Universidade Federal do Rio Grande do Norte
Centro de Tecnologia - CT
Departamento de Engenharia de Computação e Automação - DCA

RELATÓRIO TÉCNICO: Projeto 1 - Meta 2

DCA0440 - Sistemas Robóticos Autônomos

Francisco Paiva da Silva Neto
Gabriel Souto Lozano Barbosa

Natal/RN, 28 de setembro de 2023

RESUMO

Neste relatório é apresentado a lógica e os dados referentes a simulação no *software* CoppeliaSim de um gerador de caminho baseado em interpoladores de 3° grau para um robô móvel.

Palavras-chaves: CoppeliaSim; Gerador de caminho; Interpoladores de 3° grau.

Sumário

1. INTRODUÇÃO

Com a evolução dos sistemas robóticos e a dinamização da aplicação de sistemas robóticos autônomos dos mais diversos tipos, houve um encarecimento dos produtos derivados da robótica, principalmente quando o grau de autonomia dos dispositivos cresce cada vez mais, de forma a permitir que os equipamentos ganhem mais autonomia e necessitem cada vez menos de intervenção humana, de forma a conseguir conviver em paralelo com os seres humanos e suas atividades diárias.

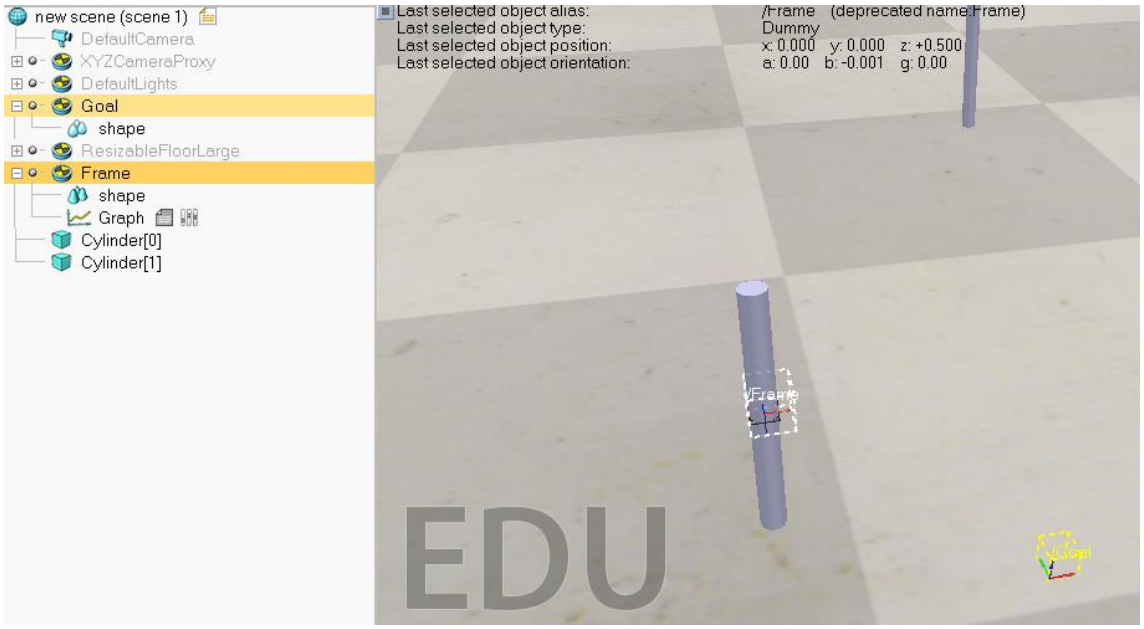
Para isso, o surgimento de softwares de simulação de robôs proporcionou uma maior dinamicidade levando em consideração que é possível ter uma ideia da modelagem e do funcionamento de protótipos antes de sua fabricação, sendo esse um recurso importante não só para validar novos sistemas robóticos autônomos, mas também evitar gastos desnecessários de recursos financeiros e tempos na construção de protótipos no mundo real.

Nesse contexto, o presente relatório visa descrever a etapa de implementação de um script capaz de gerar um caminho possível, a partir de um ponto inicial até um ponto final do ambiente de simulação, levando em consideração a orientação inicial e a orientação final desejada.

2. METODOLOGIA

Para fazer a simulação do gerador de caminho foi usado o software CoppeliaSim EDU, que se trata da versão educacional do programa de simulação robótica, a qual já vem com robôs prontos e montados previamente, permitindo ser colocado num ambiente possível de configuração e inserção de diversos objetos, embora para esta simulação tenha sido implementado frames para visualizarmos o caminho gerado.

