



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE  
CENTRO DE TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECATRÔNICA



## 1º RELATÓRIO: Controladores Cinemáticos do Robô Móvel no Simulador

CoppeliaSim 4.1.0 (V-REP)

Natal - RN  
2021

Iago Lucas Batista Galvão  
Jhonat Heberon Avelino de Souza  
Thiago de Araújo Brito

1º Relatório : Controladores Cinemáticos do Robô Móvel no Simulador  
CoppeliaSim 4.1.0 (V-REP)

Terceira e última entrega do 1º Relatório da disciplina  
EGM0007 - Sistemas Robóticos Autônomos do Mestrado  
em Engenharia Mecatrônica da Universidade Federal do  
Rio Grande do Norte (UFRN), referente a nota parcial da  
segunda unidade.

Professor(a): Pablo Javier Alsina.

Natal - RN  
2021

# **Sumário**

	<b>Páginas</b>
<b>1 INTRODUÇÃO</b>	<b>4</b>
<b>2 OBJETIVO</b>	<b>5</b>
<b>3 METODOLOGIA</b>	<b>5</b>
<b>4 SIMULAÇÃO DO MODELO CINEMÁTICO</b>	<b>7</b>
4.1 SIMULAÇÃO . . . . .	7
4.2 RESULTADOS . . . . .	10
<b>5 GERADOR DE CAMINHO</b>	<b>13</b>
<b>6 CONTROLADORES</b>	<b>18</b>
6.1 Seguidor de Caminho e Trajetória . . . . .	18
6.2 Controlador de Posição . . . . .	27
<b>7 CONCLUSÃO</b>	<b>35</b>
<b>REFERÊNCIAS</b>	<b>35</b>

# 1 INTRODUÇÃO

Com o aumento do poder computacional, houve um crescimento no número de pesquisas na área de robótica e, atualmente, o pesquisador é capaz de realizar simulações com qualidade em cenários 2D/3D com várias ferramentas revolucionárias. Os simuladores possibilitam a realização de experimentos, sem a necessidade de construir o *hardware* do robô para desempenhar vários tipos de testes. Como tal desenvolvimento custa caro, esta alternativa possibilita a pesquisa em robôs mais viável e difundida.

Existem muitos *softwares* de simulação robótica disponíveis como, por exemplo, Open HPR, Gazebo, Webots, V-REP etc (1). A plataforma utilizada nesse estudo foi a *Virtual Robot Experimentation Platform* (V-REP), pois atende muitos requisitos, como estrutura de simulação versátil e escalável, arquitetura de controle distribuída, controle por *Script*, *Plugins*, ou API cliente remota entre outras funcionalidades (1), ademais, esta é uma das mais empregadas nos estudos robóticos de todo o mundo.

A versatilidade do simulador contribui para diferentes plataformas robóticas e várias aplicações, não atentando-se somente ao fato de simular robôs com resultados iniciais. Outrossim, isenta o gasto e a necessidade da aquisição do *hardware* para resultados mais realísticos e adaptáveis.

No experimento discutido, foi criado um modelo simples de robô móvel no formato retangular ( $20x40x10\text{ cm}$ ), com três rodas, sendo estas: duas rodas traseiras tracionadas com  $12,5\text{ cm}$  de diâmetro cada e uma roda boba frontal de apoio. Apenas um sensor fora aplicado nesta simulação, o giroscópio e, dois atuadores *joints* (um em cada roda), os quais retornam as respectivas velocidades nos eixos.

Por tratar-se da segunda de três etapas do projeto, nesta fase inicial, não há necessidade do sensor GPS (*Global Positioning System* ou Sistema de Posicionamento Global) nem do giroscópio, para aferir a posição e orientação em relação a um referencial global, respectivamente. É possível medir a velocidade e a orientação do robô, apenas com a leitura dos atuadores nas rodas traseiras, onde retornam em radianos por segundos, as rotações em cada eixo. Assim, sabe-se as rotações de cada motor, logo, é possível medir a velocidade do corpo como um todo; bem como, as curvas realizadas pelas distintas rotações em cada roda, promovendo a orientação e posição do mesmo a um referencial global.

Entretanto, a nível de simplificação da simulação, o giroscópio foi aplicado, pois o *software* permite apenas um ponto a ser lido dos *joints* nas rodas, onde foi escolhido a velocidade. Com isso, a orientação é dada por este sensor e as demais aferições são realizadas a partir das leituras dos atuadores nos eixos. Ressalta-se também, que foram desprezados os erros de atuação, posição, atrito, derrapagem lateral etc. Onde foi determinado um ponto inicial fixo, aplicando as velocidades desejadas como entrada e simulando a movimentação do robô móvel ideal.

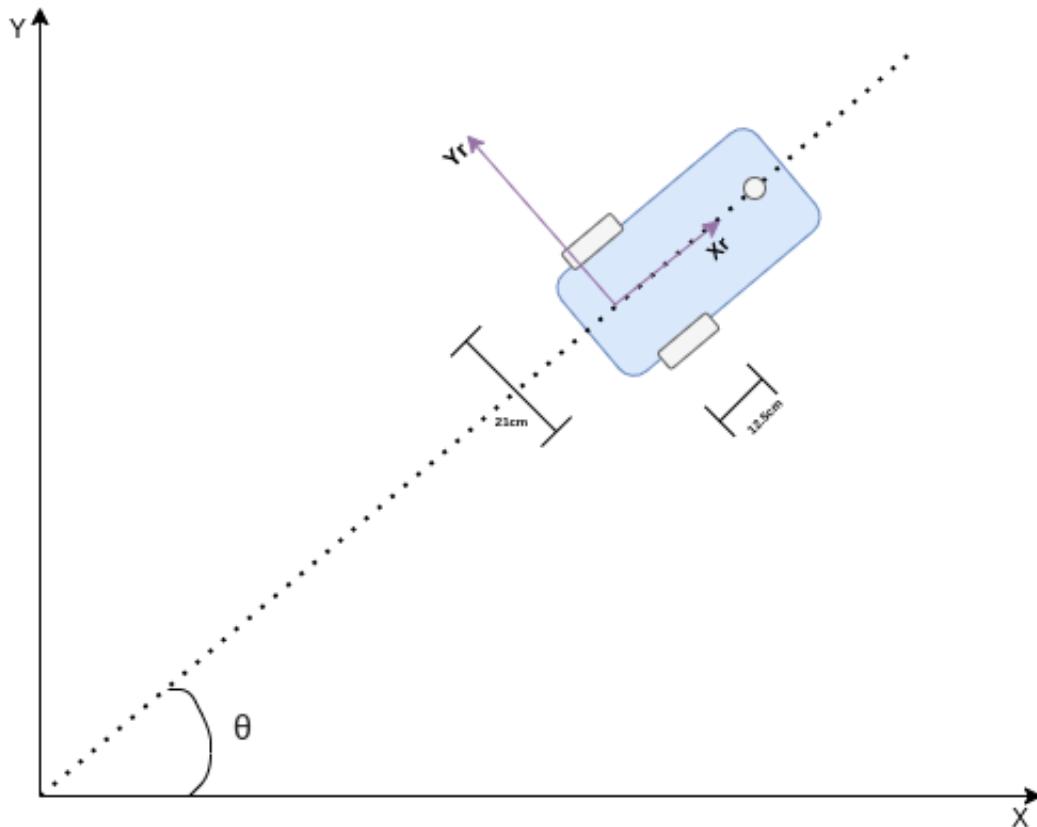
## 2 OBJETIVO

O propósito desta terceira simulação é aplicar os conhecimentos teóricos aprendidos, adotando o modelo cinemático em um robô móvel com acionamento diferencial, onde foram implementados os controladores: seguidor de caminho, seguidor de trajetória e de posição. O robô móvel deve seguir a trajetória gerada e ajustar as velocidades linear e angular de acordo com o posicionamento na curva.

## 3 METODOLOGIA

Primeiramente a uma modelagem não holonômica do robô para projetar o controlador e trajeto foi calculado com os parâmetros do robô. Dessa forma, empregou-se os conceitos aprendidos de modelagem cinemática e não holonômica de robôs, como demonstrada na Figura 1.

**Figura 1.** Modelagem do robô



*Fonte: Autores.*

Roda Direita:

- $\theta_d = -90^\circ$

- $\alpha_d = 180^\circ$

- $l_d = \frac{b}{2}$

Roda Esquerda:

- $\theta_e = 90^\circ$

- $\alpha_e = 0^\circ$

- $l_e = \frac{b}{2}$

Com base nessas informações podemos obter as equações derrapagem lateral e rolamento:

$$\begin{bmatrix} 1 & 0 & \frac{b}{2} \end{bmatrix} * ({}^i R_R(\theta))^T * q' = w_d * r_d \quad (1)$$

$$\begin{bmatrix} 1 & 0 & -\frac{b}{2} \end{bmatrix} * ({}^i R_R(\theta))^T * q' = w_e * r_e \quad (2)$$

Restrição de derrapagem lateral:

$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} * ({}^i R_R(\theta))^T * q' = 0 \quad (3)$$

Ao agruparmos as Equações 1, 2 e 5 teremos nossa modelagem cinemático do robô:

$$\begin{bmatrix} 1 & 0 & \frac{b}{2} \\ 1 & 0 & -\frac{b}{2} \\ 0 & 1 & 0 \end{bmatrix} * ({}^i R_R(\theta))^T * q' = \begin{bmatrix} r_d & 0 \\ 0 & r_e \end{bmatrix} * \begin{bmatrix} w_d \\ w_e \end{bmatrix} \quad (4)$$

para encontrar  $q'$ :

$$\begin{bmatrix} X \\ Y \\ \theta \end{bmatrix} = \begin{bmatrix} \frac{\cos(\theta)*(r_d*w_d + r_e*w_e)}{2} \\ \frac{\sin(\theta)*(r_d*w_d + r_e*w_e)}{2} \\ \frac{(r_d*w_d - r_e*w_e)}{b} \end{bmatrix} \quad (5)$$

onde  $r_d = r_e = 6,25cm$  e  $b = 21cm$ .

Além da posição e orientação, necessita-se do controle das velocidades linear e angular para concretizar o controlador da trajetória a ser seguida. Dessa forma podemos iniciar a modelagem dos controladores, os quais serão discutidos nos próximos tópicos.

