UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO
CURSO DE ENGENHARIA MECATRÔNICA E ENGENHARIA DA COMPUTAÇÃO

RELATÓRIO

Projeto primeira unidade

ATYSON JAIME DE SOUSA MARTINS: Nº 20190153956
JOSE LINDENBERG DE ANDRADE: Nº 20200150293
JÚLIA COSTA CORRÊA DE OLIVEIRA: Nº 20200149087
SAMUEL CAVALCANTI: Nº 20200149318

Natal-RN
2021

ATYSON JAIME DE SOUSA MARTINS: Nº 20190153956
JOSE LINDENBERG DE ANDRADE: Nº 20200150293
JÚLIA COSTA CORRÊA DE OLIVEIRA: Nº 20200149087
SAMUEL CAVALCANTI: Nº 20200149318

PROJETO PRIMEIRA UNIDADE

Relatório presentado à disciplina de Sistemas Robóticos Autônomos, correspondente à avaliação parcial da 1º unidade do semestre 2021.2 do curso de Engenharia Mecatrônica e Engenharia da Computação da Universidade Federal do Rio Grande do Norte, sob orientação do **Prof. Pablo Javier.**

Professor: Pablo Javier Alsina.

Natal-RN
2021

RESUMO

Esse relatório tem como objetivo relatar o processo de implementação e visualização de caminhos baseados em polinômios de terceiro grau.

Lista de Figuras

Sumário

1 INTRODUÇÃO

O objetivo desse trabalho é simular um robô móvel com acionamento diferencial em um software de simulação, onde é necessário capturar informações de orientação,posição do robô (x, y, θ) em um referencial global e exibir informações como velocidade das rodas em função do tempo, sua orientação, posição, também em função do tempo. Após a configuração da simulação é Implementado um gerador de caminhos baseado em polinômios interpoladores de terceiro grau para robô móvel e exibir o caminho gerado no simulador. Por fim foi implementando dois controladores cinemáticos onde um é um controlador seguir de trajetória e outro um controlador seguidor de posição.

2 DESENVOLVIMENTO

O desenvolvimento desse trabalho foi dividido em três partes, configuração inicial do simulador, implementação do gerador de caminhos baseado em polinômios e implementação dos controladores de posição e seguidor de trajetória.

2.1 Primeira Meta: Simulação de um robô móvel com acionamento diferencial

Continuando o que já foi apresentado em relatórios passados, utilizou-se novamente o robô Pioneer 3-DX. Além disso, usufruiu-se do RemoteApi da própria documentação do *coppelia* para possibilitar a escrita do código em Python.

Inicialmente foi preciso descobrir a identificação dos motores das rodas para posteriormente ser possível definir a velocidade do robô. Abaixo é possível observar as partes do código utilizadas para a realização dessas funções.

```python
def get_motors(client_id: int) -> tuple[int, int]:
    '''
        sim.getObjectHandle("Pioneer_p3dx_leftMotor")
    '''

    left_motor_name = 'Pioneer_p3dx_leftMotor'
    right_motor_name = 'Pioneer_p3dx_rightMotor'

    blocking_mode = sim.simx_opmode_blocking
    _, left_motor = sim.simxGetObjectHandle(
        client_id, left_motor_name, blocking_mode)

    _, right_motor = sim.simxGetObjectHandle(
        client_id, right_motor_name, blocking_mode)

    assert right_motor != -1,\
        f'não foi possível recuperar o motor direito do simulador: {right_motor_name}'
    assert left_motor_name != -1,\
        f'não foi possível recuperar o motor esquerdo do simulador: {left_motor_name} '

    return left_motor, right_motor
```

```python
def set_motor_velocity(client_id: int, motor_id: int, velocity: float):
    sim.simxSetJointTargetVelocity(
        client_id, motor_id, velocity, sim.simx_opmode_streaming)
```

Figura 1: Códigos utilizados para pegar a identificação do motor e definir a velocidade, respectivamente

Além disso, dessa mesma forma foram extraídos os dados necessários dos sensores de posição e orientação do robô, utilizando a função simxGetObjectHandle, para a criação dos controladores que serão vistos logo mais nesse relatório. E foram enviadas novamente tanto para a plotagem do caminho no V-Rap quanto para a geração de gráficos.

2.2 Segunda Meta: Gerador de caminho baseado em polinômios

O processo de implementação foi dividido em duas partes, implementação e visualização em gráficos do caminho gerado pelo polinômio e a comunicação e visualização do caminho gerado no CoppeliaSim.

