به نام خدا

درس اصول رباتیکز
نیم‌سال اول ۰۲-۰۳
استاد: دکتر آرمین سلیمی بدر

دانشکده مهندسی و علوم کامپیوتر

پاسخ تمرین سری اول

---

۱. (۲۰ نمره) یک ربات هدایت تفاضلی در موقعیت $\xi_1 = \begin{bmatrix} 1m \\ 2m \\ \frac{\pi}{2}rad \end{bmatrix}$ قرار گرفته است. می‌خواهیم این ربات را به موقعیت $\xi_2 = \begin{bmatrix} 1.5m \\ 2m \\ \frac{\pi}{2}rad \end{bmatrix}$ برسانیم. حرکت این ربات به وسیلهٔ سه‌تایی[1]هایی به شکل $(\dot{\varphi}_1, \dot{\varphi}_2, t)$ تعیین می‌شود که در آن $\dot{\varphi}_1$ سرعت چرخش چرخ راست، $\dot{\varphi}_2$ سرعت چرخش چرخ چپ و t زمان هدایت است.

۱. کمترین دستورات موردنیاز برای رساندن ربات به موقعیت موردنظر چند تاست؟ چرا؟
در یک ربات هدایت تفاضلی، هر دستور تنها می‌تواند منجر به طی شدن بخشی از یک مسیر دایره‌ای شود (یا حرکت روی یک خط صاف که همان مسیر دایره‌ای با شعاع بی‌نهایت است). اگرچه می‌توان با پیمودن یک مسیر نیم‌دایره‌ای از نقطهٔ مبدأ به نقطهٔ مقصد رسید، اما جهت ربات در نهایت درست نخواهد بود. بنابراین، حداقل نیازمند ۲ دستور برای رساندن ربات به موقعیت موردنظر هستیم.

۲. طول کوتاه‌ترین مسیر[2] طبق شرط ذکرشده در بخش قبلی چقدر است؟
ربات می‌تواند دو نیم‌دایره طی کند. اگر فاصلهٔ بین نقطهٔ مبدأ و نقطهٔ مقصد را $d = 0.5m$ در نظر بگیریم، قطر دایرهٔ این دو مسیر را می‌توان به‌ترتیب u و v در نظر گرفت به‌طوری که $u + v = d$. طول کمان این دو مسیر نیز به‌ترتیب برابر با $s_1 = \frac{\pi u}{2}$ و $s_2 = \frac{\pi v}{2}$ خواهد بود. بنابراین، طول کوتاه‌ترین مسیر طبق شرط ذکرشده در بخش قبل برابر است با:

$$s = s_1 + s_2 = \frac{\pi u}{2} + \frac{\pi v}{2} = \frac{\pi(u+v)}{2} = \frac{\pi d}{2} = \frac{\pi}{4}$$

طول این مسیر به مقادیر u و v وابسته نیست.

۳. با چه دنباله‌ای از دستورات با تعداد دلخواه می‌توان ربات را از موقعیت اولیه به موقعیت نهایی رساند؟ حداکثر سرعت هر چرخ را v و فاصلهٔ بین دو چرخ را l در نظر بگیرید.
می‌توان مسیر را مسیری با کوتاه‌ترین طول ممکن (فاصلهٔ اقلیدسی بین نقطهٔ مبدأ و نقطهٔ مقصد) در نظر گرفت. برای طی کردن چنین مسیری، ربات باید ابتدا $\frac{\pi}{2}$ رادیان به راست بچرخد، $0.5m$ رو به جلو حرکت کند و در نهایت، $\frac{\pi}{2}$ رادیان به چپ بچرخد. بنابراین، نیازمند سه دستور خواهیم بود.
سرعت خطی و سرعت زاویه‌ای ربات از طریق روابط زیر قابل محاسبه است:

$$v = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{t} = \frac{v_1 + v_2}{2}, \omega = \frac{\Delta\theta}{t} = \frac{v_1 - v_2}{l}$$

[1] Triplet (3-Tuple)
[2] Trajectory

بنابراین، دنبالهٔ دستورات به‌صورت زیر خواهد بود:

