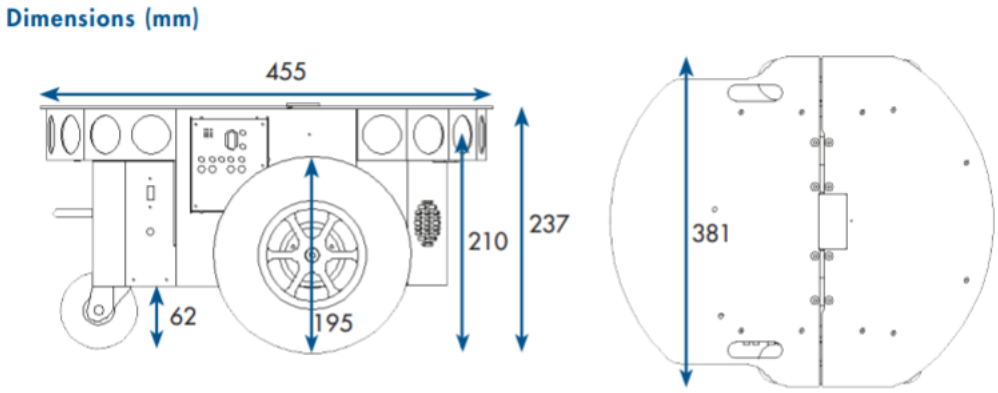
Robotica

October 4, 2021

[1]:
```
import time
import sys
import numpy as np
from numpy import sin,cos
import matplotlib.pyplot as plt
from PIL import Image
import cv2
```

---

1 Dimensões do Robô



[2]:
```
# Dimensões do Robô em metros
rodaDiametro = (195)/1000 #R
raio = rodaDiametro/2
rodaDireitaRaio = rodaDiametro/2 #Rd
rodaEsquerdaRaio = rodaDiametro/2 #Re
distanciaRodaEixo = ((381)/1000)/2 #L
velocidadeAngularRodaDireita = 0.5
velocidadeAngularRodaEsquerda = 0.5
```

---

2 Modelo I

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \underbrace{\begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0\\ \sin(\theta) & \cos(\theta) & 0\\ 0 & 0 & 1\end{bmatrix}}_{B}\underbrace{\begin{bmatrix}1 & 0\\ 0 & 0\\ 0 & 1\end{bmatrix}}_{C}\underbrace{\begin{bmatrix}\frac{r_d}{2} & \frac{r_e}{2}\\ \frac{r_d}{2l} & \frac{-r_e}{2l}\end{bmatrix}}_{D}\underbrace{\begin{bmatrix}w_d\\ w_e\end{bmatrix}}_{E}$$

[3]:
```
def ModeloDiferencialPasso(PosicaoRobo, velocidadeAngularRodaDireita = 0.5,␣
↪velocidadeAngularRodaEsquerda = 0.5, Passo=0.05):
    rotacao = float(PosicaoRobo[2])

    matrizB = np.array([cos(rotacao),   -sin(rotacao),  0,
                    sin(rotacao),   cos(rotacao),   0,
                    0           ,0,                 1]).reshape((3,3))


    matrizC = np.array([1,0,
                        0,0,
                        0,1]).reshape((3,2))

    matrizD = np.array([rodaDireitaRaio/2, rodaEsquerdaRaio/2,
                    (rodaDireitaRaio/(2*distanciaRodaEixo)), (-rodaEsquerdaRaio/
↪(2*distanciaRodaEixo))]).reshape((2,2))
    matrizE = np.array([velocidadeAngularRodaDireita,
                    velocidadeAngularRodaEsquerda]).reshape((2,1))


    matrizA = np.matmul(np.matmul(matrizB,matrizC),np.matmul(matrizD,matrizE))␣
↪#A = B*C*D*E

    deslocamento = matrizA*0.05 #velocidade * tempo

    PosicaoNova = PosicaoRobo + deslocamento

    return PosicaoNova, matrizA
```

