Amirkabir University of Technology
(Tehran Polytechnic)

کنترل ربات

سینا ربیعی، علیرضا فقیه علی‌آبادی، معین نصیری

دکتر عبداللهی

پاییز ۱۴۰۱

فهرست مطالب

۱. داده‌سازی

در این بخش داده‌های مناسب برای آموزش شبکه‌ها از ربات دریافت شده و فیلترهای مناسب روی آن‌ها پیاده می‌شود.

۱. داده‌های سنسور

برای دریافت داده‌های سنسور، در کد نوشته شده برای راه‌اندازی ربات به کمک برد raspberry pi 3، کد شماره‌ی ۱ را اضافه می‌کنیم. در این چند خط، داده‌های دریافت شده از سنسورها و ارسال شده برای موتورها به عنوان PWM در لیست‌های مربوطه اضافه شده و این لیست در فایل‌های txt. نوشته می‌شوند. سپس این فایل‌ها را برای برداشتن داده‌ها و ایجاد Data Frame در کد اصلی می‌خوانیم.

```
left_wheel.append(robot1.ctr)
right_wheel.append(robot1.ctr)
left_srf.append(robot1.sensor_left)
right_srf.append(robot1.sensor_right)

f = open("left_srf.txt","w")
f.write(str(left_srf))
f.close()

f = open("right_srf.txt","w")
f.write(str(right_srf))
f.close()

f = open("pwm_left.txt","w")
f.write(str(left_wheel))
f.close()

f = open("pwm_right.txt","w")
f.write(str(right_wheel))
f.close()
```

کد شماره‌ی ۱: نوشتن داده‌های سنسورها و موتورها در فایل درون raspberry pi

۲. فیلتر کردن و بهینه‌سازی داده‌ها

داده‌های بدست آمده از سنسورها دارای نویز و داده‌های پرت فراوانی هستند. زمانی که تمامی داده‌های آموزشی شبکه اینگونه باشند، آموزش آن تقریبا غیرممکن خواهد بود. به همین دلیل با استفاده از یک شرط if و یک فیلتر مرتبه اول ساده و همچنین تنظیم دستی داده‌ها، اعداد بدست آمده از سنسورها را بهبود می‌بخشیم. در ادامه کدهای مربوط به اطلاحات روی داده‌های و نمونه از داده‌های بهبود داده‌شده آورده شده.

```
def FirstOrder(x,dt=1,taw=9):
    a = dt/(dt + taw)
    y = [0 for i in range(len(x))]
    y[0] = x[0]
    for i in range(len(x)-1):
```

```
        y[i+1] = y[i] + a*(x[i+1] - x[i])
    return y
```

کد شماره‌ی ۲: فیلتر مرتبه اول ساده برای دمپ کردن تغییرات بزرگ

```
def reject_dist(x_new,x_old):
    if (abs(x_new-x_old)>40):
        return x_old,x_old
    else:
        return x_old,x_new
```

کد شماره‌ی ۳: شرط if برای گرفتن تغییرات و جامپ‌های موجود در مقادیر سنسور

[0, 236, 278, 277, 275, 274, 271, 82, 73, 335, 79, 75, 66, 63, 61, 60, 58, 64, 53, 51, 56, 52, 55, 44, 255, 244, 241, 46, 236, 234, 229, 97, 225, 223, 220, 217, 214, 212, 210, 209]

[100, 93, 90, 89, 87, 83, 80, 79, 77, 76, 75, 66, 63, 61, 60, 58, 55, 53, 51, 53, 54, 55, 57, 60, 65, 70, 74, 79, 83, 89, 97, 225, 223, 220, 217, 214, 212, 210, 209]

شکل ۱: سمت چپ: داده‌های اصلی، سمت راست: داده‌های تغییر داده شده

۳. وارد کردن داده‌ها به لیست و Data Frame

برای خواند فایل‌ها به کمک کتابخانه‌ها globe و os فولدرهای موجود را شناسایی کرده و تمام فایل‌های txt. که با نام‌های pwm_left، pwm_right، right_srf و left_srf موجود دارند را می‌خوانیم. خروجی این دستور به صورت رشته (String) است که به کمک کتابخانه re (regular expression) اعداد آن را پیدا کرده و در لیست‌های جدید append می‌کنیم. توجه داریم که برای برابری طول داده‌های خوانده شده در هر حرکت، این لیست‌های بدست آمده را تا طول ۱۲۰ (یکی بیشتر از طول بیشترین داده‌های حرکت) پد می‌کنیم. در نهایت لیست‌های دوبعدی بدست آمده را یک بعدی کرده و در یک Data Frame ذخیره می‌کنیم.

```
pwm_left = list()
pwm_right = list()
srf_right = list()
srf_left = list()

max_length = 120

for FolderName in [name for name in os.listdir("./Data robot Modified") if
os.path.isdir(os.path.join("./Data robot Modified", name))]:
   path = f'./Data robot Modified/{FolderName}'
   for filename in glob.glob(os.path.join(path, 'pwm_left.txt')):
      with open(os.path.join(os.getcwd(), filename), 'r') as f:
         x = re.findall(r'-?\d+\.?\d*', f.read())
         x = [eval(x[i]) for i in range(len(x))]
         for i in range(max_length-len(x)):
            x.insert(0,0)  # add zero at the begining
         pwm_left.append(x)

   for filename in glob.glob(os.path.join(path, 'pwm_right.txt')):
      with open(os.path.join(os.getcwd(), filename), 'r') as f:
         x = re.findall(r'-?\d+\.?\d*', f.read())
```

```
        x = [eval(x[i]) for i in range(len(x))]
        for i in range(max_length-len(x)):
            x.insert(0,0)  # add zero at the begining
        pwm_right.append(x)

    for filename in glob.glob(os.path.join(path, 'right_srf.txt')):
        with open(os.path.join(os.getcwd(), filename), 'r') as f:
            x = re.findall(r'-?\d+\.?\d*', f.read())
            x = [eval(x[i]) for i in range(len(x))]
            for i in range(max_length-len(x)):
                x.insert(0,0)  # add zero at the begining
            srf_right.append(x)

    for filename in glob.glob(os.path.join(path, 'left_srf.txt')):
        with open(os.path.join(os.getcwd(), filename), 'r') as f:
            x = re.findall(r'-?\d+\.?\d*', f.read())
            x = [eval(x[i]) for i in range(len(x))]
            for i in range(max_length-len(x)):
                x.insert(0,0)  # add zero at the begining
            srf_left.append(x)

pwm_left = [pwm_left[i][j] for i in range(len(pwm_left)) for j in
range(len(pwm_left[i]))]
pwm_right = [pwm_right[i][j] for i in range(len(pwm_right)) for j in
range(len(pwm_right[i]))]
srf_left = [srf_left[i][j] for i in range(len(srf_left)) for j in
range(len(srf_left[i]))]
srf_right = [srf_right[i][j] for i in range(len(srf_right)) for j in
range(len(srf_right[i]))]

controller_data = pd.DataFrame({'pwm left': np.array(pwm_left).reshape(-1,),
                                'pwm right': np.array(pwm_right).reshape(-1,),
                                'srf left': np.array(srf_left).reshape(-1,),
                               'srf right': np.array(srf_right).reshape(-1,)})
```

کد شماره‌ی ۴: خواندن فایل‌های داده‌ها و ذخیره آن‌ها در لیست‌های جدید و Data Frame

	pwm left	pwm right	srf left	srf right
0	0.0	0.0	0	0
1	0.0	0.0	0	0
2	0.0	0.0	0	0
3	0.0	0.0	0	0
4	0.0	0.0	0	0
...	...	...	...	...
78	0.0	0.0	0	0
79	0.0	0.0	0	0
80	0.0	0.0	0	0
81	50.0	80.0	244	52
82	49.0	79.0	289	50

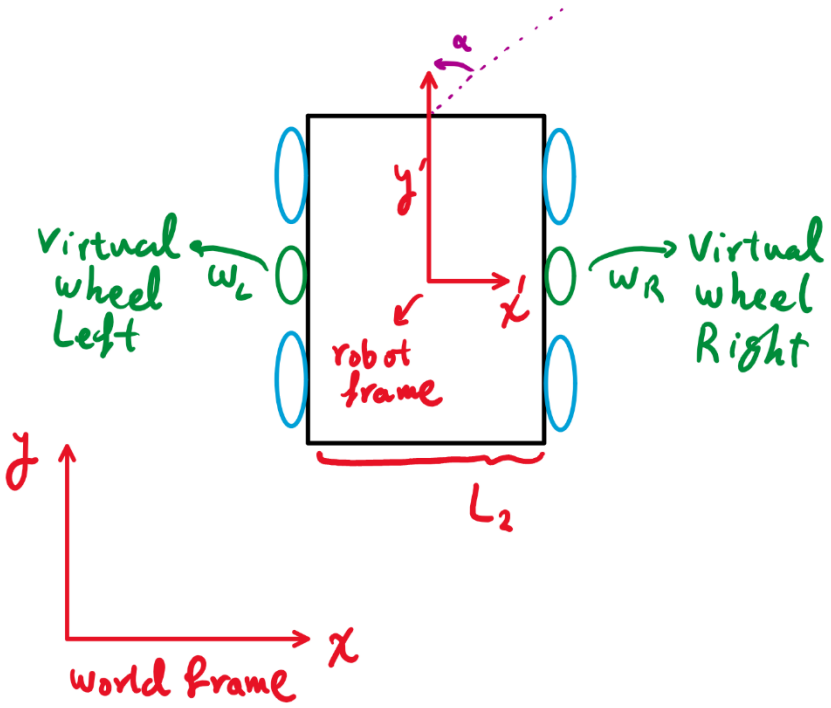
شکل ۲: داده‌های موجود در Data Frame

۴. معادلات سینماتیکی ربات

برای آموزش هرچه بهتر شبکه، داده‌های باید نرمال‌سازی شوند. از آن جهت که داده‌های موجود در این موجموعه هم مثبت و هم منفی هستند، از نرمال‌سازی استاندارد (Standard Scaling) برای این امر استفاده می‌کنیم. برای این کار از کتابخانه scikit-learn کمک می‌گیریم. در کد شماره‌ی ۷، دو تابع نرمال‌سازی استاندارد و MinMax برای نرمال‌سازی داده‌های یک Data Frame آورده شده. برای تست نرمال‌سازی MinMax نیز امتحان شد که نتایج مناسبی به همراه نداشت.

```
def position(pwm_right, pwm_left):
    r = 0.02
    L2 = 1
    max_speed = 100
    Ts = 0.1
    heading = 0
    w = 0
    x = [0]
    y = [0]
    v_right = np.array([i*max_speed*r/100. for i in pwm_right])
    v_left = np.array([i*max_speed*r/100. for i in pwm_left])

    v_x = (v_right + v_left)/2
    v_y = (v_right + v_left)/2
    w = v_left - v_right
    w = [i/L2 for i in w]
    for i in range(len(v_x)-1):
        v_x[i] = v_x[i]*(np.sin(heading))
        x.append(v_x[i]*Ts+x[i])
        v_y[i] = v_y[i]*(np.cos(heading))
        y.append(v_y[i]*Ts+y[i])
        heading = w[i]*Ts+heading

    plt.plot(x[0],y[0],'r*')
    plt.plot(x[1:],y[1:])
    plt.plot(x[len(x)-1],y[len(x)-1],'g*')

    return x, y
```