Figura 1: Ambiente de configuração no CoppeliaSim.



Fonte: Autores

Com relação a comunicação entre o software e o código, esta foi feita usando uma API em python que importa as bibliotecas disponíveis pelo programa, importando também funções que permitem controlar o ambiente por meio de um código em python fora do CoppeliaSim, de modo que a comunicação se dê por meio de um IP referenciado em um ClientID, bem como, de uma porta de comunicação que funciona como um socket entre os dois elementos que no programa recebe apenas a chamada da API que envia todos os comandos, enquanto que o código inicia a comunicação via socket e faz a movimentação e captura dos dados dos objetivos no ambiente.

Abaixo pode ser visto em ordem cronológicas das figuras o modelo de configuração e execução dos comandos para a geração do caminho.

Bloco 1: Importação das bibliotecas

```
import sim
import time
import sys
import matplotlib.pyplot as plt
import numpy as np
import math as mt
```

Fonte: Autores

A biblioteca sim é uma das bibliotecas disponibilizadas ao se baixar o software do simulador e é responsável pela comunicação via API e o envio de comandos para controle do ambiente, enquanto que a time serve para permitir pausas no código para sincronia de dados. A biblioteca sys serve para encerrar a conexão com sistema em caso de erro de conexão e a matplotlib é chamada com um método especial, o pyplot para plotar os gráficos com os dados requeridos na meta do projeto. A biblioteca numpy e math facilitam a manipulação de matrizes e ângulos.

Bloco 2: Comunicação entre os programas.

```
print ("Start")
sim.simxFinish(-1)
clientID = sim.simxStart('127.0.0.1',19999,True,True,5000,5)

if (clientID != -1):
    print('Conectado')
else:
    print('Não conectou')
    sys.exit(1)

time.sleep(1)
```

Fonte: Autores

No bloco 2 é possível visualizar a configuração do clientID que é usado como parâmetro das funções para referenciar o caminho de envio e recebimento dos dados, como também permitir que sejam coletados os parâmetros de cada um dos objetos e componentes na cena. Após o clientID ser definido, são colocados apenas condições para caso a conexão seja feita com sucesso ou caso dê algum erro, fechando a conexão.

Bloco 3: Captura e envio de posição.

```
#Configuração inicial e final do robô
qi = np.array([0, 0, 0])
qf = np.array([2,4, 60])
step = 0.1

errorcode, frame = sim.simxGetObjectHandle(clientID, 'Frame',
sim.simx_opmode_oneshot_wait)
errorcode = sim.simxSetObjectPosition(clientID, frame,-1, [qi[0],qi[1],
0], sim.simx_opmode_oneshot_wait)
errorcode = sim.simxGetObjectPosition(clientID, frame, -1,
sim.simx_opmode_oneshot_wait)
errorcode = sim.simxGetObjectOrientation(clientID, frame, -1,
sim.simx_opmode_oneshot_wait)

returnCode, goalFrame = sim.simxGetObjectHandle(clientID, 'Goal',
sim.simx_opmode_oneshot_wait)
returnCode = sim.simxSetObjectPosition(clientID, goalFrame, -1,
[qf[0],qf[1], 0], sim.simx_opmode_oneshot_wait)
```

```
returnCode = sim.simxSetObjectOrientation(clientID, goalFrame, -1, [0, 0, qf[2]], sim.simx_opmode_oneshot_wait)
```

Fonte: Autores

No CoppeliaSim os componentes são identificados como objetos, logo, para realizar quaisquer ações, são referenciados os objetos da cena projetada, logo, as funções acima fazem a captura respectivamente dos objetos que iremos manipular, sendo aplicado o parâmetro “sim.simx_opmode_oneshot_wait” que sempre espera uma resposta da função antes de executar o próximo comando e sendo direcionado para os objetos por meios dos nomes inseridos como o segundo parâmetro de cada uma das chamadas da função.

