

**دانشگاه صنعتی امیرکبیر**

**(پلی**‏**تکنیک تهران)**

**درس رباتیک**

گزارش تمرین صفر

**نگارش**

**محمدرضا حیدری : 9926053**

**استاد درس**

**دکتر جوانمردی**

**فررودین ماه 1403**

فهرست

[منطق تمرین 3](#_Toc162991205)

[پیام ها و تعریف هر یک 3](#_Toc162991206)

[سنسور 4](#_Toc162991207)

[خروجی سنسور 5](#_Toc162991208)

[کنترلر 5](#_Toc162991209)

[خروجی کنترلر 11](#_Toc162991210)

[موتور ها 11](#_Toc162991211)

[قسمتی از خروجی موتور یک 12](#_Toc162991212)

[قسمتی از خروجی موتور دو 13](#_Toc162991213)

[راهنمای اجرای کد 13](#_Toc162991214)

[Rqt\_graph 13](#_Toc162991215)

# منطق تمرین

ربات ابتدا نزدیک ترین مانع به خود را با استفاده از داده سنسور ها تشخیص داده و سپس در جهتی چرخش کند تا به سمت مانع

دور تر باشد و به سمت آن حرکت کتد تا زمانی که از مانع نزدیک تر 10 سانتی متر فاصله بگیرد.

{

    "1": "57 86 95 180",

    "2": "58 87 92 175",

    "3": "59 89 91 174",

    "4": "60 92 80 173",

    "5": "61 94 80 172",

    "6": "62 95 75 171",

    "7": "65 96 74 165",

    "8": "79 101 73 150",

    "9": "80 103 57 145",

    "10": "90 110 43 140"

}

در اولین فراخوانی کوچکترین داده 57 است و بزرگترین 180 پس باید به سمت داده بزرگتر بچرخیم و ادامه دهیم.

در دومین فراخوانی از 57 ، 10 سانتی متر فاصله نگرفته ایم و باید ادامه دهیم این روند تا عدد 79 ادامه دارد که فاصله از 57 از ده سانتی متر بیشتر است اکنون نزدیک ترین مانع 73 است و داده بزرگتر 150 و دوباره روند قبلی طی می شود

# پیام ها و تعریف هر یک

1 ) پیام proximity.msg فاصله مانع ها تا ربات را بیان می کند.

int64 left

int64 up

int64 right

int64 down

2 ( پیام order.msg دستورات ارسالی به موتور را مشخص میکند .

int64 rotate\_robot\_degree

string rotate\_wheel

int64 velocity

string rotate\_robot\_direction

rotate\_robot\_degree : مقدار چرخش ربات به درجه

rotate\_wheel : جهت چرخش چرخ ، منفی به معنای عقب گرد و مثبت به معنای حرکت به جلو است.

Velocity : بیانگر سرعت خطی است که واحد آن سانتی متر بر ثانیه است.

rotate\_robot\_direction : بیانگر جهت چرخش کل ربات است که یا ساعتگرد هست یا پاد ساعتگرد.

# سنسور

وظیفه فایل و node سنسور ، خواندن اطلاعات فایل distances.jsonدر هر فراخوانی است و درنهایت باید اطلاعات را در قالب یک پیام شخصی سازی شده publish کند.

فایل سنسور دارای سه تابع است که وظایف هر کدام به شرح زیر است :

1 ) distance\_sensor\_node\_define

تعریف نود سنسور و تعریف یک تاپیک به نام distance

2 ) read\_distance

خواندن اطلاعات از فایل distances.json

3 ) start

این تابع اصلی است که کارهای سنسور را هماهنگ کرده و از دو تابع قبلی استفاده میکند نهایتا پیام را publish می کند.

#!/usr/bin/python3

import rospy

from hwzero.msg import proximity

import json

import os

iteration = 1

def read\_distance(iteration):

    script\_dir = os.path.dirname(\_\_file\_\_)

    file\_path = os.path.join(script\_dir, 'distances.json')

    with open(file\_path, 'r') as f:

        distance\_data = json.load(f)

    return [int(x) for x in distance\_data[str(iteration)].split()]

def distance\_sensor\_node\_define():

    rospy.init\_node("sensor",anonymous=True)

    publisher = rospy.Publisher("distance",proximity,queue\_size=10)

    rate = rospy.Rate(0.75)

    return publisher,rate

def start():

 publisher,rate = distance\_sensor\_node\_define()

 global iteration

 while not rospy.is\_shutdown() and iteration<=10:

    rate.sleep()

    rospy.loginfo(iteration)

    left, up, right, down = read\_distance(iteration)

    distance\_msg = proximity()

    distance\_msg.left = left

    distance\_msg.up = up

    distance\_msg.right = right

    distance\_msg.down = down

    publisher.publish(distance\_msg)