O projeto foi dividido em três etapas: a primeira foi a simulação do modelo cinemático, discutido no tópico 4; a segunda etapa foi gerado o caminho baseado em um polinômio de  $3^\circ$ , a qual é demonstrada no tópico 5; e a terceira, implementar os controladores cinemáticos do robô móvel.

Em todas as etapas foram gerados os gráficos da simulação para análise, onde estes, são discutidos nos referentes experimentos, juntamente com os códigos.

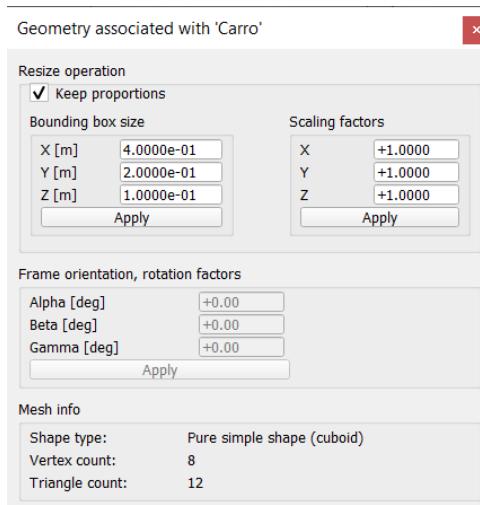
## 4 SIMULAÇÃO DO MODELO CINEMÁTICO

Com as informações do modelo cinemático, podemos implementar esse robô e modelar no **V-REP**, aplicando as dimensões da modelagem para testar a simulação e gerar os gráficos necessários, antes de implementar o controle via *script*.

### 4.1 SIMULAÇÃO

No primeiro experimento, foram inseridas as entradas das velocidades em cada motor, sendo estas, para roda direita  $4 \text{ rad/s}$  e para roda esquerda  $2 \text{ rad/s}$ , os parâmetros definidos estão ilustrados na Figura 2.

**Figura 2.** Entradas no Robô Móvel



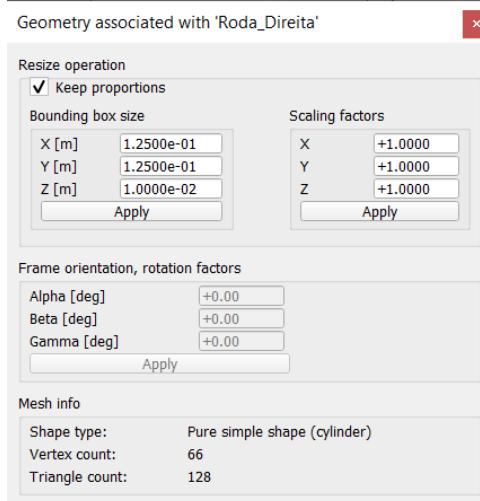
Fonte: Autores.

Similarmente, foi definido os parâmetros para cada roda. Onde são demonstrados nas Figuras 3 e 4.

Como mencionado anteriormente, também é possível fornecer as entradas via *Script*. Assim, o código a seguir fornece as mesmas velocidades, onde são definidos os valores para cada motor, atribuindo a variável a roda específica com os respectivos parâmetros, resultando em uma simulação idêntica.

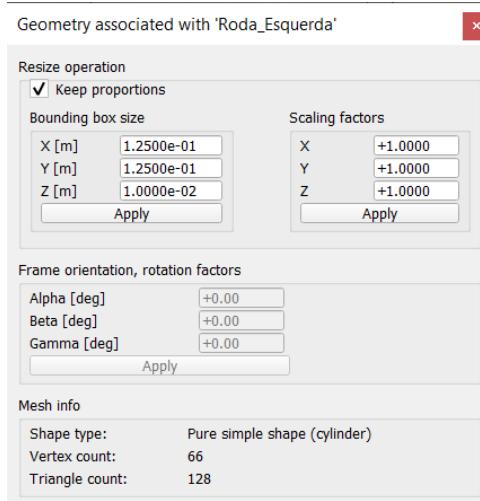
```
1 if (sim_call_type==sim.syscb_init) then
2     motorLeft=sim.getObjectHandle("Motor_Esquerdo")
3     motorRight=sim.getObjectHandle("Motor_Direito")
4 end
5
6 if (sim_call_type==sim.syscb_actuation) then
7     sim.setJointTargetVelocity(motorLeft,2)
8     sim.setJointTargetVelocity(motorRight,4)
9 end
```

**Figura 3.** Entradas na Roda Direita



Fonte: Autores.

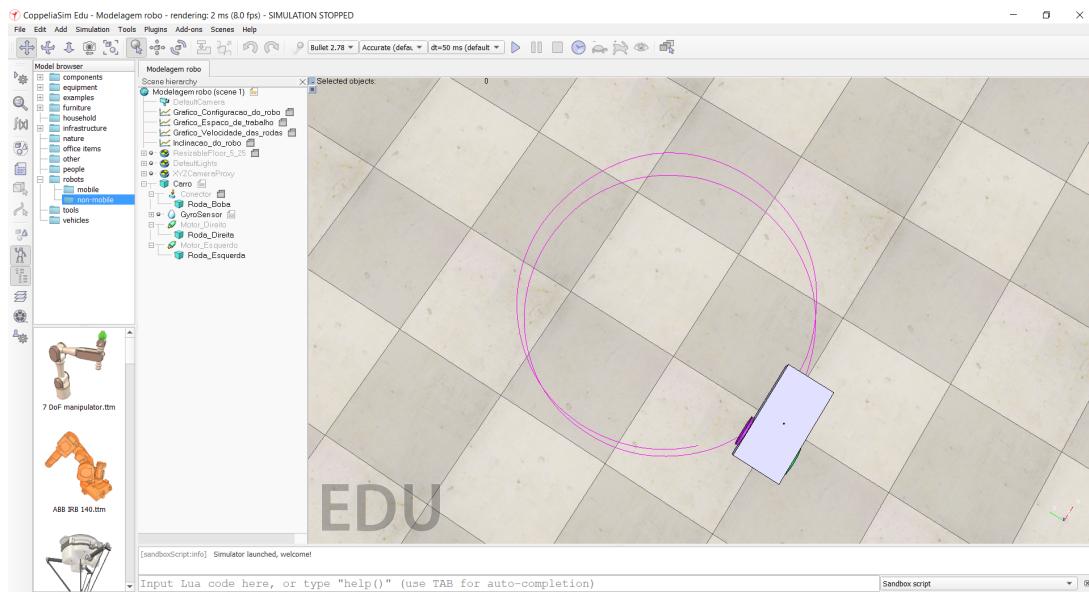
**Figura 4.** Entradas na Roda Esquerda



Fonte: Autores.

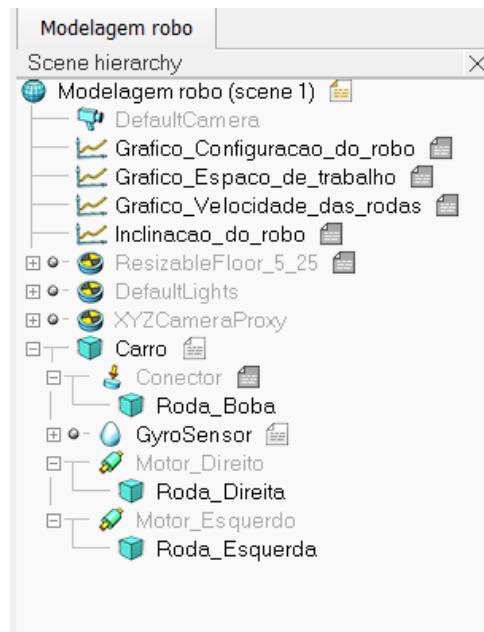
Dessa forma, o robô realiza um movimento circular anti-horário contínuo, como ilustrado na Figura 5. A hierarquização do simulador, é ilustrada na Figura 6.

**Figura 5. Trajetória do Robô**



Fonte: Autores.

**Figura 6. Hierarquização do Simulador**

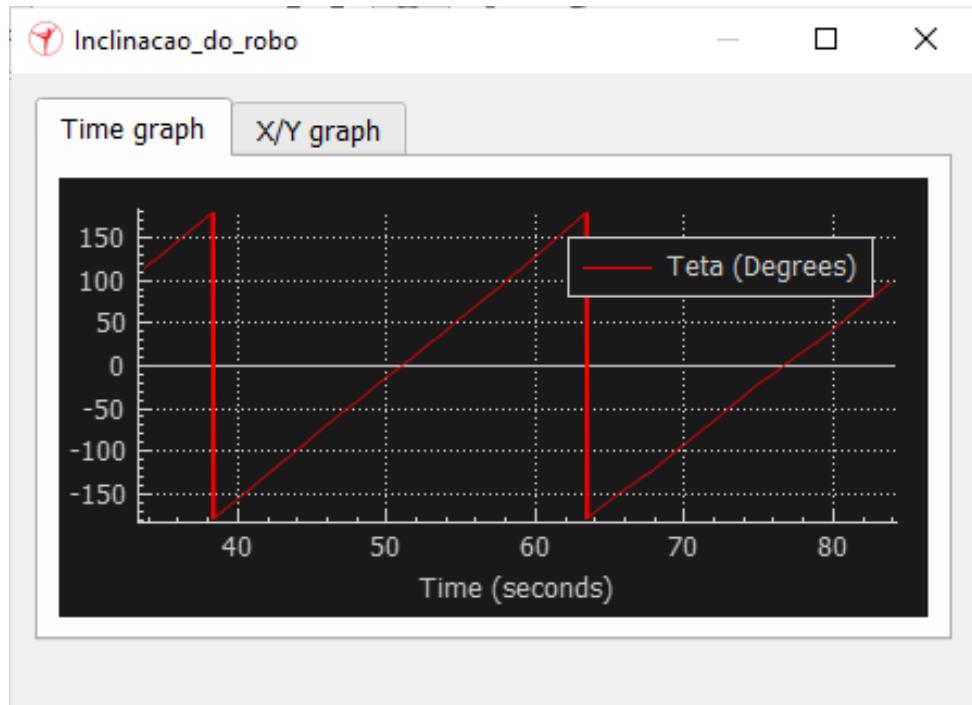


Fonte: Autores.