کد شماره‌ی ۵: توابع معادلات سینماتیکی ربات برای تبدیل PWM به موقعیت x و y



شکل ۳: نمای ربات و چرخ‌های مجازی در نظر گرفته شده

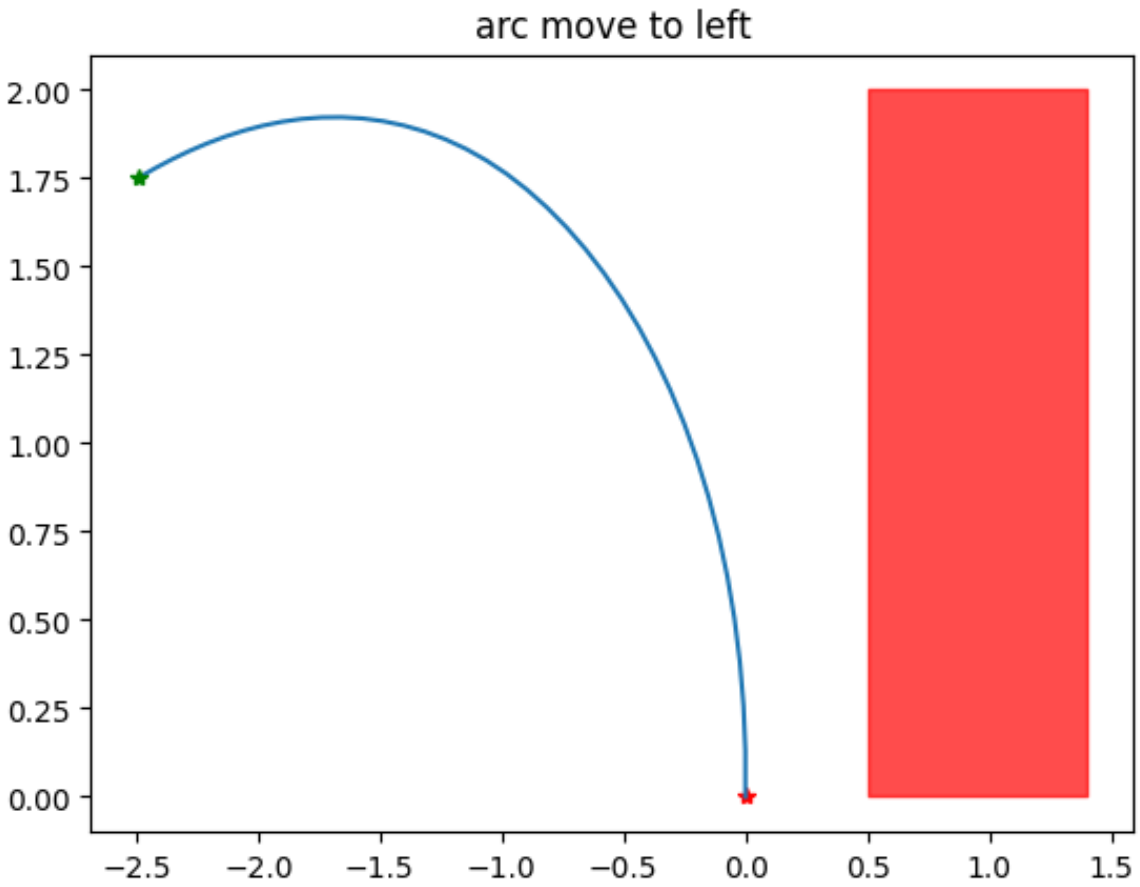
۵. رسم حرکت ربات و تشکیل دیتاست مورد نیاز برای شناسایی‌کننده

به کمک تابع تعریف شده در قسمت قبل و لیست‌های خوانده شده از فایل‌ها، مسیرهای طی شده توسط ربات در فاز جمع‌آوری داده را رسم می‌کنیم. به این ترتیب برای هرکدام از ۱۱ حرکت نمودارهای زیر آورده شده که در آن‌ها ستاره قرمز نقطه شروع و ستاره سبز نقطه پایان حرکت است.

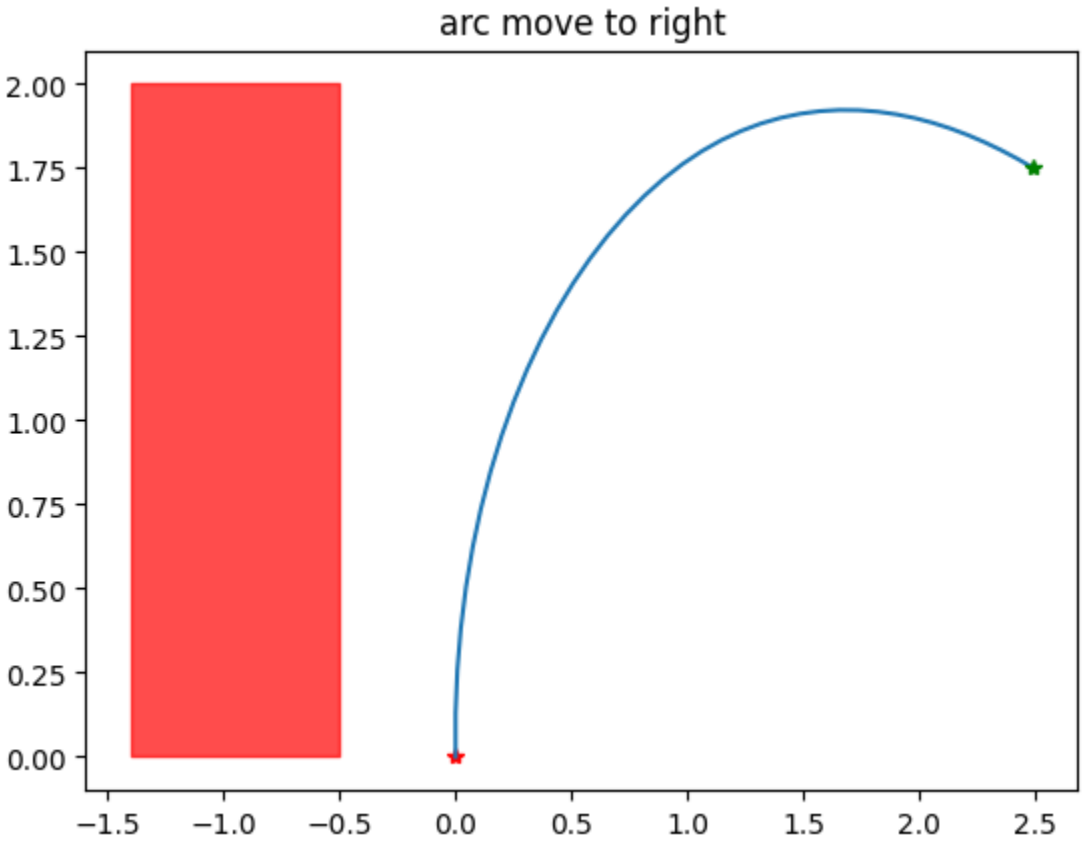
```
pos_x = list()
pos_y = list()

plt.title("arc move to left")
x,y = position(pwm_right[:120], pwm_left[:120])
pos_x.append(x)
pos_y.append(y)
plt.fill_between(np.arange(0.5,1.5,0.1), 0, 2, color='red', alpha=0.7)
plt.show()

plt.title("arc move to right")
x,y = position(pwm_right[120:240], pwm_left[120:240])
pos_x.append(x)
pos_y.append(y)
plt.fill_between(np.arange(-0.5,-1.5,-0.1), 0, 2, color='red', alpha=0.7)
plt.show()

.
.
.
.
```

```
pos_x = np.array(sum(pos_x, []))
pos_y = np.array(sum(pos_y, []))

identification_data = pd.DataFrame({'pwm left': np.array(pwm_left).reshape(-
1,),
                                    'pwm right': np.array(pwm_right).reshape(-1,),
                                    'x position': np.array(pos_x).reshape(-1,),
                                    'y position': np.array(pos_y).reshape(-1,)})
```

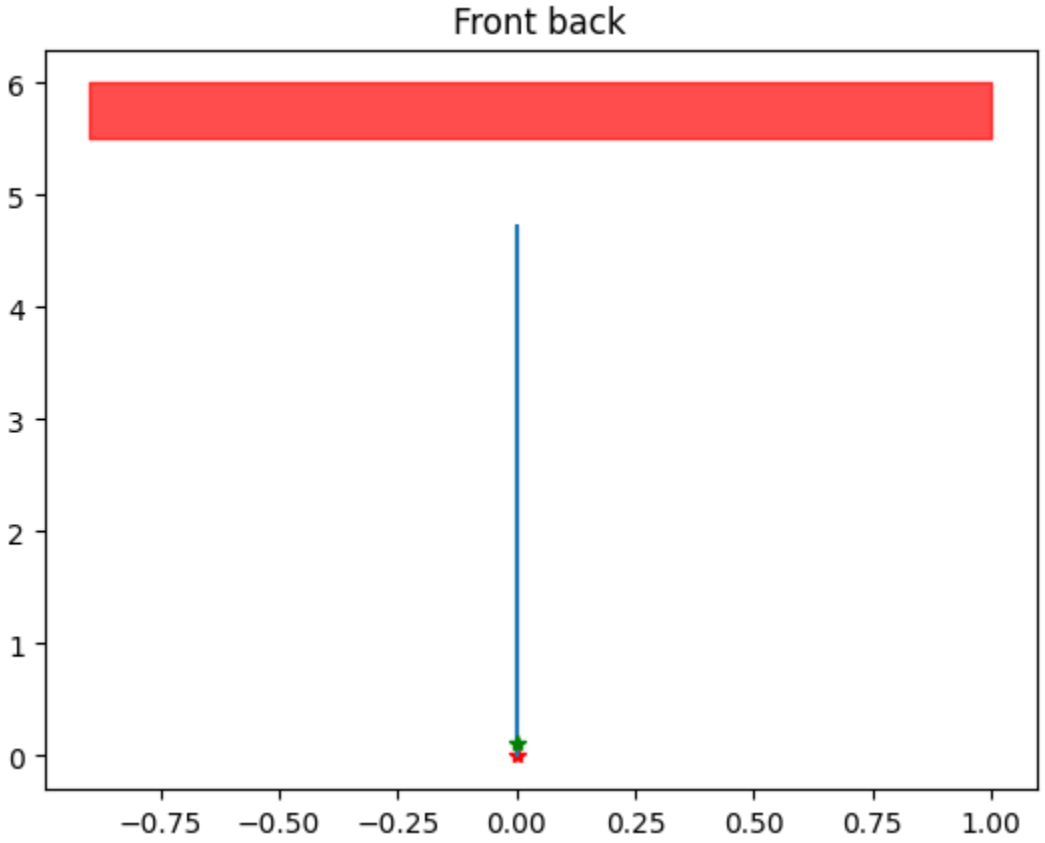
کد شماره‌ی ۶: رسم مسر حرکت ربات و بدست آوردن موقعیت x و y آن به کمک تابع تعریف شده و تشکیل Data Frame از آن‌ها



