



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



**1º RELATÓRIO: Implementar gerador de caminho baseado em
polinômios interpoladores de 3º grau para robô móvel**
CoppeliaSim 4.1.0 (V-REP)

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

1º Relatório: Implementar gerador de caminho baseado em polinômios
interpoladores de 3º grau para robô móvel
CoppeliaSim 4.1.0 (V-REP)

Segunda 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.

Sumário

	Páginas
1 INTRODUÇÃO	4
2 OBJETIVO	5
3 METODOLOGIA	5
4 SIMULAÇÃO DO MODELO CINEMÁTICO	6
4.1 SIMULAÇÃO	7
4.2 RESULTADOS	10
5 GERADOR DE CAMINHO	14
6 CONCLUSÃO	20
7 REFERÊNCIAS	20

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. Além disso, 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 ($20 \times 40 \times 10$ cm), com três rodas, sendo estas: duas rodas traseiras tracionadas com $12,5$ 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.

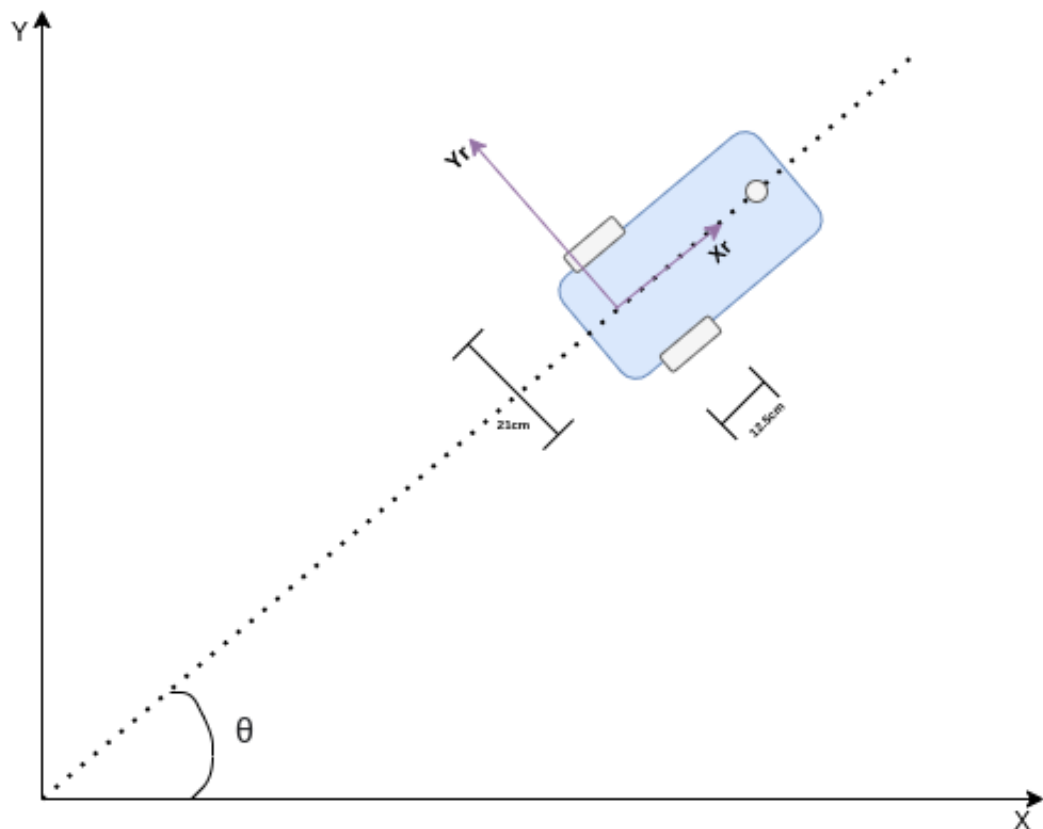
2 OBJETIVO

O propósito desta simulação é aplicar os conhecimentos teóricos aprendidos, adotando o modelo cinemático em um robô móvel com acionamento diferencial, onde foi implementado um programa gerador de caminho, baseado em um polinômio interpolado de 3°. O robô móvel deve seguir a curva; bem como, mostrar na tela o gerador de caminho durante toda a simulação.

3 METODOLOGIA

Foi realizado primeiramente a uma modelagem não holonômica do robô para projetar o controlador e trajeto. 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:

- $\theta_d = -90^\circ$
- $\alpha_d = 180^\circ$

- $l_d = b/2$

Roda Esquerda:

- $\theta_e = 90^\circ$
- $\alpha_e = 0^\circ$
- $l_e = b/2$

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

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

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

Restrição de derrapagem lateral:

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

Ao agruparmos as Equações 1, 2 e 3 teremos nossa modelagem cinemática do robô:

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

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 $b = 21cm$.

Dessa forma podemos modelar um controlador que queremos qual a coordenado e orientação do robô no mundo, e passamos o velocidade angular das rodas podemos ter a posição orientação.