## 4.2 RESULTADOS

Posteriormente, foram definidas as variáveis a serem medidas e *plotadas* em gráficos. O gráfico da inclinação do robô em graus por segundos está esboçado na Figura 7.

**Figura 7.** Gráfico da Inclinação em Graus do Robô no Tempo

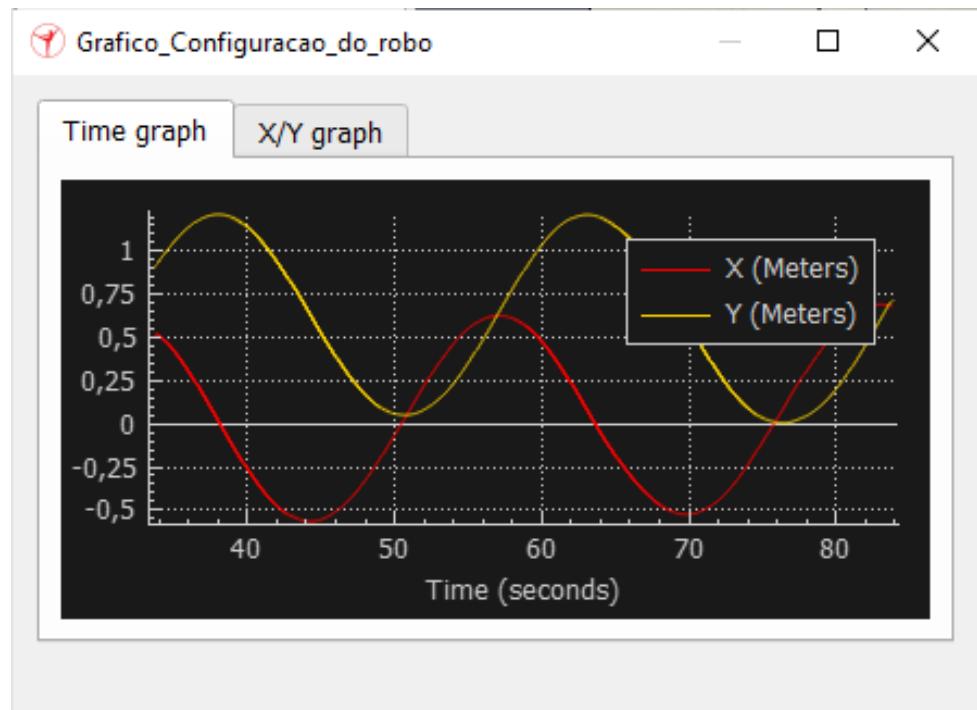


*Fonte: Autores.*

Os gráficos da configuração do robô em metro por segundo e o da velocidade de cada motor em graus por segundos, são esboçados nas Figuras 8 e 9, respectivamente.

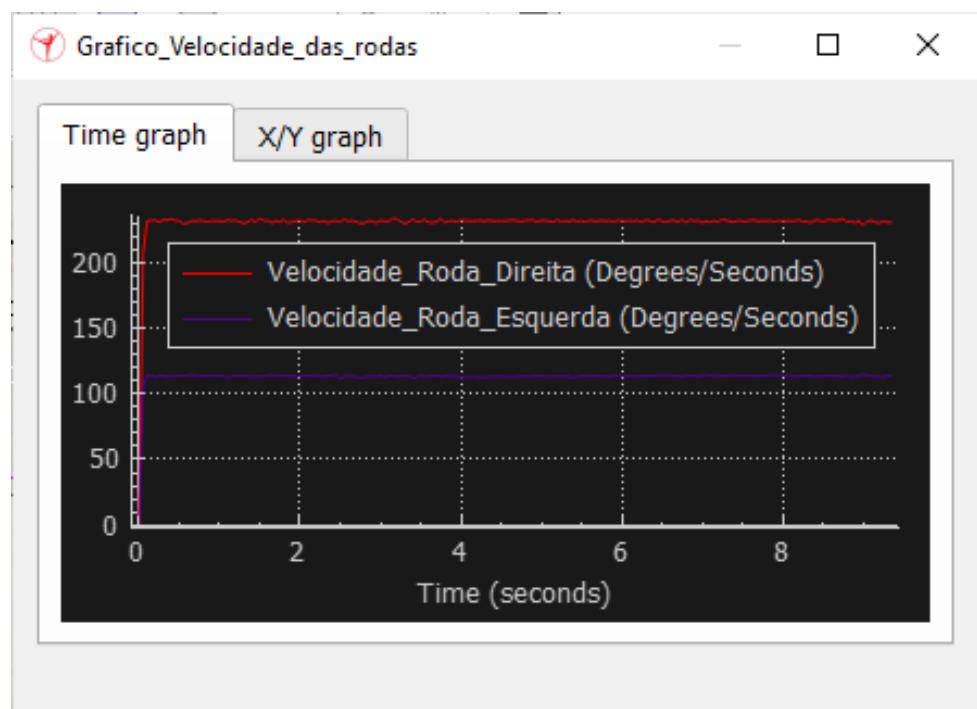
Na Figura 10 é ilustrado o gráfico da trajetória percorrida no plano de trabalho. Por fim, a Figura 11 esboça um *frame* da simulação em andamento com os descritos gráficos em tempo real.

**Figura 8.** Gráfico da Configuração do Robô



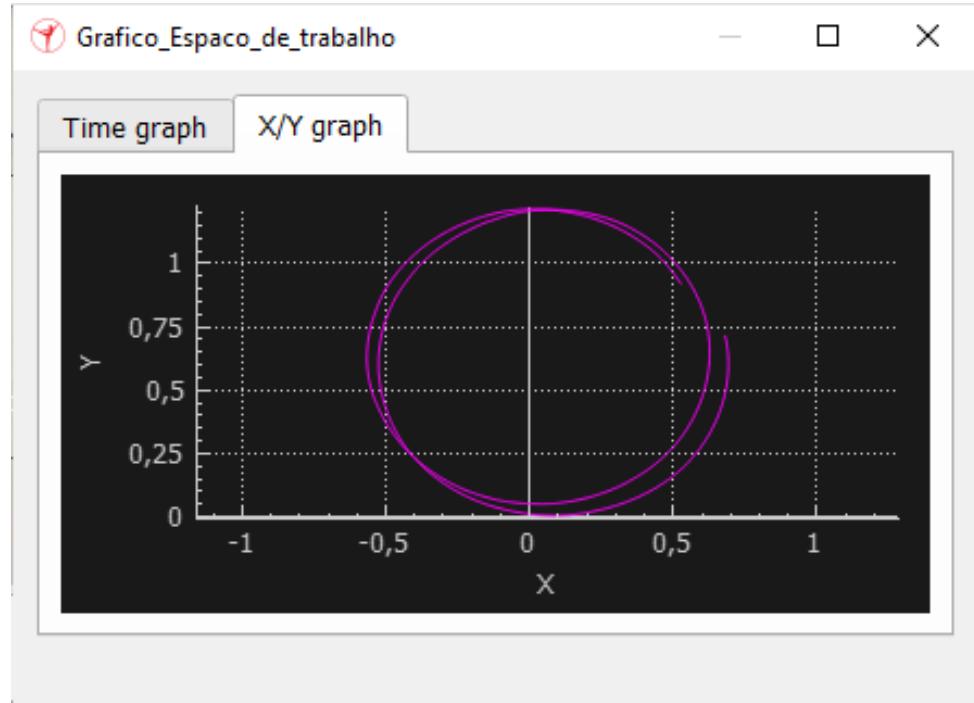
Fonte: Autores.

**Figura 9.** Gráfico da Velocidade dos Motores



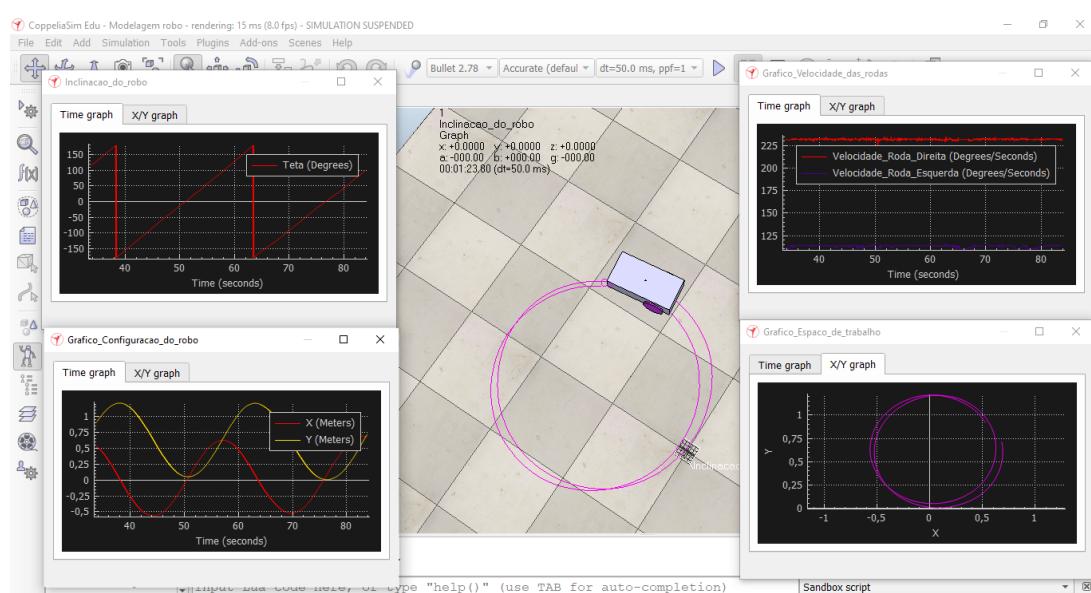
Fonte: Autores.

**Figura 10.** Gráfico da Trajetória Percorrida



Fonte: Autores.

**Figura 11.** Frame da Simulação com os Gráficos



Fonte: Autores.

## 5 GERADOR DE CAMINHO

Similarmente ao tópico anterior, neste segundo experimento foi utilizado para as mesmas condições o gerador de caminho. Contudo, foi aplicado a função seguidor de caminho com a curva interpolada.

Assim, o robô recebe os pontos gerados pela interpolação do polinômio de 3º grau fornecido e com a função "*Path planning*", este segue ponto-a-ponto o caminho traçado com cada posição e orientação predefinidas ( $x$ ,  $y$  e  $\theta$ ).

