به نام خدا

اعضای گروه: امیرحسین نوری - کیمیا ایمنی - امیرمسعود شاکر

تمرین دوم

.1

بخش اول. معادلات سینماتیک مستقیم ربات های Omni-directional:

از رابطه 3.24 کتاب مرجع داریم:

$$J_1(\beta_s)R(\theta)\dot{\xi}_I - J_2\dot{\varphi} = 0 .$$

با استفاده از رابطه بالا و ساده سازی $J_1(\beta_s)$ به J_{1f} به معادله 3.31 کتاب مرجع میرسیم:

$$\dot{\xi}_I = R(\theta)^{-1}J_{1f}^{-1}J_2\dot{\varphi} .$$

حال باید مقادیر ماتریس های $R(\theta)^{-1}$، J_{1f}^{-1} و J_2 را بیابیم.

$$R(\theta)^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} .$$

چون چارچوب مرجع محلی ربات و چارچوب مرجع جهانی در یک راستا قرار دارند، مقدار θ برابر 0 است.

بنابراین خواهیم داشت:

$$R(\theta)^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

برای محاسبه ماتریس J_{1f}^{-1} از قیود غلتش چرخ سوئدی که در معادله 3.19 کتاب مرجع آمده است، کمک میگیریم.

$$\left[\sin(\alpha+\beta+\gamma)\ -\cos(\alpha+\beta+\gamma)\ (-l)\cos(\beta+\gamma)\right]R(\theta)\dot{\xi}_I - r\dot{\varphi}\cos\gamma = 0\,.$$

برای این کار، باید مقادیر α ,β, γ را تعیین کنیم.

با توجه به شکل 3.8 کتاب مرجع، مقدار γ برای چرخ سوئدی 90 درجه برابر 0 است.

با جایگزینی این مقدار، به معادله 3.12 کتاب مرجع میرسیم.

$$\left[\sin(\alpha+\beta)\ -\cos(\alpha+\beta)\ (-l)\cos\beta\right]R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0\,.$$

با توجه به مکان خاص ربات در چارچوب مرجع محلی، مقدار α برای هر چرخ به صورت زیر است:

$\alpha_1 = \pi/3,\ \alpha_2 = \pi,\ \alpha 2 = \pi\ ,\ \ \alpha_3 = -\pi/3$

چون چرخ ها بر بدنه دایره ای ربات مماس هستند، مقدار β برای همه چرخ ها برابر 0 است.

حال میتوانیم ماتریس J_{1f} را به صورت زیر بنویسیم:

$$J_{1f} = \begin{bmatrix} \sin\frac{\pi}{3} & -\cos\frac{\pi}{3} & -l \\ 0 & -\cos\pi & -l \\ \sin-\frac{\pi}{3} & -\cos-\frac{\pi}{3} & -l \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & -l \\ 0 & 1 & -l \\ -\frac{\sqrt{3}}{2} & -\frac{1}{2} & -l \end{bmatrix}.$$

ماتریس معکوس J_{1f} با فرض l = 1 به صورت زیر است:

$$\begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & -\frac{1}{\sqrt{3}} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} \end{bmatrix}$$

طبق معادله 3.34 کتاب مرجع، ماتریس J_2 به صورت زیر است:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

در نتیجه جواب نهایی ما به صورت زیر خواهد بود:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = R(\theta)^{-1} J_{1f}^{-1} J_2 \dot{\varphi}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & -\frac{1}{\sqrt{3}} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \dot{\varphi}_3 \end{bmatrix}$$

بخش دوم. شبیه سازی:

a) در این بخش، سرعت خطی برابر 1- و سرعت زاویه ای برابر 0 است.

