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

# **RELATÓRIO TÉCNICO:** Projeto 1 - Meta 3

## **DCA0440 - Sistemas Robóticos Autônomos**

Francisco Paiva da Silva Neto
Gabriel Souto Lozano Barbosa

Natal/RN, 28 de setembro de 2023

# Resumo

Neste relatório são apresentados os dados referentes à simulação de um robô Pioneer P3DX no software CoppeliaSim com a tentativa de implementação de controlador cinemático por meio de do controlador de Samson que fornece um seguidor de caminho.
**Palavras-chaves: Robô. CoppeliaSim. Controlador Cinemático. Python.**

# Sumário

# 1. Introdução

Com o surgimento das técnicas de controle e a diversificação das técnicas que permitem a sua realização, as aplicações e os métodos de implementação de controladores também foram crescendo, ao ponto de que uma ação geral possa ser controlada por meio de sua modelagem e a criação de um controlador específico para ela.

Nesse contexto, a robótica é uma área que vem se desenvolvendo cada vez mais ao longo do tempo, e com isso a necessidade de desenvolvimento de formas melhores para controle dos robôs, de forma a automatizar os processos para que a interferência humana seja reduzida.

Com isso, o presente relatório descreve a implementação de um seguidor de trajetória por meio do controlador do Samson para fazer um robô ir de um ponto inicial a um final seguindo uma trajetória definida por um polinômio cúbico.

# 2. Metodologia

Para realizar o experimento, foi necessário gerar um caminho que neste experimento é descrito por um polinômio de grau 3, em que foram inseridas as configurações iniciais e finais na forma que as entradas e saídas são vetores com as posições em x e y, e o ângulo de orientação inicial e final.

Após ser gerado o caminho e a função Poli3 ter retornado os pontos dos eixos x e y, os coeficientes dos polinômios, bem como um vetor de orientações, esses valores são em um vetor ‘qgoal’ representado por uma matriz com o tamanho de dos vetores que a função retorna em linhas e 3 colunas, respectivas as coordenadas x,y e ao ângulo, que descreve as configurações ao longo do tempo. Após isso, é armazenado em um vetor ‘q’ a configuração atual do robô que é capturada por meio do comando ‘sim.simxGetObjectPosition()’ e ‘sim.simxGetObjectOrientation()’ , em que no último apenas o valor de gamma é salvo, referente ao teta. Vale salientar que o ‘q’ não é o vetor da equação da restrição cinemática, apenas possui o mesmo nome.

Figura 1: Formação do vetor de configuração ‘q’

```
#Captura dados da configuração dos robôs
erro, [x, y, z] = sim.simxGetObjectPosition(clientID, robot, -1, sim.simx_opmode_buffer)
erro, [alpha, beta, gamma] = sim.simxGetObjectOrientation(clientID, robot, -1, sim.simx_opmode_buffer)
x_pos.append(x)
y_pos.append(y)
teta.append((gamma*180)/mt.pi)

#Imprime os dados da configuração do robô no CoppeliaSim
sim.simxAddStatusbarMessage(clientID, "Posição em x " + str(x) + " e a Posição em y " +str(y), sim.simx_opmode_oneshot_wait)
sim.simxAddStatusbarMessage(clientID, "Ângulo teta: " + str(gamma), sim.simx_opmode_oneshot_wait)

#-----------------------------------------------------VARIAVEIS ROBO-----------------------------------------------------
q = np.array([x, y, gamma])
```

Fonte: Produzido pelo autor.

Salvo os valores, a subtração das coordenadas correspondentes no tempo de ‘qgoal’ e ‘q’ são armazenadas num vetor de erro que é multiplicado por um ganho para ser tratado como o vetor de configuração atual, vetor esse chamado de ‘qdot’, fazendo a referência ao vetor de configuração no ponto, que é representado pelo vetor ‘q’ da equação $q = [x\ y\ \theta]$.

Figura 2: Cálculo do vetor de erro

```
error[0, 0] = qgoal[i,0] - q[0]
error[1, 0] = qgoal[i,1] - q[1]
error[2, 0] = qgoal[i,2] - q[2]
qdot = np.dot(gain, error)

Minv = np.linalg.inv(Mdir) #Calcula inversa
print(Minv)
Minv_aux = np.dot(iRr(q[2]), Minv)
print(Minv_aux)
u = np.dot(Minv_aux, qdot)
print("U:", u)
```

Fonte: Produzido pelo autor.