Abaixo segue o código principal em *Python* para implementação do gerador de caminho.

```
1 # Make sure to have the server side running in CoppeliaSim:
2 # in a child script of a CoppeliaSim scene, add following command
3 # to be executed just once, at simulation start:
4 #
5 # simRemoteApi.start(19999)
6 #
7 # then start simulation, and run this program.
8 #
9 # IMPORTANT: for each successful call to simxStart, there
10 # should be a corresponding call to simxFinish at the end!
11 import math
12 import numpy as np
13 point_init = (-0.0000, -0.0000, 0.1000)
14 point_end = (1.5250, -1.4750, 0.1000)
15
16
17
18
19 try:
20     import sim
21 except:
22     print ('-----')
23     print ('"sim.py" could not be imported. This means very probably that')
24     print ('either "sim.py" or the remoteApi library could not be found.')
25     print ('Make sure both are in the same folder as this file,')
26     print ('or appropriately adjust the file "sim.py"')
27     print ('-----')
28     print ('')
29
30 import time
31 import ctypes
32
33
34 #transform from image frame to vrep frame
```

```

35 def transform_points_from_image2real (points):
36     if points .ndim < 2:
37         flipped = np .flipud(points)
38     else:
39         flipped = np .fliplr(points)
40     scale = 5/445
41     points2send = (flipped*-scale) + np .array ([2.0555+0.75280899,
42                                         -2.0500+4.96629213])
43     return points2send
44
45
46 def send_path_4_drawing(path , sleep_time = 0.07):
47     #the bigger the sleep time the more accurate the points are placed but
48     #you have to be very patient :D
49     for i in path:
50         #point2send = transform_points_from_image2real(i)
51         #print(point2send)
52         #print(type(point2send))
53         packedData=sim .simxPackFloats(i .flatten ())
54         print(packedData)
55         print(type(packedData))
56         raw_bytes = (ctypes .c_ubyte * len(packedData)) .from_buffer_copy(
57             packedData)
58         print(raw_bytes)
59         print(type(raw_bytes))
60         returnCode=sim .simxWriteStringStream(clientID , "path_coord",
61                                         raw_bytes , sim .simx_opmode_oneshot)
62         time .sleep(sleep_time)
63
64
65 def polinomio(point_init , point_end):
66     sigma = 1
67     deltaX = point_end[0] - point_init[0]
68     deltaY = point_end[1] - point_init[1]
69     alfa_init = math .tan (point_init[2])
70     alfa_end = math .tan (point_end[2])
71
72     if point_init[2] >= ((math .pi/2) - sigma) and point_init[2] <= ((math .
73         pi/2) + sigma) and point_end[2] >= ((math .pi/2) - sigma) and
74         point_end[2] >= ((math .pi/2) + sigma):
75         print('i')
76         b1 = deltaY
77         b2 = 0
78         a0 = point_init[0]
79         a1 = 0
80         a2 = 3*deltaX
81         a3 = -2*deltaX
82         b0 = point_init[1]

```

```

76     b3 = deltaY-b1-b2
77
78     elif point_init[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
79         .pi/2) + sigma):
80         print('ii')
81         a3 = -(deltaX/2)
82         b3 = 1
83         a0 = point_init[0]
84         a1 = 0
85         a2 = deltaX-a3
86         b0 = point_init[1]
87         b1 = 2*(deltaY-alfa_end*deltaX) - alfa_end*a3 + b3
88         b3 = (2*alfa_end*deltaX-deltaY) + alfa_end*a3 - 2*b3
89     elif point_end[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
90         .pi/2) + sigma):
91         print('iii')
92         a1 = 3*(deltaX/2)
93         b2 = 1
94         a0 = point_init[0]
95         a2 = 3*deltaX - 2*a1
96         a3 = a1 - 2*deltaX
97         b0 = point_init[1]
98         b1 = alfa_init*a1
99         b3 = deltaY - alfa_init*a1 - b2
100
101     else:
102         print('iv')
103         a1 = deltaX
104         a2 = 0
105         a0 = point_init[0]
106         a3 = deltaX - a1 - a2
107         b0 = point_init[1]
108         b1 = alfa_init*a1
109         b2 = 3*(deltaY-alfa_end*deltaX) + 2*(alfa_end-alfa_init)*a1 +
110             alfa_end*a2
111         b3 = 3*alfa_end*deltaX - 2*deltaY - (2*alfa_end-alfa_init)*a1 -
112             alfa_end*a2
113
114     result = []
115     x = np.arange(0,1,0.01)
116     for i in range(len(x)):
117         fx = a0 + a1*x[i] + a2*x[i]**2 + a3*x[i]**3
118         fy = b0 + b1*x[i] + b2*x[i]**2 + b3*x[i]**3
119         if fx == 0.0 and fy == 0.0:
120             #re = (fx, fy, np.arctan(0))
121             #re = (fx, fy)

```

```

119         re = np.array((fx, fy, np.arctan(0)))
120
121     else:
122         #re = (fx, fy, np.arctan(float(fy/fx)))
123         #re = (fx, fy)
124         re = np.array((fx, fy, np.arctan(float(fy/fx))))
125     #print(re)
126     #print('arc: ',np.arctan(0.9714243279370267))
127     result.append(re)
128
129     return result
130
131 print ('Program started')
132 sim.simxFinish(-1) # just in case, close all opened connections
133 clientID=sim.simxStart('127.0.0.1',19999,True,True,5000,5) # Connect to
    CoppeliaSim
134 if clientID != -1:
135     print ('Connected to remote API server')
136
137     # Now try to retrieve data in a blocking fashion (i.e. a service call):
138     res,objs=sim.simxGetObjects(clientID,sim.sim_handle_all,sim.
        simx_opmode_blocking)
139     if res==sim.simx_return_ok:
140         print ('Number of objects in the scene: ',len(objs))
141     else:
142         print ('Remote API function call returned with error code: ',res)
143
144     time.sleep(2)
145
146     caminho = polinomio(point_init,point_end)
147     send_path_4_drawing(caminho, 0.05)
148
149
150
151     # Now retrieve streaming data (i.e. in a non-blocking fashion):
152     startTime=time.time()
153     err_code,l_motor_handle = sim.simxGetObjectHandle(clientID,
        "Motor_Direito", sim.simx_opmode_blocking)
154     err_code,r_motor_handle = sim.simxGetObjectHandle(clientID,
        "Motor_Esquerdo", sim.simx_opmode_blocking)
155
156     err_code = sim.simxSetJointTargetVelocity(clientID,l_motor_handle,1.0,
        sim.simx_opmode_streaming)
157     err_code = sim.simxSetJointTargetVelocity(clientID,r_motor_handle,2.0,
        sim.simx_opmode_streaming)
158
159

```

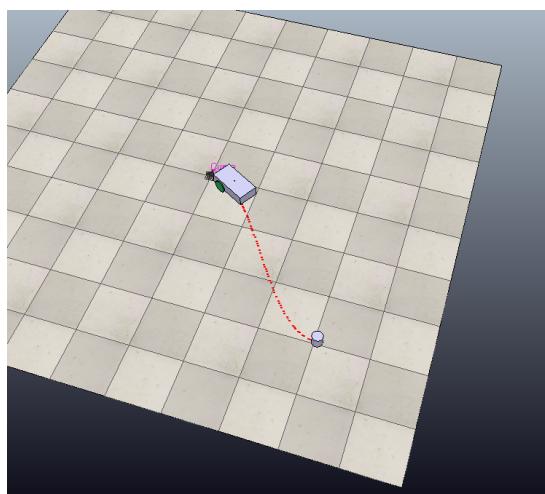
```

160     sim.simxGetIntegerParameter(clientID,sim.sim_intparam_mouse_x,sim.
161         simx_opmode_streaming) # Initialize streaming
162     # while time.time()-startTime < 5:
163     #     returnCode,data=sim.simxGetIntegerParameter(clientID,sim.
164         sim_intparam_mouse_x,sim.simx_opmode_buffer) # Try to retrieve the
165         # streamed data
166     #     if returnCode==sim.simx_return_ok: # After initialization of
167         # streaming , it will take a few ms before the first value arrives , so
168         # check the return code
169     #         print ('Mouse position x: ',data) # Mouse position x is
170         # actualized when the cursor is over CoppeliaSim's window
171     #         time.sleep(0.005)
172
173     # Now send some data to CoppeliaSim in a non-blocking fashion:
174     sim.simxAddStatusbarMessage(clientID,'Hello CoppeliaSim!',sim.
175         simx_opmode_oneshot)
176
177     # Before closing the connection to CoppeliaSim , make sure that the last
178         # command sent out had time to arrive. You can guarantee this with (
179         # for example):
180     sim.simxGetPingTime(clientID)
181
182     # Now close the connection to CoppeliaSim:
183     sim.simxFinish(clientID)
184 else:
185     print ('Failed connecting to remote API server')
186 print ('Program ended')

```

A interpolação do polinômio fornecido foi plotada ponto-a-ponto da origem até o destino. Na Figura 12 ilustra em vermelho o caminho interpolado a ser seguido.

**Figura 12.** *Interpolação do polinômio*



*Fonte: Autores.*

## 6 CONTROLADORES

Nesta seção, são discutidos a implementação de cada controlador do robô móvel. Após gerar o caminho, temos a curva interpolada que é passada para a função seguidor de caminho.

Assim, para definir a trajetória a ser seguida, o controle do caminho é limitado no tempo e o erro da posição e orientação deste é gerado ponto-a-ponto, onde o controle de velocidade em cada roda é ajustado para correção.

A partir dos ganhos proporcionais em cada motor, as velocidades linear e angular são ajustadas de acordo com o tamanho da reta ou raio da curva, respectivamente.

### 6.1 Seguidor de Caminho e Trajetória

No tópico 5 foi gerado o caminho e traçado os pontos a serem seguidos. Para seguir o caminho, o robô deve retornar sua posição atual, para ajustar os erros de posição e orientação em relação ao próximo ponto até que atinja uma distância do destino menor que *0.1cm*.