[4]:
```
def SimulaModeloUm(velocidadeAngularRodaDireita, velocidadeAngularRodaEsquerda,␣
↪Passo=0.05, TempoSimulacao = 10):


    #Periodo de amostragem e tempo de simulacao
    T = Passo # Tempo de passo. Tem de estar em concordancia com Vrep!
```

```
    Tsim = TempoSimulacao  # segundos
    t = np.arange(0,Tsim,T)
    N = len(t)

    PosicaoRobo = np.array([0,
                    0,
                    0]).reshape((3,1)) #Valores Iniciais de Posicao e angulo␣
↪zerados
    VelocidadeRobo = np.array([0,
                           0,
                           0]).reshape((3,1)) #Valores Iniciais de velocidade e␣
↪rotação zerados
    PosicaoHistorico = PosicaoRobo
    VelocidadeHistorico = VelocidadeRobo


    for i in range(0,N-1):
        PosicaoNova, Velocidades = ModeloDiferencialPasso(PosicaoRobo,
                                    velocidadeAngularRodaDireita,
                                    velocidadeAngularRodaEsquerda,
                                    Passo)
        PosicaoHistorico = np.append(PosicaoHistorico,PosicaoNova,axis=1)
        VelocidadeHistorico = np.append(VelocidadeHistorico,Velocidades,axis=1)

        PosicaoRobo = PosicaoNova

    return PosicaoHistorico
```

[5]:
```
def Simula(velocidadeAngularRodaDireita,velocidadeAngularRodaEsquerda,Passo=0.
↪05):
    try:
        import sim
    except:
        print ('--------------------------------------------------------------')
        print ('"sim.py" could not be imported. This means very probably that')
        print ('either "sim.py" or the remoteApi library could not be found.')
        print ('Make sure both are in the same folder as this file,')
        print ('or appropriately adjust the file "sim.py"')
        print ('--------------------------------------------------------------')
        print ('')

    sim.simxFinish(-1) # just in case, close all opened connections
    clientID=sim.simxStart('127.0.0.1',19997,True,True,5000,5) # Connect to␣
↪CoppeliaSim
    if clientID!=-1:
        print ('Connected to remote API server')
```