Bloco 4: Gerar caminho ponto a ponto.

```
qgoal_x, qgoal_y, qgoal_teta, a, b = Poli3(qi, qf, step)
qgoal = [qgoal_x, qgoal_y, qgoal_teta]

tetainter = interpoGamma(qi[2], qf[2], len(qgoal_x))

for i in range (len(qgoal_x)):

    errorcode = sim.simxSetObjectPosition(clientID, frame, -1, [qgoal_x[i], qgoal_y[i], 0], sim.simx_opmode_oneshot_wait)
    errorcode = sim.simxSetObjectOrientation(clientID, frame, -1, [0, 0, tetainter[i]], sim.simx_opmode_oneshot_wait)

    time.sleep(1)
```

Fonte: Autores

No bloco 4 é apresentado a lógica responsável pela criação do caminho, a partir da função “Poli3”, que estará nos anexos. A função gera o caminho, levando em consideração o ponto de partida e orientação inicial com a posição e orientação final que o robô deve ter, utilizando de polinômios interpoladores de 3° grau. Já a função “interpoGamma” é responsável por fazer a interpolação do ângulo gamma, com isso robô seguirá o caminho mantendo uma orientação coerente com o caminho criado.

O laço mostrado é setado as posições dos pontos do caminho e orientação que o robô deve apresentar em quando estiver em tal ponto.

Bloco 5: Plotar o gráfico do caminho.

```
plt.plot(x,y, linestyle='solid', color='black', marker='o',markerfacecolor='red')
plt.title("Posição do robô (x(t),y(t))")
plt.xlabel("Coordenada x (m)")
plt.ylabel("Coordenada y (m)")
plt.grid()
plt.show()
```

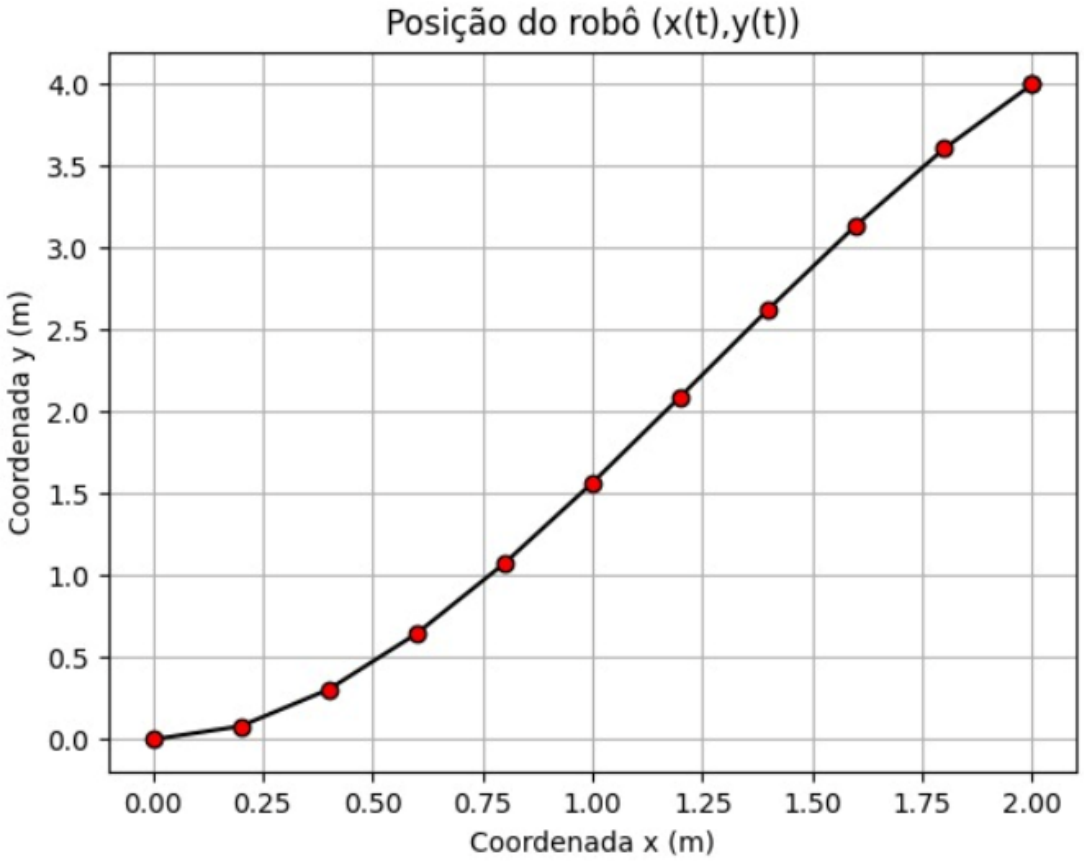
Fonte: Autores

Após todas as configurações, são executados os comandos acima que plotam o caminho gerado.