Abaixo está o código completo do controle para seguir o caminho.

```
1 # Make sure to have the server side running in CoppeliaSim:
2 # in a child script of a CoppeliaSim scene, add following command
3 # to be executed just once, at simulation start:
4 #
5 # simRemoteApi.start(19999)
6 #
7 # then start simulation, and run this program.
8 #
9 # IMPORTANT: for each successful call to simxStart, there
10 # should be a corresponding call to simxFinish at the end!
11 import math
12 import numpy as np
13 point_init = (-0.0000,-0.0000, 0.0000)
14 point_end = (+1.5250e+00, -1.4750e+00, +5.0000e-02)
15 #point_end = (1.5250e+00, 1.8000e+00, 0.0000)
16
17
18
19
20 try:
21     import sim
22 except:
23     print ('-----',
24     )
25     print ('"sim.py" could not be imported. This means very probably that')
26     print ('either "sim.py" or the remoteApi library could not be found.')
27     print ('Make sure both are in the same folder as this file,')
28     print ('or appropriately adjust the file "sim.py"')
```

```

28     print ('-----',
29         )
30     print ('')
31
32 import time
33 import ctypes
34
35
36 def send_path_4_drawing(path, sleep_time = 0.07):
37     #the bigger the sleep time the more accurate the points are placed but
38     #you have to be very patient :D
39     for i in path:
40         #point2send = transform_points_from_image2real(i)
41         #print(point2send)
42         #print(type(point2send))
43         packedData=sim.simxPackFloats(i.flatten())
44         #print(packedData)
45         #print(type(packedData))
46         raw_bytes = (ctypes.c_ubyte * len(packedData)).from_buffer_copy(
47             packedData)
48         #print(raw_bytes)
49         #print(type(raw_bytes))
50         returnCode=sim.simxWriteStringStream(clientID, "path_coord",
51             raw_bytes, sim.simx_opmode_oneshot)
52         time.sleep(sleep_time)
53
54
55 def polinomio(point_init, point_end):
56     sigma = 1
57     deltaX = point_end[0] - point_init[0]
58     deltaY = point_end[1] - point_init[1]
59     alfa_init = math.tan(point_init[2])
60     alfa_end = math.tan(point_end[2])
61
62     if point_init[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
63         pi/2) + sigma) and point_end[2] >= ((math.pi/2) - sigma) and
64         point_end[2] >= ((math.pi/2) + sigma):
65         print('i')
66         b1 = deltaY
67         b2 = 0
68         a0 = point_init[0]
69         a1 = 0
70         a2 = 3*deltaX
71         a3 = -2*deltaX
72         b0 = point_init[1]
73         b3 = deltaY-b1-b2

```

```

69 elif point_init[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
70 .pi/2) + sigma):
71     print('ii')
72     a3 = -(deltaX/2)
73     b3 = 1
74     a0 = point_init[0]
75     a1 = 0
76     a2 = deltaX-a3
77     b0 = point_init[1]
78     b1 = 2*(deltaY-alfa_end*deltaX) - alfa_end*a3 + b3
79     b3 = (2*alfa_end*deltaX-deltaY) + alfa_end*a3 - 2*b3
80 elif point_end[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
81 .pi/2) + sigma):
82     print('iii')
83     a1 = 3*(deltaX/2)
84     b2 = 1
85     a0 = point_init[0]
86     a2 = 3*deltaX - 2*a1
87     a3 = a1 - 2*deltaX
88     b0 = point_init[1]
89     b1 = alfa_init*a1
90     b3 = deltaY - alfa_init*a1 - b2
91 else:
92     print('iv')
93     a1 = deltaX
94     a2 = 0
95     a0 = point_init[0]
96     a3 = deltaX - a1 - a2
97     b0 = point_init[1]
98     b1 = alfa_init*a1
99     b2 = 3*(deltaY-alfa_end*deltaX) + 2*(alfa_end-alfa_init)*a1 +
100        alfa_end*a2
101    b3 = 3*alfa_end*deltaX - 2*deltaY - (2*alfa_end-alfa_init)*a1 -
102        alfa_end*a2
103
104
105    result = []
106    orien = []
107    x = np.arange(0,1,0.01)
108    for i in range(len(x)):
109        fx = a0 + a1*x[i] + a2*x[i]**2 + a3*x[i]**3
110        fy = b0 + b1*x[i] + b2*x[i]**2 + b3*x[i]**3
111        if fx != 0:
112            orientation = np.arctan(float(fy/fx))
113        else:
114            orientation = 0

```

```

112     position = np.array((fx , fy , 0))
113     result.append(position)
114
115     return (result , orientation)
116
117 def coeficientes (origem , destino):
118     sigma=1
119     deltaX = origem[0] - destino[0]
120     deltaY = origem[1] - destino[1]
121     alfa_init = math.tan(point_init[2])
122     alfa_end = math.tan(point_end[2])
123
124     if point_init[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
125         .pi/2) + sigma) and point_end[2] >= ((math.pi/2) - sigma) and
126         point_end[2] >= ((math.pi/2) + sigma):
127         print('i')
128         b1 = deltaY
129         b2 = 0
130         a0 = point_init[0]
131         a1 = 0
132         a2 = 3*deltaX
133         a3 = -2*deltaX
134         b0 = point_init[1]
135         b3 = deltaY-b1-b2
136
137     elif point_init[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
138         .pi/2) + sigma):
139         print('ii')
140         a3 = -(deltaX/2)
141         b3 = 1
142         a0 = point_init[0]
143         a1 = 0
144         a2 = deltaX-a3
145         b0 = point_init[1]
146         b1 = 2*(deltaY-alfa_end*deltaX) - alfa_end*a3 + b3
147         b3 = (2*alfa_end*deltaX-deltaY) + alfa_end*a3 - 2*b3
148
149     elif point_end[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
150         .pi/2) + sigma):
151         print('iii')
152         a1 = 3*(deltaX/2)
153         b2 = 1
154         a0 = point_init[0]
155         a2 = 3*deltaX - 2*a1
156         a3 = a1 - 2*deltaX
157         b0 = point_init[1]
158         b1 = alfa_init*a1
159         b3 = deltaY - alfa_init*a1 - b2

```

```

155     else :
156         print('iv')
157         a1 = deltaX
158         a2 = 0
159         a0 = point_init[0]
160         a3 = deltaX - a1 - a2
161         b0 = point_init[1]
162         b1 = alfa_init*a1
163         b2 = 3*(deltaY-alfa_end*deltaX) + 2*(alfa_end-alfa_init)*a1 +
164             alfa_end*a2
165         b3 = 3*alfa_end*deltaX - 2*deltaY - (2*alfa_end-alfa_init)*a1 -
166             alfa_end*a2
167
168         a = [a0,a1,a2,a3]
169         b = [b0,b1,b2,b3]
170
171         x = np.arange(0,1,0.01)
172         for i in range(len(x)):
173             fx = a0 + a1*x[i], a2*x[i]**2 + a3*x[i]**3
174             fy = b0 + b1*x[i] + b2*x[i]**2 + b3*x[i]**3
175             #if fx != 0:
176                 #orientation = np.arctan(float(fy / fx))
177             #else:
178                 # orientation = 0
179
180         return (a,b)
181
182     print ('Program started')
183     sim.simxFinish(-1) # just in case, close all opened connections
184     clientID=sim.simxStart('127.0.0.1',19999,True,True,5000,5) # Connect to
185         CoppeliaSim
186     if clientID !=-1:
187         print ('Connected to remote API server')
188
189     # Now try to retrieve data in a blocking fashion (i.e. a service call):
190     res,objs=sim.simxGetObjects(clientID,sim.sim_handle_all,sim.
191         simx_opmode_blocking)
192     if res==sim.simx_return_ok:
193         print ('Number of objects in the scene: ',len(objs))
194     else:
195         print ('Remote API function call returned with error code: ',res)
196
197     time.sleep(2)

```

```

198 #####
199 #teste controle estabilizante
200 caminho, orientation = polinomio(point_init, point_end)
201 send_path_4_drawing(caminho, 0.05)
202 #v, w = controlador_posicao(posicao_robo, orientacao_robo)
203
204 #####
205 #teste controle estabilizante
206 #####
207 #Fazendo os handles
208 err_code, motor_direito = sim.simxGetObjectHandle(clientID, "Motor_Direito", sim.simx_opmode_blocking)
209 err_code, motor_esquerdo = sim.simxGetObjectHandle(clientID, "Motor_Esquerdo", sim.simx_opmode_blocking)
210 err_code, carro = sim.simxGetObjectHandle(clientID, "Carro", sim.simx_opmode_blocking)
211
212 #Zerando as velocidades das rodas
213 err_code = sim.simxSetJointTargetVelocity(clientID, motor_direito, 0, sim.simx_opmode_streaming)
214 err_code = sim.simxSetJointTargetVelocity(clientID, motor_esquerdo, 0, sim.simx_opmode_streaming)
215
216 #Posicao e orientacao do robo
217 err_code, posicao_robo = sim.simxGetObjectPosition(clientID, carro, -1, sim.simx_opmode_blocking)
218 err_code, orientacao_robo = sim.simxGetObjectOrientation(clientID, carro, -1, sim.simx_opmode_streaming)
219
220 #Ganhos do controlador
221 k_theta = 2.0
222 k_l = 0.1
223
224 #parametros do robo
225 i = 1
226 v = 0.5
227 d = 0.21 #distancia do eixo entre as rodas
228 rd = 0.0625 #raio roda direita
229 re = 0.0625 #raio roda esquerda
230
231 caminho, orientacao = polinomio(point_init, point_end)
232 lamb = 0
233 while True:
234     #Posicao e orientacao do robo
235     err_code, posicao_robo = sim.simxGetObjectPosition(clientID, carro, -1, sim.simx_opmode_blocking)
236     err_code, orientacao_robo = sim.simxGetObjectOrientation(clientID, carro,