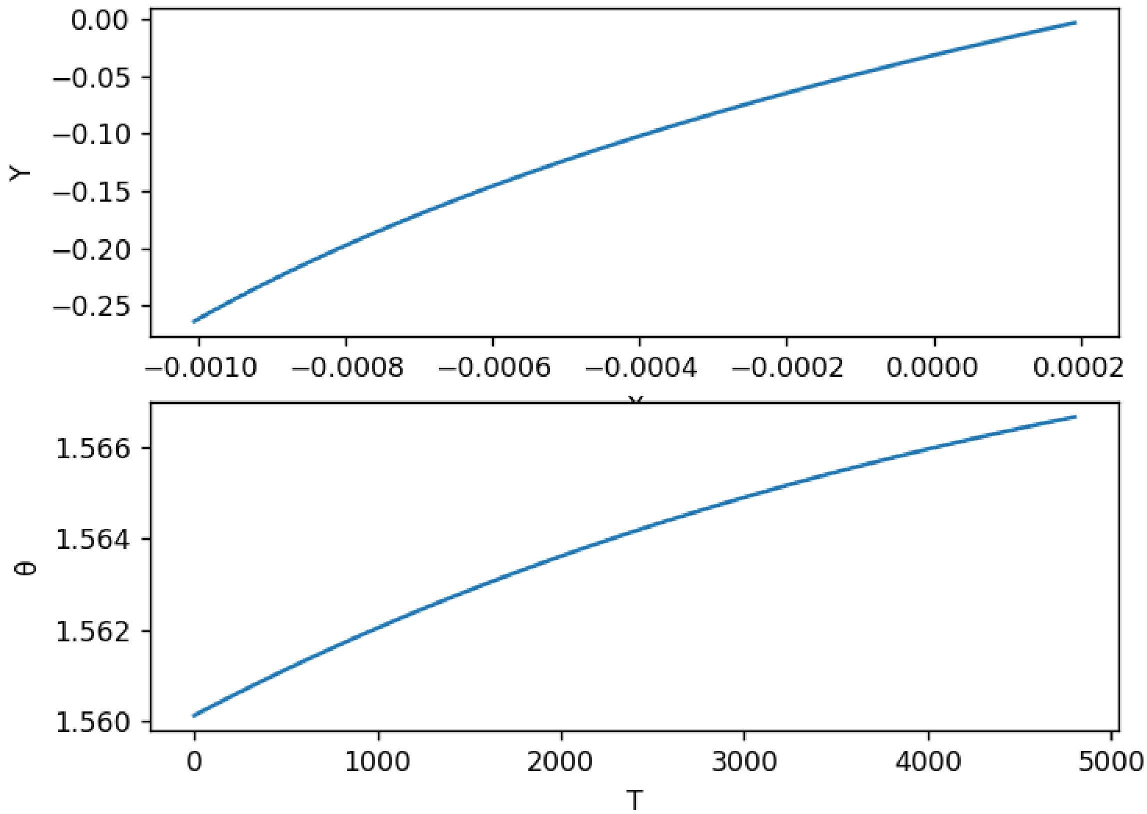
بنابراین باید سرعت چرخ ها را به گونه ای تعیین کنیم که ربات به سمت عقب حرکت کند.

سرعت چرخ 1 برابر 1- و سرعت چرخ 2 برابر 1 تعیین شده است.

کد این قسمت و نمودار رسم شده:

```python
from controller import Robot, Motor
import math
import numpy as np
import matplotlib.pyplot as plt

TIME_STEP = 64

MAX_SPEED = 6.28

# create the Robot instance.
robot = Robot()

# get a handler to the motors and set target position to infinity
(speed control)
wheel0 = robot.getDevice('wheel0_joint')
wheel1 = robot.getDevice('wheel1_joint')
wheel2 = robot.getDevice('wheel2_joint')

wheel0.setPosition(float('inf'))
wheel1.setPosition(float('inf'))
wheel2.setPosition(float('inf'))

# set up the motor speeds
wheel0.setVelocity(0)
wheel1.setVelocity(-1)
wheel2.setVelocity(1)

# GPS
gps = robot.getDevice('gps')
gps.enable(TIME_STEP)

# Compass
compass = robot.getDevice('compass')
compass.enable(TIME_STEP)

c = 1
t = 0

# create x, y, theta, time lists
```

```python
x = []
y = []
theta = []
time = []

while robot.step(TIME_STEP) != -1:
    gps_value = gps.getValues()
    print(gps_value)
    x.append(gps_value[0])
    y.append(gps_value[1])

    compass_value = compass.getValues()
    time.append(t)
    theta.append(math.atan2(compass_value[1], compass_value[0]))

    if (c > 75):
        break
    c += 1
    t += TIME_STEP

# Plot
fig, ax = plt.subplots(2)

ax[0].set(xlabel='X', ylabel='Y')
ax[0].plot(x, y)

ax[1].plot(time, theta)
ax[1].set(xlabel='T', ylabel='θ')
plt.show()
```



b) در این بخش، سرعت خطی برابر 1 و سرعت زاویه ای برابر 0 است.