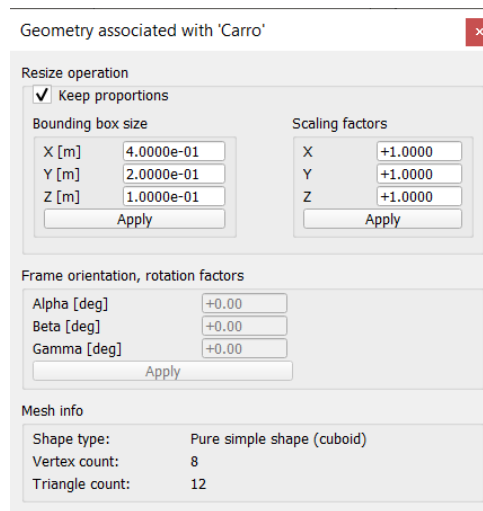
4 SIMULAÇÃO DO MODELO CINEMÁTICO

Com as informações do modelo cinemático, podemos implementar esse robô e modelar no **V-REP**, aplicando as dimensões da modelagem.

4.1 SIMULAÇÃO

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

Figura 2. Entradas no Robô Móvel



Geometry associated with 'Carro'

Resize operation

☒ Keep proportions

Bounding box size

X [m]	4.0000e-01
Y [m]	2.0000e-01
Z [m]	1.0000e-01

Apply

Scaling factors

X	+1.0000
Y	+1.0000
Z	+1.0000

Apply

Frame orientation, rotation factors

Alpha [deg]	+0.00
Beta [deg]	+0.00
Gamma [deg]	+0.00

Apply

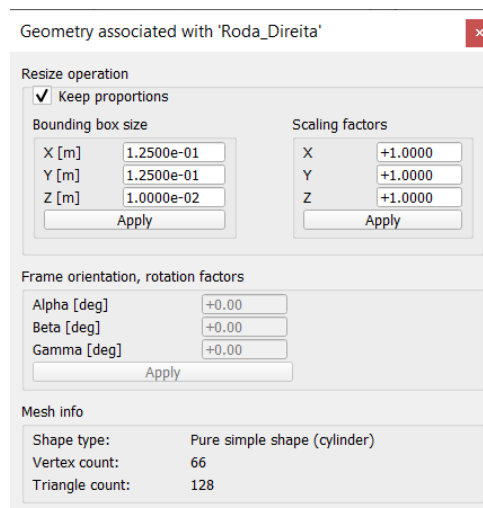
Mesh info

Shape type:	Pure simple shape (cuboid)
Vertex count:	8
Triangle count:	12

Fonte: Autores.

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

Figura 3. Entradas na Roda Direita



Geometry associated with 'Roda_Direita'

Resize operation

☒ Keep proportions

Bounding box size

X [m]	1.2500e-01
Y [m]	1.2500e-01
Z [m]	1.0000e-02

Apply

Scaling factors

X	+1.0000
Y	+1.0000
Z	+1.0000

Apply

Frame orientation, rotation factors

Alpha [deg]	+0.00
Beta [deg]	+0.00
Gamma [deg]	+0.00

Apply

Mesh info

Shape type:	Pure simple shape (cylinder)
Vertex count:	66
Triangle count:	128

Fonte: Autores.

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

Figura 4. Entradas na Roda Esquerda

Geometry associated with 'Roda_Esquerda'

Resize operation

☒ Keep proportions

Bounding box size

X [m] 1.2500e-01

Y [m] 1.2500e-01

Z [m] 1.0000e-02

Apply

Scaling factors

X +1.0000

Y +1.0000

Z +1.0000

Apply

Frame orientation, rotation factors

Alpha [deg] +0.00

Beta [deg] +0.00

Gamma [deg] +0.00

Apply

Mesh info

Shape type: Pure simple shape (cylinder)

Vertex count: 66

Triangle count: 128

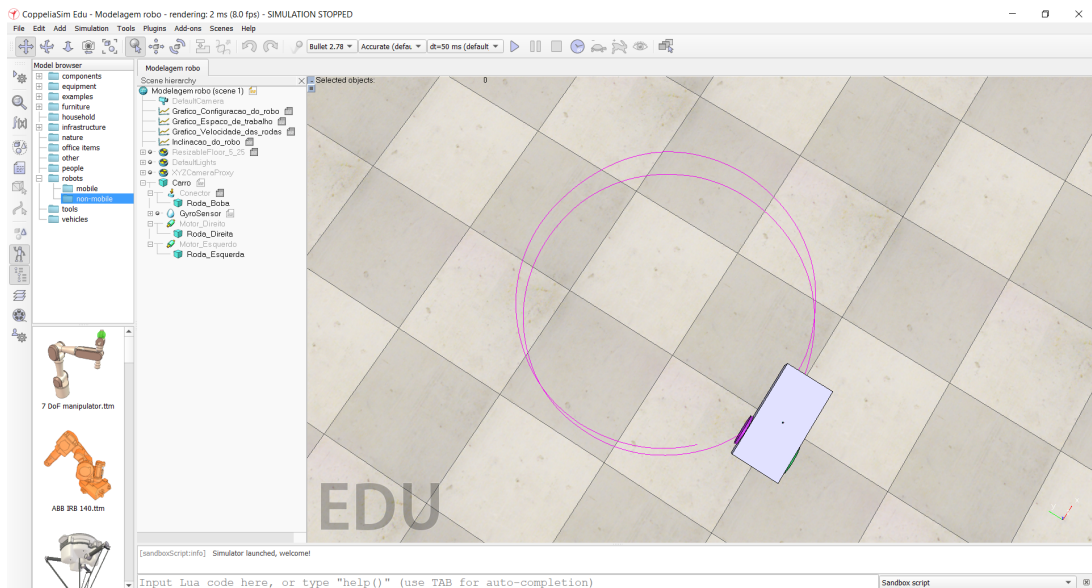
Fonte: Autores.