2.2.1 Implementação do polinômio de 3 grau

Dados os seguintes polinômios $x(\lambda)$, $y(\lambda)$ e $\theta(\lambda)$ que representação a configuração do robô em em dado λ

$$x(\lambda) = a_0 + a_1\lambda + a_2\lambda^2 + a_3\lambda^3$$
$$y(\lambda) = b_0 + b_1\lambda + b_2\lambda^2 + b_3\lambda^3$$
$$\theta(\lambda) = tan^{-1}(\frac{b1 + 2b_2\lambda + 3b_3\lambda^2}{a1 + 2a_2\lambda + 3a_3\lambda^2})$$

Uma vez que temos a posição inicial P_i e final P_f, definimos que, uma posição é formada pela tupla da sua coordenada x, sua coordenada y e seu angulo θ. logo a posição P_i e P_f, são:

$$P_i = (x_i, y_i, \theta_i)$$
$$P_f = (x_f, y_f, \theta_f)$$

então definimos $\Delta x = x_f - x_i$, $\Delta y = y_f - y_i$,$d_i = tan(\theta_i)$ e $d_f = tan(\theta_f)$. Para encontrar os coeficientes desse sistema, dizemos que:

$$x(0), y(0), \theta(0) = (x_i, y_i, \theta_i)$$
$$x(1), y(1), \theta(1) = (x_f, y_f, \theta_f)$$

Esse problema foi resolvido pro Diogo Pinheiro Fernandes Pedrosa em sua tese de mestrado, então a seguir vamos apresentar a solução dele

se $\theta_i \approx 90^\circ$ **e** $\theta_f \approx 90^\circ$

$$
\begin{aligned}
a_0 &= x_i \\
a_1 &= 0 \\
a_2 &= 3\Delta x \\
a_3 &= -2\Delta x \\
b_0 &= y_i \\
b_1 &= \Delta y \\
b_2 &= 0 \\
b_3 &= \Delta y - b_1 - b_2
\end{aligned}
$$

se apenas $\theta_i \approx 90^\circ$

$$
\begin{aligned}
a_0 &= x_i \\
a_1 &= 0 \\
a_2 &= \frac{3\Delta x}{2} \\
a_3 &= \frac{-\Delta x}{2} \\
b_0 &= y_i \\
b_1 &= 2(\Delta y - d_f \Delta x) - d_f a_3 \\
b_2 &= 2d_f \Delta x - \Delta y + d_f a_3 \\
b_3 &= 0
\end{aligned}
$$

se apenas $\theta_f \approx 90^\circ$

$$a_0 = x_i$$
$$a_1 = \frac{3\Delta x}{2}$$
$$a_2 = 3\Delta x - 2a_1$$
$$a_3 = a_1 - 2\Delta x$$
$$b_0 = y_i$$
$$b_1 = d_i a_1$$
$$b_2 = -\Delta y$$
$$b_3 = \Delta y - d_i a_1 - b_2$$

se nenhum dos ângulos for próximo a 90 graus

$$a_0 = x_i$$
$$a_1 = \Delta x$$
$$a_2 = 0$$
$$a_3 = \Delta x - a_2 - a_1$$
$$b_0 = y_i$$
$$b_1 = d_i a_1$$
$$b_2 = 3\Delta y - 3d_f \Delta x + d_f a_2 - 2a_1(d_i - d_f)$$
$$b_3 = 3d_f \Delta x - 2\Delta y - d_f a_2 - a_1(2d_f - d_i)$$

Após implementado esse conjunto de *if*s e *else*s foram realizados gráficos para testar a implementação, que podem ser vistos na figura 2.

