|  |  |  |
| --- | --- | --- |
| C:\Users\mohammad\Downloads\image(1).png | به نام خدا | C:\Users\mohammad\Pictures\amirkabirLogo.png |
| **دانشگاه صنعتی امیرکبیر**  **دانشکده‌ مهندسی کامپیوتر**  **اصول علم ربات**  **پروژه نهایی** | | |

|  |  |
| --- | --- |
| مهدی رحمانی – هادی ناظمی اسفندیاری | نام و نام خانوادگی |
| 9731701- 9730034 | شماره‌ دانشجویی |
|  | تاریخ ارسال گزارش |

­

**فهرست گزارش سوالات**

[آماده‌سازی محیط ROS 3](#_Toc139092067)

[سناریوی اول 4](#_Toc139092068)

[سناریوی دوم 21](#_Toc139092069)

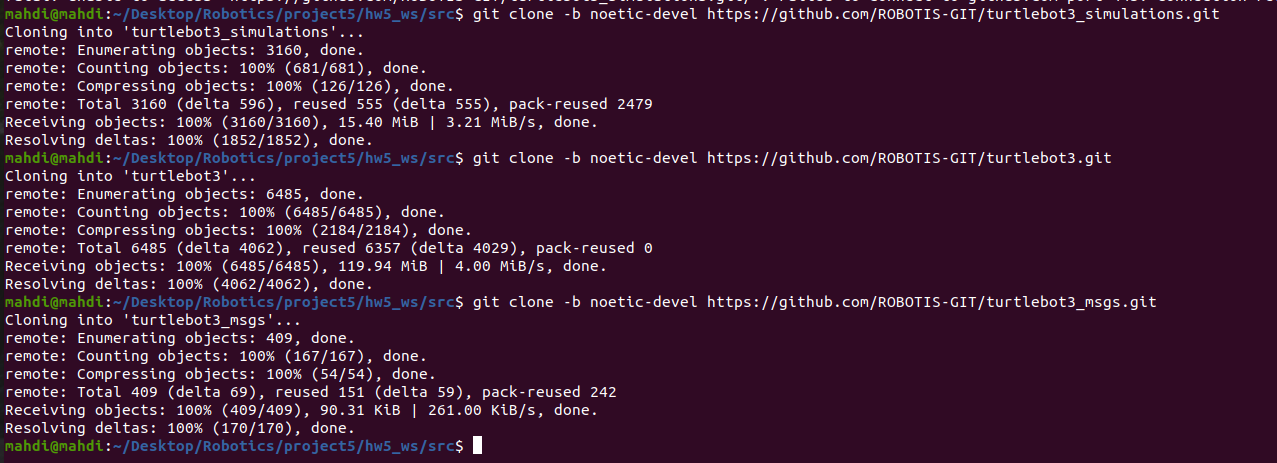
# آماده‌سازی محیط ROS

A picture containing text, screenshot, font

Description automatically generatedابتدا یک Work space برای این پروژه میسازیم و آن را initialize میکنیم:

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

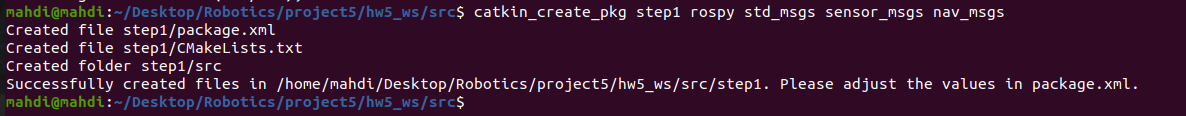


# سناریوی اول

* **آماده‌سازی‌های اولیه گام اول**

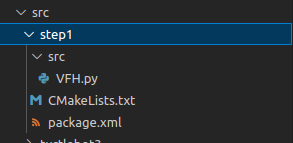
برای این قسمت یک پکیج جداگانه درست میکنیم به نام step1 میکنیم و dependencyهایی که ممکن است به کار بیایند را هم اضافه کنیم.

catkin\_create\_pkg step1 rospy std\_msgs sensor\_msgs nav\_msgs



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:

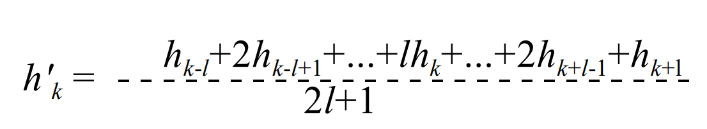
* Starting point: (0, 0)
* Goal point: (13, 7)

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 5 درجه 5 درجه را باید یک sector تعریف کرد. پس مقدار و n به صورت زیر است که n برابر با تعداد sectorها میباشد.