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
```

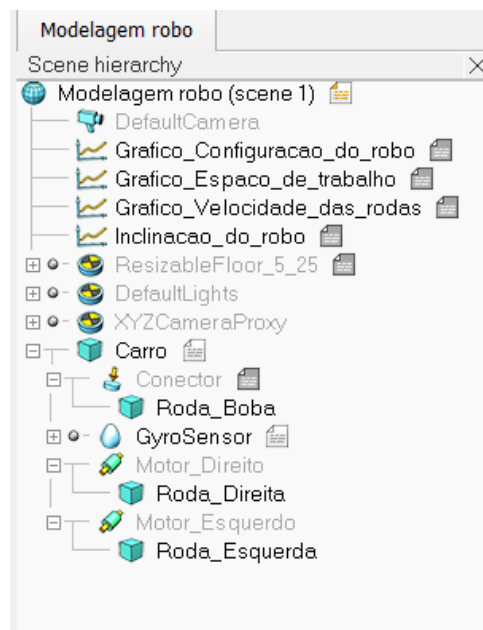
Dessa forma, o robô realiza um movimento circular anti-horário contínuo, como ilustrado na Figura 5. A hierarquização do simulador, é ilustrada na Figura 6.

Figura 5. Trajetória do Robô



Fonte: Autores.

Figura 6. Hierarquização do Simulador

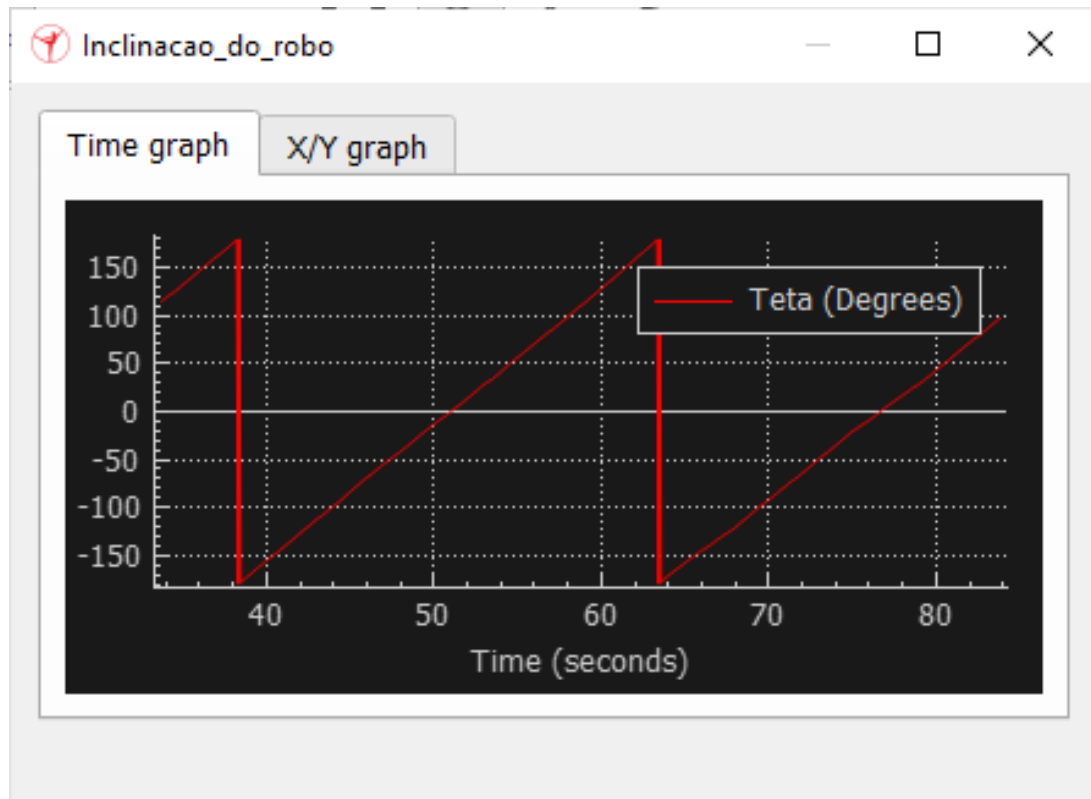


Fonte: Autores.

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

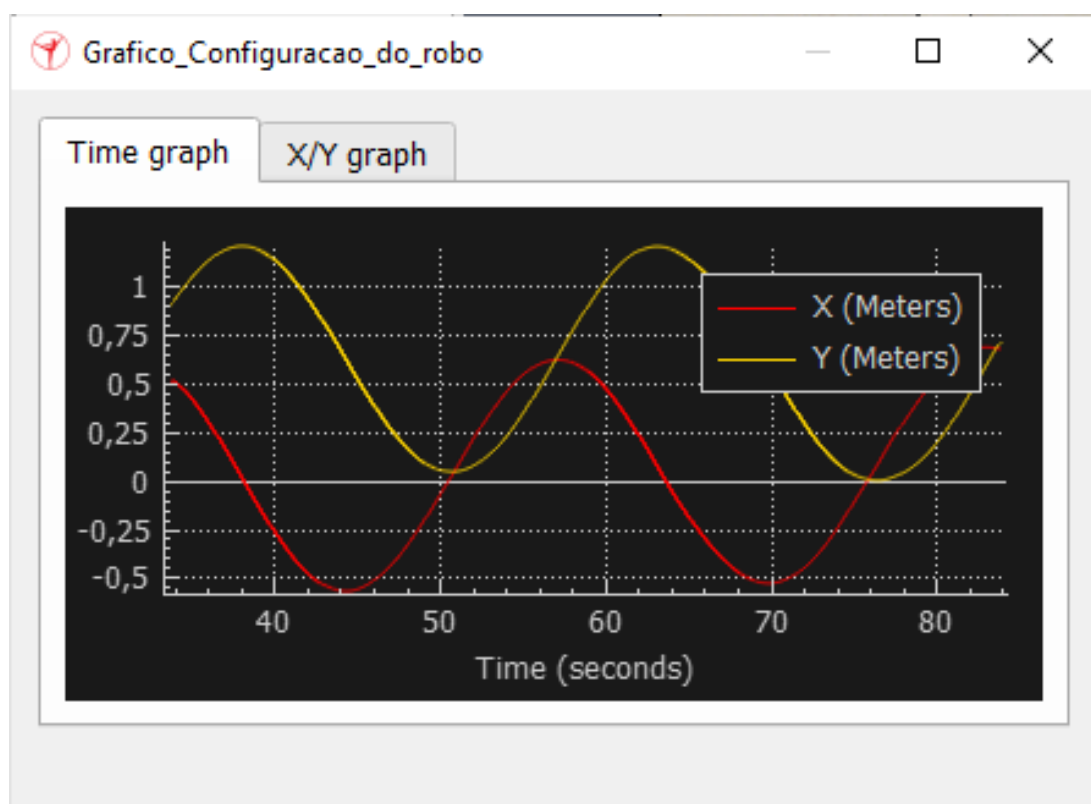


Fonte: Autores.

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.

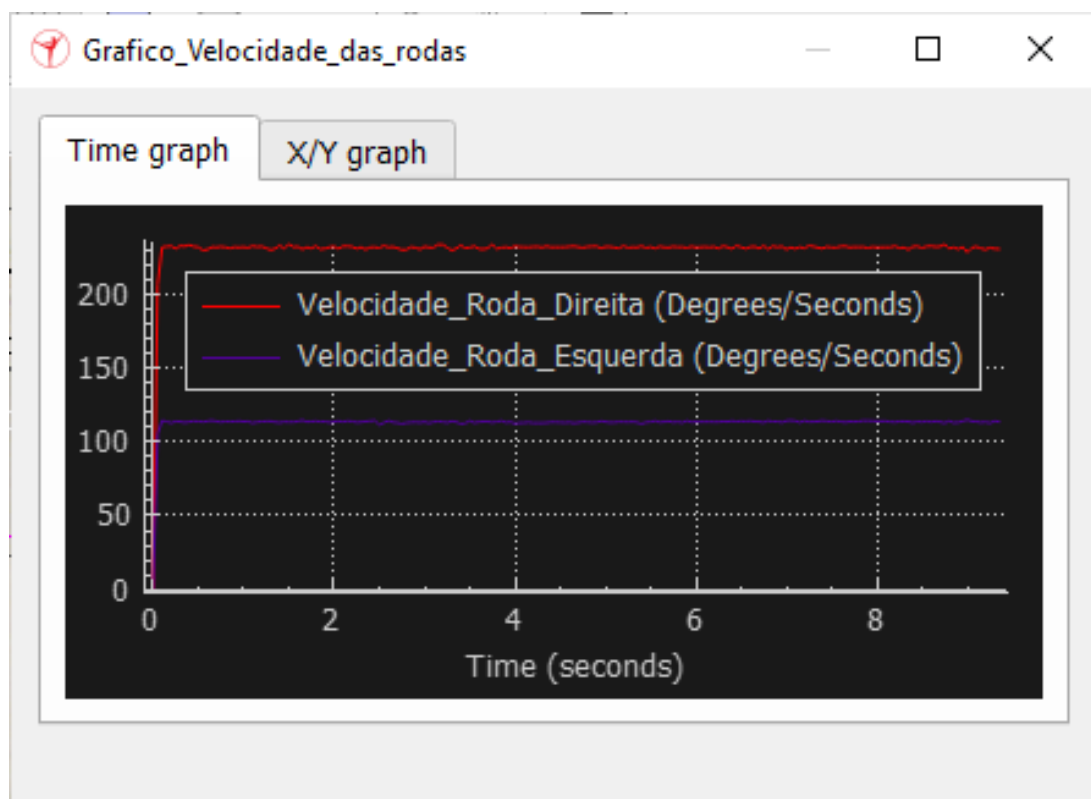