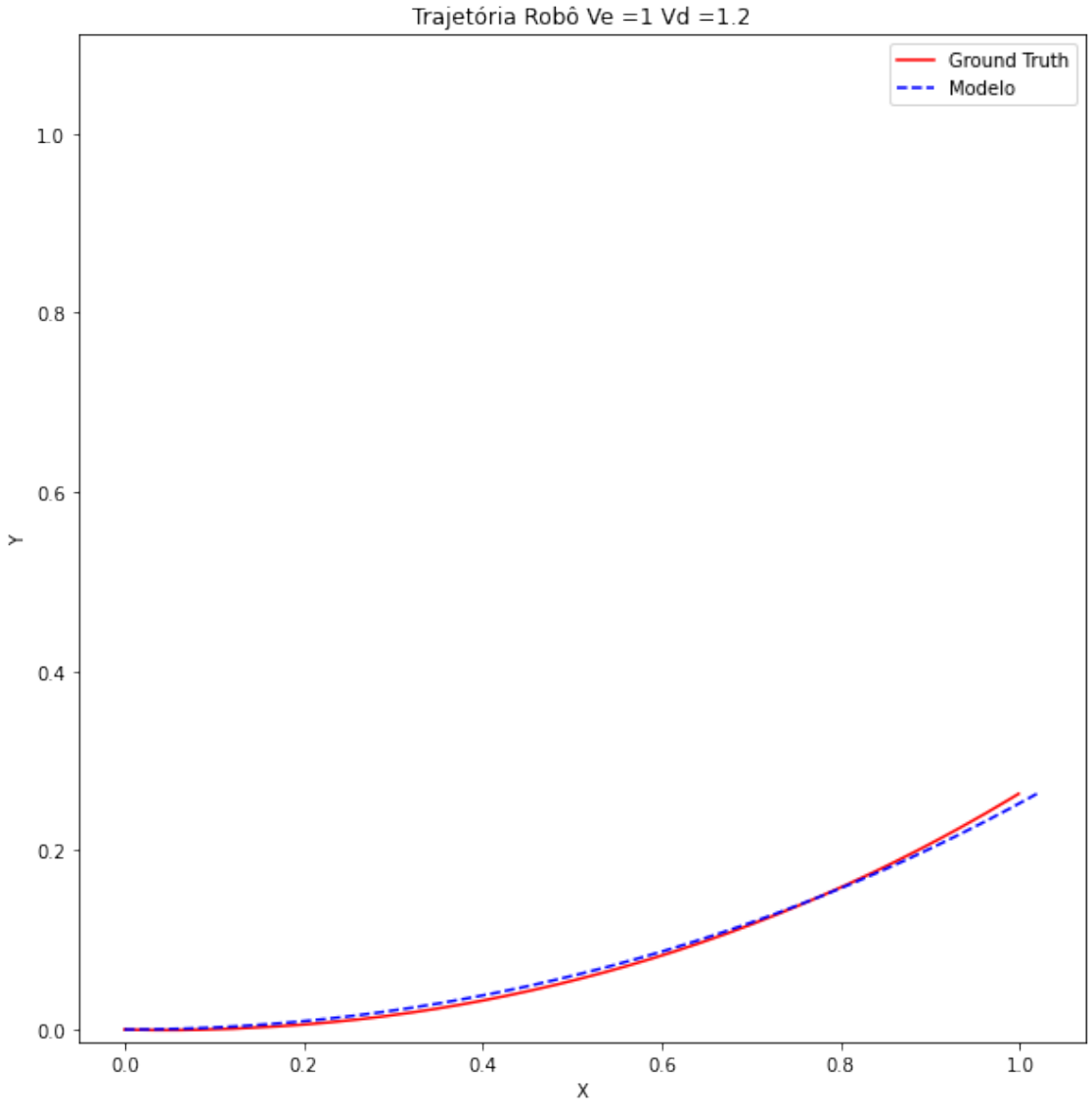
```
        print("Simulation Started", end =" ")
        # enable the synchronous mode on the client:
        sim.simxSynchronous(clientID,True)

        # start the simulation:
        sim.simxStartSimulation(clientID,sim.simx_opmode_blocking)

        ## Handle
        returnCode,Robot = sim.simxGetObjectHandle(clientID,'Pioneer_p3dx',sim.
↪simx_opmode_blocking)
        returnCode,left_Motor= sim.
↪simxGetObjectHandle(clientID,'Pioneer_p3dx_leftMotor',sim.
↪simx_opmode_blocking)
        returnCode,right_Motor= sim.
↪simxGetObjectHandle(clientID,'Pioneer_p3dx_rightMotor',sim.
↪simx_opmode_blocking)
        returnCode,front_Sensor =sim.
↪simxGetObjectHandle(clientID,'Pioneer_p3dx_ultrasonicSensor5',sim.
↪simx_opmode_blocking)
        returnCode,camera= sim.simxGetObjectHandle(clientID,'Vision_sensor',sim.
↪simx_opmode_blocking)
        returnCode,resolution,image = sim.
↪simxGetVisionSensorImage(clientID,camera,1,sim.simx_opmode_streaming)


        #Periodo de amostragem e tempo de simulacao
        T = Passo # Tempo de passo. Tem de estar em concordancia com Vrep!
        Tsim = 10  # segundos
        t = np.arange(0,Tsim,T)
        N = len(t)

        #Pose inicial
        returnCode,Real_Robot_position = sim.
↪simxGetObjectPosition(clientID,Robot,-1,sim.simx_opmode_blocking)
        P = np.array(Real_Robot_position).reshape((3,1))
        PosicaoInicial = P
        PosicaoRealHistorico = P - PosicaoInicial


        # Now step a few times:
        for i in range(0,N-1):
            #Anda um passo de simulacao
            sim.simxSynchronousTrigger(clientID)


            #Velocidade de referencia
```

```
            v_motor_l= velocidadeAngularRodaEsquerda
            v_motor_r= velocidadeAngularRodaDireita

            returnCode=sim.
↪simxSetJointTargetVelocity(clientID,left_Motor,v_motor_l,sim.
↪simx_opmode_blocking)
            returnCode=sim.
↪simxSetJointTargetVelocity(clientID,right_Motor,v_motor_r,sim.
↪simx_opmode_blocking)


            returnCode,Real_Robot_position = sim.
↪simxGetObjectPosition(clientID,Robot,-1,sim.simx_opmode_blocking)
            P = np.array(Real_Robot_position).reshape((3,1)) - PosicaoInicial
            PosicaoRealHistorico = np.append(PosicaoRealHistorico,P,axis=1)
            b = "Simulando " + str((i/(N-1))*100) + '%'
            #print (b)
            sys.stdout.write('\r'+b)
            returnCode,resolution,image =sim.
↪simxGetVisionSensorImage(clientID,camera,1,sim.simx_opmode_buffer)
            if returnCode == sim.simx_return_ok :
                imageAcquisitionTime=sim.simxGetLastCmdTime(clientID)
                #time.sleep(.5)
                img = np.array(image,dtype=np.uint8)
                img.resize([resolution[1],resolution[0]])

                img = np.flip(img)
                img = np.fliplr(img)
                cv2.imshow('image',img)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break


        # stop the simulation:
        sim.simxStopSimulation(clientID,sim.simx_opmode_blocking)
        print()
        print('Simulation Stopped')

        # Now close the connection to CoppeliaSim:
        sim.simxFinish(clientID)
        print('Simulation Closed')


    else:
        print ('Failed connecting to remote API server')
        print ('Program ended')
    return PosicaoRealHistorico
```