- $(-v, v, \frac{\pi l}{4v})$
- $(v, v, \frac{d}{v})$
- $(v, -v, \frac{\pi l}{4v})$

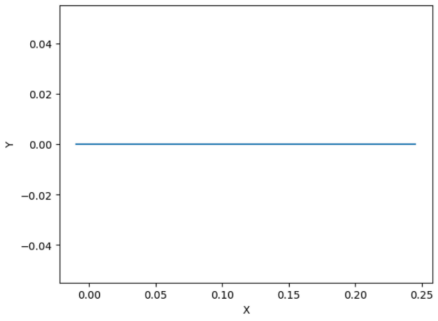
اگر بخواهیم به‌جای سرعت خطی چرخ‌ها از سرعت زاویه‌ای چرخ‌ها استفاده کنیم، باید سرعت چرخ‌ها را بر شعاع‌شان تقسیم کرد (شعاع چرخ‌ها را r فرض می‌کنیم):

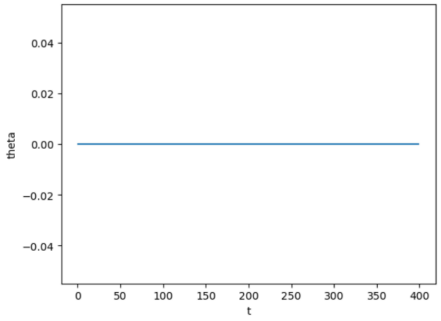
- $(-\frac{v}{r}, \frac{v}{r}, \frac{\pi l}{4v})$
- $(\frac{v}{r}, \frac{v}{r}, \frac{d}{v})$
- $(\frac{v}{r}, -\frac{v}{r}, \frac{\pi l}{4v})$

۴. طول مسیر به‌دست‌آمده در بخش قبلی چقدر است؟
فاصلهٔ اقلیدسی بین نقطهٔ مبدأ و نقطهٔ مقصد، که همان $0.5m$ است.

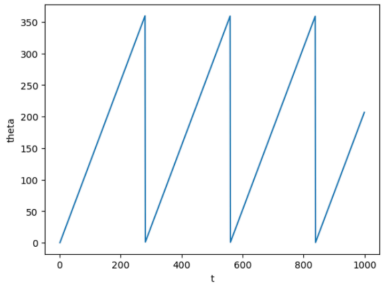
۲. (۱۵ نمره) سرعت‌های زیر را به چرخ‌های ربات e-puck[3] اعمال کرده و در هر مورد، نمودار مسیر حرکت $(X - Y)$ و جهت سر ربات نسبت به زمان $(\theta - t)$ را رسم کنید:

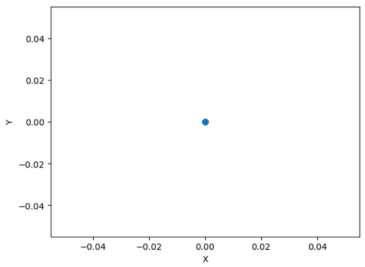
۱. $\dot{\varphi}_1 = 1rad/s, \dot{\varphi}_2 = 1rad/s$





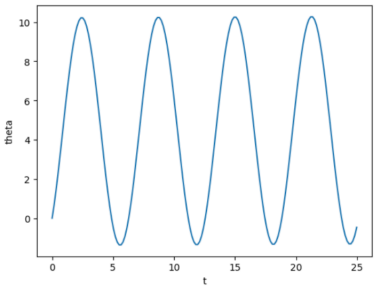
۲. $\dot{\varphi}_1 = 1rad/s, \dot{\varphi}_2 = -1rad/s$

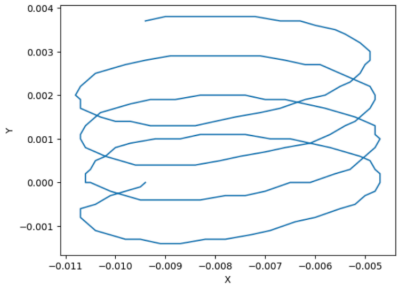




۳. $\dot{\varphi}_1(t) = \sin t\ rad/s, \dot{\varphi}_2(t) = -\cos t\ rad/s$

[3] https://www.cyberbotics.com/doc/guide/epuck?version=cyberbotics:R2019a





منظور از t، گام زمانی در شبیه‌ساز است.