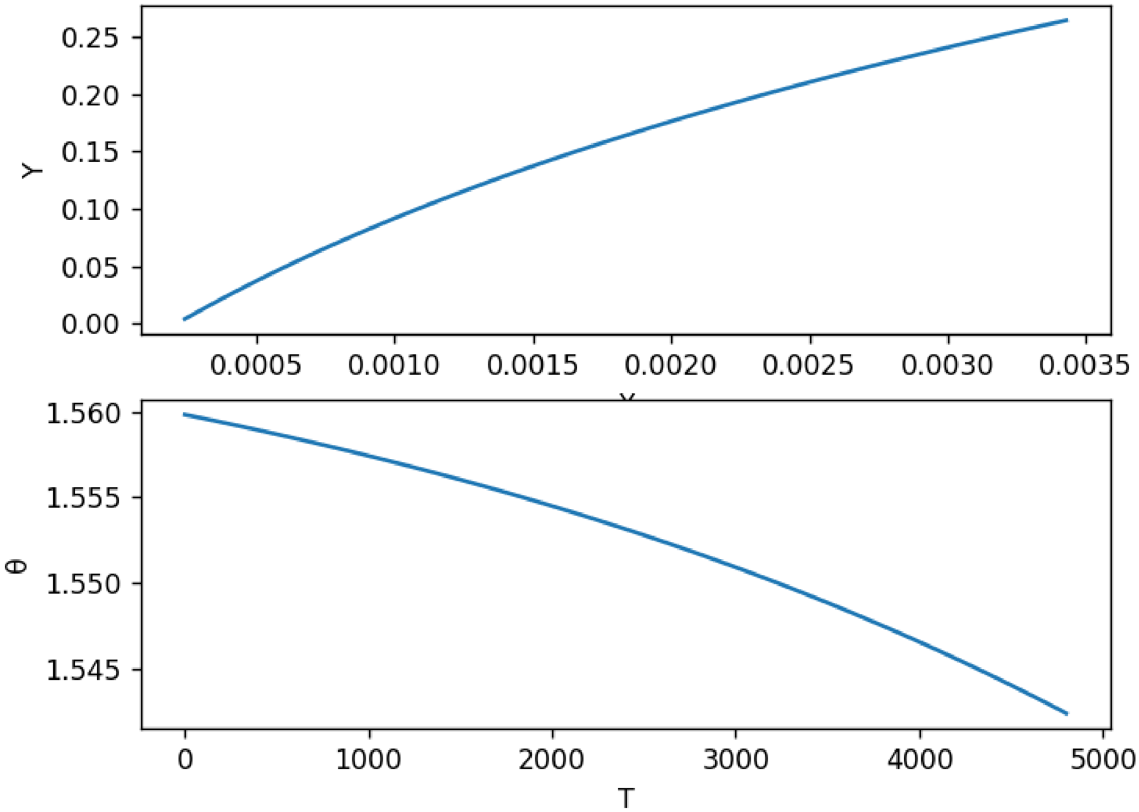
بنابراین باید سرعت چرخ ها را به گونه ای تعیین کنیم که ربات به سمت جلو حرکت کند.

سرعت چرخ 1 برابر 1 و سرعت چرخ 2 برابر 1- تعیین شده است.

کد این قسمت و نمودار رسم شده:

```python
from controller import Robot, Motor
import math
import numpy as np
import matplotlib.pyplot as plt

TIME_STEP = 64

MAX_SPEED = 6.28

# create the Robot instance.
robot = Robot()

# get a handler to the motors and set target position to infinity
(speed control)
wheel0 = robot.getDevice('wheel0_joint')
wheel1 = robot.getDevice('wheel1_joint')
wheel2 = robot.getDevice('wheel2_joint')

wheel0.setPosition(float('inf'))
wheel1.setPosition(float('inf'))
wheel2.setPosition(float('inf'))

# set up the motor speeds
wheel0.setVelocity(0)
wheel1.setVelocity(1)
wheel2.setVelocity(-1)

# GPS
gps = robot.getDevice('gps')
gps.enable(TIME_STEP)

# Compass
compass = robot.getDevice('compass')
compass.enable(TIME_STEP)

c = 1
t = 0
```

```python
# create x, y, theta, time lists
x = []
y = []
theta = []
time = []

while robot.step(TIME_STEP) != -1:
    gps_value = gps.getValues()
    print(gps_value)
    x.append(gps_value[0])
    y.append(gps_value[1])

    compass_value = compass.getValues()
    time.append(t)
    theta.append(math.atan2(compass_value[1], compass_value[0]))

    if (c > 75):
        break
    c += 1
    t += TIME_STEP

# Plot
fig, ax = plt.subplots(2)

ax[0].set(xlabel='X', ylabel='Y')
ax[0].plot(x, y)

ax[1].plot(time, theta)
ax[1].set(xlabel='T', ylabel='θ')
plt.show()
```



c) در این بخش، سرعت خطی برابر 0 و سرعت زاویه ای برابر 1 است.

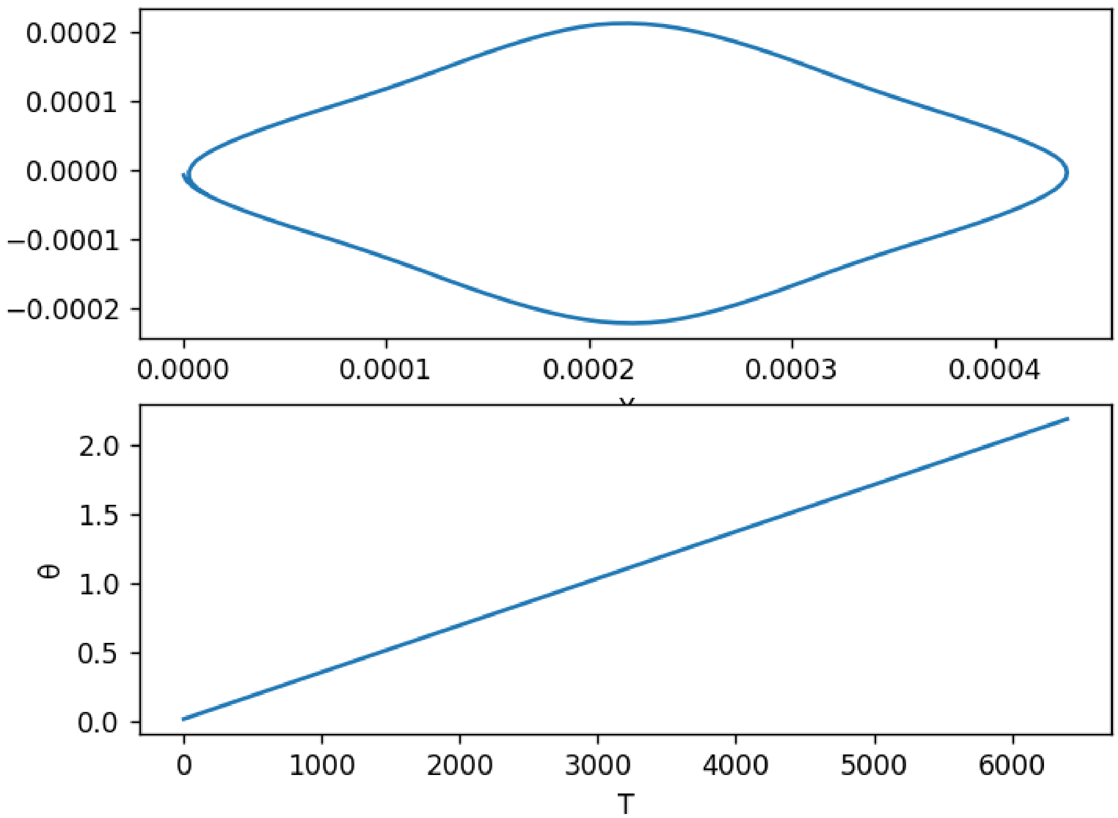
بنابراین باید سرعت چرخ ها را به گونه ای تعیین کنیم که ربات حرکت خطی نداشته و به دور خود بچرخد.

سرعت همه چرخ ها برابر 1 قرار داده شده است.

کد این قسمت و نمودار رسم شده:

```python
from controller import Robot, Motor
import math
import numpy as np
import matplotlib.pyplot as plt

TIME_STEP = 64

MAX_SPEED = 6.28

# create the Robot instance.
robot = Robot()

# get a handler to the motors and set target position to infinity
(speed control)
wheel0 = robot.getDevice('wheel0_joint')
wheel1 = robot.getDevice('wheel1_joint')
wheel2 = robot.getDevice('wheel2_joint')

wheel0.setPosition(float('inf'))
wheel1.setPosition(float('inf'))
wheel2.setPosition(float('inf'))

# set up the motor speeds
wheel0.setVelocity(1)
wheel1.setVelocity(1)
wheel2.setVelocity(1)

# GPS
gps = robot.getDevice('gps')
gps.enable(TIME_STEP)

# Compass
compass = robot.getDevice('compass')
compass.enable(TIME_STEP)

c = 1
t = 0

# create x, y, theta, time lists
```

```python
x = []
y = []
theta = []
time = []

while robot.step(TIME_STEP) != -1:
    gps_value = gps.getValues()
    print(gps_value)
    x.append(gps_value[0])
    y.append(gps_value[1])

    compass_value = compass.getValues()
    time.append(t)
    theta.append(math.atan2(compass_value[1], compass_value[0]))

    if (c > 100):
        break
    c += 1
    t += TIME_STEP

# Plot
fig, ax = plt.subplots(2)

ax[0].set(xlabel='X', ylabel='Y')
ax[0].plot(x, y)

ax[1].plot(time, theta)
ax[1].set(xlabel='T', ylabel='θ')
plt.show()
```



.2

برای به دست آوردن موقعیت ربات در لحظه بعد، میتوانیم از روابط زیر استفاده کنیم:

$$\frac{x_1 - x_0}{\Delta t} = \dot{x} \longrightarrow \boxed{x_1 = \dot{x}\Delta t + x_0}$$

$$\frac{y_1 - y_0}{\Delta t} = \dot{y} \longrightarrow \boxed{y_1 = \dot{y}\Delta t + y_0}$$

$$\frac{\theta_1 - \theta_0}{\Delta t} = \dot{\theta} \longrightarrow \boxed{\theta_1 = \dot{\theta}\Delta t + \theta_0}$$

موقعیت حال حاضر ربات : θ_0, y_0, x_0

موقعیت ربات در لحظه بعد : θ_1, y_1, x_1

مدت زمان جابجایی از موقعیت فعلی به بعدی : Δt

سرعت خطی و زاویه ای : $\dot{\theta}, \dot{y}, \dot{x}$

طبق روابط بالا، نیاز داریم مقادیر سرعت خطی و زاویه را که در چارچوب مرجع جهانی هستند بیابیم.

برای این کار میتوانیم از روابط کتاب مرجع استفاده کنیم:

$$^{I}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} .$$

$$v = k_\rho \rho ,$$

$$\omega = k_\alpha \alpha + k_\beta \beta ,$$

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} .$$

$$\alpha = -\theta + \operatorname{atan2}(\Delta y, \Delta x) .$$

$$\beta = -\theta - \alpha .$$

مقادیر k، طبق کتاب مرجع در شبیه سازی انجام شده به صورت زیر تعیین شده است:

$$k = (k_\rho, k_\alpha, k_\beta) = (3, 8, -1.5) .$$

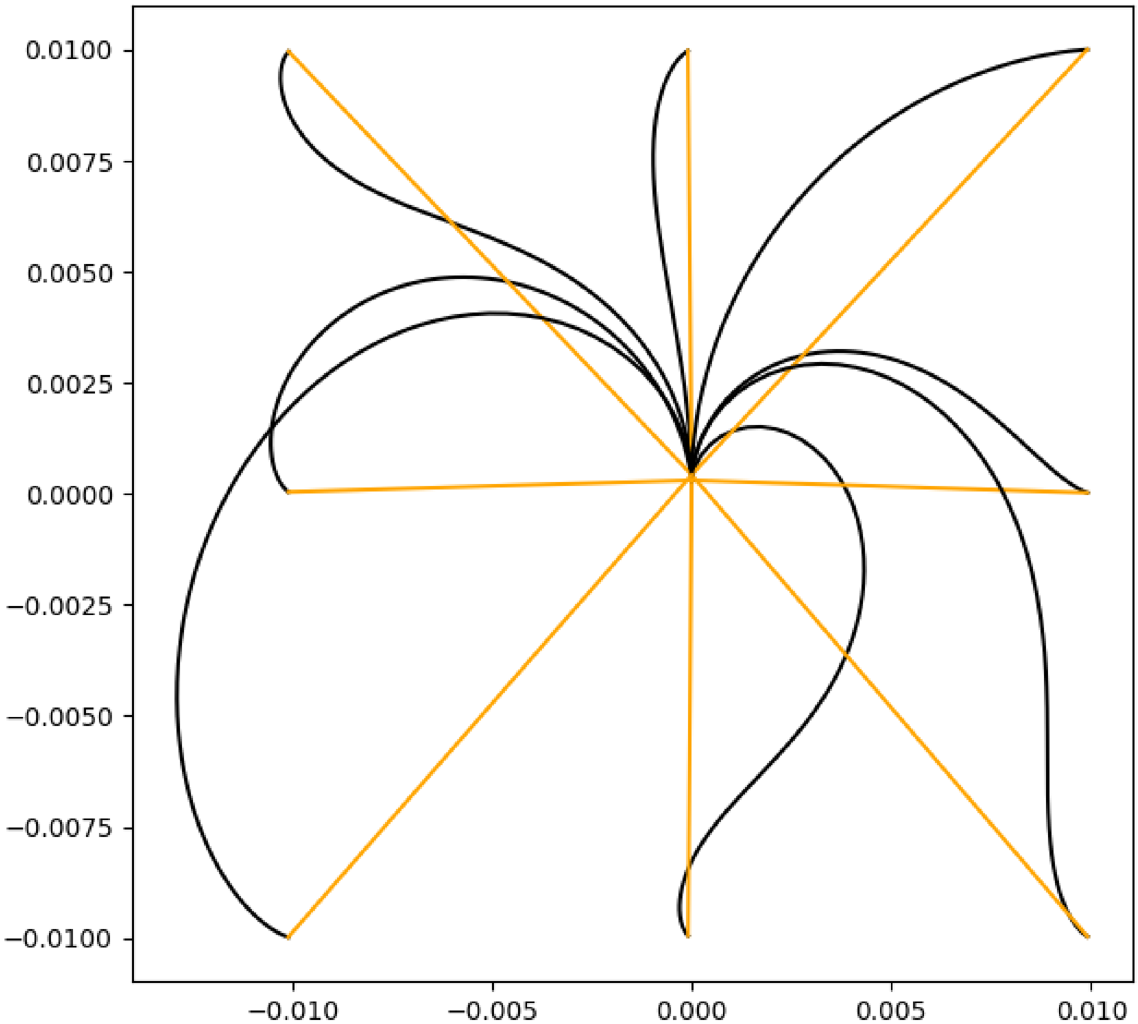
در شکل زیر، ربات باید 8 بار از مرکز و با زاویه 90 درجه نسبت به محور افقی به سمت نقاط مختلف حرکت کند.

کد این سوال و نمودار رسم شده:

```python
import math
import numpy as np
import matplotlib.pyplot as plt


def p_controller(x_goal, y_goal, x_initial = 0, y_initial = 0,
theta_initial = math.pi/2, delta_t = 0.01, threshold = 0.0001):
    x_coordinates = []
    y_coordinates = []

    theta = theta_initial
    x_cur = x_initial
    y_cur = y_initial

    while math.sqrt(math.pow((x_goal-x_cur), 2) + math.pow((y_goal-
y_cur), 2)) > threshold:
        delta_x = x_goal - x_cur
        delta_y = y_goal - y_cur

        ro = math.sqrt(math.pow(delta_x, 2) + math.pow(delta_y, 2))

        alpha = -theta + math.atan2(delta_y, delta_x)
        while(alpha > math.pi):
          alpha = alpha - 2 * math.pi
        while(alpha < -math.pi):
          alpha = alpha + 2 * math.pi

        beta = -theta - alpha
        while(beta > math.pi):
          beta = beta - 2 * math.pi
        while(beta < -math.pi):
          beta = beta + 2 * math.pi

        k = (3, 8, -1.5) # (k_ro, k_alpha, k_beta)

        v =  k[0] * ro
        omega = k[1] * alpha + k[2] * beta
```

```
        x_dot, y_dot, theta_dot = v * math.cos(theta), v *
math.sin(theta), omega

        x_cur, y_cur, theta = x_cur + x_dot*delta_t, y_cur +
y_dot*delta_t, theta + omega*delta_t

        x_coordinates.append(x_cur)
        y_coordinates.append(y_cur)


    return x_coordinates, y_coordinates

plt.figure(figsize=(7, 7), dpi=100)

xx, yy = p_controller(0, 10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(0, -10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(10/1000, 0)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(-10/1000, 0)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(10/1000, 10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(-10/1000, 10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(10/1000, -10/1000)
```

```
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(-10/1000, -10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')
```



.3

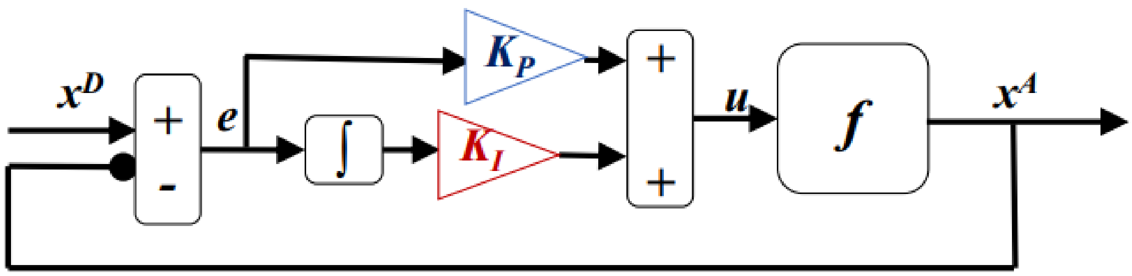
مشکل کنترلر P این است که ورودی آن متناسب با میزان خطاست.

در ابتدا که خطا زیاد است دستورات بزرگتر ایجاد میشود.

به مرور با کاهش خطا، خروجی کنترلی هم کاهش می یابد.

یک راه حل برای آن، استفاده از مجموع (انتگرال خطا) است که در طول زمان افزایش می یابد.

افزودن مضربی از انتگرال خطا میتواند به خطای ثابت پایانی کنترلر P کمک کند.

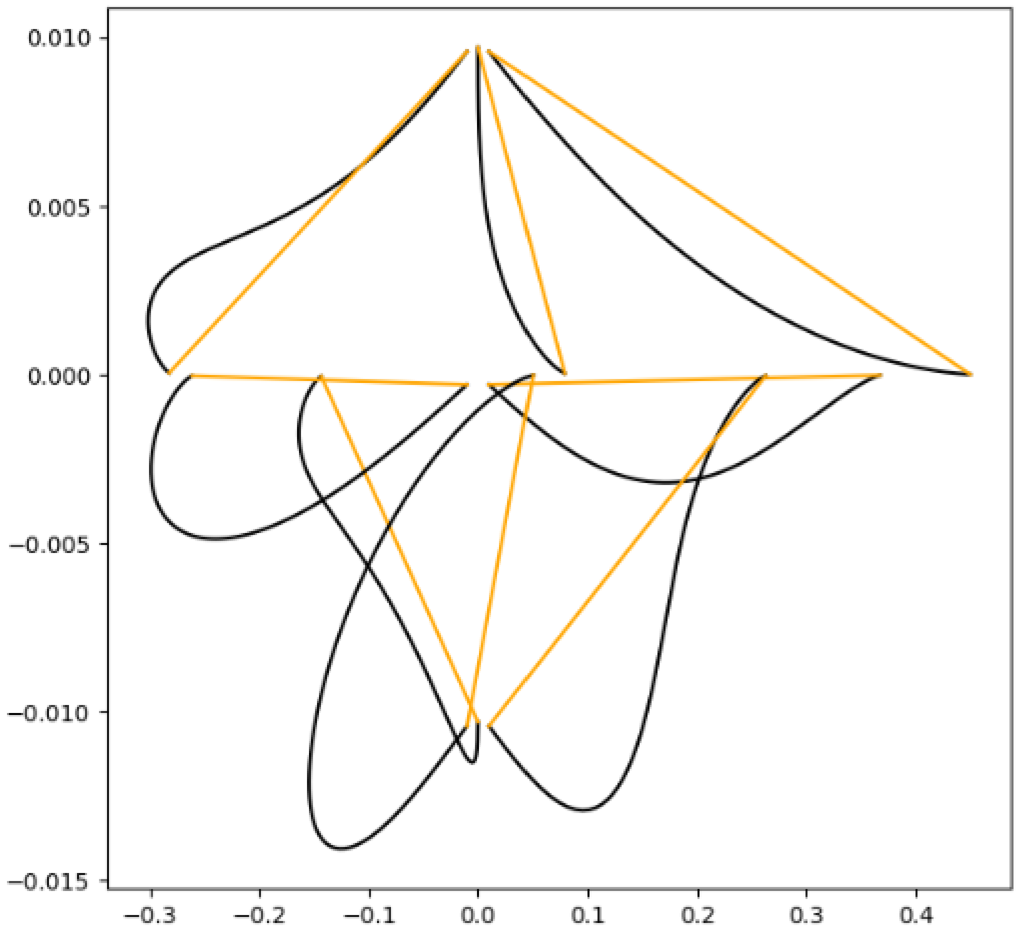


کد و نمودار رسم شده:

```
import math
import numpy as np
import matplotlib.pyplot as plt


def p_controller(x_goal, y_goal, x_initial=0, y_initial=0,
theta_initial=math.pi/2, delta_t=0.01, threshold=0.0001):
    x_errors = []
    y_errors = []

    theta = theta_initial
    x_cur = x_initial
    y_cur = y_initial

    e_x = 0
    e_y = 0
```

```python
    while math.sqrt(math.pow((x_goal-x_cur), 2) + math.pow((y_goal-
y_cur), 2)) > threshold:
        delta_x = x_goal - x_cur
        delta_y = y_goal - y_cur

        ro = math.sqrt(math.pow(delta_x, 2) + math.pow(delta_y, 2))

        alpha = -theta + math.atan2(delta_y, delta_x)
        while(alpha > math.pi):
          alpha = alpha - 2 * math.pi
        while(alpha < -math.pi):
          alpha = alpha + 2 * math.pi

        beta = -theta - alpha
        while(beta > math.pi):
          beta = beta - 2 * math.pi
        while(beta < -math.pi):
          beta = beta + 2 * math.pi

        k = (3, 8, -1.5)  # (k_ro, k_alpha, k_beta)

        v = k[0] * ro
        omega = k[1] * alpha + k[2] * beta

        x_dot, y_dot, theta_dot = v * \
            math.cos(theta), v * math.sin(theta), omega

        x_cur, y_cur, theta = x_cur + x_dot*delta_t, y_cur + \
            y_dot*delta_t, theta + omega*delta_t

        e_x += x_goal - x_cur
        e_y = + y_goal - y_cur

        x_errors.append(e_x)
        y_errors.append(e_y)

    return x_errors, y_errors
```

```
plt.figure(figsize=(7, 7), dpi=100)

xx, yy = p_controller(0, 10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(0, -10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(10/1000, 0)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(-10/1000, 0)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(10/1000, 10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(-10/1000, 10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(10/1000, -10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')

xx, yy = p_controller(-10/1000, -10/1000)
plt.plot(xx, yy, color='black')
plt.plot([xx[0], xx[len(xx)-1]], [yy[0], yy[len(yy)-1]], 'orange')
```


0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4