Para encontrar os valores das velocidades em x, y e a velocidade angular, é resolvida a equação de restrições cinemáticas de um robô com acionamento diferencial abaixo.

Figura 3: Equação de restrições cinemáticas

$$\begin{bmatrix} 1 & 0 & b/2 \\ 1 & 0 & -b \\ 0 & 1 & 0 \end{bmatrix} \cdot ({}^{I}R_{R}(\theta))^{T}.q' = \begin{bmatrix} r_D & 0 \\ 0 & r_E \end{bmatrix}$$

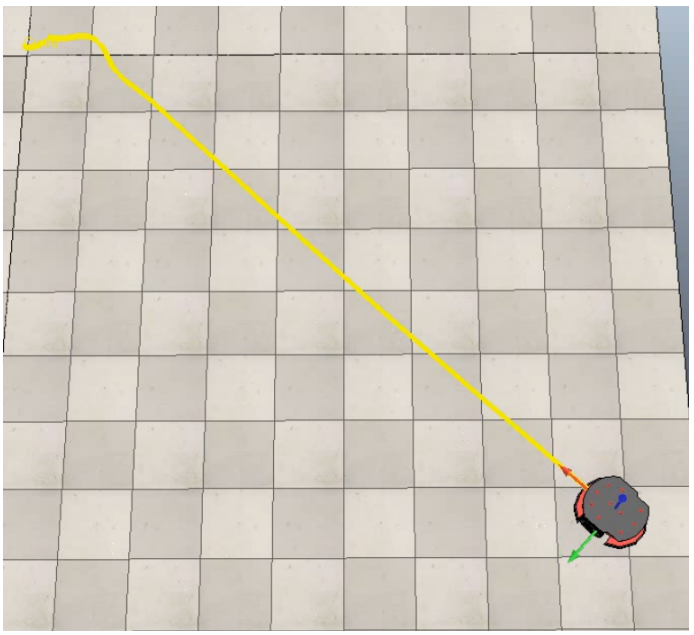
Fonte: Produzido pelo autor.

Resolvendo a equação para q’ no próprio código, chegamos na velocidade para o controlador, como mostra na figura 2 abaixo do vetor qdot que serve como vetor de entrada para nossa equação.

# 3. Resultados

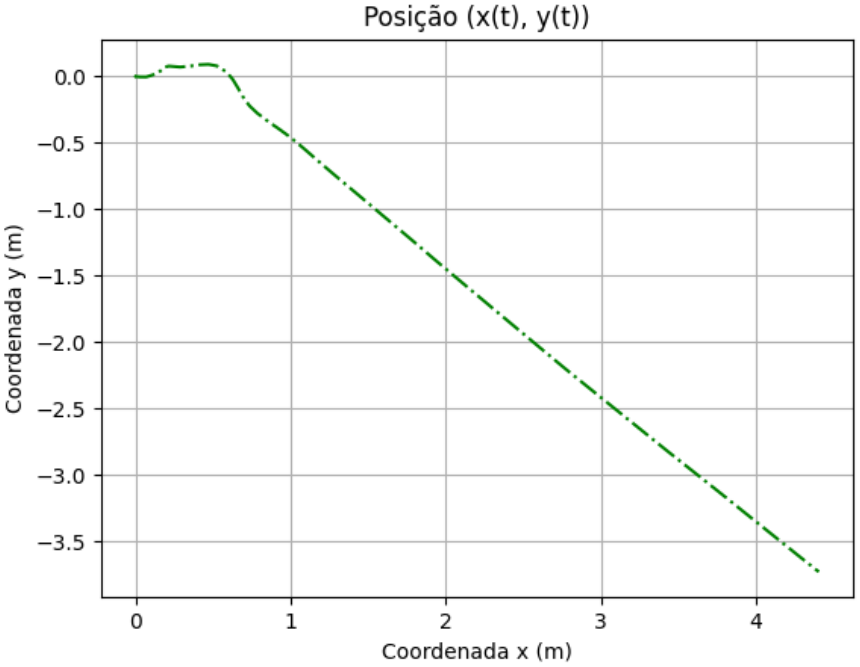
Nas simulações, o robô demonstrou uma trajetória errática se comparado com o que foi gerado pela função de geração de trajetória, devido erros na implementação oriundos da interpretação prévia do controlador, como pode ser visto nas figuras abaixo que mostram a trajetória do robô no CoppeliaSim, no python e a trajetória que deveria ter seguido, bem como a evolução de suas variáveis no tempo.