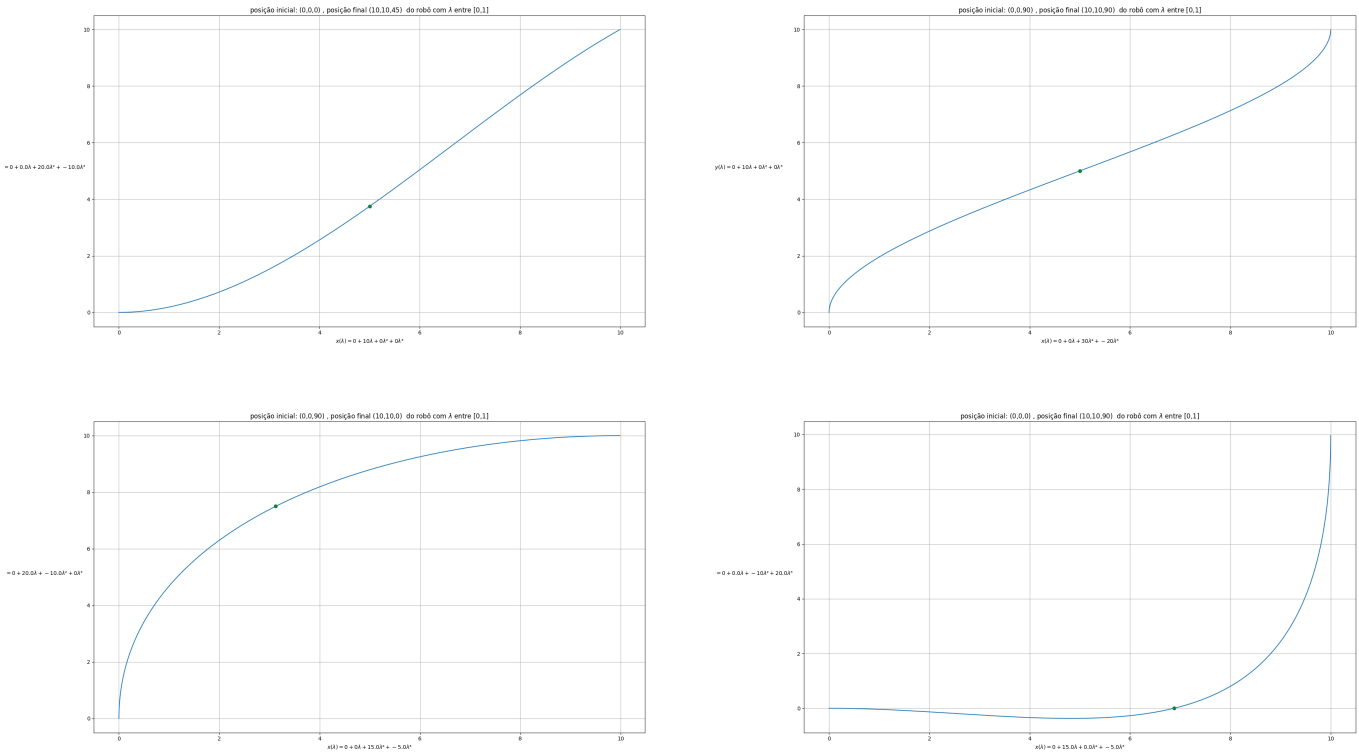

Figura 2: Caminhos gerados para a posição inicial (0,0) até (10,10), variando a sua orientação θ

2.2.2 Visualização do caminho no Simulador

Como pode ser visto na Figura 3 inicialmente foram calculados os polinômios geradores de terceiro grau, a partir do quais foram obtidos os gráficos mostrados na Figura 2. Após isso foi necessário calcular as posições (x, y, z) para o λ variando entre 0 e 1, sendo o eixo Z definido com um valor fixo (no caso, 0.05), tendo em vista que o robô não se move nele. Após a obtenção dos pontos no plano real do simulador, foi necessário transformá-los em uma string para, então, enviar essa informação para o *Coppelia*.

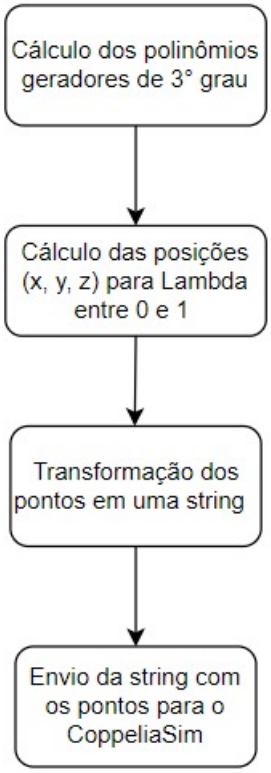

Figura 3: Fluxograma da visualização do caminho no CoppeliaSim

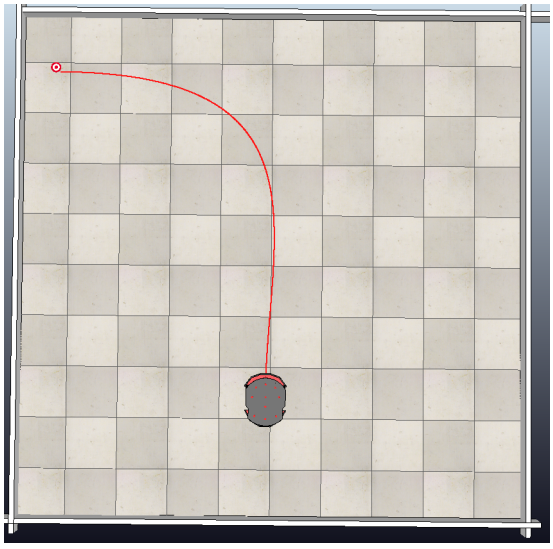

Figura 4: Caminho a ser seguido pelo robô no espaço de trabalho do CoppeliaSim

2.3 Terceira Meta: Controladores cinemáticos do robô móvel

2.3.1 Controlador Frederico

Um dos objetivos desse trabalho é observar e implementar o controlador de posição, chamado de Frederico. O objetivo de um controlador cinemático de posição é dado uma posição, x, y, como referência, o controlador deve enviar sinais para os atuadores de modo que quando o tempo tende ao infinito, a posição do robô esteja na referência, para isso esse controlador deve receber a cada

