربات را به انتها رساند



## سناریوی دوم

در این بخش پس از مطالعه سایت داده شده و دانلود پکیج مورد نظر در پکیجمان میتوانیم به پیاده اسزی بپردازیم.

ابتدا به کالیبره کردن دوربین میپردازیم. ربات وقتی به موانع نرسیده است به کمک کانتراست خطوط به راحتی خط را دنبال میکند. وقتی به مانع رسید باید یک سری state تعریف کنیم و براساس آن‌ها پیش برویم:

حالت Lane Detection در این حالت فقط خط را دنبال میکند و به موانع کاری ندارد.

حالت GO Straight که در آن به صورت مستقیم در جهت مورد نظر حرکت میکند و فقط سرعت خطی داریم

حالت چرخش که خود دوتاست:

۱- CLOCKWISE
۲- COUNTER_CLOCKWISE

وقتی به مانع رسیدیم به اندازه ۴۵ درجه چرخیده و سپس به صورت مستقیم میرویم و بعد در نهایت خط را دنبال میکنیم.