3. RESULTADOS

O resultado obtido foi considerado o objeto partindo da posição [xi, yi] = [0, 0] e gamma = 0° e terminando em [xf, yf] = [2.0 , 4.0] e gamma = 60°. O gráfico a seguir mostra o caminho gerado a ponto a ponto para que o robô siga o melhor caminho possível, considerando um terreno sem nenhum obstáculo.

Gráfico 1: Caminho gerado ponto a ponto.

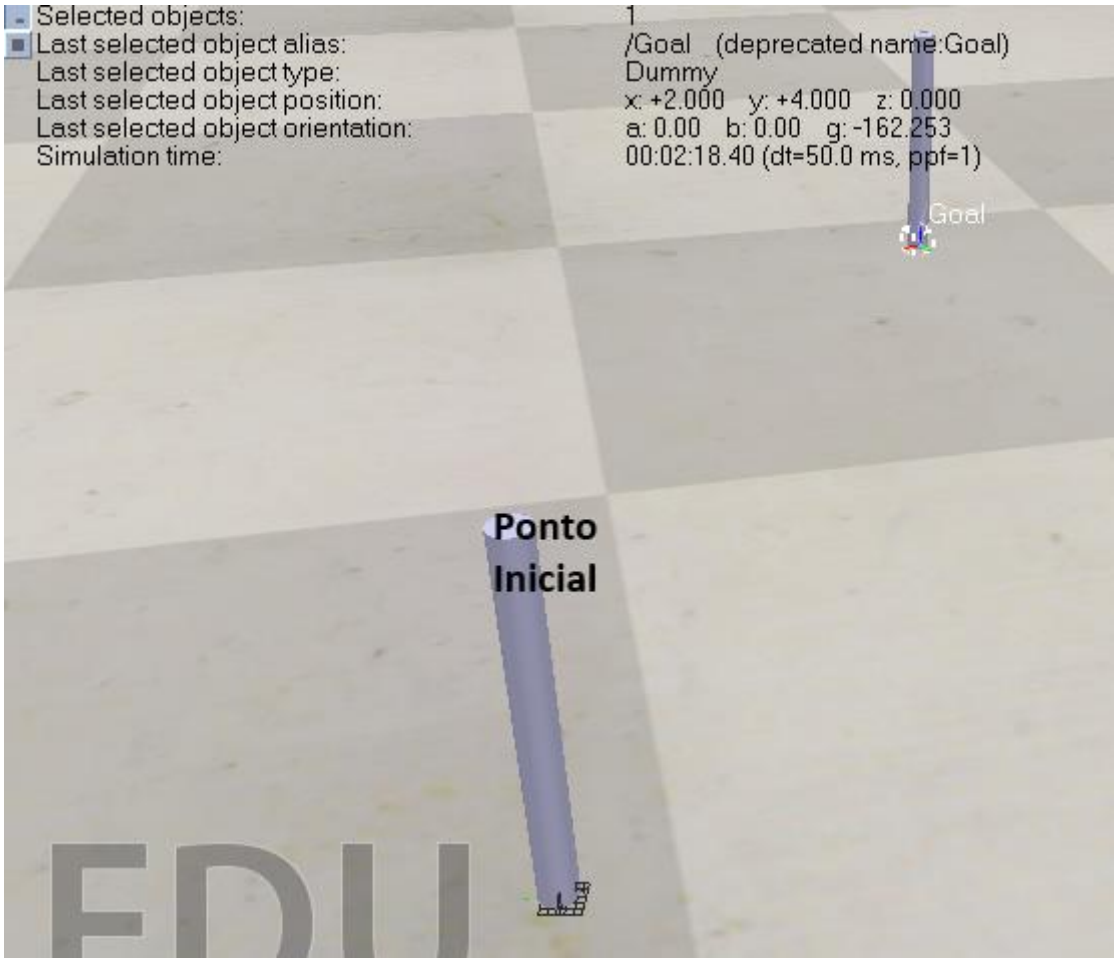


Fonte: Autores

O resultado, para o exemplo anterior, teve como saída dos pontos criados os seguintes vetores: no eixo x = [0.0, 0.2, 0.4, 0.6, 0.8 , 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.0] e no y = [0.0, 0.08082309, 0.30514875, 0.6457616, 1.07544624, 1.5669873, 2.09316937, 2.62677706, 3.14059499, 3.60740777, 4.0, 4.0]. É possível perceber que se repete os pontos finais, tanto no eixo x, como no y, isso ocorre pois o objeto ele chegar no destino final a orientação um pouco diferente da que é peça para finalizar, por este motivo, é feito um ajuste de orientação, já com o objeto localizado no ponto final.