Na Figura 10 é ilustrado o gráfico da trajetória percorrida no plano de trabalho. Por fim, a Figura 11 esboça um *frame* da simulação em andamento com os descritos gráficos em tempo real.

Figura 8. *Gráfico da Configuração do Robô*



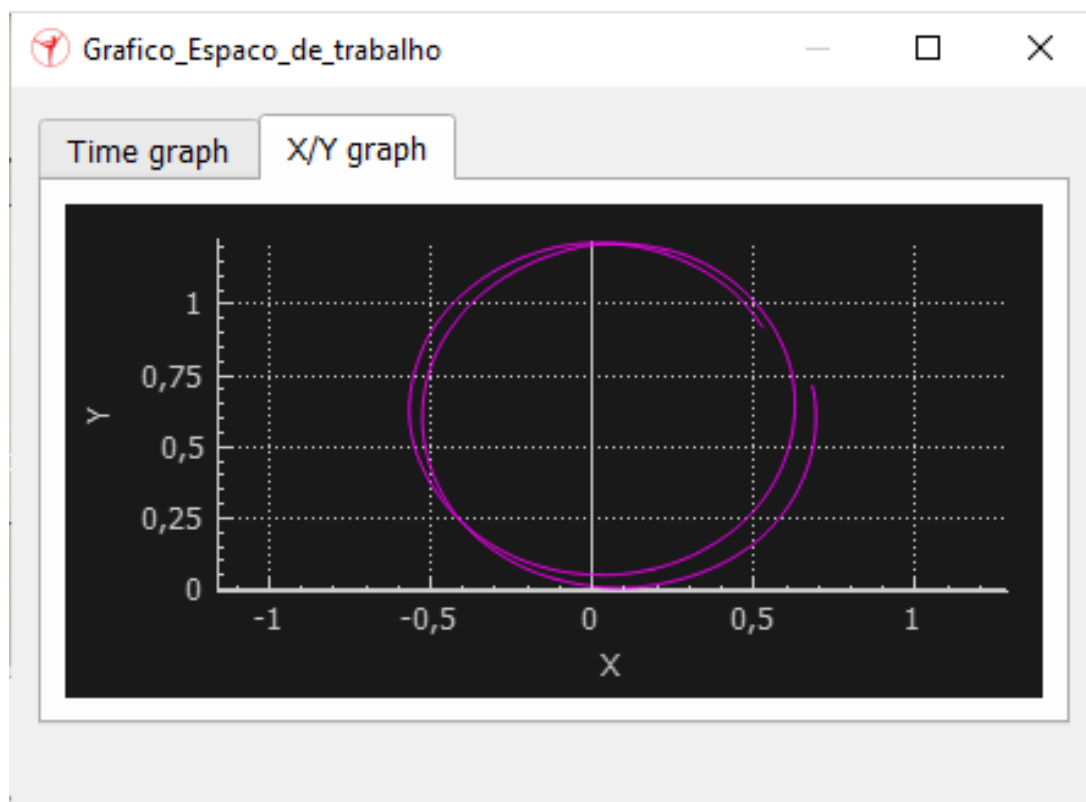
Fonte: Autores.

Figura 9. *Gráfico da Velocidade dos Motores*



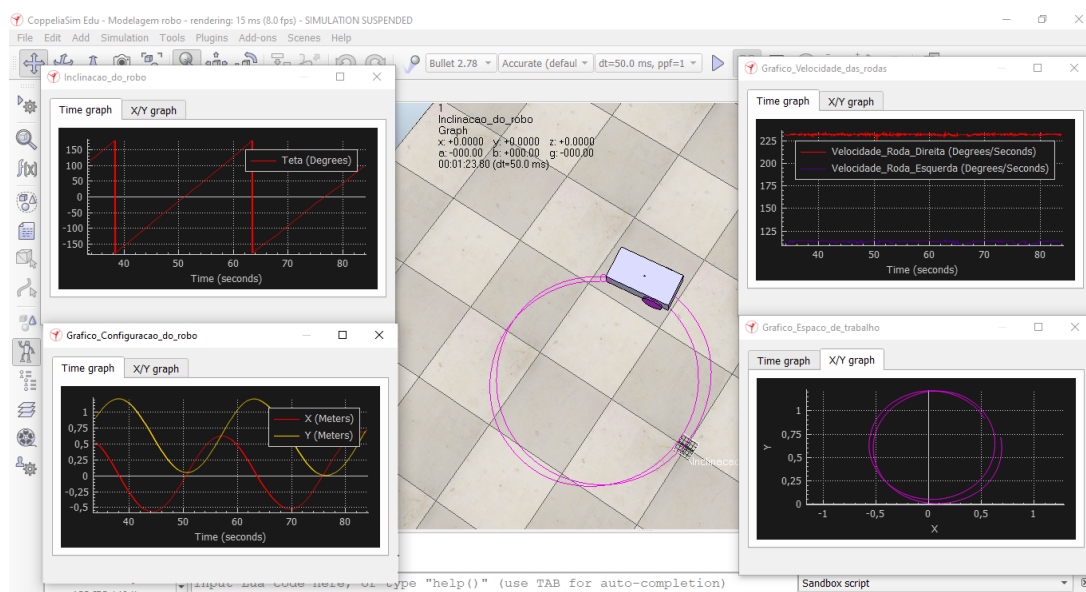
Fonte: Autores.

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, foi utilizado nas mesmas condições o gerador de caminho. Contudo, foi aplicado a função de seguidor de caminho, com a curva interpolada.

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

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