Figura 4: Trajeto do robô no CoppeliaSim



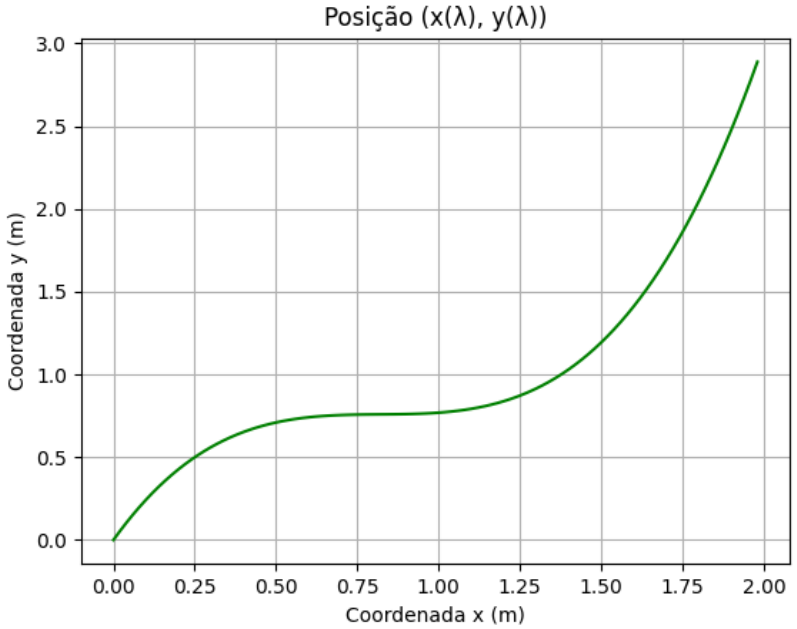
Fonte: Produzido pelo autor.

Figura 5: Trajeto do robô no python



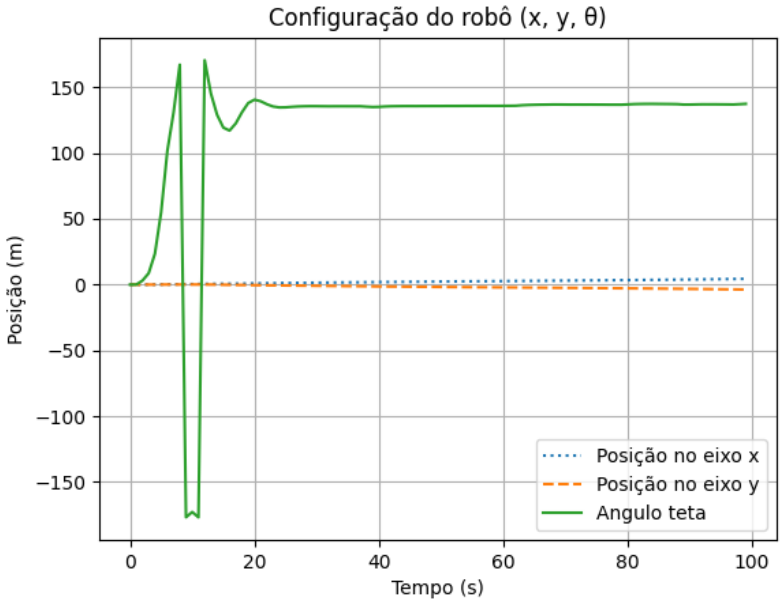

Fonte: Produzido pelo autor.

Figura 6: Trajeto que o robô deveria seguir




Fonte: Produzido pelo autor.

Figura 7: Evolução das variáveis do robô no tempo




Fonte: Produzido pelo autor.

# 4. Conclusão

Ao final do experimento foi possível concluir que existem erros na implementação do controlador que devem ser corrigidos, devido os resultados observados ao longo das simulações, dessa forma, em versões futuras o controlador deverá ter os ganhos ajustados, bem como sua implementação alterada para permitir a entrada correta das entradas e saídas referentes às velocidades para o robô como saída, e os erros calculados para a entrada do controlador.