instante de tempo o valor atual de sua configuração (x, y, θ). O controlador Frederico, foi proposto por Frederico Carvalho Vieira, em usa tese de mestrado. O controlador Frederico, é basicamente a combinação de dois controladores P.I.D que visam minimizar para zero o um erro de posição e de orientação do robô. A função custo é definida da seguinte forma:

$$e_\theta = \Delta\phi$$
$$e_s = \Delta l cos(\Delta\phi)$$

onde e_θ é o erro orientação e e_s erro de posição.

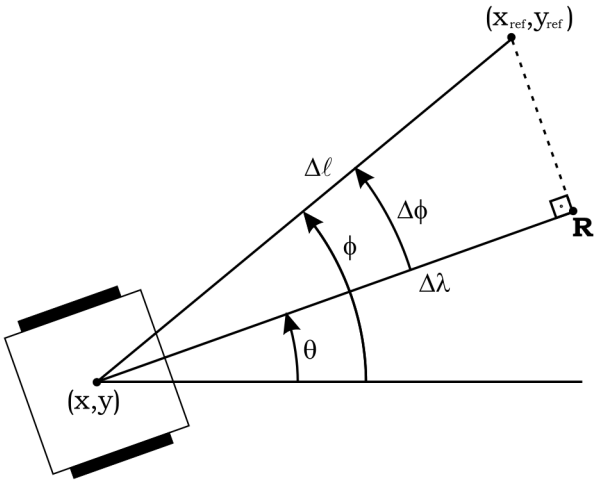

Figura 5: Definições envolvidas no problema de controle

perceba que $\Delta l cos(\Delta\phi) = \Delta\lambda$ e $\Delta\phi = \phi - \theta$, uma vez que buscamos que θ seja igual a ϕ, logo $\Delta\phi$ tende a 0, por tanto, minimizar o $\Delta\phi$ e $\Delta\lambda$, tende a minimizar Δl. Essa relação permite resolver esse problema dividindo ele em dois sub-problemas, minimizar o erro de orientação e de posição, onde cada sub problema possui um controlador P.I.D próprio. Nesse trabalho, foi implementado um controlador P.I.D com windup guard. Onde cada P.I.D ficou com a seguintes configurações:

P.I.D de posição

$$k_p = 0.9145$$
$$k_i = \frac{k_p}{1.9079}$$
$$k_d = k_p 0.085$$

P.I.D de orientação

$$k_p = 0.4$$
$$k_i = 0.15$$
$$k_d = k_p 0.0474$$

em ambos os controladores o windup guard é igual 5. detalhes de implementação podem ser encontrados no arquivo *PID.py* dentro do repositório do projeto, `https://github.com/samuel-cavalcanti/controle_cinematico_sistemas_autonomos_ufrn`

2.3.2 Controlador Frederico seguindo uma trajetória

Em um controlador de posição, o seu alvo, ou seja, a sua posição desejada não varia com o tempo. Já quando o objetivo é seguir uma trajetória, esse alvo possui sua posição variando com o tempo e o controlador deve ser capaz de seguir e idealmente ter uma distância próxima a zero do seu alvo. Uma vez tendo um caminho adquirido através de polinômios de terceiro grau e um controlador capaz de seguir um uma posição x, y. O que foi feito foi mapear esse caminho que está em λ para o domínio do tempo e fazer para que cada instante de tempo t, uma nova posição desejada fosse para o controlador Frederico que buscará minimizar a distância do robô com a do alvo móvel. Mapear o caminho que está em função de λ para o tempo se traduz em resolver a seguinte relação:

$$v(t)dt = d\lambda \sqrt{(\frac{dx}{d\lambda})^2 + (\frac{dy}{d\lambda})^2}$$

onde $v(t)$ é a velocidade no domínio do tempo. Podemos rearranjar e equação para encontrarmos a relação de $\lambda(t)$

$$d\lambda = \frac{v(t)dt}{\sqrt{(\frac{dx}{d\lambda})^2 + (\frac{dy}{d\lambda})^2}}$$

$$\lambda = \int \frac{v(t)dt}{\sqrt{(\frac{dx}{d\lambda})^2 + (\frac{dy}{d\lambda})^2}}$$

Como a resolução dessa integral não é trivial, optou-se por uma resolução numérica. Onde

$$\Delta\lambda = \frac{v(t)\Delta t}{\sqrt{(\frac{dx}{d\lambda})^2 + (\frac{dy}{d\lambda})^2}}$$

$$\lambda_k = \lambda_{k-1} + \Delta\lambda$$
$$\textbf{onde } \lambda_{-1} = 0$$

A função de velocidade $v(t)$ utilizada foi

$$v(t) = \frac{V_{\max}(1 - cos(\frac{2\pi t}{T_{\max}})}{2}$$
$$V_{\max} = 2lT_{\max}$$

$$l = \int_0^1 \sqrt{(\frac{dx}{d\lambda})^2 + (\frac{dy}{d\lambda})^2}$$

onde $V_{\max}$ e $T_{\max}$ são a velocidade máxima e o tempo máximo, no nosso trabalho foi arbitrado, um tempo máximo de 5 segundos e foi calculado a integral numericamente através do método do trapézio, dessa forma era obtido o valor de velocidade máxima que era necessária para o calculo de $v(t)$. Tendo o valor de $v(t)$ era realizado o calculo para obter o λ_k, que por sua vez era usado nos polinômios $x(\lambda)$, $y(\lambda)$, $\theta(\lambda)$, que retornava a posição P_t do alvo a ser seguido pelo controlador Frederico. Detalhes de implementação sobre a criação da trajetória a ser seguida pode ser encontrada no arquivo *trajectory_follow.py* no repositório `https://github.com/samuel-cavalcanti/controle_cinematico_sistemas_autonomos_ufrn`

2.3.3 Algoritmo do funcionamento do sistema de controle do robô móvel

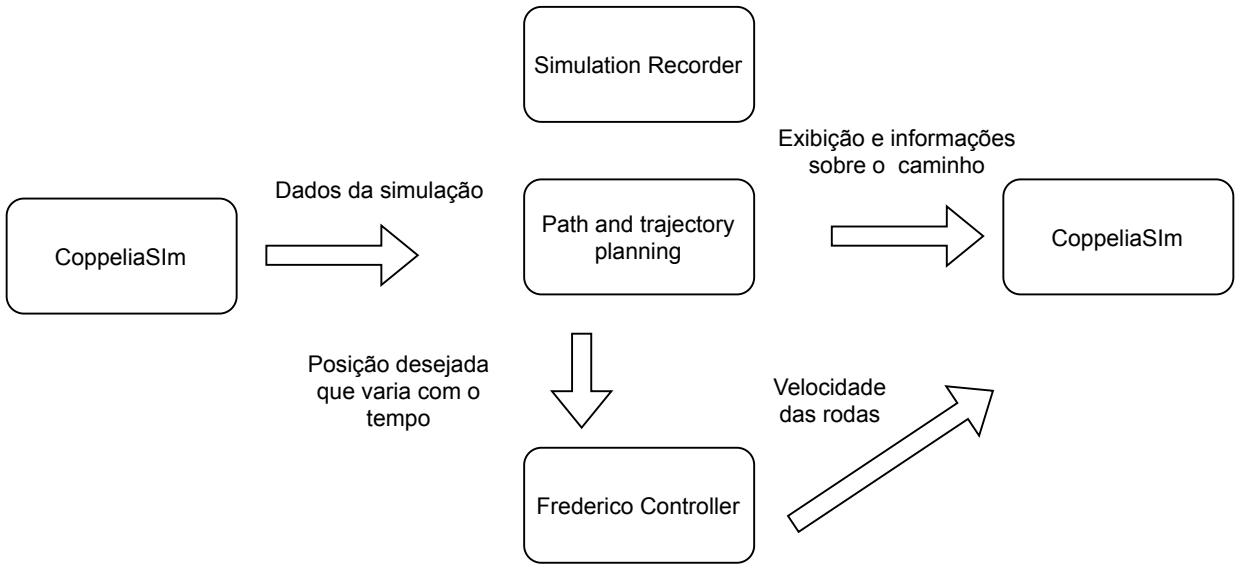

Figura 6: Funcionando do sistema

Uma vez que já tenhamos configurado o simulador, implementando os controladores é chegado o momento de integrar cada modulo do nosso sistema e executar a simulação, coletando dados que serão discutidos na próxima seção de resultados. O sistema funciona da seguinte maneira, primeiro é inicializado a simulação, a qual possui scrips escritos em lua, que inicializam a comunicação via socket com o a nossa aplicação python, a qual possui três principais módulos, o Simulation Recorder que vai gravar todos os dados que vem do simulador e que vão para o simulador, ele no final da simulação gera um arquivo .csv a qual foi utilizado para criação dos gráficos. Caso a simulação for o teste com apenas o controlador Frederico, o módolo Frederico estará ativado e receberá a posição desejada e a configuração atual do robô a qual retorna os valores de velocidade das rodas que será enviado de volta para o simulador. Também foi criado um objeto auxiliar durante a simulação, um disco chamado Target, a qual será posicionado para a posição desejada que o controlador deverá posicionar o robô. Caso queiramos testar o controlador Frederico com um planejador de trajetórias baseado em polinômios, então ativamos o modulo path and trajectory planning a qual recebe o tempo atual da aplicação e retorna a configuração desejada do robô, que será enviada junto com a configuração atual do robô para o controlador Frederico a qual responde com as velocidades das rodas a qual será enviado para o simulador. Como o modulo path and trajectory planning está ativado então ele envia para o simulador o caminho a qual o robô deverá percorrer e também o disco Target irá ser atualizado constantemente com a posição desejada que o robô deveria está. Foi simulador o robô móvel me duas tarefas, o primeiro é dado uma posição x, y desejada, o robô deve ser capaz de sozinho chegar nele, para esse cenário foi utilizado apenas o controlador Frederico, ou seja, não foi utilizado o gerador de trajetórias. A partir dessa simulação foi gerador os gráficos das figuras 7, 8. Um vídeo demonstrando o funcionamento pode ser encontrado no seguinte link: `https://www.youtube.com/watch?v=OEe0XtFPN4g&ab_channel=SamuelCavalcanti`