```
1 # This example illustrates how to use the path/motion
2 # planning functionality from a remote API client.
3 #
4 # Load the demo scene 'motionPlanningServerDemo.ttt' in CoppeliaSim
5 # then run this program.
6 #
7 # IMPORTANT: for each successful call to simxStart, there
8 # should be a corresponding call to simxFinish at the end!
9
10 import sim
11
12 print ('Program started')
13 sim.simxFinish(-1) # just in case, close all opened connections
14 clientID=sim.simxStart('127.0.0.1',19997,True,True,-500000,5) # Connect to
    CoppeliaSim, set a very large time-out for blocking commands
15 if clientID!=-1:
16     print ('Connected to remote API server')
17
18     emptyBuff = bytearray()
19
20     # Start the simulation:
21     sim.simxStartSimulation(clientID,sim.simx_opmode_oneshot_wait)
22
23     # Load a robot instance: res,retInts,retFloats,retStrings,retBuffer=
        sim.simxCallScriptFunction(clientID,'remoteApiCommandServer',sim.
        sim_scripttype_childscript,'loadRobot',[],[0,0,0,0],['d:/
        coppeliaRobotics/qrelease/release/test.ttm'],emptyBuff,sim.
        simx_opmode_oneshot_wait)
24     # robotHandle=retInts[0]
25
26     # Retrieve some handles:
27     res,robotHandle=sim.simxGetObjectHandle(clientID,'IRB4600#',sim.
        simx_opmode_oneshot_wait)
28     res,target1=sim.simxGetObjectHandle(clientID,'testPose1#',sim.
        simx_opmode_oneshot_wait)
29     res,target2=sim.simxGetObjectHandle(clientID,'testPose2#',sim.
        simx_opmode_oneshot_wait)
```