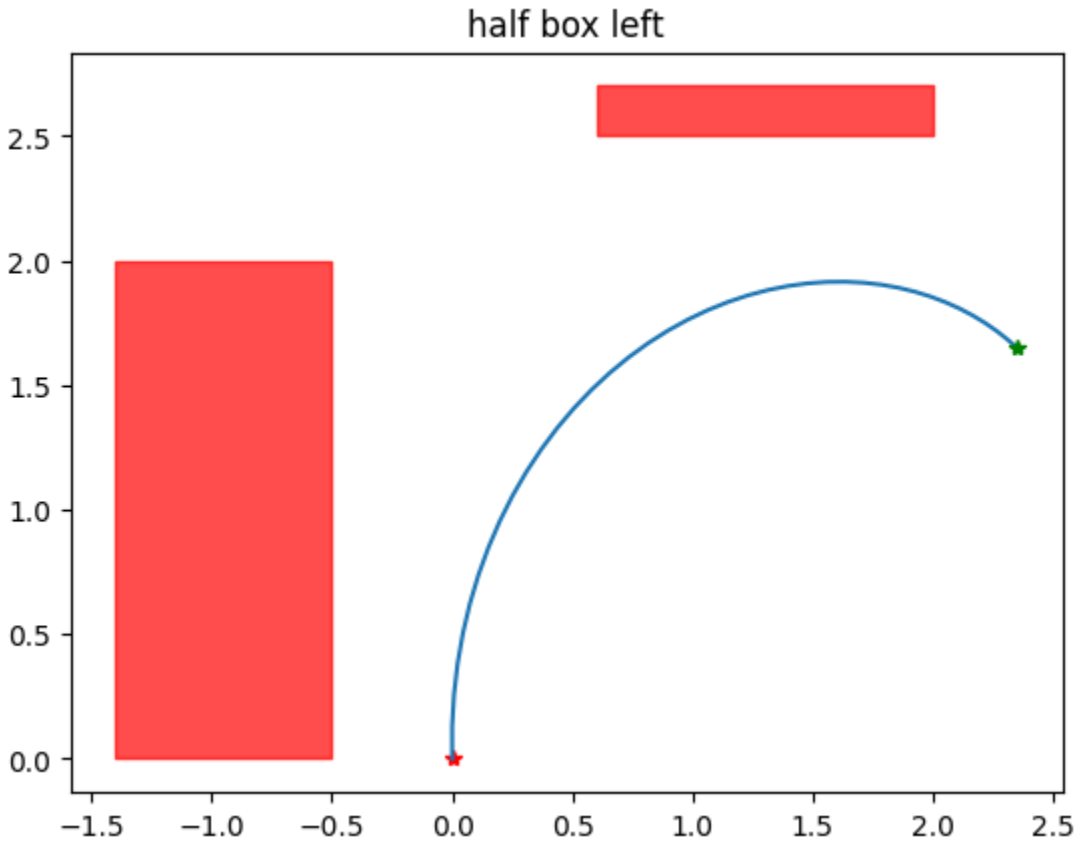
شکل ۴: حرکت قوسی به سمت چپ



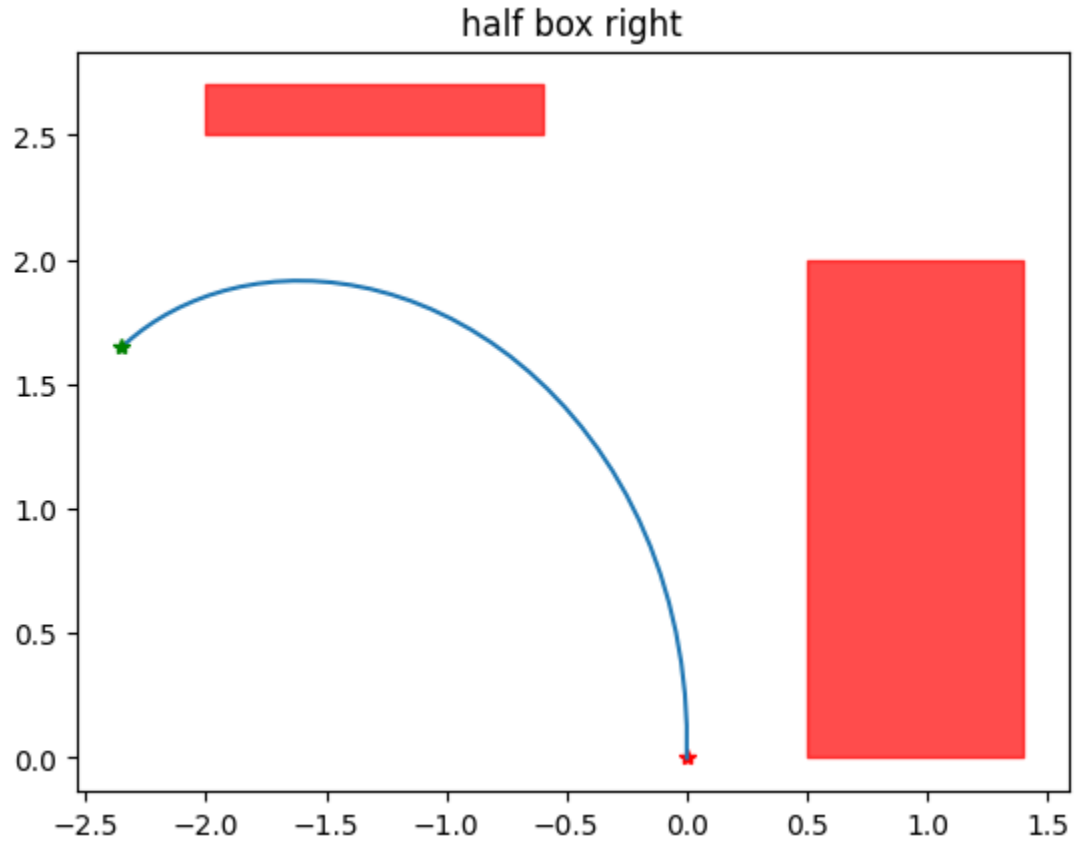
شکل ۵: حرکت قوسی به سمت راست



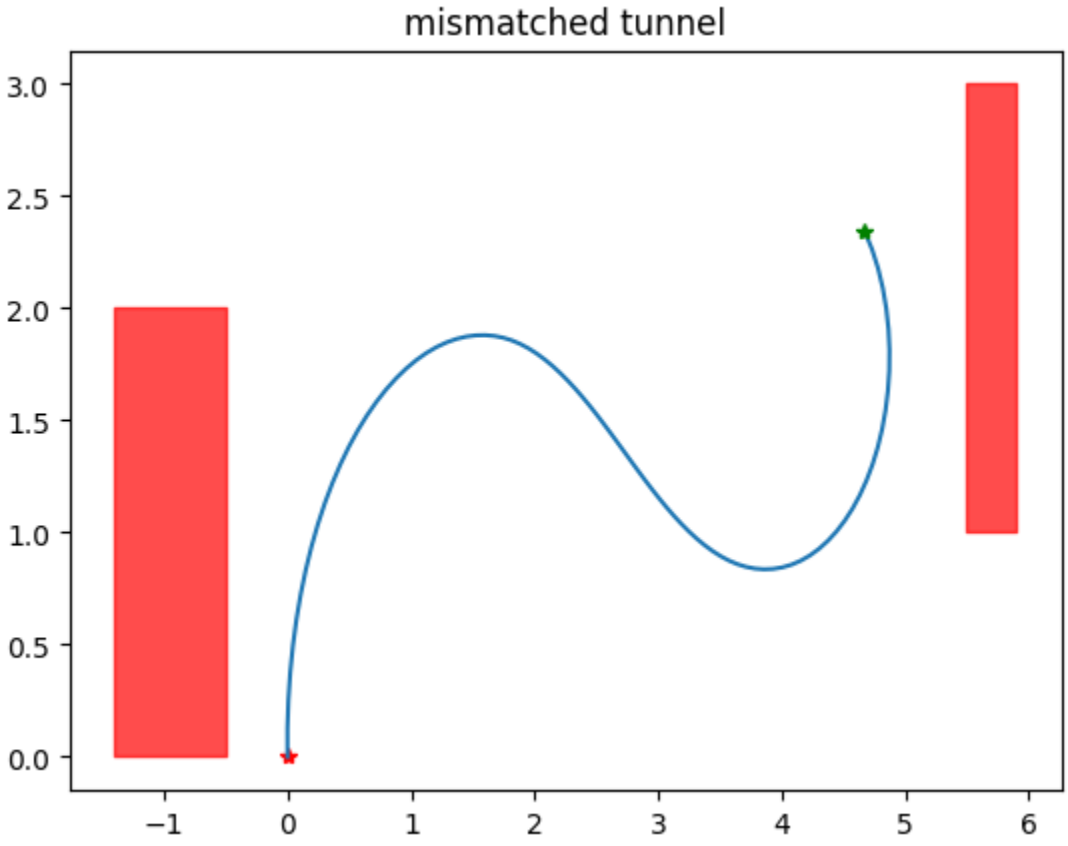
شکل ۶: حرکت به جلو و برگشت به عقب



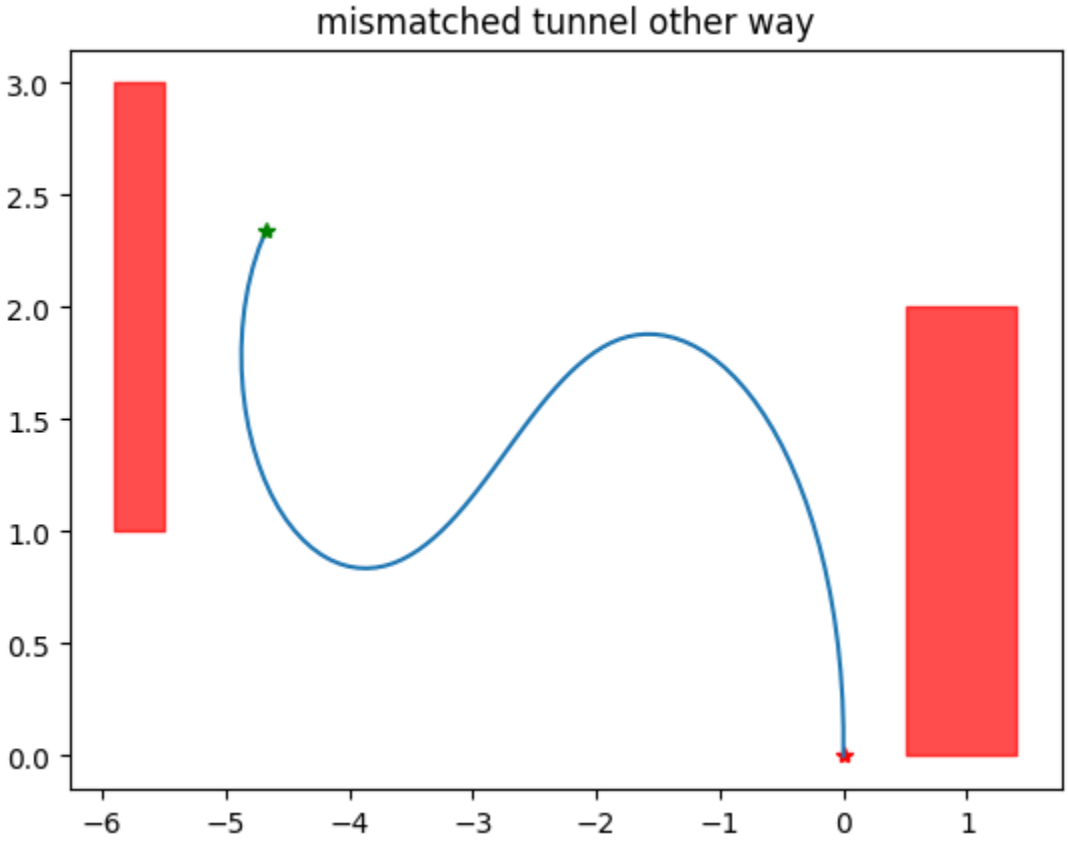
شکل ۷: حرکت قوسی به چپ در یک نیمه جعبه