[6]:
```
velocidadeAngularRodaEsquerda = 1
velocidadeAngularRodaDireita = 1.2
PosicaoRealHistorico =␣
 ↪Simula(velocidadeAngularRodaDireita,velocidadeAngularRodaEsquerda,Passo=0.05)

PosicaoModeloHistorico =␣
 ↪SimulaModeloUm(velocidadeAngularRodaDireita,velocidadeAngularRodaEsquerda,␣
 ↪Passo=0.05)

plt.rcParams["figure.figsize"] = (10,10)

PosicaoRealX = PosicaoRealHistorico[0,:]
PosicaoRealY = PosicaoRealHistorico[1,:]
AnguloReal = PosicaoRealHistorico[2,:]

plt.plot(PosicaoRealX,PosicaoRealY,color='red', label="Ground Truth")

PosicaoX = PosicaoModeloHistorico[0,:]
PosicaoY = PosicaoModeloHistorico[1,:]
Angulo = PosicaoModeloHistorico[2,:]
plt.plot(PosicaoX,PosicaoY,color='blue',ls='--', label="Modelo")
Title = 'Trajetória Robô Ve ='+ str(velocidadeAngularRodaEsquerda) + ' Vd =' +␣
 ↪str(velocidadeAngularRodaDireita)
plt.title(Title)
plt.xlabel('X')
plt.ylabel('Y')
plt.axis('square')
plt.legend()
```

```
Connected to remote API server
Simulando 99.49748743718592%
Simulation Stopped
Simulation Closed
```