```

30     res , target3=sim.simxGetObjectHandle( clientID , 'testPose3#',sim.
        simx_opmode_oneshot_wait)
31     res , target4=sim.simxGetObjectHandle( clientID , 'testPose4#',sim.
        simx_opmode_oneshot_wait)
32
33     # Retrieve the poses (i.e. transformation matrices , 12 values , last row
        is implicit) of some dummies in the scene
34     res , retInts , target1Pose , retStrings , retBuffer=sim.simxCallScriptFunction
        ( clientID , 'remoteApiCommandServer' ,sim.sim_scripttype_childscript , '
        getObjectPose' , [ target1 ] , [] , [] , emptyBuff , sim.
        simx_opmode_oneshot_wait)
35     res , retInts , target2Pose , retStrings , retBuffer=sim.simxCallScriptFunction
        ( clientID , 'remoteApiCommandServer' ,sim.sim_scripttype_childscript , '
        getObjectPose' , [ target2 ] , [] , [] , emptyBuff , sim.
        simx_opmode_oneshot_wait)
36     res , retInts , target3Pose , retStrings , retBuffer=sim.simxCallScriptFunction
        ( clientID , 'remoteApiCommandServer' ,sim.sim_scripttype_childscript , '
        getObjectPose' , [ target3 ] , [] , [] , emptyBuff , sim.
        simx_opmode_oneshot_wait)
37     res , retInts , target4Pose , retStrings , retBuffer=sim.simxCallScriptFunction
        ( clientID , 'remoteApiCommandServer' ,sim.sim_scripttype_childscript , '
        getObjectPose' , [ target4 ] , [] , [] , emptyBuff , sim.
        simx_opmode_oneshot_wait)
38
39     # Get the robot initial state:
40     res , retInts , robotInitialState , retStrings , retBuffer=sim.
        simxCallScriptFunction( clientID , 'remoteApiCommandServer' ,sim.
        sim_scripttype_childscript , 'getRobotState' , [ robotHandle ] , [] , [] ,
        emptyBuff , sim.simx_opmode_oneshot_wait)
41
42     # Some parameters:
43     approachVector=[0,0,1] # often a linear approach is required. This
        should also be part of the calculations when selecting an
        appropriate state for a given pose
44     maxConfigsForDesiredPose=10 # we will try to find 10 different states
        corresponding to the goal pose and order them according to distance
        from initial state
45     maxTrialsForConfigSearch=300 # a parameter needed for finding
        appropriate goal states
46     searchCount=2 # how many times OMPL will run for a given task
47     minConfigsForPathPlanningPath=400 # interpolation states for the OMPL
        path
48     minConfigsForIkPath=100 # interpolation states for the linear approach
        path
49     collisionChecking=1 # whether collision checking is on or off
50
51     # Display a message:

```