```

```

        , -1 , sim . simx_opmode_streaming )

237
238     theta_robo = orientacao_robo [2] + math . pi /2
239
240     #raio de giro
241     a,b = coeficientes ( point_init , point_end )
242     dx = a [1] + 2*a [2]*lamb + 3*a [3]*(lamb**2)
243     dy = b [1] + 2*b [2]*lamb + 3*b [3]*(lamb**2)
244     d2x = 2*a [2] + 6*a [3]*lamb
245     d2y = 2*b [2] + 6*b [3]*lamb
246     r = (((dx **2)+(dy **2)) **1.5 )/((d2y*dx)-(d2x*dy)))
247     k = (1/r)

248
249     #delta theta
250     theta_SF = math . atan ((b [1] + 2*b [2]*lamb + 3*b [3]*lamb**2)/(a [1] + 2*
251                         a [2]*lamb + 3*a [3]*lamb**2))
252     Delta_theta = theta_robo - theta_SF
253
254     #garantir o sinal correto de delta L
255     theta_posicao_robo = math . atan2 (posicao_robo [1] , posicao_robo [0])
256     delta_L = np . linalg . norm (posicao_robo - caminho [0])
257     ponto_curva = caminho [0]
258     for i in range (len (caminho) -1):
259         distance = np . linalg . norm (posicao_robo - caminho [i+1])
260         if (delta_L > distance):
261             delta_L = distance
262             ponto_curva = caminho [i+1]
263     theta_ref = math . atan2 (ponto_curva [1] , ponto_curva [0])

264     if (theta_ref > theta_posicao_robo):
265         delta_L = -delta_L
266
267     i = i +1

268     u = -(k_theta*Delta_theta + (k_l*delta_L*v*math . sin (Delta_theta) /
269             Delta_theta))

270     w = u + ((k*v*math . cos (Delta_theta))/(1-(k*delta_L)))
271
272     wd = (v / rd) + (d/(2*rd))*w
273     we = (v / re) - (d/(2*re))*w
274
275     err_code = sim . simxSetJointTargetVelocity (clientID , motor_direito , wd ,
276             sim . simx_opmode_streaming )
277     err_code = sim . simxSetJointTargetVelocity (clientID , motor_esquerdo , we ,
278             sim . simx_opmode_streaming )
279     print ('lamb : ' , lamb)

```

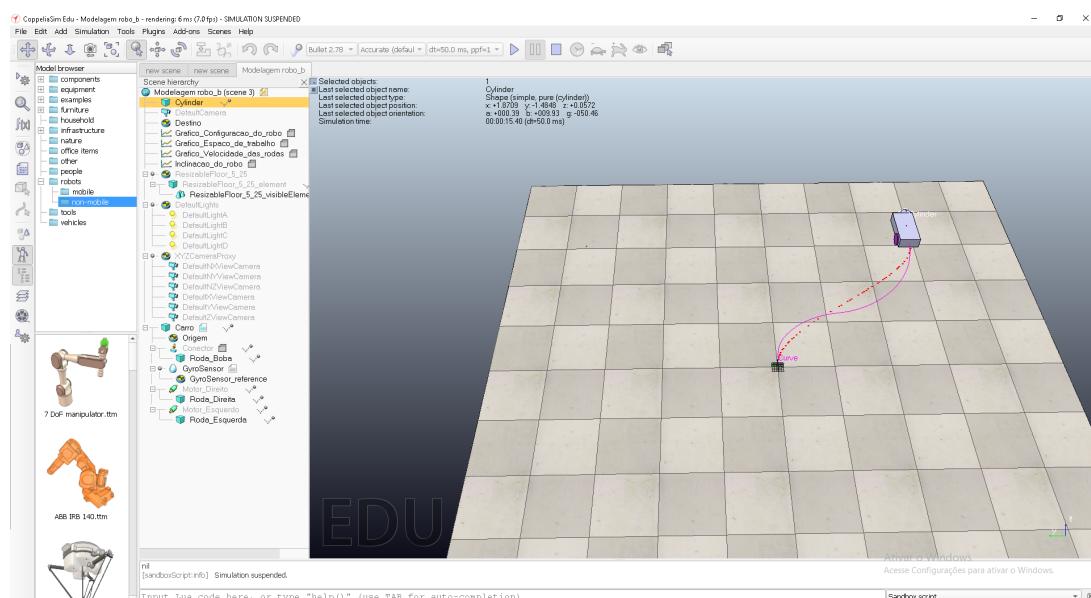
```

279     if (lamb >= 1):
280         err_code = sim.simxSetJointTargetVelocity(clientID, motor_direito
281             , 0.0, sim.simx_opmode_streaming)
282         err_code = sim.simxSetJointTargetVelocity(clientID, motor_esquerdo
283             , 0.0, sim.simx_opmode_streaming)
284         break
285
286     lamb = lamb + 0.01
287
288     err_code = sim.simxSetJointTargetVelocity(clientID, motor_direito, 0.0,
289         sim.simx_opmode_streaming)
290     err_code = sim.simxSetJointTargetVelocity(clientID, motor_esquerdo, 0.0,
291         sim.simx_opmode_streaming)
292     print ('Finalizado seguidor de caminho')
293
294     # Now close the connection to CoppeliaSim:
295     sim.simxFinish(clientID)
296 else:
297     print ('Failed connecting to remote API server')
298 print ('Program ended')

```

A trajetória é baseada no tempo, assim, as velocidades das rodas podem ser definidas baseadas nos erros. Em outras palavras, a aceleração é definida da relação do ponto atual ao ponto a ser seguido. A Figura 13 ilustra o *frame* da simulação ao atingir o destino desejado.

**Figura 13.** Trajetória percorrida pelo robô móvel

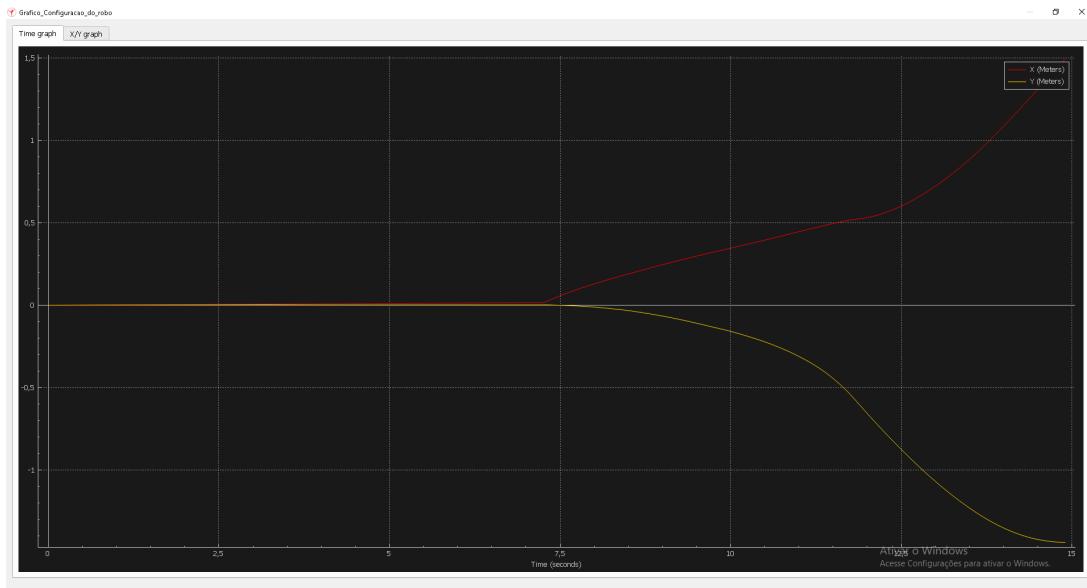


*Fonte:* Autores.

Os gráficos abaixo foram gerados para analisar o comportamento da função e uma melhor compreensão da rotina de simulação e ajustes dos ganhos. A configuração do robô para X e Y é

demonstrada na Figura 14.

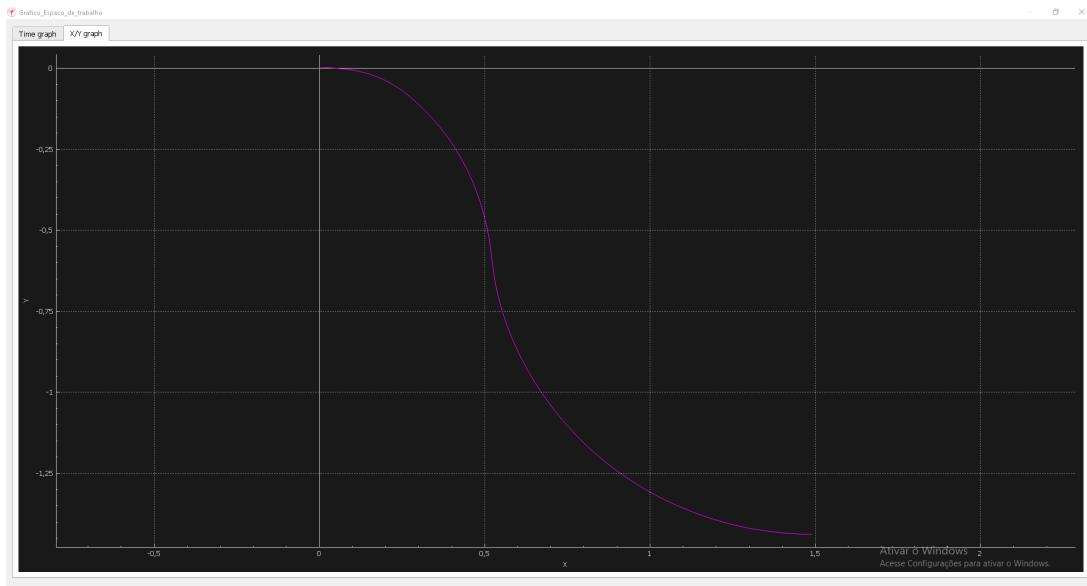
**Figura 14.** Configuração do robô móvel para o plano XY



Fonte: Autores.

O caminho seguido, aproxima-se da curva interpolada, onde este está demonstrado na Figura 15.

**Figura 15.** Trajetória percorrida no plano XY



Fonte: Autores.

## 6.2 Controlador de Posição

Como mencionado anteriormente de forma indireta, para o carro seguir a trajetória, necessita-se do controle de posição e orientação. Este controle é ajustado pela variação do tamanho da reta e do raio de curvatura do ângulo, ambos em relação ao ponto/destino a ser alcançado.

Dessa maneira, os erros de posição e orientação do robô são minimizados com o aumento ou redução da velocidade na roda. O direcionamento deste é realizado pelas aplicações de velocidades distintas em cada motor, onde a parte frontal é inclinada de forma a reduzir o ângulo do eixo central do robô ao ponto a ser seguido.