*همچنین در رابطه گفته شده که مقادیر a و b به صورت زیر میباشند:*

*همچنین گفته شده که d حداکثر فاصله میشود باید مقدار a- bd برابر با 0 شود. بنابراین داریم:*

*همچنین در رابطه زیر که برای smoothing هست باید مقدار l را برابر با 1 در نظر بگیریم:*

*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 طبق مقاله داریم:*

*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 داریم که به صورت زیر تعیین میشود:*

*همچنین یک مقدار داریم که در این جا و با ایده گیری از مقاله برابر با :*

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

*یک مقدار نیاز است که ماکسیسم سرعت زاویه‌ای yaw ربات میباشد که برابر با :*

* **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()

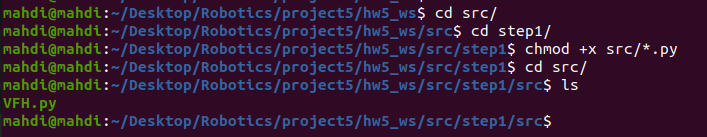
    except rospy.ROSInterruptException:

        rospy.loginfo("contoller terminated.")

* **اجرایی کردن کدهای پایتون**

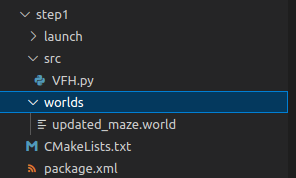
در مرحله بعد باید تمامی کدهای پایتون را executable کنیم. برای این کار لازم است در ترمینال در پکیج step2 کد زیر را اجرا کنیم:

chmod +x src/\*.py



* **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"/>

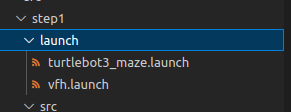
        <arg name="yaw" value="0.0"/>

    </include>

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

</launch>

در نهایت پوشه مربوطه به صورت زیر در می اید:

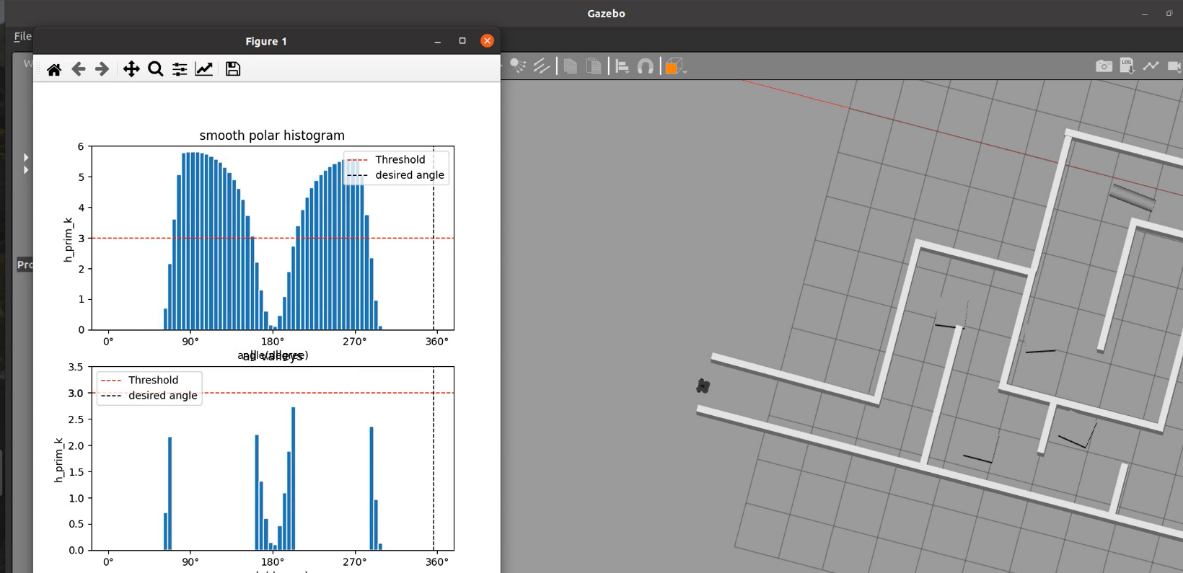


سپس در آخر لازم است تا به دایرکتوری ورک اسپیس برویم و catkin\_make را صدا بزنیم. سپس برای استفاده لازم است تا ابتدا سورس کنیم و سپس ربات را اکسپورت کنیم و در نهایت roslaunch را صدا بزنیم:

* . devel/setup.bash
* export TURTLEBOT3\_MODEL=waffle
* roslaunch step1 vfh.launch

A screenshot of a computer program

Description automatically generated with medium confidence

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:

1. CLOCKWISE
2. COUNTER\_CLOCKWISE

وقتی به مانع رسیدیم به اندازه 45 درجه چرخیده و سپس به صورت مستقیم میرویم و بعد در نهایت خط را دنبال میکنیم.