```

52     res , retInts , retFloats , retStrings , retBuffer=sim.simxCallScriptFunction(
        clientID , 'remoteApiCommandServer' , sim.sim_scripttype_childscript , '
        displayMessage' , [], [], [ 'Computing and executing several path
        planning tasks for a given goal pose.&&nSeveral goal states
        corresponding to the goal pose are tested.&&nFeasability of a linear
        approach is also tested. Collision detection is on.' ] , emptyBuff , sim
        .simx_opmode_oneshot_wait)
53
54     # Do the path planning here (between a start state and a goal pose ,
        including a linear approach phase):
55     inInts=[robotHandle , collisionChecking , minConfigsForIkPath ,
        minConfigsForPathPlanningPath , maxConfigsForDesiredPose ,
        maxTrialsForConfigSearch , searchCount]
56     inFloats=robotInitialState+target1Pose+approachVector
57     res , retInts , path , retStrings , retBuffer=sim.simxCallScriptFunction(
        clientID , 'remoteApiCommandServer' , sim.sim_scripttype_childscript , '
        findPath_goalIsPose' , inInts , inFloats , [] , emptyBuff , sim .
        simx_opmode_oneshot_wait)
58
59     if (res==0) and len(path)>0:
60         # The path could be in 2 parts: a path planning path , and a linear
            approach path:
61         part1StateCnt=retInts [0]
62         part2StateCnt=retInts [1]
63         path1=path [: part1StateCnt*6]
64
65         # Visualize the first path:
66         res , retInts , retFloats , retStrings , retBuffer=sim .
            simxCallScriptFunction( clientID , 'remoteApiCommandServer' , sim .
            sim_scripttype_childscript , 'visualizePath' , [ robotHandle
            , 255 , 0 , 255 ] , path1 , [] , emptyBuff , sim . simx_opmode_oneshot_wait)
67         line1Handle=retInts [0]
68
69         # Make the robot follow the path:
70         res , retInts , retFloats , retStrings , retBuffer=sim .
            simxCallScriptFunction( clientID , 'remoteApiCommandServer' , sim .
            sim_scripttype_childscript , 'runThroughPath' , [ robotHandle ] , path1
            , [] , emptyBuff , sim . simx_opmode_oneshot_wait)
71
72         # Wait until the end of the movement:
73         runningPath=True
74         while runningPath:
75             res , retInts , retFloats , retStrings , retBuffer=sim .
                simxCallScriptFunction( clientID , 'remoteApiCommandServer' , sim
                . sim_scripttype_childscript , 'isRunningThroughPath' , [
                robotHandle ] , [] , [] , emptyBuff , sim . simx_opmode_oneshot_wait)
76             runningPath=retInts [0]==1

```



```

77
78     path2=path[part1StateCnt*6:]
79
80     # Visualize the second path (the linear approach):
81     res,retInts,retFloats,retStrings,retBuffer=sim.
        simxCallScriptFunction(clientID,'remoteApiCommandServer',sim.
            sim_scripttype_childscript,'visualizePath',[robotHandle
                ,0,255,0],path2,[],emptyBuff,sim.simx_opmode_oneshot_wait)
82     line2Handle=retInts[0]
83
84     # Make the robot follow the path:
85     res,retInts,retFloats,retStrings,retBuffer=sim.
        simxCallScriptFunction(clientID,'remoteApiCommandServer',sim.
            sim_scripttype_childscript,'runThroughPath',[robotHandle],path2
               ,[],emptyBuff,sim.simx_opmode_oneshot_wait)
86
87     # Wait until the end of the movement:
88     runningPath=True
89     while runningPath:
90         res,retInts,retFloats,retStrings,retBuffer=sim.
            simxCallScriptFunction(clientID,'remoteApiCommandServer',sim.
                .sim_scripttype_childscript,'isRunningThroughPath',[
                    robotHandle],[],[],emptyBuff,sim.simx_opmode_oneshot_wait)
91         runningPath=retInts[0]==1
92
93     # Clear the paths visualizations:
94     res,retInts,retFloats,retStrings,retBuffer=sim.
        simxCallScriptFunction(clientID,'remoteApiCommandServer',sim.
            sim_scripttype_childscript,'removeLine',[line1Handle],[],[],
                emptyBuff,sim.simx_opmode_oneshot_wait)
95     res,retInts,retFloats,retStrings,retBuffer=sim.
        simxCallScriptFunction(clientID,'remoteApiCommandServer',sim.
            sim_scripttype_childscript,'removeLine',[line2Handle],[],[],
                emptyBuff,sim.simx_opmode_oneshot_wait)
96
97     # Get the robot current state:
98     res,retInts,robotCurrentConfig,retStrings,retBuffer=sim.
        simxCallScriptFunction(clientID,'remoteApiCommandServer',sim.
            sim_scripttype_childscript,'getRobotState',[robotHandle],[],[],
                emptyBuff,sim.simx_opmode_oneshot_wait)
99
100    # Display a message:
101    res,retInts,retFloats,retStrings,retBuffer=sim.
        simxCallScriptFunction(clientID,'remoteApiCommandServer',sim.
            sim_scripttype_childscript,'displayMessage',[],[],['Computing
                and executing several path planning tasks for a given goal state
                . Collision detection is on.'],emptyBuff,sim.

```

```

simx_opmode_one-shot_wait)

102
103 # Do the path planning here (between a start state and a goal state
    ):
104 inInts=[robotHandle ,collisionChecking ,minConfigsForPathPlanningPath
    ,searchCount]
105 inFloats=robotCurrentConfig+robotInitialState
106 res ,retInts ,path ,retStrings ,retBuffer=sim.simxCallScriptFunction(
    clientID ,'remoteApiCommandServer',sim.sim_scripttype_childscript
    , 'findPath_goalIsState',inInts ,inFloats ,[],emptyBuff,sim.
    simx_opmode_one-shot_wait)