Abaixo está o código completo do controle de posição/orientação.

```
1 # Make sure to have the server side running in CoppeliaSim:
2 # in a child script of a CoppeliaSim scene, add following command
3 # to be executed just once, at simulation start:
4 #
5 # simRemoteApi.start(19999)
6 #
7 # then start simulation, and run this program.
8 #
9 # IMPORTANT: for each successful call to simxStart, there
10 # should be a corresponding call to simxFinish at the end!
11 import math
12 import numpy as np
13
14
15 point_init = (-0.0000,-0.0000, 0.0000)
16 point_end = (1.5250e+00, -1.4750e+00, 0.0000)
17 #Ganhos do controlador
18 k_theta = 0.1;
19 k_l = 0.1;
20 #Robot model
21 d = 0.21 #wheel axis distance
22 r_w = 0.0625 #wheel radius
23
24 try:
25     import sim
26 except:
27     print ('-----',
28           )
29     print ('"sim.py" could not be imported. This means very probably that')
30     print ('either "sim.py" or the remoteApi library could not be found.')
31     print ('Make sure both are in the same folder as this file,')
32     print ('or appropriately adjust the file "sim.py"')
33     print ('-----',
34           )
35     print ('')
```

```

34
35 import time
36 import ctypes
37
38
39 #transform from image frame to vrep frame
40 def transform_points_from_image2real (points):
41     if points.ndim < 2:
42         flipped = np.flipud(points)
43     else:
44         flipped = np.fliplr(points)
45     scale = 5/445
46     points2send = (flipped*-scale) + np.array([2.0555+0.75280899,
47         -2.0500+4.96629213])
48     return points2send
49
50
51 def send_path_4_drawing(path, sleep_time = 0.07):
52     #the bigger the sleep time the more accurate the points are placed but
53     #you have to be very patient :D
54     for i in path:
55         packedData=sim.simxPackFloats(i.flatten())
56         raw_bytes = (ctypes.c_ubyte * len(packedData)).from_buffer_copy(
57             packedData)
58         resultCode=sim.simxWriteStringStream(clientID , "path_coord",
59             raw_bytes , sim.simx_opmode_oneshot)
60         time.sleep(sleep_time)
61
62
63 def polinomio(point_init, point_end):
64     sigma = 1
65     deltaX = point_end[0] - point_init[0]
66     deltaY = point_end[1] - point_init[1]
67     alfa_init = math.tan(point_init[2])
68     alfa_end = math.tan(point_end[2])
69
70     if point_init[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
71         pi/2) + sigma) and point_end[2] >= ((math.pi/2) - sigma) and
72         point_end[2] >= ((math.pi/2) + sigma):
73         print('i')
74         b1 = deltaY
75         b2 = 0
76         a0 = point_init[0]
77         a1 = 0
78         a2 = 3*deltaX
79         a3 = -2*deltaX
80         b0 = point_init[1]
81         b3 = deltaY-b1-b2

```

```

75
76     elif point_init[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
77         .pi/2) + sigma):
78         print('ii')
79         a3 = -(deltaX/2)
80         b3 = 1
81         a0 = point_init[0]
82         a1 = 0
83         a2 = deltaX-a3
84         b0 = point_init[1]
85         b1 = 2*(deltaY-alfa_end*deltaX) - alfa_end*a3 + b3
86         b3 = (2*alfa_end*deltaX-deltaY) + alfa_end*a3 - 2*b3
87     elif point_end[2] >= ((math.pi/2) - sigma) and point_init[2] <= ((math.
88         .pi/2) + sigma):
89         print('iii')
90         a1 = 3*(deltaX/2)
91         b2 = 1
92         a0 = point_init[0]
93         a2 = 3*deltaX - 2*a1
94         a3 = a1 - 2*deltaX
95         b0 = point_init[1]
96         b1 = alfa_init*a1
97         b3 = deltaY - alfa_init*a1 - b2
98     else:
99         print('iv')
100        a1 = deltaX
101        a2 = 0
102        a0 = point_init[0]
103        a3 = deltaX - a1 - a2
104        b0 = point_init[1]
105        b1 = alfa_init*a1
106        b2 = 3*(deltaY-alfa_end*deltaX) + 2*(alfa_end-alfa_init)*a1 +
107            alfa_end*a2
108        b3 = 3*alfa_end*deltaX - 2*deltaY - (2*alfa_end-alfa_init)*a1 -
109            alfa_end*a2
110
111    result = []
112    orien = []
113    x = np.arange(0,1,0.01)
114    for i in range(len(x)):
115        fx = a0 + a1*x[i] + a2*x[i]**2 + a3*x[i]**3
116        fy = b0 + b1*x[i] + b2*x[i]**2 + b3*x[i]**3
117        if fx != 0:
118            orientation = np.arctan(float(fy/fx))
119        else:

```

```

118         orientation = 0
119         position = np.array((fx, fy, 0))
120         result.append(position)
121
122     return (result, orientation)
123
124 def controlador_posicao(delta_x, delta_y, orientacao):
125     Kr = 0.6
126     kdelta = 0.1
127     ksigma = 0.1
128     R = (delta_x**2+delta_y**2)**(1/2)
129     delta = np.arctan((delta_y/delta_x))
130     sigma = delta + orientacao
131     v = Kr*R*math.cos(delta)
132     w = kdelta*delta + (Kr*(delta + ksigma*sigma)*(math.sin(delta)*math.cos(delta)))/delta
133     return v,w
134
135
136 print ('Program started')
137 sim.simxFinish(-1) # just in case, close all opened connections
138 clientID=sim.simxStart('127.0.0.1',19999,True,True,5000,5) # Connect to
139     CoppeliaSim
140 if clientID != -1:
141     print ('Connected to remote API server')
142
143     # Now try to retrieve data in a blocking fashion (i.e. a service call):
144     res,objs=sim.simxGetObjects(clientID,sim.sim_handle_all,sim.
145         simx_opmode_blocking)
146     if res==sim.simx_return_ok:
147         print ('Number of objects in the scene: ',len(objs))
148     else:
149         print ('Remote API function call returned with error code: ',res)
150
151
152
153
154     # Now retrieve streaming data (i.e. in a non-blocking fashion):
155     startTime=time.time()
156     err_code,l_motor_handle = sim.simxGetObjectHandle(clientID, "
157         Motor_Direito", sim.simx_opmode_blocking)
158     err_code,r_motor_handle = sim.simxGetObjectHandle(clientID, "
159         Motor_Esquerdo", sim.simx_opmode_blocking)
160
161     err_code, posicao_robo = sim.simxGetObjectHandle(clientID, "Origem", sim.

```

```

        simx_opmode_blocking)
160    err_code , posicao_robo = sim.simxGetObjectPosition(clientID , posicao_robo
161        ,-1 , sim.simx_opmode_blocking)

162    err_code , destino_robo = sim.simxGetObjectHandle(clientID , "Destino" , sim
163        .simx_opmode_blocking)
164    err_code , destino_robo = sim.simxGetObjectPosition(clientID , destino_robo
165        ,-1 , sim.simx_opmode_blocking)

166    err_code , orientacao_robo = sim.simxGetObjectHandle(clientID , "GyroSensor"
167        , sim.simx_opmode_blocking)
168    err_code , orientacao_robo = sim.simxGetObjectOrientation(clientID ,
169        orientacao_robo ,-1 , sim.simx_opmode_blocking)
170    print ('posicao robo:', posicao_robo)
171    print ('Destino robo:', destino_robo)
172    print ('orientacao do robo:', orientacao_robo)

173    caminho , orientation = polinomio(point_init , point_end)
174    send_path_4_drawing(caminho , 0.05)

175    ######
176    #teste controle estabilizante
177    while True:
178        err_code , posicao_robo = sim.simxGetObjectHandle(clientID , "Origem" ,
179            sim.simx_opmode_blocking)
180        err_code , posicao_robo = sim.simxGetObjectPosition(clientID ,
181            posicao_robo ,-1 , sim.simx_opmode_blocking)

182        err_code , orientacao_robo = sim.simxGetObjectHandle(clientID ,
183            "GyroSensor" , sim.simx_opmode_blocking)
184        err_code , orientacao_robo = sim.simxGetObjectOrientation(clientID ,
185            orientacao_robo ,-1 , sim.simx_opmode_blocking)

186        theta_robo = orientacao_robo[2]
187        print("theta_robo: " , theta_robo)

188        if (theta_robo > math.pi):
189            theta_robo = theta_robo + 2*math.pi
190        elif (theta_robo < -math.pi):
191            theta_robo = theta_robo - 2*math.pi
192        #calculo dos delta x,y
193        delta_x = point_end[0] - posicao_robo[0]
194        delta_y = point_end[1] - posicao_robo[1]

```

```

197     #if (theta_robo < 0 ):
198     #    theta_robo = theta_robo + math.pi/2
199
200     #theta_estrela/referencial
201     theta_ref = np.arctan(delta_y/delta_x)
202     print("theta_ref: ", theta_ref)
203     #calculo do delta_l/referencial , delta_theta
204     delta_l_ref = np.sqrt((delta_x)**2 + (delta_y)**2)
205     delta_theta = theta_ref - theta_robo
206
207     if (delta_theta > math.pi):
208         delta_theta = delta_theta - 2*math.pi
209     elif (delta_theta < -math.pi):
210         delta_theta = delta_theta + 2*math.pi
211     #calculo do delta_l
212
213     delta_l = delta_l_ref*np.cos(delta_theta)
214
215     v = k_l*delta_l
216     w = k_theta*delta_theta
217     #v, w = controlador_posicao(delta_x, delta_y, delta_theta)
218     wd = (v/r_w) + (d/(2*r_w))*w
219     we = (v/r_w) - (d/(2*r_w))*w
220
221     err_code = sim.simxSetJointTargetVelocity(clientID,l_motor_handle,we,
222                                              sim.simx_opmode_streaming)
222     err_code = sim.simxSetJointTargetVelocity(clientID,r_motor_handle,wd,
223                                              sim.simx_opmode_streaming)
223
224     if delta_l_ref <= 0.1:
225         break
226
227     #teste controle estabilizante
228 ##########
229
230     err_code = sim.simxSetJointTargetVelocity(clientID,l_motor_handle,0,sim
231                                              .simx_opmode_streaming)
231     err_code = sim.simxSetJointTargetVelocity(clientID,r_motor_handle,0,sim
232                                              .simx_opmode_streaming)
233
233     sim.simxGetIntegerParameter(clientID,sim.sim_intparam_mouse_x,sim.
234                                 simx_opmode_streaming) # Initialize streaming
234
235     # Now send some data to CoppeliaSim in a non-blocking fashion:
236     sim.simxAddStatusbarMessage(clientID,'Hello CoppeliaSim!',sim.
237                                 simx_opmode_oneshot)