برای دسترسی به موقعیت ربات، از حسگرهای GPS[4] و Compass[5] استفاده کنید.

برای رسم نمودارها، استفاده از هر ابزار دلخواهی مجاز است.

با تشکر از سید حسن مجید زنوزی، محمدمهدی پرچمی جلال و پگاه بیک‌زاده

۳. (۲۰ نمره) تابعی برای سینماتیک مستقیم ربات هدایت تفاضلی پیاده‌سازی کنید.

پارامترهای ورودی:

- x: مؤلفهٔ افقی موقعیت فعلی ربات
- y: مؤلفهٔ عمودی موقعیت فعلی ربات
- θ: زاویهٔ سر ربات
- v_1: سرعت چرخ راست
- v_2: سرعت چرخ چپ
- t: زمان هدایت
- l: فاصلهٔ بین دو چرخ ربات

مقادیر خروجی:

- x_n: مؤلفهٔ افقی موقعیت نهایی ربات
- y_n: مؤلفهٔ عمودی موقعیت نهایی ربات
- θ_n: زاویهٔ نهایی سر ربات

```
import math


def diffdrive(x, y, theta, v_1, v_2, t, l):
    if v_1 == v_2:
        x_n = x + v_2 * t * math.cos(theta_n)
        y_n = y + v_2 * t * math.sin(theta_n)

        return x_n, y_n, theta_n

    R = l / 2.0 * ((v_2 + v_1) / (v_1 - v_2))
```

[4] https://cyberbotics.com/doc/reference/gps
[5] https://cyberbotics.com/doc/reference/compass

```
    ICC_x = x - R * math.sin(theta)
    ICC_y = y + R * math.cos(theta)
    omega = (v_1 - v_2) / l
    dtheta = omega * t
    x_n = math.cos(dtheta) * (x - ICC_x) - math.sin(dtheta) * (y -
        ICC_y) + ICC_x
    y_n = math.sin(dtheta) * (x - ICC_x) + math.cos(dtheta) * (y -
        ICC_y) + ICC_y
    theta_n = theta + dtheta

    return x_n, y_n, theta_n
```

فرض کنید رباتی در موقعیت $\xi = \begin{bmatrix} 1.5m \\ 2m \\ \frac{\pi}{2}rad \end{bmatrix}$ قرار گرفته است. دنبالهٔ زیر از دستورات را بر روی آن اجرا می‌کنیم:

۱. $c_1 = (v_1 = 0.3m/s, v_2 = 0.3m/s, t = 3s)$
۲. $c_2 = (v_1 = 0.1m/s, v_2 = -0.1m/s, t = 1s)$
۳. $c_3 = (v_1 = 0.2m/s, v_2 = 0m/s, t = 2s)$

به کمک تابعی که پیاده‌سازی کردید، موقعیت ربات را پس اعمال هر دستور محاسبه کنید. فاصلهٔ بین دو چرخ ربات را $l = 0.5m$ در نظر بگیرید.

```
l = 0.5
x, y, theta = 1.5, 2.0, math.pi / 2.0
print('starting pose: x: %f, y: %f, theta: %f' % (x, y, theta))
v_1 = 0.3
v_2 = 0.3
t = 3
x, y, theta = diffdrive(x, y, theta, v_2, v_1, t, l)
print('after motion 1: x: %f, y: %f, theta: %f' % (x, y, theta))
v_1 = -0.1
v_2 = 0.1
t = 1
x, y, theta = diffdrive(x, y, theta, v_2, v_1, t, l)
print('after motion 2: x: %f, y: %f, theta: %f' % (x, y, theta))
v_1 = 0.0
v_2 = 0.2
t = 2
x, y, theta = diffdrive(x, y, theta, v_2, v_1, t, l)
print('after motion 3: x: %f, y: %f, theta: %f' % (x, y, theta))
```