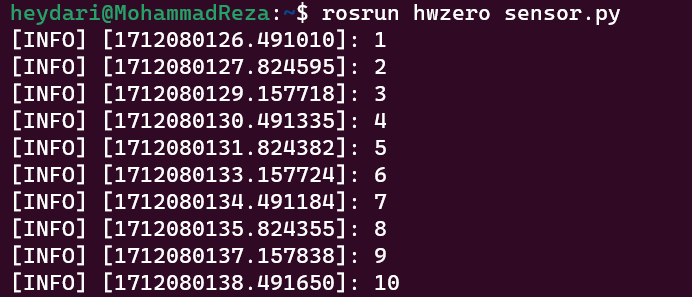
    iteration = iteration+1

if \_\_name\_\_ == '\_\_main\_\_':

    start()

## خروجی سنسور

هر عدد بیانگر تعداد دفعات خواندن اطلاعات از فایل json است.



# کنترلر

وظیفه فایل و نود کنترلر تصمیم گیری بر اساس اطلاعات سنسور و فرستادن دستور مناسب به هر یک از موتور ها است

فایل کنترلر دارای پنج تابع اصلی است که وظایف هر یک به شرح زیر است :

1 ) callback

آماده سازی اطلاعات ارسالی از سنسور برای استفاده دیگر توابع

2 ) controller\_node

تعریف نود کنترلر و دو تاپیک برای دستور به موتور ها همچنین تعریف یک دریافت کننده اطلاعات از سنسور

3 ) create\_order\_msg

آماده سازی پیام مناسب جهت ارسال به موتور ها ، قسمت زیادی از منطق برنامه در این تابع است.

4 ) create\_zero\_messaage

آماده سازی پیام مناسب به کنترلر وقتی نیازی به چرخش ربات نیست و ربات باید به جهت حرکت خود ادامه دهد.

5 )     start2

این تابع اصلی است که کارهای کنترلر را هماهنگ کرده و از چهار تابع قبلی استفاده میکند نهایتا پیام ها را publish می کند.قسمتی از منطق های برنامه نیز در این تابع به طور جداگانه برنامه ریزی شده است.

#!/usr/bin/python3

import rospy

from hwzero.msg import proximity

from hwzero.msg import order

import random

import time

Data = None

Data\_Old = None

def callback(data):

    global Data

    Data = data

def controller\_node():

    rospy.init\_node('controller', anonymous=True)

    engine1\_pub = rospy.Publisher('engine1\_order', order, queue\_size=10)

    engine2\_pub = rospy.Publisher('engine2\_order', order, queue\_size=10)

    rospy.Subscriber('distance', proximity, callback)

    return engine1\_pub, engine2\_pub

def start2():

    engine1\_pub, engine2\_pub = controller\_node()

    while not rospy.is\_shutdown():

        if Data:

            start\_time = time.time()

            engine\_msg1,engine\_msg2 = create\_order\_msg(Data)

            engine\_msg1.velocity = 0

            engine\_msg2.velocity = 0

            engine1\_pub.publish(engine\_msg1)

            engine2\_pub.publish(engine\_msg2)

            rospy.loginfo("first data recevied")

            Data\_Old = Data

            Data\_List = [Data.left,Data.up,Data.down,Data.right]

            Data\_old\_list = [Data\_Old.left,Data\_Old.up,Data\_Old.down,Data\_Old.right]

            min\_number = min(Data\_old\_list)

            min\_index = Data\_old\_list.index(min\_number)

            min\_element = Data\_List[min\_index]

            while not rospy.is\_shutdown():

                Data\_List = [Data.left,Data.up,Data.down,Data.right]

                Data\_old\_list = [Data\_Old.left,Data\_Old.up,Data\_Old.down,Data\_Old.right]

                min\_number = min(Data\_old\_list)

                min\_index = Data\_old\_list.index(min\_number)

                min\_element = Data\_List[min\_index]

                if max(Data\_List) == 140 and max(Data\_old\_list)==140:

                    rospy.signal\_shutdown("Shutting down")

                    exit()

                rospy.loginfo('min of old distance : %s' , min\_number )

                rospy.loginfo('now we are in  : %s' , min\_element )

                if min\_element - min\_number > 10 or min\_element - min\_number < -10 :

                    engine\_msg1,engine\_msg2 = create\_order\_msg(Data)

                    elapsed\_time = time.time() - start\_time

                    engine\_msg1.velocity = abs(int((min\_element - min\_number)/elapsed\_time))

                    engine\_msg2.velocity = engine\_msg1.velocity

                    start\_time = time.time()

                    rospy.loginfo("we should rotate")

                    engine1\_pub.publish(engine\_msg1)

                    engine2\_pub.publish(engine\_msg2)

                    Data\_Old = Data

                else:

                    engine\_msg1,engine\_msg2 = create\_zero\_messaage()

                    rospy.loginfo("do not need rotate")

                    elapsed\_time = time.time() - start\_time

                    engine\_msg1.velocity = abs(int((min\_element - min\_number)/elapsed\_time))

                    engine\_msg2.velocity = engine\_msg1.velocity

                    start\_time = time.time()

                    engine1\_pub.publish(engine\_msg1)

                    engine2\_pub.publish(engine\_msg2)

                rate = rospy.Rate(.75)

                rate.sleep()

def create\_order\_msg(Data):

    Data2 = [Data.left,Data.up,Data.down,Data.right]

    def find\_way(Data):

        if max(Data2) == Data.left:

            maxIndex = "left"

        if max(Data2) == Data.right:

            maxIndex = "right"

        if max(Data2) == Data.up:

            maxIndex = "up"

        if max(Data2) == Data.down:

            maxIndex = "down"

        if min(Data2) == Data.left:

            minIndex = "left"

        if min(Data2) == Data.right:

            minIndex = "right"

        if min(Data2) == Data.up:

            minIndex = "up"

        if min(Data2) == Data.down:

            minIndex = "down"

        if maxIndex == "left":

            if minIndex == "right":

                way = 2

                sgn = 1

            if minIndex == "down":

                way = 1

                sgn = 1

            if minIndex == "up":

                way = 1

                sgn = -1

        if maxIndex == "right":

            if minIndex == "left":

                way = 2

                sgn = 1

            if minIndex == "down":

                way = 1

                sgn = -1

            if minIndex == "up":

                way = 1

                sgn = 1

        if maxIndex == "down":

            if minIndex == "right":

                way = 1

                sgn = 1

            if minIndex == "left":

                way = 1

                sgn = -1

            if minIndex == "up":

                way = 2

                sgn = 1

        if maxIndex == "up":

            if minIndex == "right":

                way = 1

                sgn = -1

            if minIndex == "down":

                way = 2

                sgn = 1

            if minIndex == "left":

                way = 1

                sgn = 1

        return way,sgn

    way , sgn = find\_way(Data)

    def UpLeft\_LeftDown\_DownRight\_RightUp(sgn):

        engine\_msg1 = order()

        engine\_msg2 = order()

        if sgn == 1:

            rospy.loginfo("robot should rotate 90 degree ccw")

            engine\_msg1.rotate\_robot\_degree = 90

            engine\_msg1.rotate\_wheel = "negative"

            engine\_msg1.velocity = None

            engine\_msg1.rotate\_robot\_direction = "ccw"

            engine\_msg2.rotate\_robot\_degree = 90

            engine\_msg2.rotate\_wheel = "positive"

            engine\_msg2.velocity = None

            engine\_msg2.rotate\_robot\_direction = "ccw"

        if sgn == -1:

            rospy.loginfo("robot should rotate 90 degree cw")

            engine\_msg1.rotate\_robot\_degree = 90

            engine\_msg1.rotate\_wheel = "positive"

            engine\_msg1.velocity = None

            engine\_msg1.rotate\_robot\_direction = "cw"

            engine\_msg2.rotate\_robot\_degree = 90

            engine\_msg2.rotate\_wheel = "negative"

            engine\_msg2.velocity = None

            engine\_msg2.rotate\_robot\_direction = "cw"

        return engine\_msg1,engine\_msg2

    def UpDown\_LeftRight(sgn):

        sgn = random.choice([1, -1])

        engine\_msg1 = order()

        engine\_msg2 = order()

        rospy.loginfo("robot should rotate 180 degree cw or ccw")

        if sgn == 1:

            engine\_msg1.rotate\_robot\_degree = 180

            engine\_msg1.rotate\_wheel = "positive"

            engine\_msg1.velocity = None

            engine\_msg1.rotate\_robot\_direction = "cw"

            engine\_msg2.rotate\_robot\_degree = 180

            engine\_msg2.rotate\_wheel = "negative"