107
108 if (res==0) and len(path)>0:
109     # Visualize the path:
110     res ,retInts ,retFloats ,retStrings ,retBuffer=sim.
        simxCallScriptFunction(clientID ,'remoteApiCommandServer',sim.
        .sim_scripttype_childscript , 'visualizePath',[robotHandle
        ,255,0,255],path,[],emptyBuff,sim.simx_opmode_one-shot_wait)
111     lineHandle=retInts [0]
112
113     # Make the robot follow the path:
114     res ,retInts ,retFloats ,retStrings ,retBuffer=sim.
        simxCallScriptFunction(clientID ,'remoteApiCommandServer',sim.
        .sim_scripttype_childscript , 'runThroughPath',[robotHandle ],
        path,[],emptyBuff,sim.simx_opmode_one-shot_wait)

115
116     # Wait until the end of the movement:
117     runningPath=True
118     while runningPath:
119         res ,retInts ,retFloats ,retStrings ,retBuffer=sim.
            simxCallScriptFunction(clientID ,'remoteApiCommandServer'
            ,sim.sim_scripttype_childscript , 'isRunningThroughPath',[
            robotHandle ],[],[],emptyBuff,sim.
            simx_opmode_one-shot_wait)
120         runningPath=retInts [0]==1
121
122     # Clear the path visualization:
123     res ,retInts ,retFloats ,retStrings ,retBuffer=sim.
        simxCallScriptFunction(clientID ,'remoteApiCommandServer',sim.
        .sim_scripttype_childscript , 'removeLine',[lineHandle ],[],[],
        emptyBuff,sim.simx_opmode_one-shot_wait)

124
125     # Collision checking off:
126     collisionChecking=0
127
128     # Display a message:
129     res ,retInts ,retFloats ,retStrings ,retBuffer=sim.

```

```

simxCallScriptFunction(clientID,'remoteApiCommandServer',sim
.sim_scripttype_childscript,'displayMessage',[[]],[[]],[
Computing and executing several linear paths, going through
several waypoints. Collision detection is OFF.'],emptyBuff,
sim.simx_opmode_oneshot_wait)

130
131 # Find a linear path that runs through several poses:
132 inInts=[robotHandle,collisionChecking,minConfigsForIkPath]
133 inFloats=robotInitialState+target2Pose+target1Pose+target3Pose+
target4Pose
134 res,retInts,path,retStrings,retBuffer=sim.
simxCallScriptFunction(clientID,'remoteApiCommandServer',sim
.sim_scripttype_childscript,'findIkPath',inInts,inFloats,[],
emptyBuff,sim.simx_opmode_oneshot_wait)

135
136 if (res==0) and len(path)>0:
137     # Visualize the path:
138     res,retInts,retFloats,retStrings,retBuffer=sim.
simxCallScriptFunction(clientID,'remoteApiCommandServer',
sim.sim_scripttype_childscript,'visualizePath',[
robotHandle,0,255,255],path,[],emptyBuff,sim.
simx_opmode_oneshot_wait)
139     line1Handle=retInts[0]
140
141     # Make the robot follow the path:
142     res,retInts,retFloats,retStrings,retBuffer=sim.
simxCallScriptFunction(clientID,'remoteApiCommandServer',
sim.sim_scripttype_childscript,'runThroughPath',[
robotHandle],path,[],emptyBuff,sim.
simx_opmode_oneshot_wait)

143
144     # Wait until the end of the movement:
145     runningPath=True
146     while runningPath:
147         res,retInts,retFloats,retStrings,retBuffer=sim.
simxCallScriptFunction(clientID,'
remoteApiCommandServer',sim.
sim_scripttype_childscript,'isRunningThroughPath',[
robotHandle],[[]],[[]],emptyBuff,sim.
simx_opmode_oneshot_wait)
148         runningPath=retInts[0]==1
149
150     # Clear the path visualization:
151     res,retInts,retFloats,retStrings,retBuffer=sim.
simxCallScriptFunction(clientID,'remoteApiCommandServer',
sim.sim_scripttype_childscript,'removeLine',[
line1Handle],[[]],[[]],emptyBuff,sim.

```

```

simx_opmode_one-shot_wait)
152
153 # Stop simulation:
154 sim.simxStopSimulation(clientID, sim.simx_opmode_one-shot_wait)
155
156 # Now close the connection to CoppeliaSim:
157 sim.simxFinish(clientID)
158 else:
159     print('Failed connecting to remote API server')
160 print('Program ended')

```

6 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 de aplicar 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. Outro sim, destaca-se a importância dessas ferramentas simulatórias, para o caso atual de pandemia, onde as aulas são remotas.

Analisando os gráficos demonstrados, é notória melhoria na percepção do comportamento de um robô móvel. Com uma melhor clareza, percebe-se que para uma simples trajetória, a disposição para esse tipo de robô segue uma lógica cíclica. Nos próximos experimentos, serão inseridos obstáculos, os quais devem ser evitados no caminho previamente definido pelo usuário.

7 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.