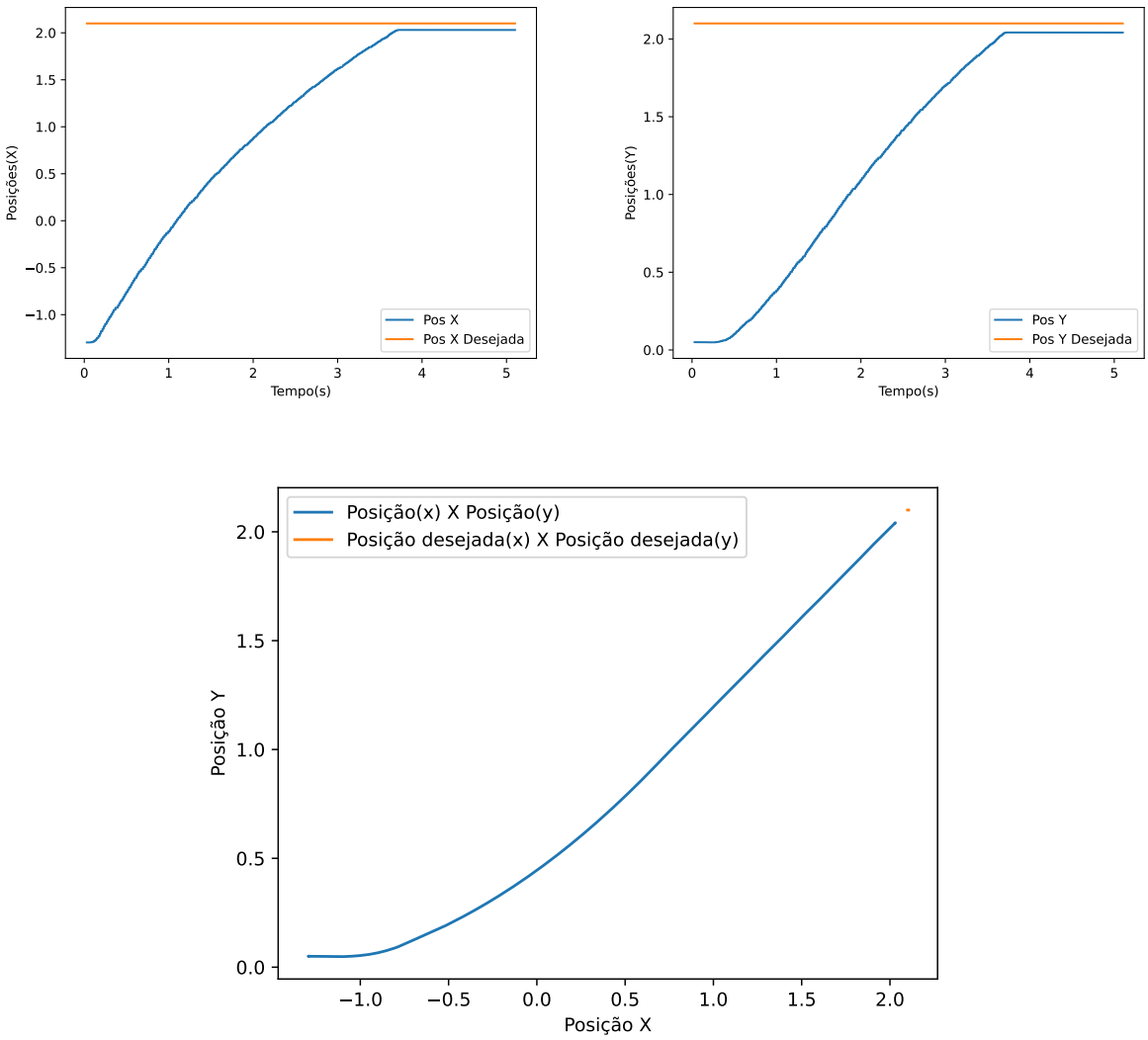

Figura 7: Posições do robô com apenas o controlador Frederico

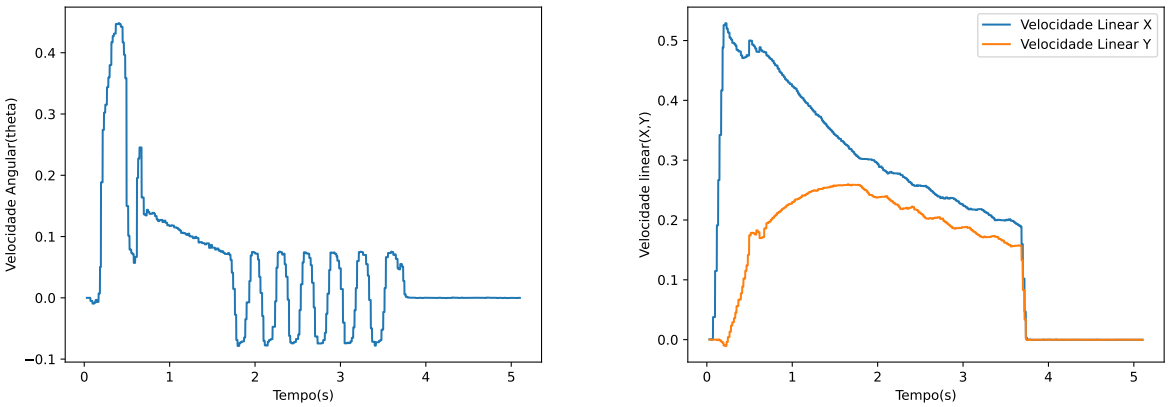

Figura 8: velocidades robô com apenas o controlador Frederico

A segunda tarefa foi dado uma configuração final x, y, θ para o gerador de trajetórias, a qual em um dado instante de tempo t, gerava uma configuração x_t, y_t, θ_t que o robô deveria ser capaz de seguir ou executar a trajetória gerada. Simulando esse segundo cenário foi gerado os gráficos da seguintes

figuras 9, 10. Também foi gravado o vídeo da simulação: `https://www.youtube.com/watch?v=pNImd-6fzWw&ab_channel=SamuelCavalcanti`

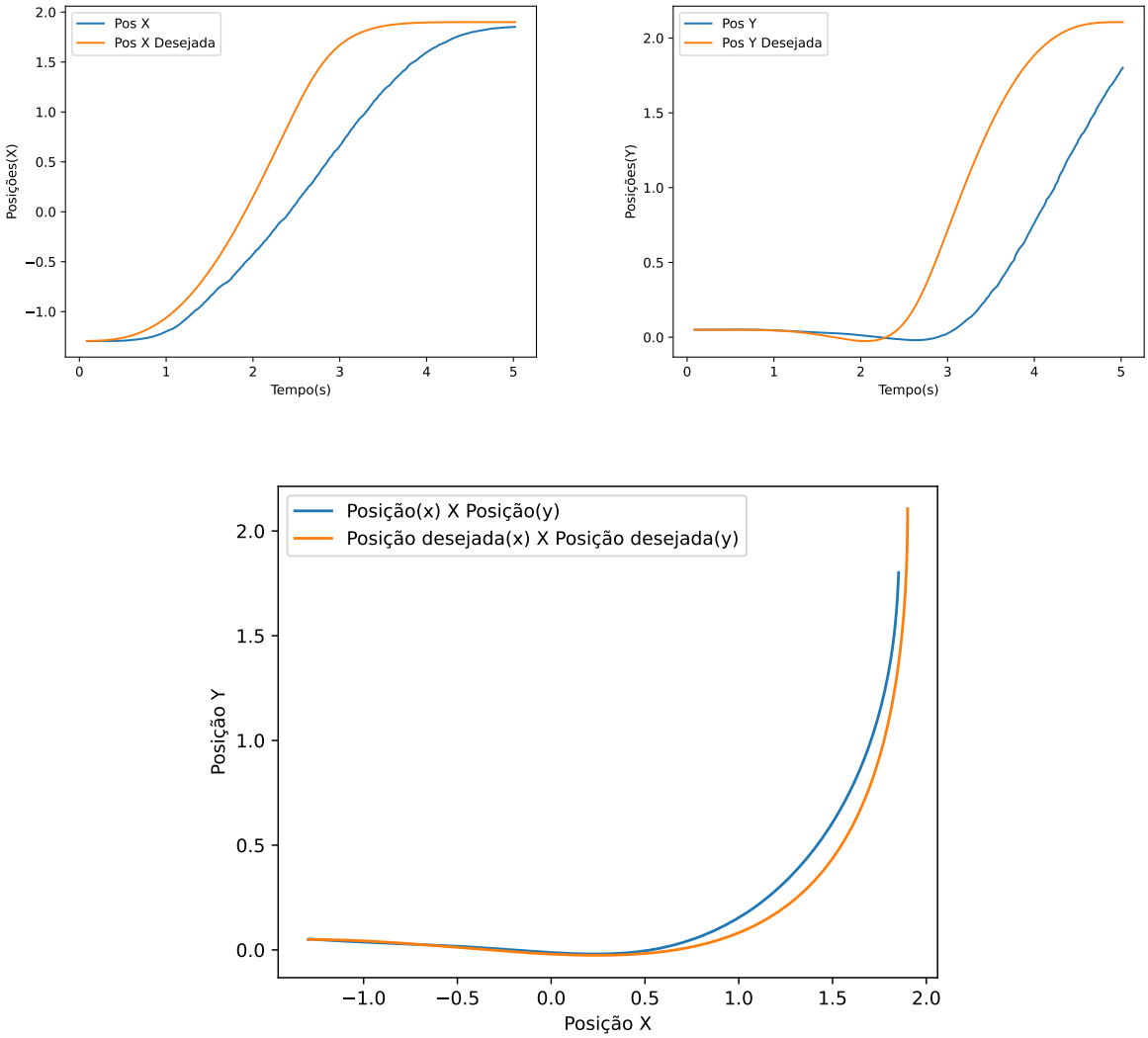

Figura 9: Posição X, Y do robô em comparação ao seguidor de trajetória

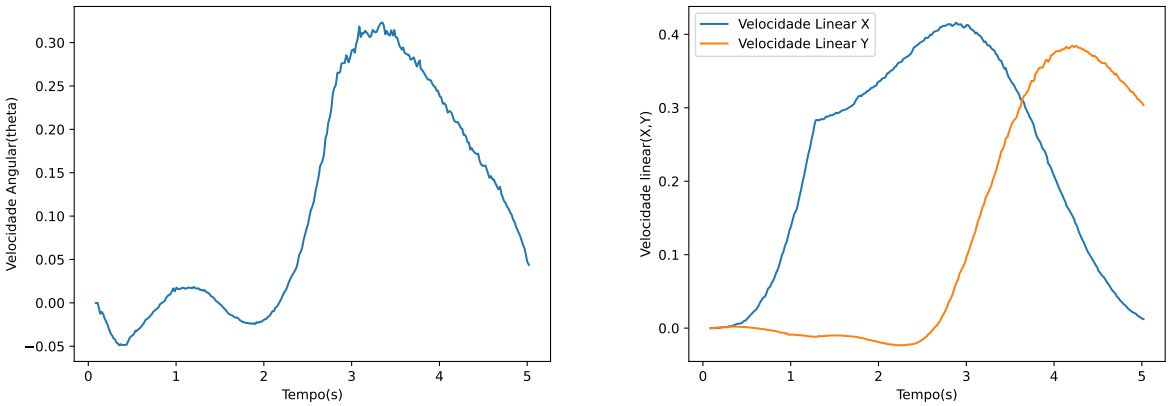

Figura 10: velocidades robô com o seguidor de trajetória

Referências

[1] Simulador CoppeliaSim. Coppelia Robotics, 2021. Disponível em: <https://www.coppeliarobotics.com/>. Acesso em 22 de nov. de 2021.

[2] API do V-RAP. Coppelia Robotics, 2021. Disponível em: <https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm#simxGetObjectPosition>. Acesso em 22 de nov. de 2021.

[3] Remote API. Coppelia Robotics, 2021. Disponível em: <https://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm>. Acesso em 22 de nov. de 2021.

[4] ROS Interface. Coppelia Robotics, 2021. Disponível em: <https://www.coppeliarobotics.com/helpFiles/en/rosInterf.htm>. Acesso em 22 de nov. de 2021.

[5] ROS 2 tutorial. Coppelia Robotics, 2021. Disponível em: <https://www.coppeliarobotics.com/helpFiles/en/ros2Tutorial.htm>. Acesso em 22 de nov. de 2021.