[6]:
```
<matplotlib.legend.Legend at 0x1b1d10d8430>
```

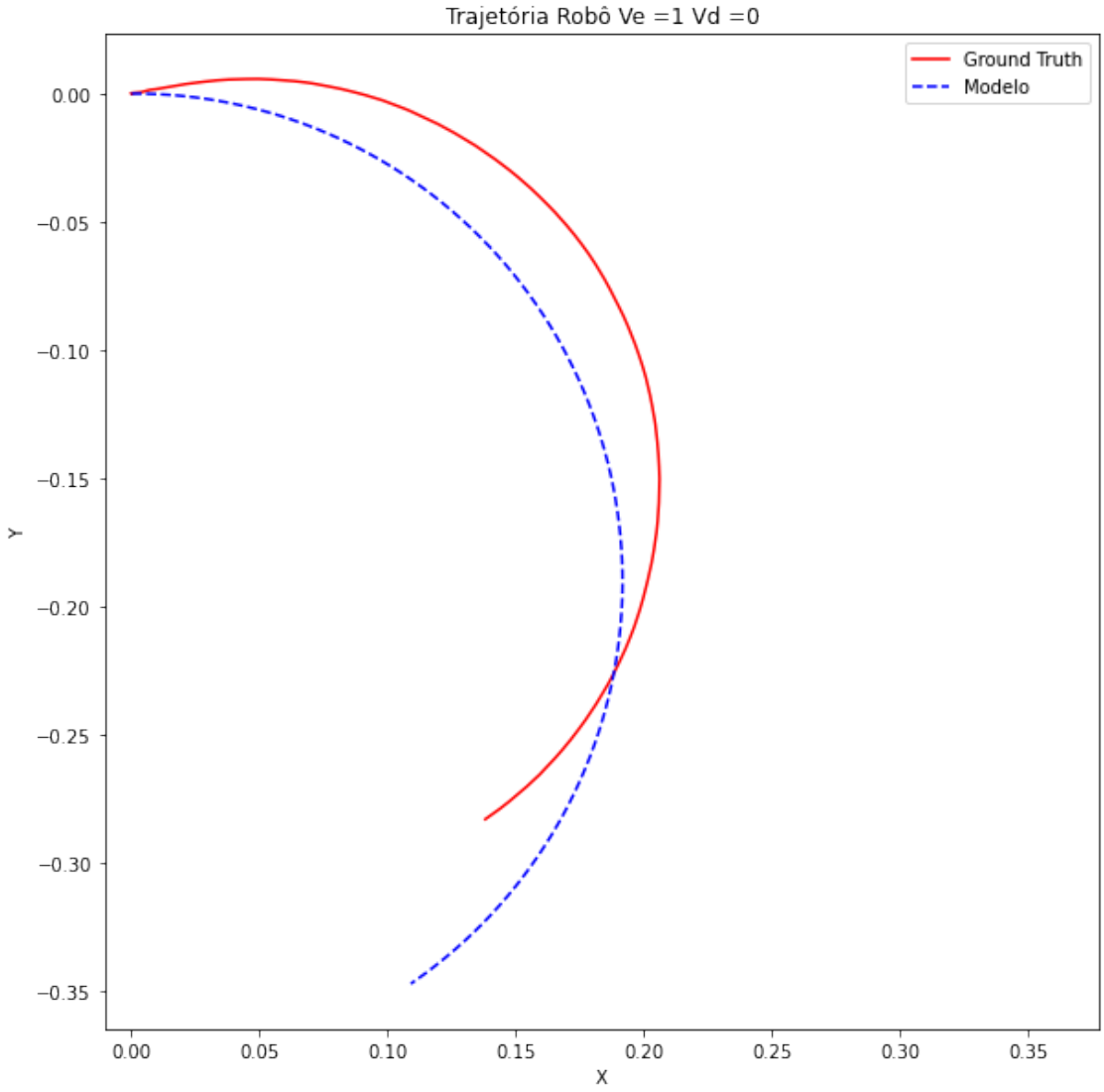


[7]:
```
velocidadeAngularRodaEsquerda = 1
velocidadeAngularRodaDireita = 0
PosicaoRealHistorico =␣
 ↪Simula(velocidadeAngularRodaDireita,velocidadeAngularRodaEsquerda,Passo=0.05)

PosicaoModeloHistorico =␣
 ↪SimulaModeloUm(velocidadeAngularRodaDireita,velocidadeAngularRodaEsquerda,␣
 ↪Passo=0.05)

plt.rcParams["figure.figsize"] = (10,10)

PosicaoRealX = PosicaoRealHistorico[0,:]
```

```
PosicaoRealY = PosicaoRealHistorico[1,:]
AnguloReal = PosicaoRealHistorico[2,:]

plt.plot(PosicaoRealX,PosicaoRealY,color='red', label="Ground Truth")

PosicaoX = PosicaoModeloHistorico[0,:]
PosicaoY = PosicaoModeloHistorico[1,:]
Angulo = PosicaoModeloHistorico[2,:]
plt.plot(PosicaoX,PosicaoY,color='blue',ls='--', label="Modelo")
Title = 'Trajetória Robô Ve ='+ str(velocidadeAngularRodaEsquerda) + ' Vd =' +␣
 ↪str(velocidadeAngularRodaDireita)
plt.title(Title)
plt.xlabel('X')
plt.ylabel('Y')
plt.axis('square')
plt.legend()
```

```
Connected to remote API server
Simulando 99.49748743718592%
Simulation Stopped
Simulation Closed
```