```

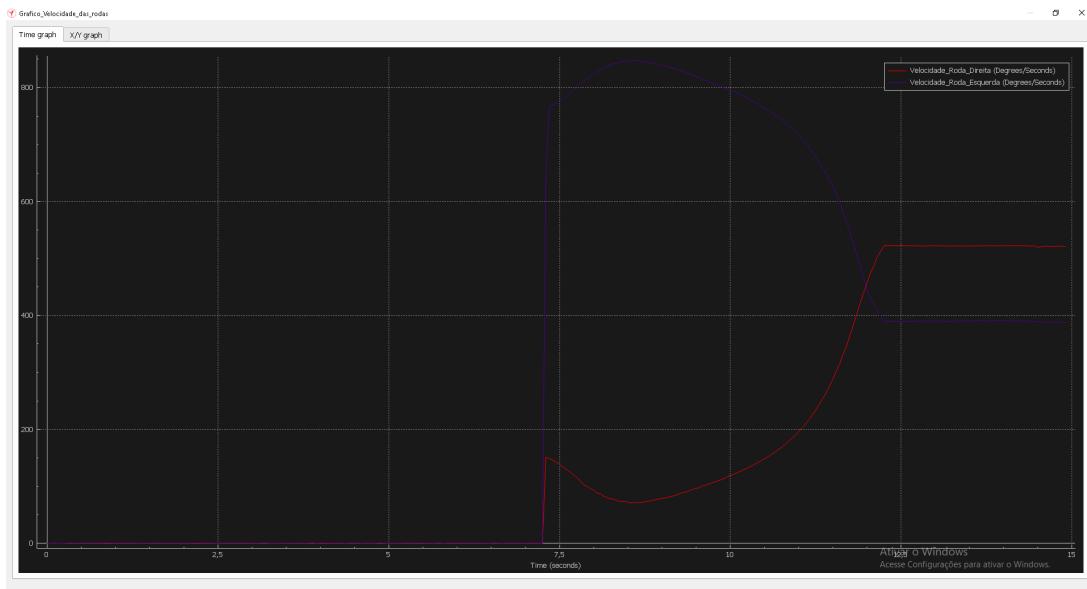
```

238 # Before closing the connection to CoppeliaSim, make sure that the last
239     command sent out had time to arrive. You can guarantee this with (
240         for example):
241     sim.simxGetPingTime(clientID)
242
243     # Now close the connection to CoppeliaSim:
244     sim.simxFinish(clientID)
245 else:
246     print ('Failed connecting to remote API server')
247 print ('Program ended')

```

A Figura 16 representa o gráfico da velocidade de cada rodas do robô por toda trajetória percorrida.

**Figura 16.** Velocidades das rodas

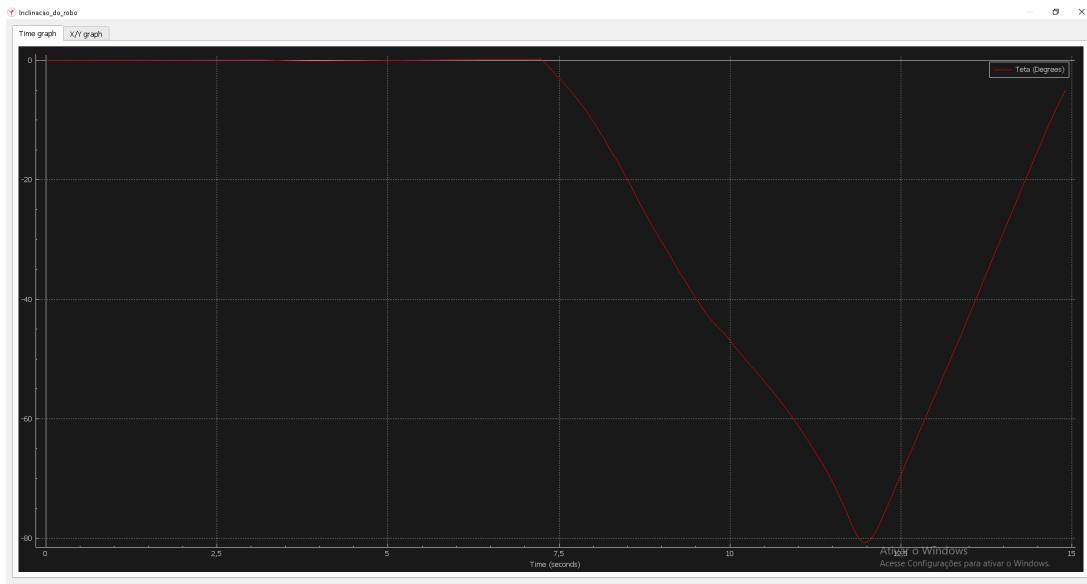


*Fonte: Autores.*

Por fim, as Figuras 17 e 18 representam os gráficos de inclinação do carro em relação ao ponto de destino e a configuração do robô para o volume *XYZ*, respectivamente. Onde o controlador deve ajustar cada velocidade com a posição e orientação atual do carro até o destino final.

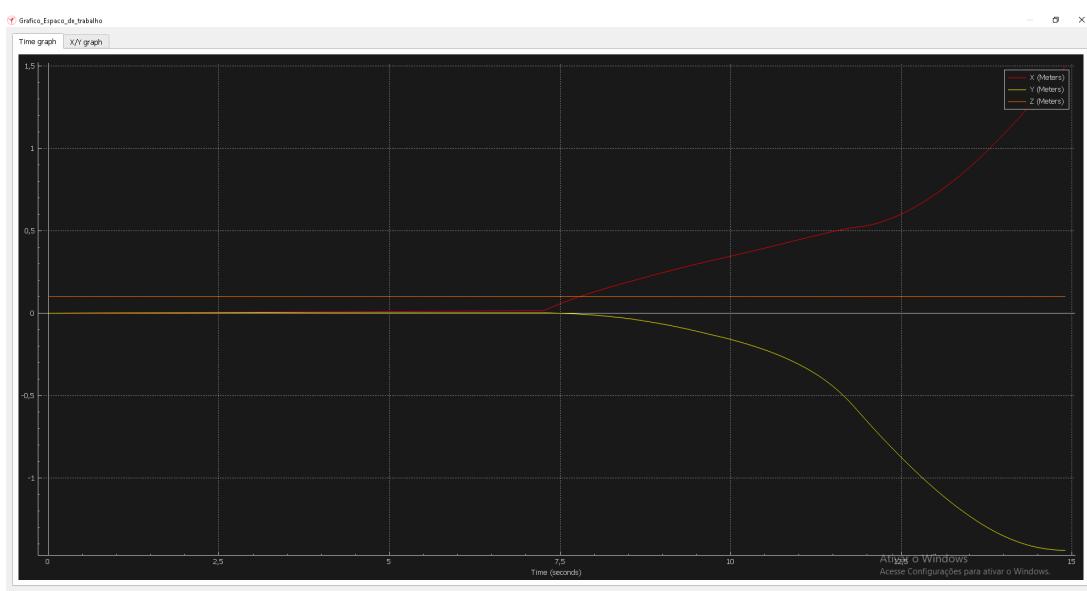
A curva (em rosa) gerada pelo caminho seguido, pode ser ajustada pelos ganhos linear e angular. Onde tal ajuste influênciaria na precisão da trajetória seguida, a qual deve ser a mais próxima e suave da curva interpolada possível.

**Figura 17.** Inclinação  $\theta$  do robô móvel



Fonte: Autores.

**Figura 18.** Configuração XYZ do robô móvel



Fonte: Autores.

## 7 CONCLUSÃO

Com a inovação de simuladores como este, pesquisadores de todos os níveis podem realizar suas pesquisas sem a necessidade física do dispositivo robótico. Assim, reduzindo ao máximo os possíveis erros de programação e custos, antes da aplicação no robô físico.

A partir desta, também, facilitou a compreensão da teoria mostrada em aula, principalmente, nos casos de escassez ou ausência dos materiais necessários para tal finalidade. Ademais, destaca-se a importância dessas ferramentas simulatórias, para o caso atual de pandemia, onde as aulas são remotas.

O controle da trajetória foi implementado com sucesso, porém necessita de ajustes finos para suavizar mais as curvas e aumentar a velocidade, sem perder a precisão do controle de posição.

Analizando os gráficos demonstrados, é notória melhoria na percepção do comportamento de um robô móvel com o ajuste dos ganhos. Com uma melhor clareza, percebe-se que para uma simples trajetória, o controle requer muito processamento para receber e enviar remotamente, as informações de comando para locomoção e estado posição/orientação atual do robô.

Como o processamento é feito em tempo real, os gráficos obtidos pela simulação são muito próximos ao comportamento em um experimento com *hardware* de predefinições similares e em boas condições estruturais. Assim, a certeza nas análises destes é mais pontual e econômica. Outrossim, é a experiência proporcionada em um simulador utilizado tanto no âmbito acadêmico, como comercial.

Finalmente, o experimento resultou nas proposições desejadas e este serve de base para os próximos, os quais serão inseridos obstáculos. Estes serão evitados, onde o caminho gerado deverá encontrar o melhor trajeto para o destino final, sem nenhuma colisão.

## REFERÊNCIAS

- [1] ROHMER, E.; SINGH, S. P. N.; FREESE, M. V-rep: A versatile and scalable robot simulation framework. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.: s.n.], 2013. p. 1321–1326. 4
- [2] NASCIMENTO, L. B. P. et al. Introdução ao v-rep: Uma plataforma virtual para simulação de robôs. In: *Alex Oliveira Barradas Filho; Pedro Porfirio Muniz Farias; Ricardo de Andrade Lira Rabêlo. (Org.). Minicursos da ERCEMAPI e EAComp 2019. 1ed.* Porto Alegre: Sociedade Brasileira de Computação. [S.l.: s.n.], 2019. p. 49–68.