            engine\_msg2.velocity = None

            engine\_msg2.rotate\_robot\_direction = "cw"

        if sgn == -1:

            engine\_msg1.rotate\_robot\_degree = 180

            engine\_msg1.rotate\_wheel = "negative"

            engine\_msg1.velocity = None

            engine\_msg1.rotate\_robot\_direction = "ccw"

            engine\_msg2.rotate\_robot\_degree = 180

            engine\_msg2.rotate\_wheel = "positive"

            engine\_msg2.velocity = None

            engine\_msg2.rotate\_robot\_direction = "ccw"

        return engine\_msg1,engine\_msg2

    if way == 1:

        engine\_msg1,engine\_msg2 = UpLeft\_LeftDown\_DownRight\_RightUp(sgn)

    if way == 2:

        engine\_msg1,engine\_msg2 = UpDown\_LeftRight(sgn)

    return engine\_msg1,engine\_msg2

def create\_zero\_messaage():

    engine\_msg1 = order()

    engine\_msg2 = order()

    engine\_msg1.rotate\_robot\_degree = 0

    engine\_msg1.rotate\_wheel = "positive"

    engine\_msg1.velocity = None

    engine\_msg1.rotate\_robot\_direction = 0

    engine\_msg2.rotate\_robot\_degree = 0

    engine\_msg2.rotate\_wheel = "positive"

    engine\_msg2.velocity = None

    engine\_msg2.rotate\_robot\_direction = "None"

    return engine\_msg1,engine\_msg2

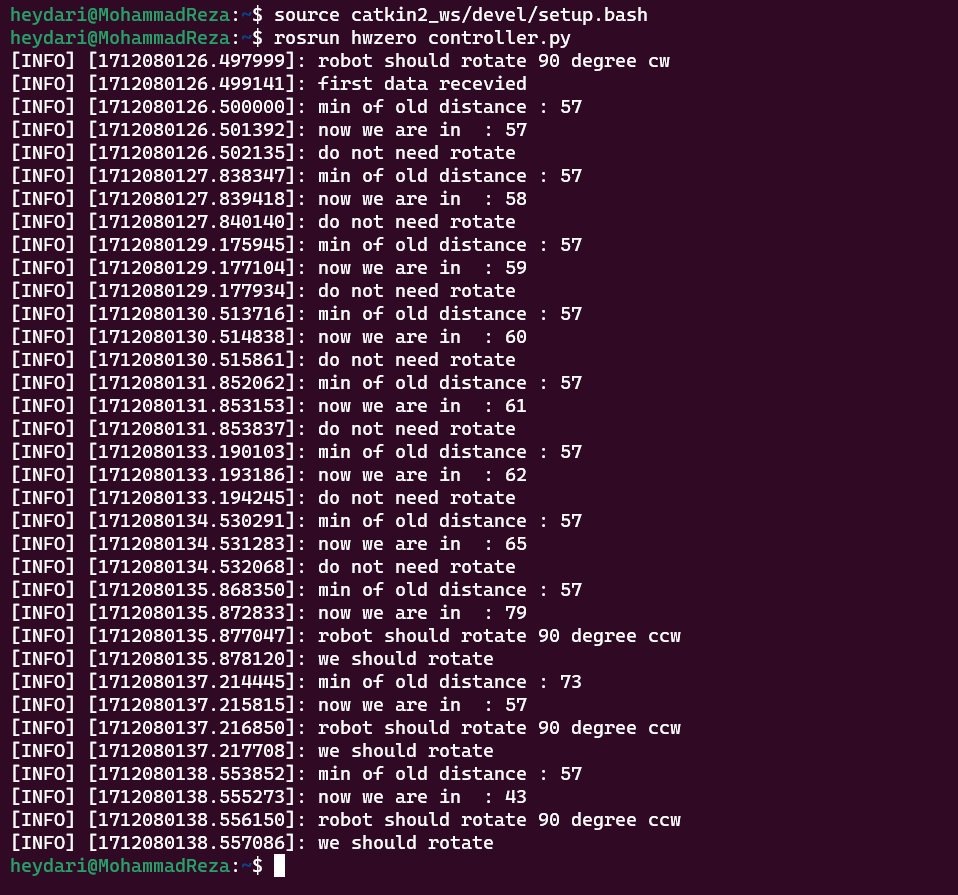
if \_\_name\_\_ == '\_\_main\_\_':

    start2()

نکات تکمیلی کد کنترلر :

* متغیر way در صورت برابر بودن با عدد یک یعنی یک واحد چرخش نیاز است و برابر بودن با عدد دو یعنی دو واحد یا 180 در جه چرخش نیاز است.
* متغیر sgn در صورت برابر بودن با یک یعنی پاد ساعتگرد و برابر بودن با منفی یک یعنی ساعتگرد
* سرعت های لحظه ای در موتور نمایش داده می شود که در دیتای دریافتی اول صفر نشان داده می شود.