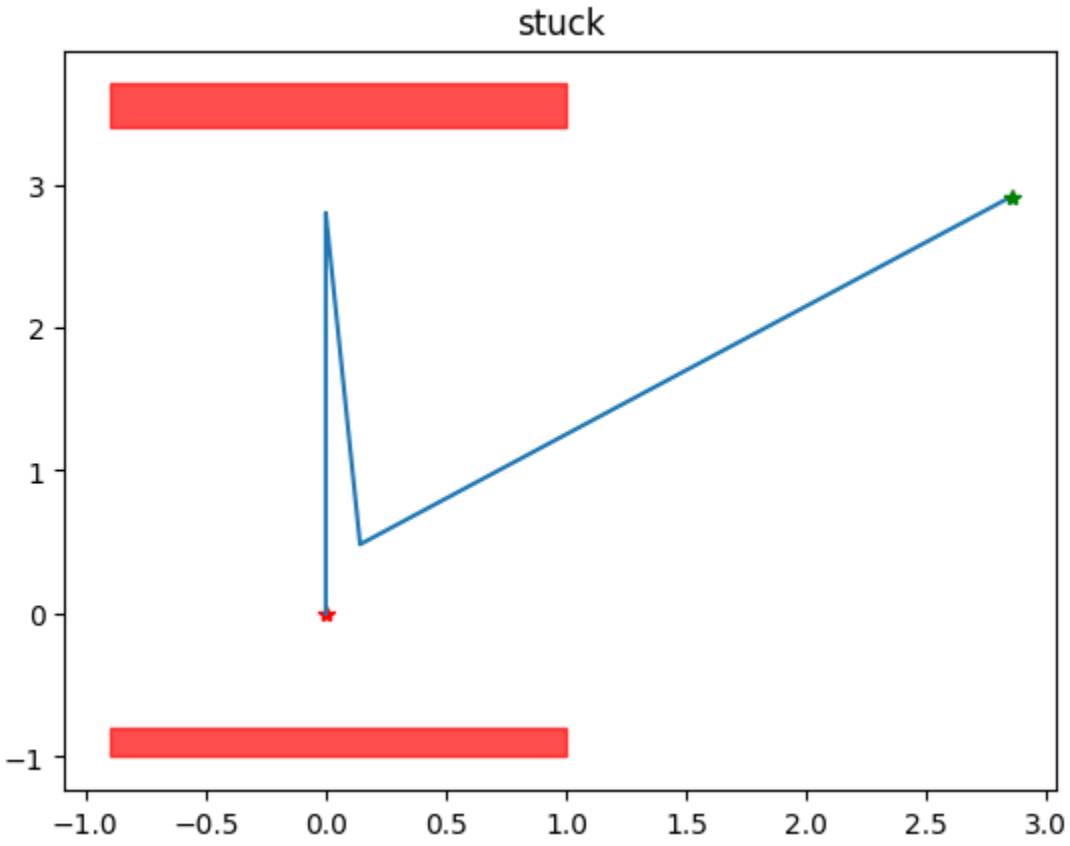
شکل ۸: حرکت قوسی به راست در یک نیمه جعبه



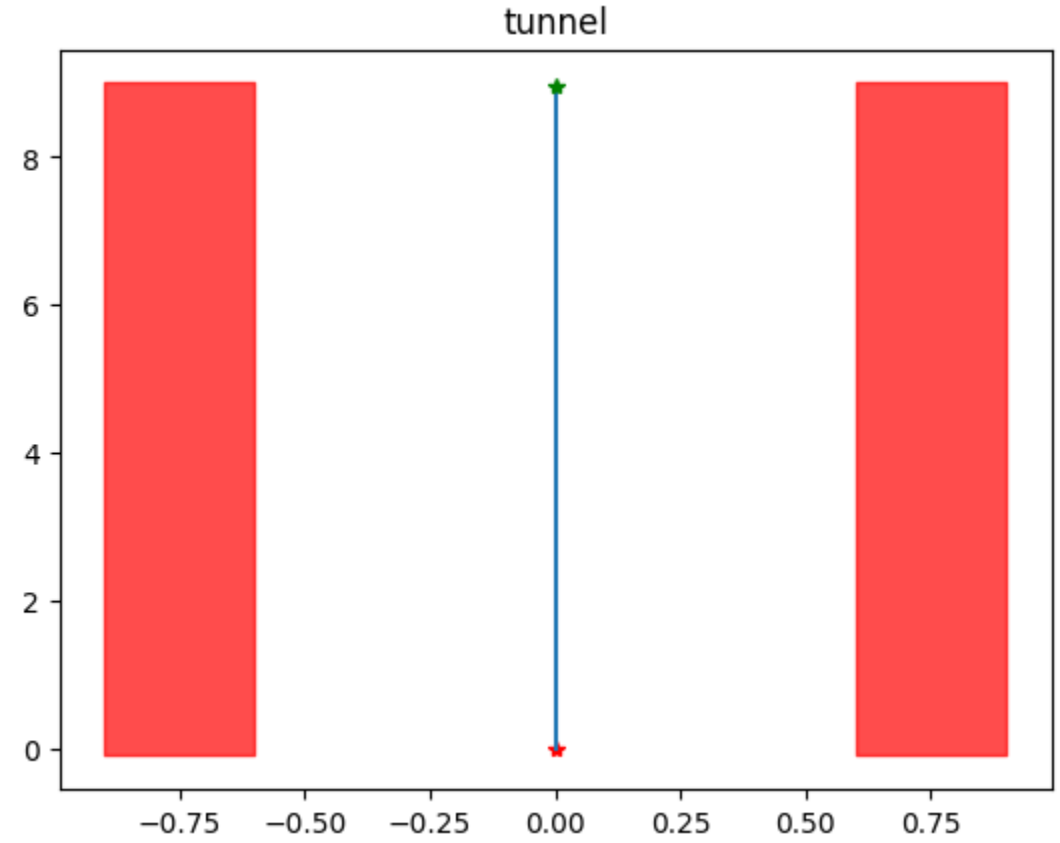
شکل ۹: حرکت در یک تونل غیرهمسان



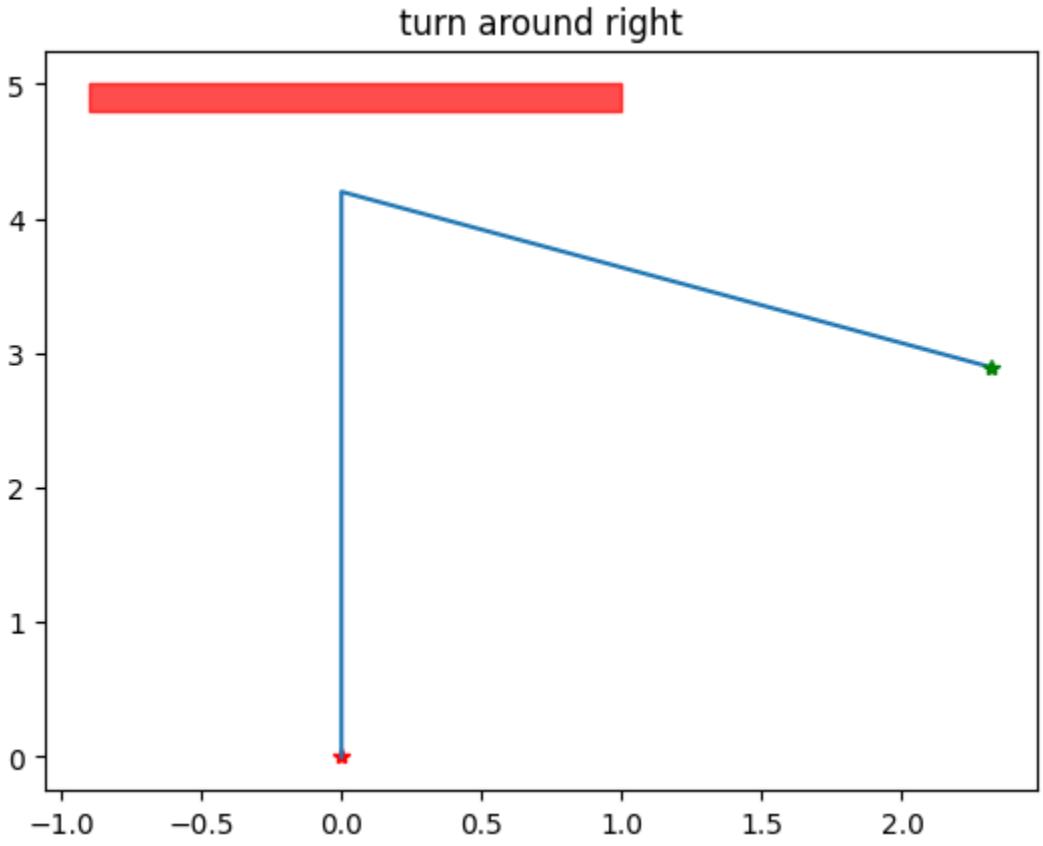
شکل ۱۰: حرکت در یک تونل غیرهمسان از جهت دیگر



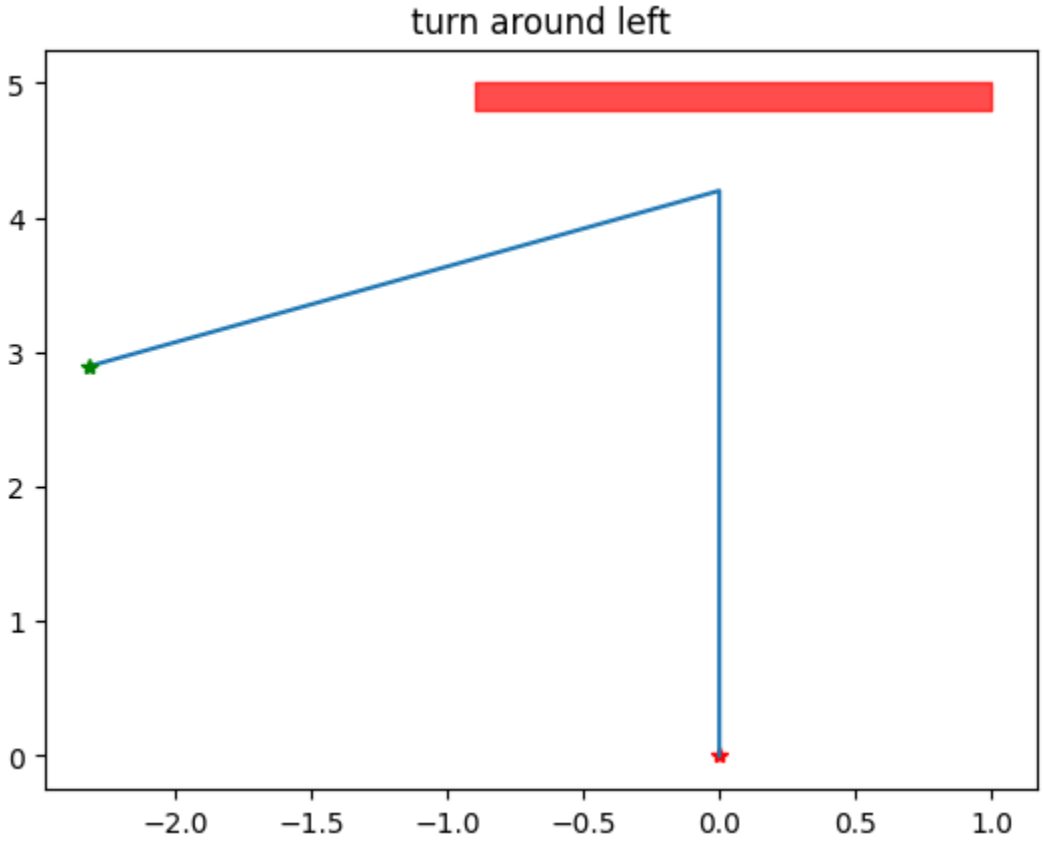
شکل ۱۱: حرکت عقب‌گرد در گیر کردن بین دو دیوار



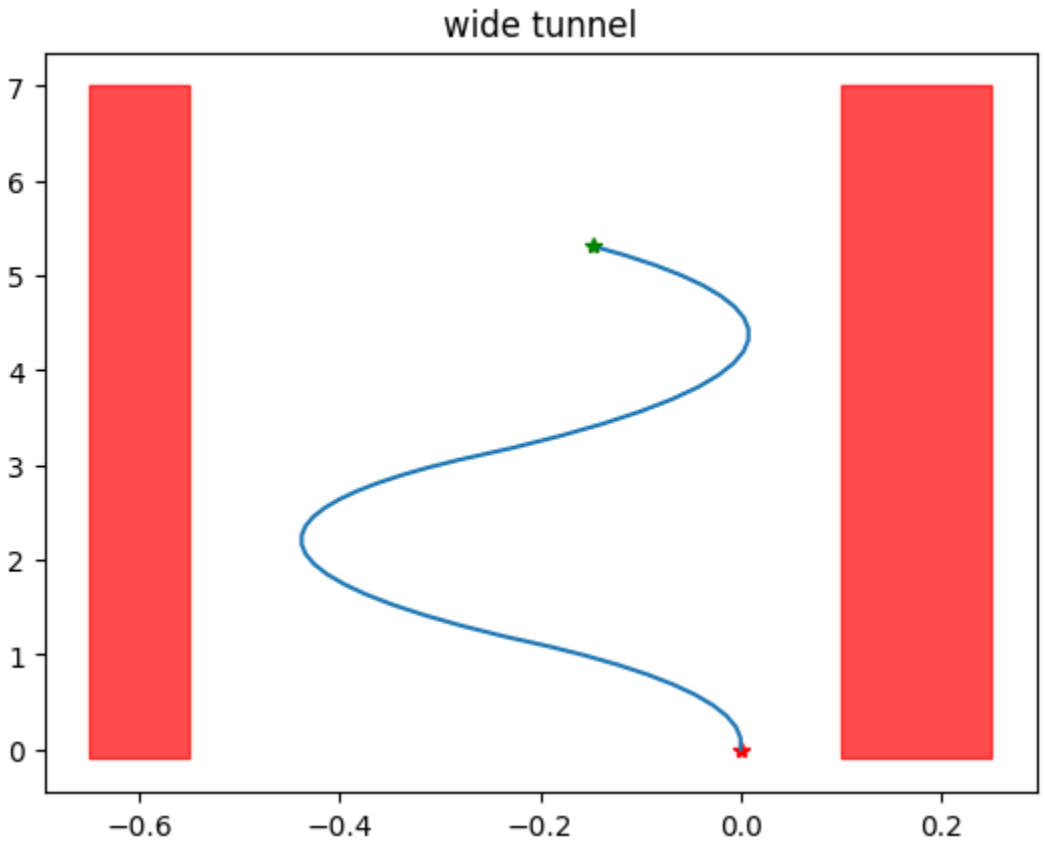
شکل ۱۲: حرکت در تونل باریک و همسان



شکل ۱۳: گردش به راست در برخورد با دیوار



شکل ۱۴: گردش به چپ در برخورد با دیوار



شکل ۱۵: حرکت در تونل پهن و همسان

۶. نرمال‌سازی

برای آموزش هرچه بهتر شبکه، داده‌های باید نرمال‌سازی شوند. از آن جهت که داده‌های موجود در این موجوعه هم مثبت و هم منفی هستند، از نرمال‌سازی استاندارد (Standard Scaling) برای این امر استفاده می‌کنیم. برای این کار از کتابخانه scikit-learn کمک می‌گیریم. در کد شماره‌ی ۷، دو تابع نرمال‌سازی استاندارد و MinMax برای نرمال‌سازی داده‌های یک Data Frame آورده شده. برای تست نرمال‌سازی MinMax نیز امتحان شد که نتایج مناسبی به همراه نداشت.

```
def feature_normalizer_std(x):
    scaler = StandardScaler()
    if isinstance(x,pd.Series):
        x = scaler.fit_transform(x.values.reshape((-1,1)))
    else:
        for i in x.columns:
            x[i] = scaler.fit_transform(x[i].values.reshape((-1,1)))
    return x

def feature_normalizer_minmax(x):
    scaler = MinMaxScaler()
    if isinstance(x,pd.Series):
        x = scaler.fit_transform(x.values.reshape((-1,1)))
    else:
```

```
        for i in x.columns:
            x[i] = scaler.fit_transform(x[i].values.reshape((-1,1)))
    return x

identification_data = feature_normalizer_std(identification_data)
identification_data.head(85)
```

کد شماره‌ی ۷: توابع نرمال‌سازی داده‌های یک Data Frame

	pwm left	pwm right	x position	y position
0	-0.688395	-0.675795	-0.019648	-0.771307
1	-0.688395	-0.675795	-0.019648	-0.771307
2	-0.688395	-0.675795	-0.019648	-0.771307
3	-0.688395	-0.675795	-0.019648	-0.771307
4	-0.688395	-0.675795	-0.019648	-0.771307
...	...	...	...	...
80	-0.688395	-0.675795	-0.019648	-0.771307
81	0.777618	1.641976	-0.019648	-0.771307
82	0.748298	1.613004	-0.019648	-0.690265
83	0.718978	1.584031	-0.026000	-0.610614
84	0.689658	1.555059	-0.038484	-0.532630

شکل ۱۶: نمونه Data Frame نرمال شده

۲. شناسایی

در این بخش با داده‌های آماده شده به شناسایی مدل ربات به کمک شبکه عصبی می‌پردازیم.

۱. قرار دادن داده‌ها در حالت سری زمانی

در این پروژه، ماهیت داده‌های بدست آمده از ربات سری زمانی می‌باشد، به این ترتیب که داده‌های x و y هر لحظه به x و y مرحله قبل و همچنین PWMهای مرحله قبل وابسته است. به همین دلیل با استفاده از توابع تعریف شده در کد شماره‌ی ۸ این داده را به فرم سری‌های زمانی در می‌آوریم. تابع split_sequences برای پنجره زنی روی داده‌ها به کار می‌رود، به این ترتیب که از دیتاست ورودی ۴ ستون اول را تا سطر n_steps برای مجموعه seq_x انتخاب کرده و داده‌ی موجود در دو ستون آخر سطر n_step را برای y_seq. به این ترتیب سری که منجر به ایجاد y_seq می‌شود را بدست آورده‌ایم. این سری در x و سری خروجی در y اضافه می‌شوند که آرایه این دو خروجی تابع می‌باشند.

در تابع time_series_MLP ورودی دیتاستی است که قصد ایجاد سری‌های زمانی از آن را داریم به همراه اسم ستون‌های مربوط به ورودی (۴ ستون) و اسم ستون‌های خروجی (۲ ستون) و همچنین تعداد زمان‌های موجود در هر نمونه از سری‌های زمانی. پس از جداسازی ستون‌های مورد نظر به کمک تابع hstack این ستون‌ها را کنار هم قرار داده و آرایه بدست آمده را به تابع split_sequence می‌دهیم. توجه داریم که سری‌های انتخاب شده دارای ۸۰ نمونه زمانی و ۴ ورودی و ۲ خروجی هستند. به همین دلیل خروجی

x_train_seq یک آرایه سه بعدی است (آرایه‌ای از آرایه‌های دو بعدی) که باید به آرایه دو بعدی تبدیل شود. بعد دوم آرایه همان تعداد نمونه‌ها و بعد سوم آن همان تعداد attributeها (۴) می‌باشد. خروجی این تابع داده‌های آموزش آماده و ابعاد شبکه است.

```
def split_sequences(sequences, n_steps):
# split a multivariate sequence into samples
    x, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, : -2], sequences[end_ix, -2:]
        x.append(seq_x)
        y.append(seq_y)
    return np.array(x), np.array(y)

def time_series_MLP(data, inseq1, inseq2, inseq3, inseq4, outseq1, outseq2,
n_steps):
    # define input sequence
    # convert to [rows, columns] structure
    in_seq1 = np.array(data[[inseq1]]).reshape(-1, 1)
    in_seq2 = np.array(data[[inseq2]]).reshape(-1, 1)
    in_seq3 = np.array(data[[inseq3]]).reshape(-1, 1)
    in_seq4 = np.array(data[[inseq4]]).reshape(-1, 1)
    out_seq1 = np.array(data[[outseq1]]).reshape(-1, 1)
    out_seq2 = np.array(data[[outseq2]]).reshape(-1, 1)
    # horizontally stack columns
    data_set = np.hstack((in_seq1, in_seq2, in_seq3, in_seq4, out_seq1,
out_seq2))
    x_train_seq, y_train_seq = split_sequences(data_set, n_steps)
    n_input = x_train_seq.shape[1] * x_train_seq.shape[2]
    n_output = y_train_seq.shape[1]
    # flatten input
    x_train_seq = x_train_seq.reshape((x_train_seq.shape[0], n_input))

    return x_train_seq, y_train_seq, n_input, n_output

x_train_seq, y_train_seq, n_input, n_output =
time_series_MLP(identification_data, "pwm left", "pwm right", "x position", "y
position", "x position", "y position", 80)
```

کد شماره‌ی ۸: توابع نرمال‌سازی داده‌های یک Data Frame

۲. مدل شبکه و آموزش

با توجه به قضیه universal approximation، یک شبکه عصبی دولایه (۱ لایه پنهان) برای تخمین هر تابعی می‌تواند به خوبی عمل کند. به کمک keras یک شبکه دولایه با تابع فعال‌ساز tanh (داده‌های مثبت و منفی داریم) برای لایه پنهان ایجاد می‌کنیم. در این شبکه از لایه‌های dropout استفاده کردیم که به طور رندوم در هر epoch تعداد از نورون‌ها را (که در آرگومان ورودی آن مشخص شده) خاموش می‌کند. این موضوع به generality شبکه کمک می‌کند. بعد ورودی و خروجی شبکه در تابع قبلی بدست آمده.

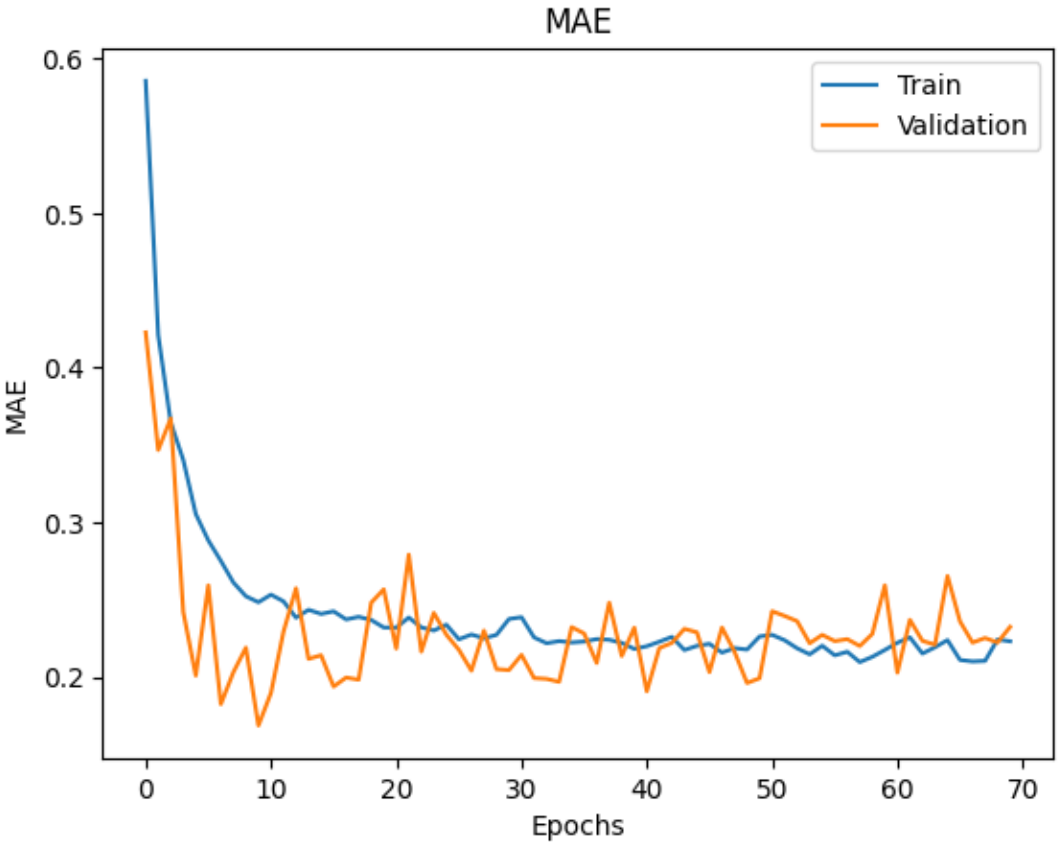
```
model_NN = tf.keras.Sequential()
model_NN.add(tf.keras.layers.Input(n_input))
model_NN.add(tf.keras.layers.Dropout(0.15))
model_NN.add(tf.keras.layers.Dense(30,activation='tanh'))
model_NN.add(tf.keras.layers.Dropout(0.1))
model_NN.add(tf.keras.layers.Dense(n_output))

model_NN.summary()
```

کد شماره‌ی ۹: تعریف مدل شبکه در keras

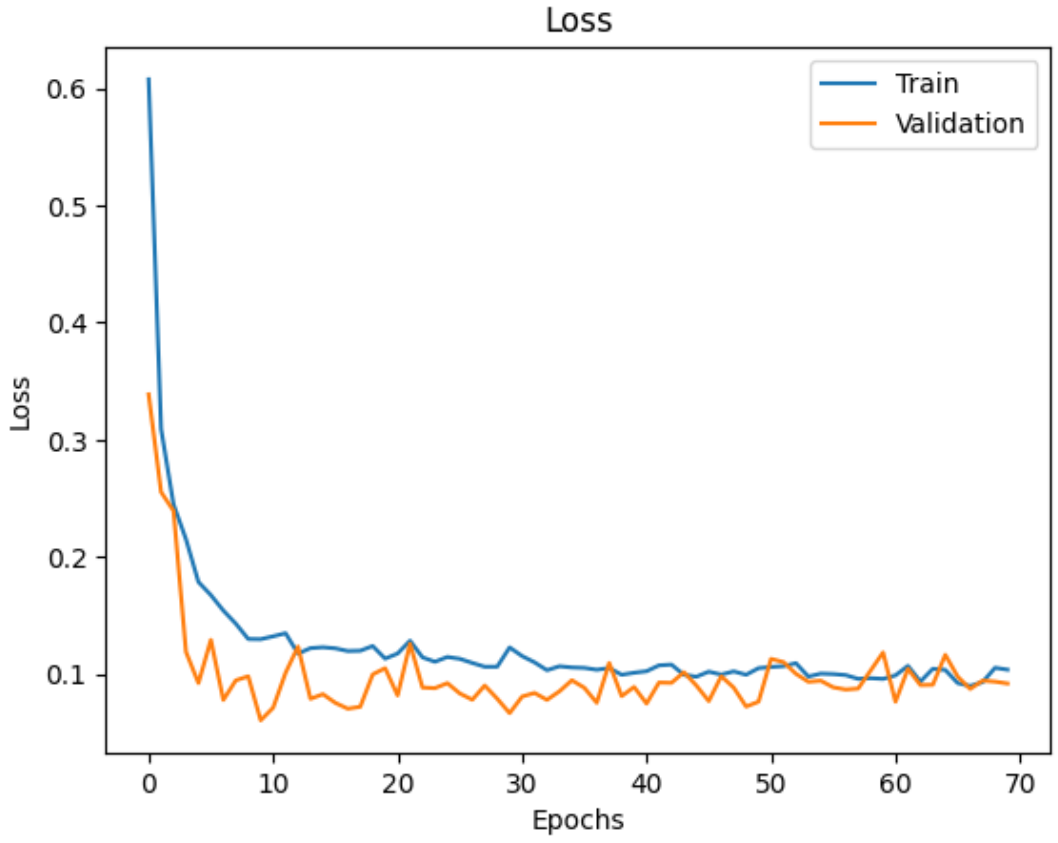
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dropout (Dropout)           (None, 480)               0

 dense (Dense)               (None, 30)                14430

 dropout_1 (Dropout)         (None, 30)                0

 dense_1 (Dense)             (None, 2)                 62

=================================================================
Total params: 14,492
Trainable params: 14,492
Non-trainable params: 0
```

شکل ۱۷: خلاصه مشخصات شبکه تشکیل داده شده

برای آموزش شبکه از Adam optimizer با ضریب یادگیری ۰/۰۰۱ استفاده می‌کنیم. توجه داریم که به علت کم بودن داده‌های سایز batchها را کوچک در نظر می‌گیریم (mini batch). چون عمل مورد نظر برای این شبکه regression است معیار آن را نیز mae قرار می‌دهیم. در ادامه نتایج بدست آمده از آموزش این شبکه آورده شده.



شکل ۱۸: کاهش mae با پیشرفت epochها



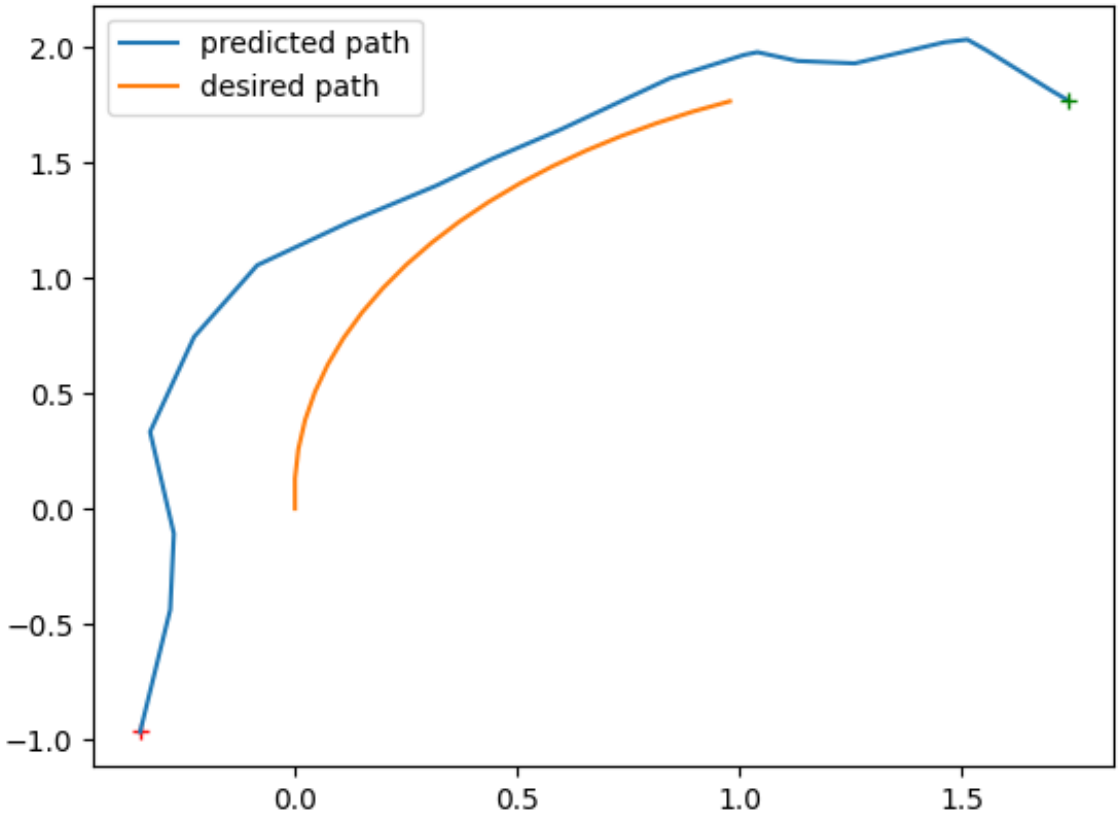
شکل ۱۹: کاهش loss با پیشرفت epochها

۳. تست شبکه

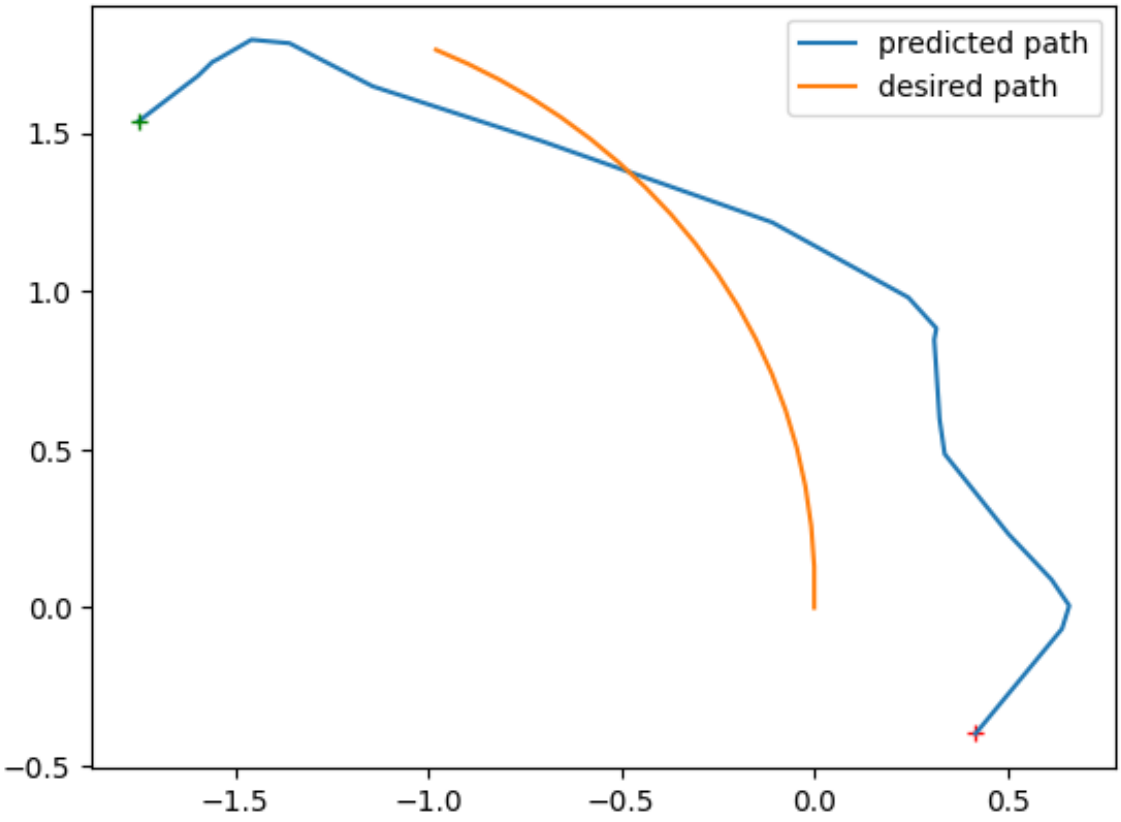
برای تست شبکه با ایجاد یک سری PWM و آماده‌سازی داده‌ها به روش مشابه، از متد predict استفاده کرده و داده‌های بدست آمده و مطلوب را را با تابع position رسم می‌کنیم. همچنین به کمک معیار mean_squared_error موجود در scikit-learn خطای خروجی بدست آمده با خروجی مطلوب را می‌سنجیم. همانطور که از نتایج فاز آموزش انتظار می‌رفت، نمودار خروجی با تقریب خوبی نمودار مطلوب را دنبال کرده و خطای بدست آمده با معیار تعریف شده تقریبا ۰/۹۵ است.

```
pwm_right1 = np.arange(80,60,-1)
pwm_left1 = np.arange(50,30,-1)

pwm_right = np.pad(pwm_right1, (max_length-len(pwm_right1),0), 'constant',
constant_values=(0,))
pwm_left = np.pad(pwm_left1, (max_length-len(pwm_left1),0), 'constant',
constant_values=(0,))
x,y= position(pwm_right, pwm_left)

test_data = pd.DataFrame({'pwm_left': np.array(pwm_left).reshape(-1,),
                          'pwm_right': np.array(pwm_right).reshape(-1,),
                          'x position': np.array(x).reshape(-1,),
                          'y position': np.array(y).reshape(-1,)})

test_data = feature_normalizer_std(test_data)

x_test, y_test, n_input, n_output = time_series_MLP(test_data, "pwm_left",
"pwm_right", "x position", "y position", "x position", "y position", 80)

prediction = model_NN.predict(x_test)
print(f"mean squared error of the test
{np.sqrt(mean_squared_error(y_test,prediction))}")
plt.plot(prediction[0,0],prediction[0,1],'r+')
plt.plot(prediction[:,0],prediction[:,1],label="predicted path")
plt.plot(prediction[-1:,0],prediction[-1:,1],'g+')
plt.plot(x,y,label="desired path")
plt.legend()
plt.show()
```

کد شماره‌ی ۱۰: تعریف مسیر تست و آماده‌سازی داده‌های آن. استفاده از متد predict برای بدست آوردن خروجی شبکه



شکل ۲۰: تخمین قوس راست توسط شبکه



شکل ۲۱: تخمین قوس چپ توسط شبکه

۴. ذخیره سازی مدل و وزن‌ها

در آخر برای استفاده مدل در موارد بعدی به کمک متد save مدل را ذخیره می‌کنیم. همچنین وزن‌های لایه‌ها را در یک فایل txt. ذخیره می‌کنیم. حجم مدل ذخیره شده، به دلیل سادگی بسیار ناچیز است.

```
wights_NN = list()
for layer in model_NN.layers:
    wights_NN.append(layer.get_weights())
f = open("weights_NN.txt","w")
f.write(str(wights_NN))
f.close()

model_NN.save(filepath='ident_model.h5', include_optimizer=True)
```

کد شماره‌ی ۱۱: ذخیره‌سازی مدل و وزن‌ها لایه‌ها

۳. کنترلر

در این بخش با داده‌های آماده شده در بخش اول به طراحی کنترلر برای obstacale avoidance ربات می‌پردازیم.

۱. قرار دادن داده‌ها در حالت سری زمانی

همانطور که پیشتر گفته شد، داده‌های به دست آمده از تست عملی ربات نیاز به پیش پردازش دارند تا در حالت سری زمانی قرار بگیرند. توجه داریم که داده‌های قرار داده شده در لیست‌ها همگی مربوط به یک بازه نمونه‌برداری هستند و به همین دلیل باید داده‌های سنسورها را در دیتاست یک ردیف به بالا شیفت بدهیم. برای این کار با اضافه کردن یک سطر صفر و شیفت دادن دو ستون سنسورها به بالا و همچنین حذف سطر NaN بوجود آمده در انتها داده‌ها را برای پنجره زنی و قرار دادن در سری زمانی آماده می‌کنیم.

```
LSTM_set = pd.DataFrame({
    "pwm left": [0.0],
    "pwm right": [0.0],
    "srf left": [0.0],
    "srf right": [0.0]},)

LSTM_set = LSTM_set.append(controller_data)
LSTM_set[['srf left','srf right']] = LSTM_set[['srf left','srf right']].shift(-
1)
LSTM_set = LSTM_set.dropna(axis=0)
# LSTM_set = feature_normalizer_std(LSTM_set)
LSTM_set.head(85)
```

کد شماره‌ی ۱۲: جلو بردن داده‌های سنسور به اندازه یک بازه نمونه‌برداری

	pwm left	pwm right	srf left	srf right
0	0.0	0.0	0.0	0.0
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
...	...	...	...	...
79	0.0	0.0	0.0	0.0
80	0.0	0.0	244.0	52.0
81	50.0	80.0	289.0	50.0
82	49.0	79.0	274.0	50.0
83	48.0	78.0	274.0	49.0

شکل ۲۲: داده‌های شیفت داده شده

برای ایجاد سری‌ها به این صورت عمل می‌کنیم که هر time step یک بردار ۴ المانی است که هر کدام از این المان‌ها مربوط به یکی از ورودی‌های شبکه (داده‌های سنسور و PWMها) می‌باشد. این بردارها را به لیست‌های دو بعدی تبدیل می‌کنیم که پس از تشکیل رشته‌ای از نمونه‌ها بتوان آن‌ها را به عنوان هر time step جدا کرد. در ادامه با جلو رفتن در هر سطر، داده‌های سطرهای قبلی باید به انتها اضافه شوند تا فرم کلی سری زمانی شکل بگیرد. برای این کار از دستور cumsum() در کتابخانه pandas استفاده می‌کنیم و داده‌های بدست آمده را در یک ستون جدید در دیتاست قرار می‌دهیم. همچنین چون خروجی ما دو بعدی است (دو PWM)، آن ها را در لیست قرار می‌دهیم.

```
# Put inputs into a single list
LSTM_set['single_input_vector'] = LSTM_set[['srf left','srf right','pwm
left','pwm right']].apply(tuple, axis=1).apply(list)
# Double-encapsulate list so we can sum it in the next step and keep time steps
as separate elements
LSTM_set['single_input_vector'] = LSTM_set.single_input_vector.apply(lambda x:
[list(x)])
# Using .cumsum() to include previous row vectors in the current row list of
vectors (creating time series)
LSTM_set['cumulative_input_vectors'] = LSTM_set.single_input_vector.cumsum()

LSTM_set['output_vector'] = LSTM_set[['pwm left','pwm right']].apply(tuple,
axis=1).apply(list)
```

کد شماره‌ی ۱۳: تنظیم سری‌های زمانی با بردارهای ۴ مولفه‌ای

لیست‌های بدست آمده در ستون cumulative_input_vectors هم طول نیستند و برای ورودی شبکه باید هم طول شوند. به همین دلیل ماکزیمم طول آن‌ها را پیدا می‌کنیم و به کمک تابع pad_sequences کتابخانه pandas آن‌ها را تا همان طول ماکزیمم پد می‌کنیم و در یک ستون جدید ذخیره می‌کنیم.

```
max_sequence_length = LSTM_set.cumulative_input_vectors.apply(len).max()
padded_sequences =
tf.keras.preprocessing.sequence.pad_sequences(LSTM_set.cumulative_input_vectors
.tolist(), max_sequence_length).tolist()
```

```
LSTM_set['padded_input_vectors'] =
pd.Series(padded_sequences).apply(np.asarray)

LSTM_set.head(85)
```

کد شماره‌ی ۱۴: هم طول کردن سری‌های بدست آمده در مرحله قبل

	pwm left	pwm right	srf left	srf right	single_input_vector	cumulative_input_vectors	output_vector	padded_input_vectors
0	0.0	0.0	0.0	0.0	[[0.0, 0.0, 0.0, 0.0]]	[[0.0, 0.0, 0.0, 0.0]]	[0.0, 0.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
1	0.0	0.0	0.0	0.0	[[0.0, 0.0, 0.0, 0.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0]]	[0.0, 0.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
2	0.0	0.0	0.0	0.0	[[0.0, 0.0, 0.0, 0.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [...	[0.0, 0.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
3	0.0	0.0	0.0	0.0	[[0.0, 0.0, 0.0, 0.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [...	[0.0, 0.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
4	0.0	0.0	0.0	0.0	[[0.0, 0.0, 0.0, 0.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [...	[0.0, 0.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
...	...	...	...	...	...	...	...	...
80	0.0	0.0	0.0	0.0	[[0.0, 0.0, 0.0, 0.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [...	[0.0, 0.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
81	0.0	0.0	244.0	52.0	[[244.0, 52.0, 0.0, 0.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [...	[0.0, 0.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
82	50.0	80.0	289.0	50.0	[[289.0, 50.0, 50.0, 80.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [...	[50.0, 80.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
83	49.0	79.0	274.0	50.0	[[274.0, 50.0, 49.0, 79.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [...	[49.0, 79.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...
84	48.0	78.0	274.0	49.0	[[274.0, 49.0, 48.0, 78.0]]	[[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0], [...	[48.0, 78.0]	[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0,...

شکل ۲۳: دیتاست جدید

بعد از ایجاد دیتاست جدید، ستون پد شده را به عنوان x_train_seq قرار می‌دهیم. این ستون باید برای ورودی به LSTM به فرم ۳ بعدی باشد به این ترتیب که ابعاد ورودی برابر تعداد attributeها، طول ورودی برابر time step های موجود در هر نمونه و تعداد کلی نمونه‌ها برابر با تعداد سطرهای دیتاست است. همچنین بعد خروجی بسیار واضح و برابر با ۲ است.

```
x_train_init = np.asarray(LSTM_set.padded_input_vectors)

# Use hstack to and reshape to make the inputs a 3d vector
x_train = np.hstack(x_train_init).reshape(len(LSTM_set),max_sequence_length,4)
y_train =
np.hstack(np.asarray(LSTM_set.output_vector)).reshape(len(LSTM_set),2)
print(x_train_init.shape)
print(x_train.shape)

# Input length is the length for one input sequence (i.e. the number of rows
for your sample)
# Input dim is the number of dimensions in one input vector (i.e. number of
input columns)
# Output dimensions is the shape of a single output vector
input_length = x_train.shape[1]
input_dim = x_train.shape[2]
output_dim = len(y_train[0])
```

کد شماره‌ی ۱۵: آماده‌سازی ابعاد داده‌های ورودی و خروجی شبکه

بعد از ایجاد دیتاست جدید، ستون پد شده را به عنوان x_train_seq قرار می‌دهیم. این ستون باید برای ورودی به LSTM به فرم ۳ بعدی باشد به این ترتیب که ابعاد ورودی برابر تعداد attributeها، طول ورودی برابر time step های موجود در هر نمونه و تعداد کلی نمونه‌ها برابر با تعداد سطرهای دیتاست است. همچنین بعد خروجی بسیار واضح و برابر با ۲ است.

توجه داریم که در این دیتاست، نرمال‌سازی نباید انجام شود. زیرا داده‌ای صفر اضافه شده به عنوان padding در نرمال‌سازی شده به معنای صفر نیستند و باعث خطای شبکه در تخمین می‌شوند (هرچند معیارهای آموزش و mse بر مطلوب بودن شبکه دلالت کنند).

۲. مدل شبکه و آموزش

با توجه به این موضوع که این کنترلر باید توسط بک برد raspberry pi 3 اجرا شود که توان محاسباتی آن به اندازه یک کامپیوتر مستقل نیست، ترجیح بر این است که ساختار شبکه تا حد ممکن ساده و حجم آن کم باشد تا سر هر زمان نمونه برداری داده‌های جدید برای کنترل ربات آماده باشد. هم همین دلیل فقط از یک لایه LSTM با ۱۰۰ یونیت و تابع فعال‌ساز tanh (به دلیل مشابه قسمت قبل) استفاده می‌کنیم. خوبی عمل کند. به کمک keras یک شبکه دولایه با تابع فعال‌ساز tanh (داده‌های مثبت و منفی داریم) برای لایه پنهان ایجاد می‌کنیم. در این شبکه از لایه‌های dropout استفاده کردیم که به طور رندوم در هر epoch تعداد از نورون‌ها را (که در آرگومان ورودی آن مشخص شده) خاموش می‌کند. این موضوع به generality شبکه کمک می‌کند. بعد ورودی و خروجی شبکه در تابع قبلی بدست آمده.

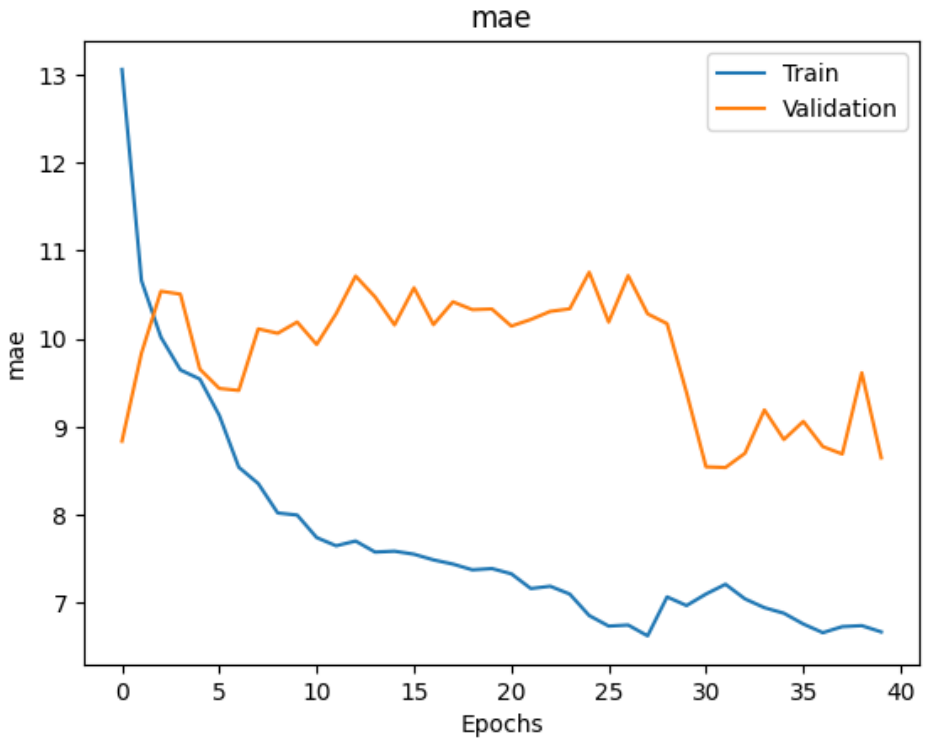
```
model_control = tf.keras.Sequential()
model_control.add(tf.keras.layers.LSTM(100, input_dim = input_dim, input_length
= input_length))
model_control.add(tf.keras.layers.Dense(output_dim))

model_control.summary()
```

کد شماره‌ی ۱۶: تعریف مدل شبکه کنترلر در keras

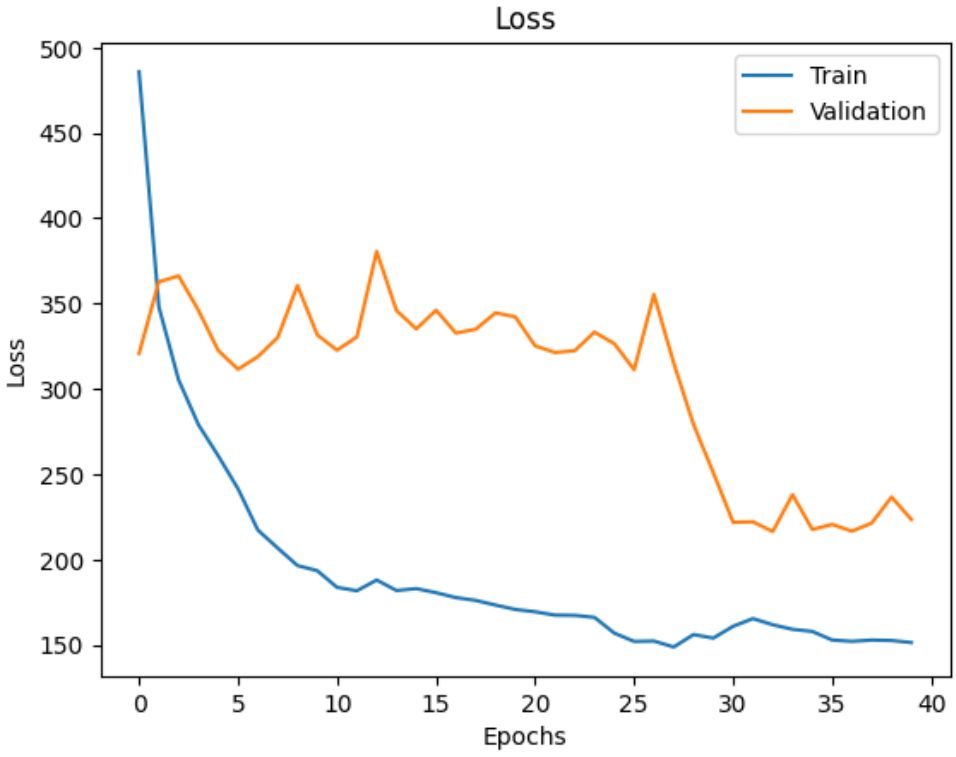
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 100)               42000

 dense_6 (Dense)             (None, 2)                 202

=================================================================
Total params: 42,202
Trainable params: 42,202
Non-trainable params: 0
```

شکل ۲۴: خلاصه مشخصات شبکه کنترلر تشکیل داده شده

برای آموزش شبکه از روش Stochstic Gradient Descend استفاده شده. در ادامه نتایج بدست آمده از آموزش این شبکه آورده شده.



شکل ۲۵: کاهش mae با پیشرفت epochها و رد کردن مینیمم محلی



شکل ۲۶: کاهش loss با پیشرفت epochها و رد کردن مینیمم محلی

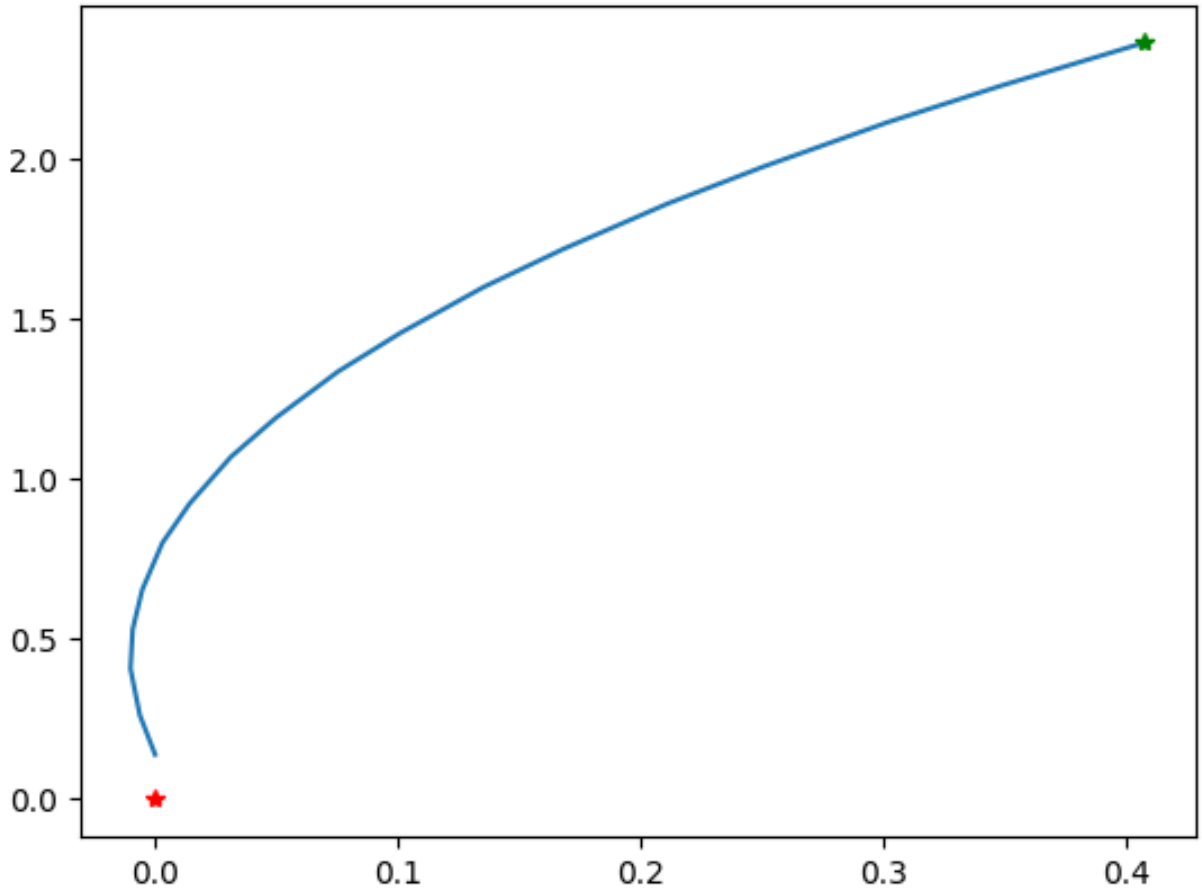
۳. تست شبکه

برای تست شبکه با ایجاد یک سری PWM و آماده‌سازی داده‌ها به روش مشابه، از متد predict استفاده کرده و داده‌های بدست آمده و مطلوب را را با تابع position رسم می‌کنیم. همچنین به کمک معیار

mean_squared_error موجود در scikit-learn خطای خروجی بدست آمده با خروجی مطلوب را می‌سنجیم. همانطور که از نتایج فاز آموزش انتظار می‌رفت، نمودار خروجی با تقریب خوبی نمودار مطلوب را دنبال کرده و خطای بدست آمده با معیار تعریف شده تقریبا ۰/۹۵ است.

```
srf_left = np.arange(40,80,2)
srf_right = np.random.randint(200,250,len(srf_left))
pwm_right = np.array([0])
pwm_left = np.array([0])
predict = list()
test_set = pd.DataFrame({"pwm left": pwm_left.reshape(-1,),
                         "pwm right": pwm_right.reshape(-1,),
                         "srf left": srf_left[0].reshape(-1,),
                         "srf right": srf_right[0].reshape(-1,)})

for i in range(len(srf_left)-1):
    test_set = pd.DataFrame({"pwm left": pwm_left.reshape(-1,),
                             "pwm right": pwm_right.reshape(-1,),
                             "srf left": srf_left[i].reshape(-1,),
                             "srf right": srf_right[i].reshape(-1,)})
    test_set['single_input_vector'] = test_set[['srf left','srf right','pwm
left','pwm right']].apply(tuple, axis=1).apply(list)
    test_set['single_input_vector'] = test_set.single_input_vector.apply(lambda
x: [list(x)])
    test_set['cumulative_input_vectors'] =
test_set.single_input_vector.cumsum()
    test_set['output_vector'] = test_set[['pwm left','pwm right']].apply(tuple,
axis=1).apply(list)
    padded_sequences =
tf.keras.preprocessing.sequence.pad_sequences(test_set.cumulative_input_vectors
.tolist(), max_sequence_length).tolist()
    test_set['padded_input_vectors'] =
pd.Series(padded_sequences).apply(np.asarray)

    x_test_init = np.asarray(test_set.padded_input_vectors)
    x_test =
np.hstack(x_test_init).reshape(len(test_set),max_sequence_length,4)
    y_test =
np.hstack(np.asarray(test_set.output_vector)).reshape(len(test_set),2)

    prediction = model_control.predict(x_test)
    pwm_right = np.array(prediction[0,1])
    pwm_left = np.array(prediction[0,0])
    predict.append(prediction.tolist())

predict = np.array(sum(predict, []))
print(predict)
x,y = position(predict[:,1], predict[:,0])
```

کد شماره‌ی ۱۷: تعریف مسیر تست و آماده‌سازی داده‌های آن. استفاده از متد predict برای بدست آوردن خروجی شبکه



شکل ۲۷: تخمین قوس راست توسط شبکه

۴. ذخیره سازی مدل و وزن‌ها

در آخر برای استفاده مدل در موارد بعدی به کمک متد save مدل را ذخیره می‌کنیم. همچنین وزن‌های لایه‌ها را در یک فایل txt. ذخیره می‌کنیم. حجم مدل ذخیره شده، به دلیل سادگی بسیار ناچیز است.

```
wights_contoller = list()
for layer in model_control.layers:
    wights_contoller.append(layer.get_weights())
f = open("weights_controller.txt","w")
f.write(str(wights_contoller))
f.close()

model_control.save(filepath='controller_model.h5', include_optimizer=True)
```

کد شماره‌ی ۱۸: ذخیره‌سازی مدل و وزن‌ها لایه‌ها