موقعیت ابتدایی ربات و موقعیت ربات پس از انجام هر دستور به‌صورت زیر خواهد بود:

$$(1.500000, 2.000000, 1.570796)$$
$$(1.500000, 2.900000, 1.570796)$$
$$(1.500000, 2.900000, 1.170796)$$

$$(1.639676, 3.035655, 0.370796)$$

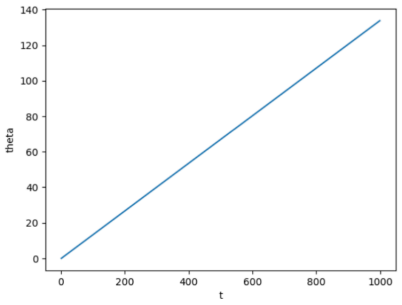
۴. (۱۵ نمره) مدل سینماتیک معکوس ربات هدایت تفاضلی[۶] را در قالب تابعی به‌دست آورید که با دریافت سرعت خطی و سرعت زاویه‌ای ربات، سرعت چرخش چرخ‌های ربات را محاسبه کرده و برگرداند. سپس، حالت‌های زیر را با استفاده از آن توسط ربات e-puck شبیه‌سازی کرده و در هر مورد، نمودار مسیر حرکت ربات $(X - Y)$ و جهت سر ربات نسبت به زمان $(\theta - t)$ را رسم کنید:

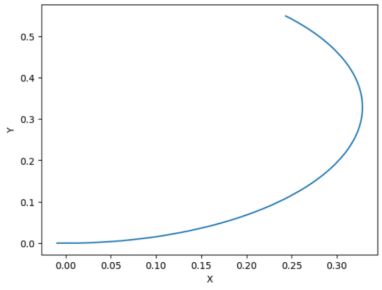
۱. $v = 0.03m/s, \omega = 0.1rad/s$

۲. $v = 0m/s, \omega = 0.5rad/s$

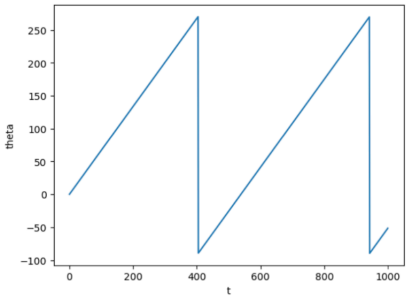
```
def inverse_kinematic(xp, thetap):
    s = xp * (2 / (DIAMETER / 2))
    sub = thetap * (CHASSIS_LENGTH / (DIAMETER / 2))
    return ((s + sub) / 2), (s - (s + sub) / 2)
```

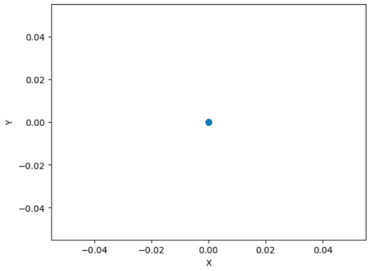
در بخش اول، سرخت چرخ اول برابر با 1.304 و سرعت چرخ چپ برابر با 1.096 خواهد بود. نمودار مسیر حرکت ربات و جهت سر ربات نسبت به زمان در این حالت به‌صورت زیر است:





در بخش دوم، سرخت چرخ اول برابر با 0.512 و سرعت چرخ چپ برابر با -0.512 خواهد بود. نمودار مسیر حرکت ربات و جهت سر ربات نسبت به زمان در این حالت به‌صورت زیر است:



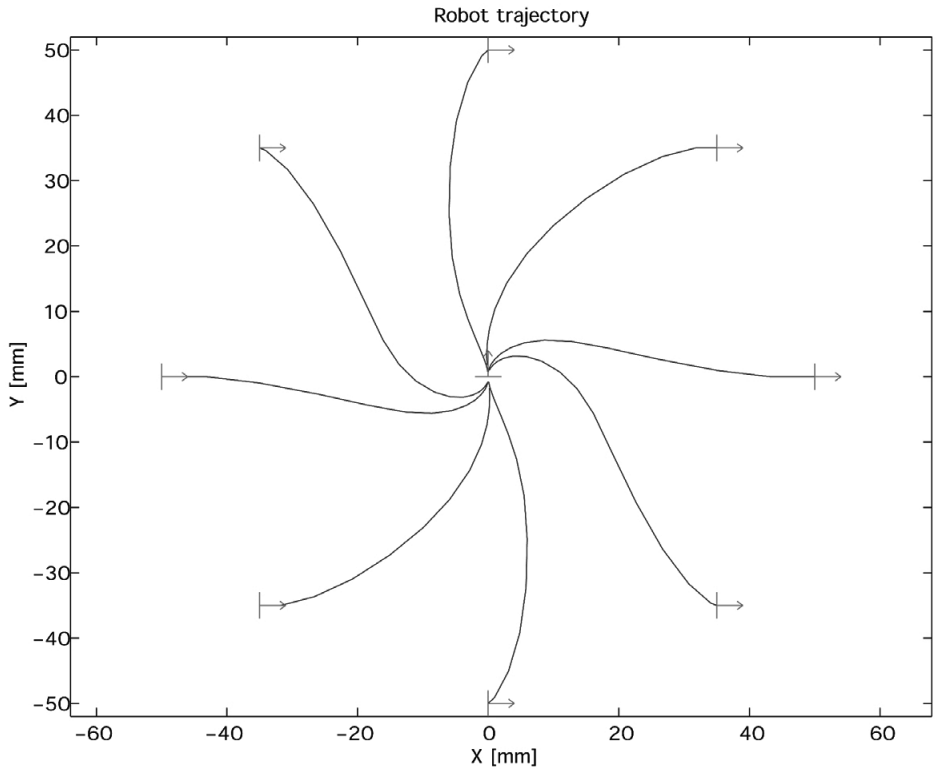


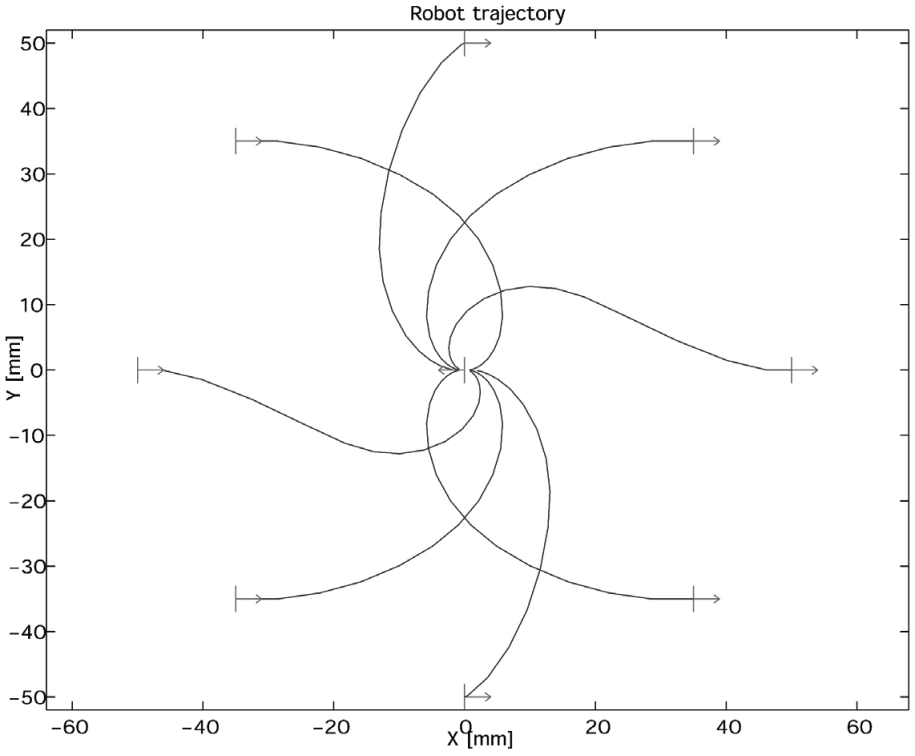
با تشکر از سید حسن مجید زنوزی، محمدمهدی پرچمی جلال و پگاه بیک‌زاده