Para essa simulação foi utilizado 2 frames, um que irá realizar os pontos, de forma visual no ambiente de simulação, e outro que representa o destino final que queremos. Os cilindros mostrados na figura 2, são apenas marcos de demonstração do ponto inicial e do ponto final.

Figura 2: Ambiente de simulação.



Fonte: Autores

4. CONCLUSÃO

De forma geral, foi possível implementar um gerador de caminho com interpolador de polinômios de 3 grau e explorar mais funcionalidades do programa de simulação, visto que as funcionalidades foram implementadas por meio das funções disponibilizadas pela API do CoppeliaSim. Nesse contexto, foi possível ainda ter noção de como funcionam outros conceitos relacionados a programação, já que foi usado na conexão entre os programas por meio da API em python, além de, obter os conhecimentos necessários para se realizar um controlador capaz de gerar um caminho capaz de ser implementado em robôs moveis. Desta forma, a experiência referente a segunda meta do projeto foi cumprida no que se refere a implementação e interação com a tecnologia.

ANEXO

Bloco A1: Função Poli3

```
def Poli3 (qi, qf, step) :
    dx = qf[0] - qi[0]
    dy = qf[1] - qi[1]
    teta_init = (qi[2]*mt.pi) / 180
    teta_final = (qf[2]*mt.pi) / 180
    di = mt.tan(teta_init)
    df = mt.tan(teta_final)

    teta_func = []

    if (((teta_init > 89) and (teta_init < 91)) & ((teta_final > 89) &
(teta_final < 91))):
        a0 = qi[0]
        a1 = 0
        a2 = 3 * dx
        a3 = (-2) * dx
        b0 = qi[1]
        b1 = dy
        b2 = 0
        b3 = dy - b1 - b2
    elif ((teta_init > 89) and (teta_init < 91)):
        a0 = qi[0]
        a1 = 0
        a3 = (-1) * dx / 2
        a2 = dx (-1) * a3
        b0 = qi[1]
        b2 = 1
        b1 = 2 * (dy - df * dx) - df * a3 + b2
        b3 = (2 * df * dx - dy) + df * a3 - 2 * b2
    elif ((teta_final > 89) and (teta_final < 91)):
        a0 = qi[0]
        a1 = 3*dx/2
        a2 = 3*dx - 2*a1
        a3 = a1 - 2*dx
        b0 = qi[1]
        b1 = di*a1
        b2 = 1
        b3 = dy - di*a1 - b2
    else:
        a0 = qi[0]
        a1 = dx
        a2 = 0
        a3 = dx - a1 - a2
        b0 = qi[1]
        b1 = di*a1
        b2 = 3*(dy - df*dx) + 2*(df - di)*a1 + df*a2
```

```
        b3 = 3*df*dx - 2*dy - (2*df - di)*a1 - df*a2

    a = np.array([a0, a1, a2, a3])
    b = np.array([b0, b1, b2, b3])
    lambda_poli = np.arange(0, 1+step, step)

    X = a0 + a1*lambda_poli + a2*pow(lambda_poli, 2) +
a3*pow(lambda_poli, 3)
    Y = b0 + b1*lambda_poli + b2*pow(lambda_poli, 2) +
b3*pow(lambda_poli, 3)

    for i in range (len(lambda_poli)):
        teta_func.append(mt.atan((b1 + b2*lambda_poli[i] +
b3*pow(lambda_poli[i], 2))/(a1 + a2*lambda_poli[i] +
b3*pow(lambda_poli[i],2)))*180/mt.pi)
        #teta_func.append(mt.atan((a1 + a2*lambda_poli[i] +
b3*pow(lambda_poli[i],2))/(b1 + b2*lambda_poli[i] +
b3*pow(lambda_poli[i], 2)))*180/mt.pi)

    X = np.append(X, qf[0])
    Y = np.append(Y, qf[1])
    teta_func.append(qf[2])

    #Retorno dos dados
    return X, Y, teta_func, a, b
```

Fonte: Autores

Bloco A2: Função interpoGamma

```
def interpoGamma (qi, qf, step):

    teta_interp = np.linspace(qi, qf, step)
    return teta_interp
```

Fonte: Autores.