## خروجی کنترلر



# موتور ها

وظیفه دو فایل مربوط به موتور فقط تعریف دو نود و دریافت و چاپ دستورات کنترلر است.

منظور از موتور اول موتور سمت چپ و موتور دوم موتور سمت راست است.

برای مثال فایل موتور اول به شرح زیر است :

#!/usr/bin/python3

import rospy

from hwzero.msg import order

def engine\_node\_callback1(data):

    rospy.loginfo("engine1 :%s",data)

def start3():

    rospy.init\_node('engine\_node', anonymous=True)

    rate = rospy.Rate(.75)

    while not rospy.is\_shutdown():

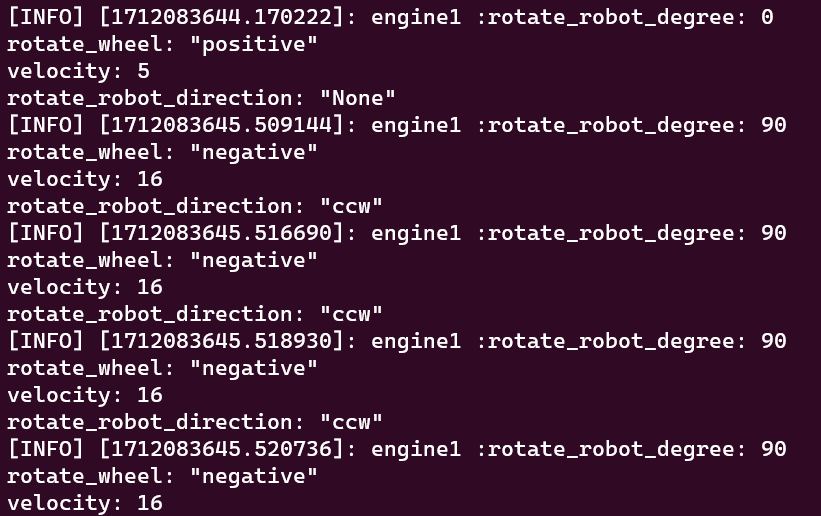
        rate.sleep()

        rospy.Subscriber('engine1\_order', order, engine\_node\_callback1)

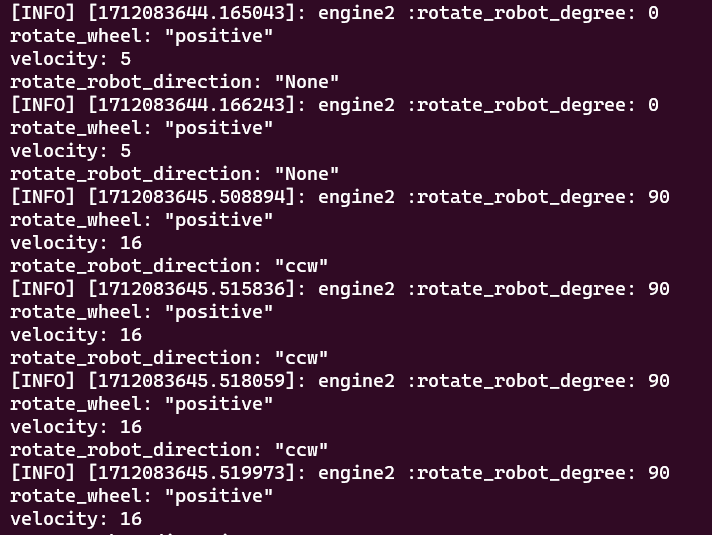
if \_\_name\_\_ == '\_\_main\_\_':

    start3()

## قسمتی از خروجی موتور یک



## قسمتی از خروجی موتور دو



# راهنمای اجرای کد

آخرین فایلی که اجرا می شود باید sensor باشد چون sensor به محض راه اندازی دیتا ارسال میکند و به خاط کم بودن دیتا ها این باعث می شود بعد از ده ثانیه اطلاعات تمام شود پس حتما اخرین فایلی که اجرا میکنید باید sensor باشد.

ترتیب اجرای دیگر فایل ها مهم نیست.

# Rqt\_graph