[7]: <matplotlib.legend.Legend at 0x1b1d18717c0>

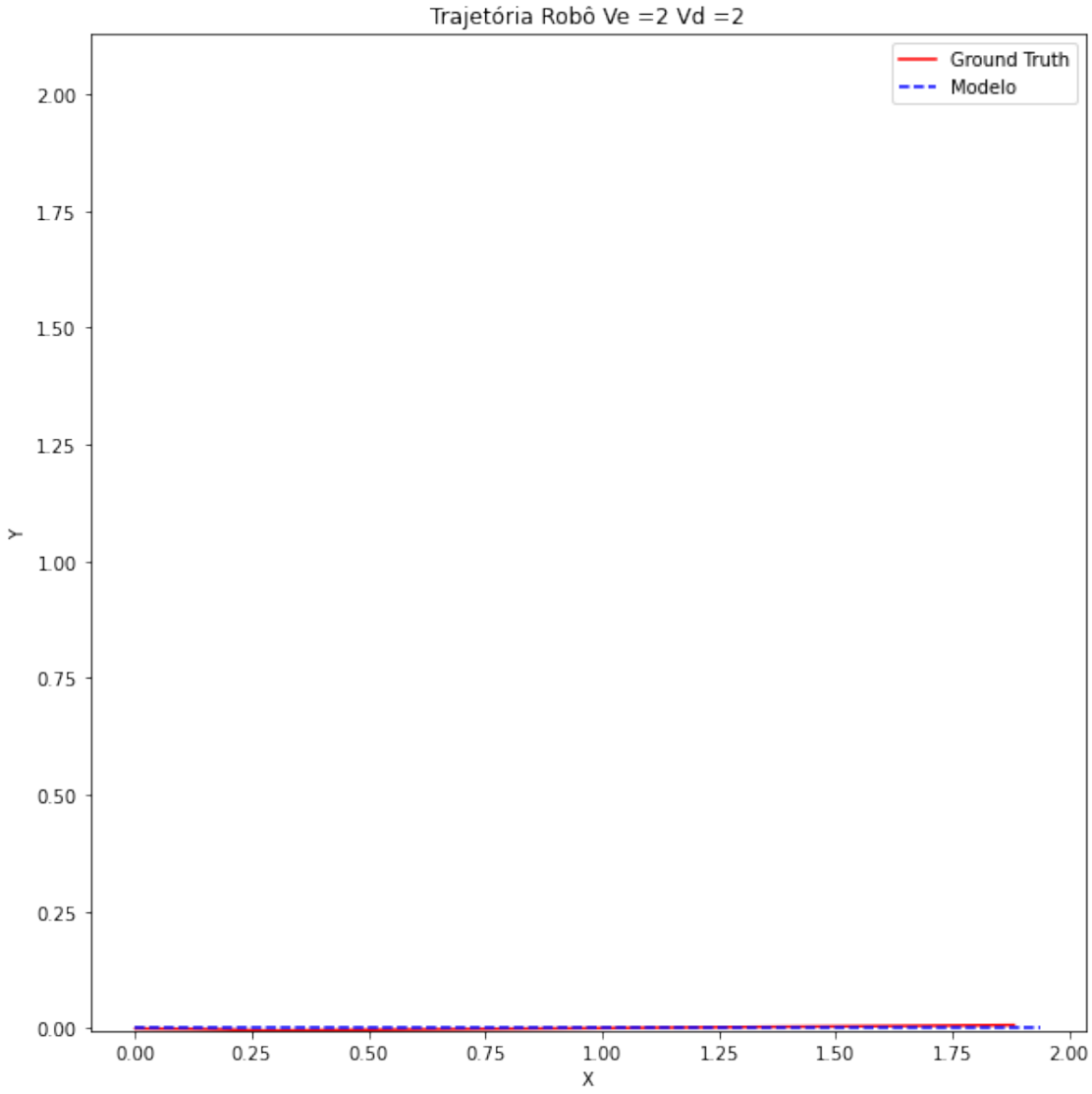


[8]:
```
velocidadeAngularRodaEsquerda = 2
velocidadeAngularRodaDireita = 2
PosicaoRealHistorico =␣
 ↪Simula(velocidadeAngularRodaDireita,velocidadeAngularRodaEsquerda,Passo=0.05)

PosicaoModeloHistorico =␣
 ↪SimulaModeloUm(velocidadeAngularRodaDireita,velocidadeAngularRodaEsquerda,␣
 ↪Passo=0.05)

plt.rcParams["figure.figsize"] = (10,10)

PosicaoRealX = PosicaoRealHistorico[0,:]
PosicaoRealY = PosicaoRealHistorico[1,:]
```

```
AnguloReal = PosicaoRealHistorico[2,:]

plt.plot(PosicaoRealX,PosicaoRealY,color='red', label="Ground Truth")

PosicaoX = PosicaoModeloHistorico[0,:]
PosicaoY = PosicaoModeloHistorico[1,:]
Angulo = PosicaoModeloHistorico[2,:]
plt.plot(PosicaoX,PosicaoY,color='blue',ls='--', label="Modelo")
Title = 'Trajetória Robô Ve ='+ str(velocidadeAngularRodaEsquerda) + ' Vd =' +␣
 ↪str(velocidadeAngularRodaDireita)
plt.title(Title)
plt.xlabel('X')
plt.ylabel('Y')
plt.axis('square')
plt.legend()
```

```
Connected to remote API server
Simulando 99.49748743718592%
Simulation Stopped
Simulation Closed
```

[8]: <matplotlib.legend.Legend at 0x1b1d14aebe0>


Trajetória Robô Ve =2 Vd =2
Ground Truth
Modelo
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
X

[ ]: