UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

FEELT49081 – SISTEMAS DIGITAIS PARA MECATRÔNICA

Trabalho Prático de Sistemas Digitais para Mecatrônica

Prof. Éder Alves de Moura

TRABALHO 1:

DRONE EM 2D

Eduardo Cunha de Carvalho	11621EMT003
João Alberto Andraus Gassan	11211EMT003
João Victor Medeiros Rocha	41621ETE005
Matheus Alves de Paula	11521EMT008
Nicolas Fischmann	12011EMT032

Uberlândia

Março de 2022

1. OBJETIVOS

Este trabalho prático tem como objetivo desenvolver um jogo de controle de um drone em 2D utilizando Python + Pygame sendo necessário desenvolver em linhas de código a simulação das leis físicas e estratégia de controle implementando-se a simulação de um sistema de controle de posição com base na modelagem cinemática e dinâmica do drone desenvolvida ao longo das aulas.

2. BIBLIOTECA

2.1. PYGAME

Para o presente projeto foi utilizado uma biblioteca do Python conhecida como Pygame, sendo que conforme o site oficial do desenvolvedor, esta biblioteca é usada na criação de aplicações multimídia como jogos. Ela é uma camada de abstração construída sobre a biblioteca SDL, e assim como a SDL ela é altamente portável e roda em praticamente todas plataformas e sistemas operacionais.

Algumas características dessa biblioteca são listadas abaixo:

- Não necessita do OpenGL;
- Pode ser facilmente usado com processadores de múltiplos núcleos;
- Utiliza C otimizado e código Assembly para funções principais;
- Fácil instalação, não necessita a instalação de diversos pacotes para funcionar em diversos sistemas operacionais;
- Portátil;
- É simples e fácil de usar.

2.2. MATPLOTLIB, NUMPY E TIME

A biblioteca time permite obter o tempo muito preciso em formato de long. Vamos usar essa biblioteca para criar um relógio que limita a frequência de cálculos dentro do programa.

A biblioteca numpy permite criar listas de listas e fazer operações matriciais. Usamos essa biblioteca para os cálculos matriciais e o gerenciamento de dados como as listas de posições que usamos depois para gerar figuras com Matplotlib.

3. DESENVOLVIMENTO

3.1. ELEMENTOS

Para o desenvolvimento do projeto foram definidos dois elementos principais:

- Drone: elemento que será controlado com base na sua posição e atitude (orientação);
- Tela: onde acontece a ação do sistema.

3.2. IMPLEMENTAÇÃO

3.2.1. CLASSE GAME

A classe game contém todos os elementos do jogo, ou seja o jogador (drone), os muros e a posição de comando no nosso caso.

3.2.2. CLASSE PLAYER

A classe player contém a dinâmica do drone, os parâmetros do drone, os controladores, e um método de rebote que gera a interação entre o drone e um muro em caso de colisão do tipo vertical ou horizontal.

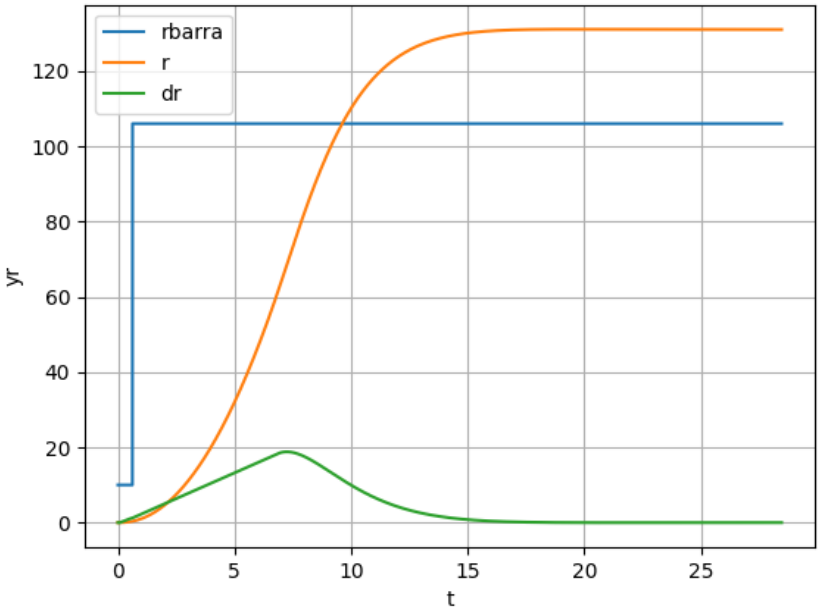

Figura X: Evolução da posição do drone sobre o eixo y_r

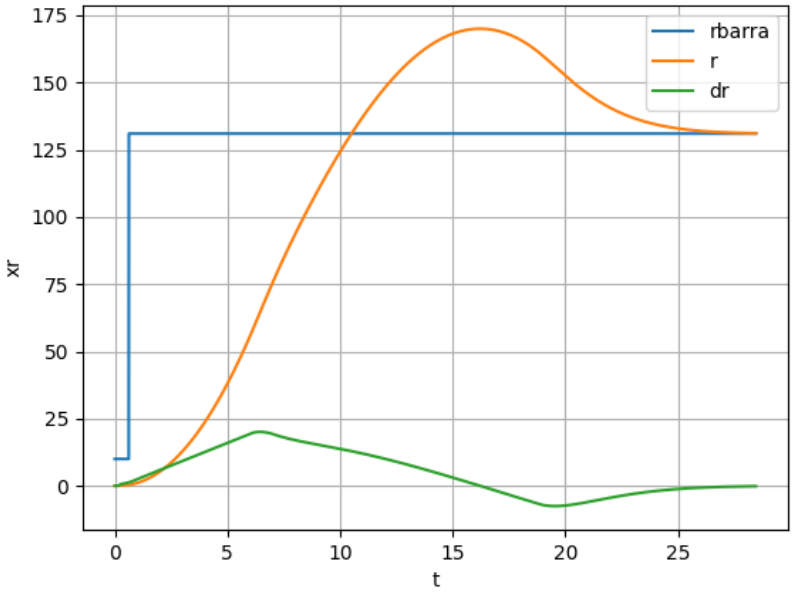

Figura X:Evolução da posição do drone sobre o eixo x_r

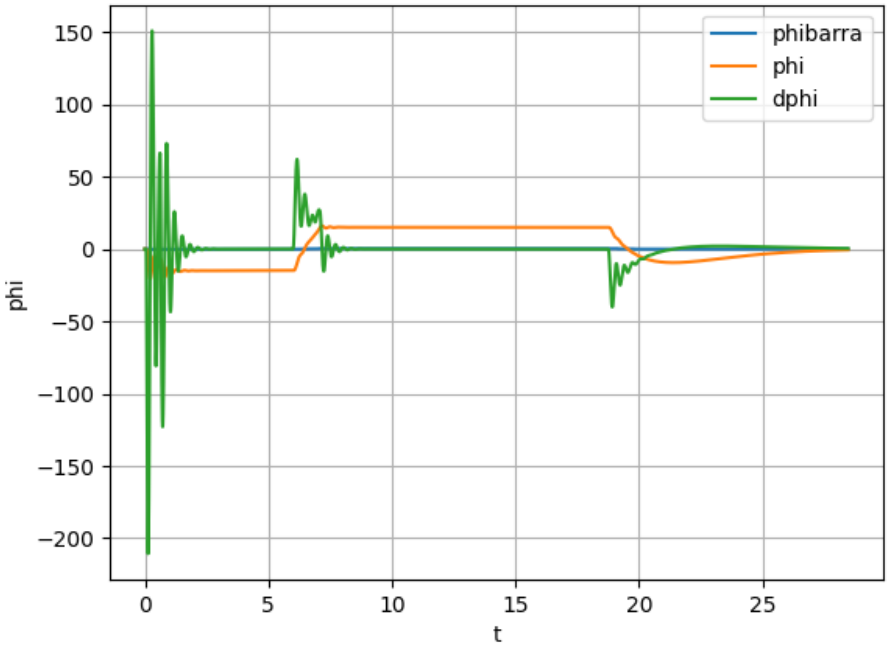

Figura X: Evolução do ângulo do drone no tempo

Podemos perceber que o drone tem um erro estático na posição vertical por causa da força peso. Podemos zerar esse erro com um controlador que contém um integrador ou aumentando o ganho proporcional, mas assim arriscamos a perda da estabilidade do drone.

Finalmente, a função *atualizar_posicao_tela()* permite gerar a orientação e a posição da imagem da tela.

3.2.3. CLASSE WALL

A classe Wall define o objeto muro como um retângulo delimitado por dois pontos (p1 e p2). Um muro pode ser do tipo vertical ou horizontal para ter um rebote respetivamente horizontal ou vertical. Um muro tem um lado, se o drone entrar num muro marcado como “right”, o drone vai rebotar para a esquerda.

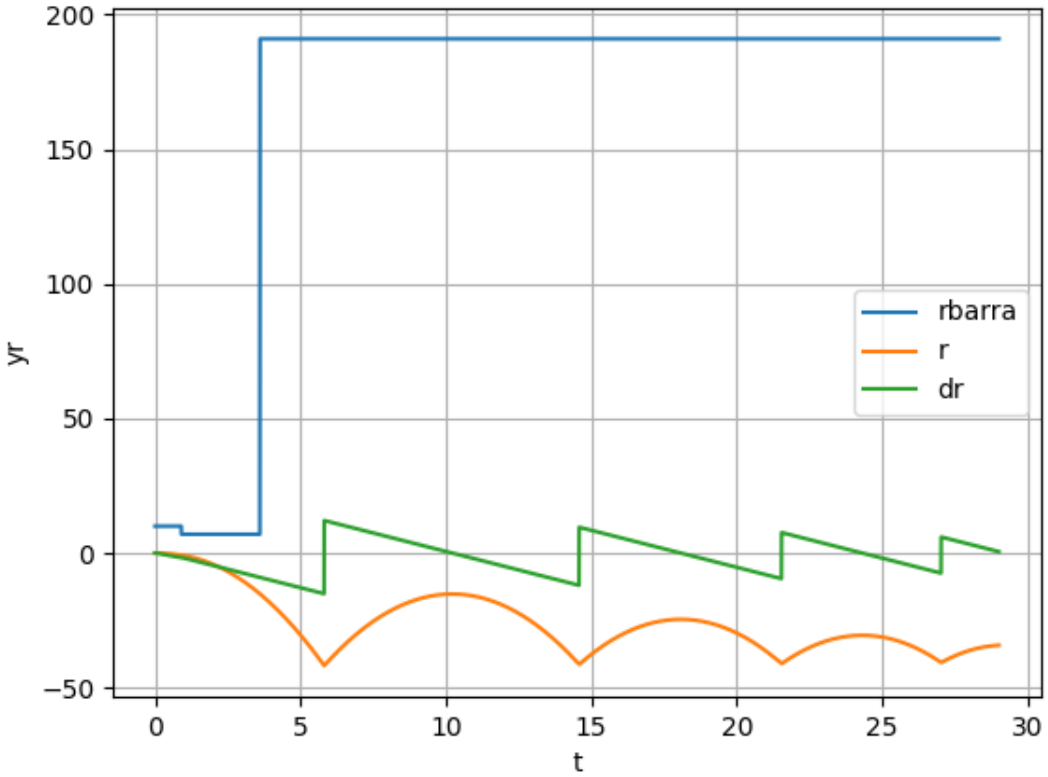

Figura X: Exemplo de rebote do drone sobre o chão (muro de baixo)

3.2.4. CÓDIGO MAIN

O código principal contém o loop principal do jogo com a biblioteca pygame. Ele gerencia o tempo com a biblioteca time.time, detecta o clic do mouse na tela para definir a posição de comando enviada para o drone, chama as funções de detecção de colisão dos muros no objeto Game para mudar as variáveis de colisão “colisaoh” e “colisaov” o objeto Game.Player em caso de colisão.

Após um tempo definido, o jogo termina e gera as figuras da evolução da posição do drone no tempo que vimos em cima usando a biblioteca matplotlib.

Finalmente, para os cálculos vetoriais e cálculos matemáticos, usamos as bibliotecas numpy e math.

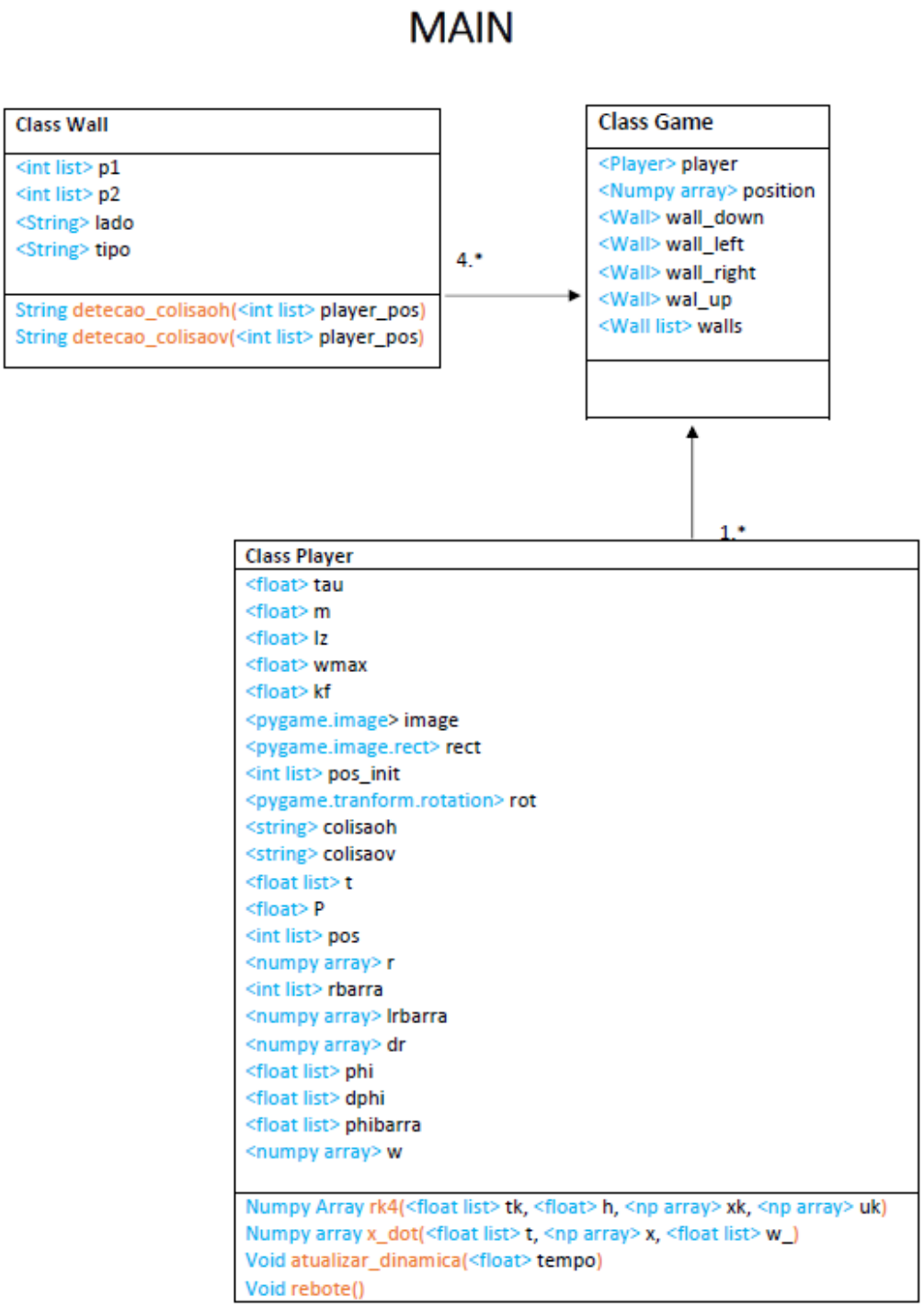

Figura X: Diagrama UML do programa

3.3. DIFICULDADES

Nós enfrentamos várias dificuldades durante o desenvolvimento do projeto. A primeira dificuldade foi mostrar na tela o drone com um certo ângulo. Para isso, tivemos que criar uma imagem orientada com a função *transform.rotate* do pygame e fazer ela aparecer na boa posição. Infelizmente, o eixo de rotação da imagem era o ponto em cima na esquerda do quadro da imagem, e não o centro. Assim, tivemos que jogar com a posição inicial, a o centro da imagem rotacionada e a posição do drone na simulação dentro da função *atualizar_posicao_tela()* do objeto Player a fim de mostrar uma trajetória do drone que faz sentido.

A nossa segunda dificuldade foi a diferença entre a base das coordenadas na tela e as coordenadas cartesianas que usamos para a simulação do drone. Temos que inverter o y_r na tela para o drone não ir no sentido errado. Isso podia causar muitos erros.

A terceira dificuldade foi o gerenciamento dos dados e das listas dentro da simulação para obter as listas que permitem gerar as figuras. Foi preciso muitos testes e uma análise de cada variável dentro do programa para debugar vários problemas em relação ao tamanho das listas, o concatenamento, o armazenamento dos valores, a otimização das listas, etc.

A quarta dificuldade foi o tempo de atraso do programa. Nós queríamos uma frequência de atualização dos controladores de 0.005 segundos, mas o cálculo de uma iteração dura geralmente mais ou menos 0.05 segundos. Tivemos que colocar como parâmetro o tempo real no código de simulação para ter uma movimentação com uma velocidade próxima do real.

Finalmente, nós perdemos muito tempo tentando refazer sozinhos a dinâmica do drone, mas nossa simulação funcionava relativamente usando a função *odeint* em vez de rk4. O problema da função *odeint* foi o tempo de cálculo que era abusivamente longo e podia às vezes ser exponencial.

4. REFERÊNCIAS

[1] Real Python, 2022. Disponível em: https://realpython.com/pygame-a-primer/#pygame-concepts.

[2] Python Brasil, 2022. Disponível em: https://python.org.br/games/

[3] Modelos e códigos do curso de Sistemas Embarcados II, Éder Alves de Moura, 2022, UFU