۵. (۱۵ نمره) یک ربات هدایت تفاضلی را در نظر بگیرید که بر روی نقطه‌ای از دایره‌ای با شعاع $r = 0.5m$ با زاویهٔ دلخواه قرار گرفته است. می‌خواهیم به سمت مرکز دایره حرکت کنیم. یک کنترل‌کنندهٔ P برای این ربات طراحی کنید و این حرکت را با استفاده از این کنترل‌کننده به ازای موقعیت‌های اولیهٔ مختلف انجام دهید. نمودار مسیرهای موردانتظار و مسیرهای پیموده‌شده را نیز رسم کنید.

[6] Differential-Drive Robot

تصاویری از مسیرهای پیموده‌شده توسط ربات (با فرض این که شعاع دایره برابر با $r = 50mm$ است) در شکل‌های زیر آمده است:





برای حل این سؤال، مطالعهٔ بخش ۳-۶ (صفحات ۹۱ تا ۹۹) کتاب مرجع درس[۷] توصیه می‌شود.

[7] Introduction to Autonomous Mobile Robots, second edition

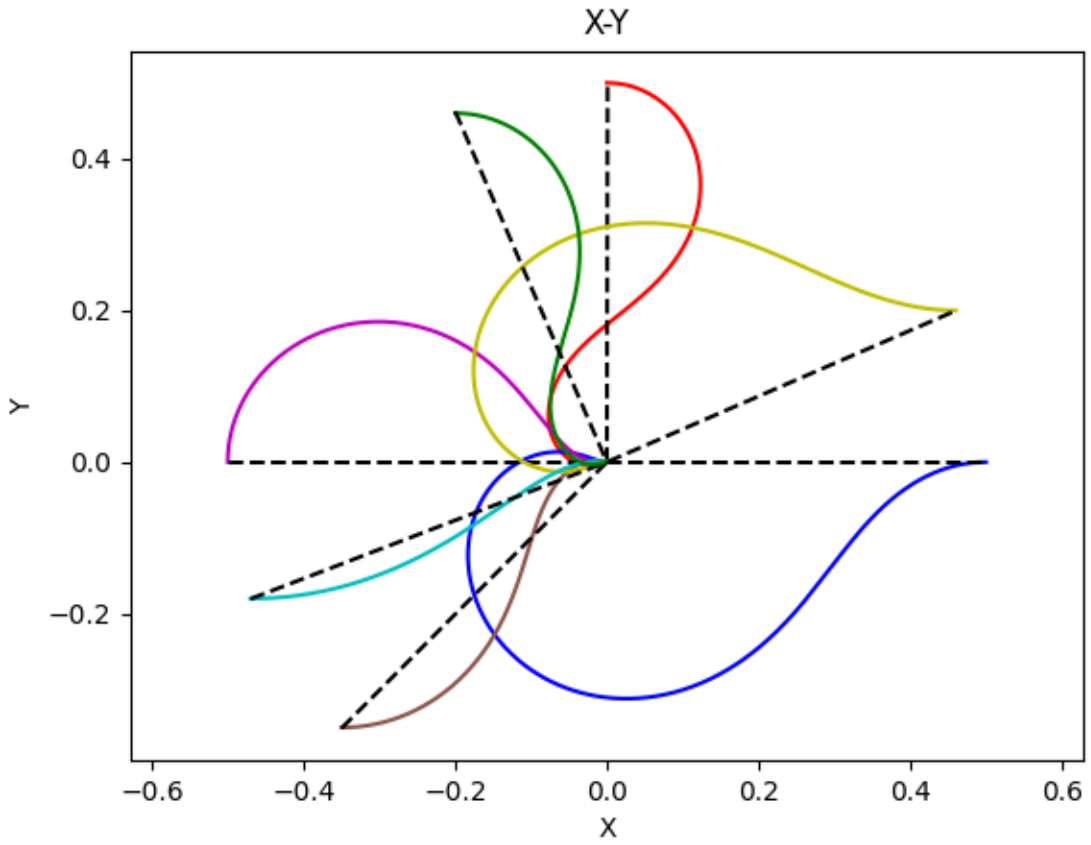
یک نمونهٔ پیاده‌سازی‌شده به‌صورت زیر است:

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

def move(kesiI, v, w):
    x, y, teta = kesiI
    Vx = v
    Vy = 0
    omega = w
    kesiDotR = np.array([[Vx],
                         [Vy],
                         [omega]])
    RInverse = np.array([[math.cos(-teta), math.sin(-teta), 0],
                [-math.sin(-teta), math.cos(-teta), 0],
                [0, 0, 1]])
    kesiDotI = RInverse.dot(kesiDotR)
    newKesiI = np.array([(x + 0.05*kesiDotI[0])[0],
                         (y + 0.05*kesiDotI[1])[0],
                         (teta + 0.05*kesiDotI[2])[0]]).tolist()
    return newKesiI

def getAlpha(currentCoordinate):
    teta = currentCoordinate[2]
    f = math.atan2(0 - currentCoordinate[1], 0 - currentCoordinate
       [0])
    alpha = f - teta
    while alpha > math.pi:
        alpha -= 2 * math.pi
    while alpha < -math.pi:
        alpha += 2 * math.pi
    return alpha

def getBeta(teta, alpha):
    beta = -teta - alpha
    while beta > math.pi:
        beta -= 2 * math.pi
    while beta < -math.pi:
        beta += 2 * math.pi
    return beta

def getRo(currentCoordinate):
    return math.sqrt(currentCoordinate[0] ** 2 + currentCoordinate
       [1] ** 2)
```

```
def main(currentCoordinate):
    alpha = getAlpha(currentCoordinate)
    beta = getBeta(currentCoordinate[2], alpha)
    ro = getRo(currentCoordinate)

    if alpha <= math.pi/2 and alpha >= -math.pi/2:
            Kro = 1/10
            Kb = -1/10
            Ka = 2/10
    else:
            Kro = -1/10
            Kb = 1/10
            Ka = -2/10

    xValues = []
    yValues = []

    while True:
        xValues.append(currentCoordinate[0])
        yValues.append(currentCoordinate[1])
        v = Kro * ro
        w = Kb * beta + Ka * alpha
        currentCoordinate = move(currentCoordinate, v, w)
        alpha = getAlpha(currentCoordinate)
        beta = getBeta(currentCoordinate[2], alpha)
        ro = getRo(currentCoordinate)
        if alpha < 0.0001 and beta < 0.0001 and ro < 0.0001:
            break
    return xValues, yValues


differentStarts = [[0, 0.5, 0], [0.5, 0, math.pi], [-0.35, -0.35,
   0], [-0.5, 0, math.pi/2], [0.46, 0.2, math.pi], [-0.47, -0.18,
   0], [-0.2, 0.46, 0]]
colors = ['r', 'b', 'g', 'y', 'c', 'm', 'tab:pink', 'tab:gray', '
   tab:brown', 'tab:orange', 'tab:olive']

plt.figure()
for startPoint in differentStarts:
    xValues, yValues = main(startPoint)
    plt.plot(xValues, yValues, colors[random.randint(0, 9)])

    x, y = startPoint[0], startPoint[1]
    if x > 0:
        t = np.linspace(0.001, x, 400)
    else:
        t = np.linspace(x, 0.001, 400)
```

```
        f = (y / (x + 0.001)) * t
        plt.plot(t, f, '--k')

plt.xlabel('X')
plt.ylabel('Y')
plt.title('X-Y')
plt.axis('equal')
plt.show()
```

خروجی به‌صورت زیر خواهد بود:



با تشکر از محمدمتین مؤمنی راوندی، محمدسجاد صریریان مبارکه و صدرا برزگی