Universidade Federal de Uberlândia-UFU
Faculdade de Engenharia Mecânica
Graduação em Engenharia Mecatrônica
Sistemas Digitais para Mecatrônica
Prof.: Éder

SIMULAÇÃO DRONE

Guilherme Gonzaga Silva	11621EMT021
João Pedro Silva Fagundes	11711EMT014
Thalles Jonathan Trigueiro de Almeida	11711EMT020
Lucas Boaventura Pereira	11711EMT021

Uberlândia-MG
Outubro/2021

Sumário

1. Bibliotecas Utilizadas

A decisão do grupo foi de utilizar a biblioteca Pygame para implementação do jogo a ser desenvolvido.

1.1. Pygame

Pygame é uma biblioteca de jogos multiplataforma feita para ser utilizada em conjunta com a linguagem de programação Python baseada em SDL. É voltada para o desenvolvimento e games e interfaces gráficas, o Pygame fornece acesso a áudios, teclados, controles, mouses e hardwares gráficos via OpenGL e Direct3D.

Tem como vantagens:

- CPU’s multi core podem ser usadas facilmente;
- Usa código C e Assembly otimizado para funções básicas;
- Vem com muitos sistemas operacionais;
- Verdadeiramente portátil;
- É simples;
- Muitos jogos foram publicados;
- Você controla seu loop principal;
- Não requer uma GUI para usar todas as funções;
- Resposta rápida aos bugs relatados;
- Pequena quantidade de código;
- Modular.

Figura 1: Pygame.

2. Case e Implementação

2.1. Case

O projeto tem como intuito a modelagem cinemática de um Drone desenvolvida ao longo do semestre da disciplina de Sistemas Digitais para Mecatrônica (Sistemas Embarcados 2). A implementação possui dois tipos de ação:

- Movimentação por meio de waypoints;
- Movimentação por meio do teclado.

2.2. Implementação na biblioteca

Toda estrutura foi montada na biblioteca Pygame. Temos uma classe que é o drone e outra chamada de “Simulation”, onde foram implementados os requisitos dispostos no case, como a implementação de controle no drone.

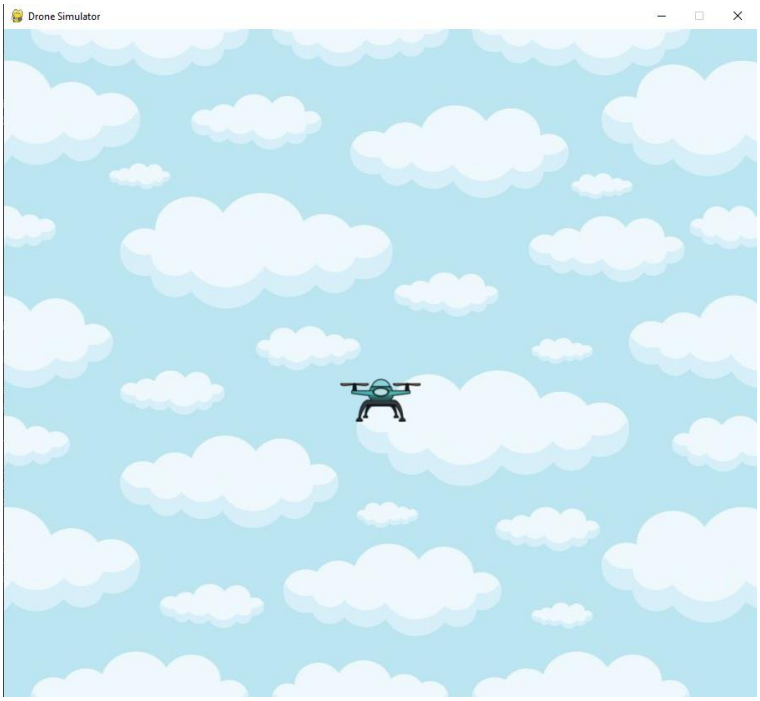

Figura 2: Simulação do código.

2.2.1. Classe “Drone”

Nessa classe, definimos todas as características do drone que foi utilizado na simulação.

Temos uma função chamada “__init__”, essa função recebe uma imagem e a posição inicial do drone, ambos serão passados no código main quando a função for chamada.

Na função “update” os elementos do jogo serão atualizados quando uma tecla é acionada, um exemplo disso são os cenários, a posição do drone e sua posição angular. Essa função recebe a posição em x, em y e o ângulo.

E por fim, a função “draw” cria uma janela para imagem a ser alocada para o drone.

```
class Drone(pygame.sprite.Sprite):
    def __init__(self, image_path, init_pos):
        super().__init__()
        self.image = pygame.image.load(f'images/{image_path}')
        self.original_image = self.image
        self.rect = self.image.get_rect(center=init_pos)

    def update(self, pos_x, pos_y, angle):
        self.rect.move_ip(pos_x - self.rect.centerx, pos_y - self.rect.ce
ntery)

        self.image = pygame.transform.rotozoom(self.original_image, angle
 * 180.0 / pi, 1)
        self.rect = self.image.get_rect(center=self.rect.center)

    def draw(self, surface):
        surface.blit(self.image, self.rect)
```

2.2.2. Classe “Simulation”

Nessa sessão, são definidas as características físicas do drone e do ambiente em que ele se encontra, como: massa, gravidade, tamanho, constante de força, momento de inércia, etc. Também foi implementado o sistema de controle para agir na movimentação do drone através dos waypoints seguindo a trajetória da figura 3.

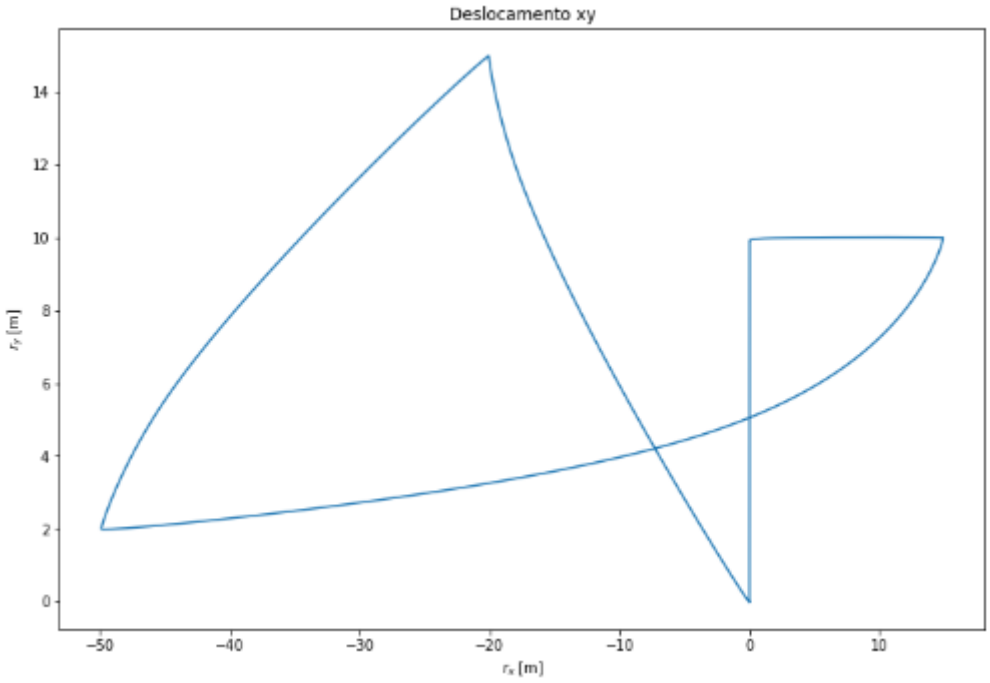

Figura 3: Trajetória pré estabelecida do drone no plano xy.

Foi implementado um controlador proporcional derivativo (PD) para gerar a ação de controle.

Além disso, foi utilizado o método de Runge-Kutta para resolução de EDO's, conhecido como uma otimização do método de Euler. Esse método consiste em comparar um polinômio de Taylor apropriado para eliminar o cálculo das derivadas, fazendo-se várias avaliações da função a cada passo.

Buscando uma melhor estimativa da derivada com a avaliação da função em mais pontos no intervalo $[x_n, x_{n+1}]$ do que se comparado ao método de Euler. Um método de Runge-Kutta de ordem n possui um erro da ordem de $O\ (h^{n+1})$.

De modo geral, o método de Runge-Kutta é dado por

$$y_{n+1} = y_n + h\ (\text{Inclinação})$$

No qual a Inclinação é uma constante que é obtida através do cálculo da inclinação em vários pontos no interior do subintervalo.

O controlador PD, assim como a ação de controle, as saturações implementadas no motor, cálculos dos erros e das posições, além do método numérico para aproximação das derivadas, estão representadas no código simulation.py disponibilizada na pasta do GitHub.

2.2.3. Código “main”

O código main pode ser divido em duas partes:

- Definição das constantes físicas que serão utilizadas;
- Loop principal, onde o “jogo” é simulado.

No começo, foi definido o tamanho da interface, onde encontramos um problema de que o a classe “simulation” nos entregava os valores em metro e a biblioteca Pygame lia em pixels, para corrigir isso, foi criada uma função chamada “interpolate”, que basicamente fazia essa conversão de escala. Além disso, havia um problema de referencial onde, ao utilizar o comando move.ip, precisávamos da posição absoluta a ser passada, e tínhamos a posição em relação a posição que a imagem está, para isso foi descontado o centro do retângulo.

Também foram determinados os waypoints que seriam passados para a classe “simulation”, além das imagens que utilizadas de background e como referência do drone.

No loop principal, foi colocada uma condição de que o jogo irá rodar até o jogador pressionar o “QUIT”. Ao simular o jogo, o drone irá fazer a trajetória dos waypoints que foram passados para a classe “simulation” e, caso o jogador pressione uma tecla direcional, o movimento irá encerrar e o próprio jogador tem o controle do drone. Caso o movimento dos waypoint acabe o drone, que irá encerrar o movimento no meio da tela, estará sobre o controle do jogador até que o mesmo pressione QUIT.

3. Bibliografia

[1] PYGAME ORG. Pygame.org, 2021. Disponível em: <http://pygame.org/>. Acesso em: 11 de setembro de 2021.

[2] PyGame: A Primer on Game Programming in Python. Disponível em: < https://realpython.com/pygame-a-primer/>. Acesso em: 11 de setembro de 2021.

[3] Ogata, K. - Engenharia de Controle Moderno, Prentice-Hall, 4ª. ed., 2004. Kuo, B.C. – Automatic Control Systems, 7th Edition, Prentice Hall, 1995.

[4] VALLE, K. N. F. Métodos numéricos de Euler e Runge-Kutta. 2012. 40 f. Monografia (Especialização) - Programa de Pós-graduação em Matemática Para Professores Com

Ênfase em Cálculo, Universidade Federal de Minas Gerais, Belo Horizonte, 2012. Disponível em: . Acesso em: 20 de setembro de 2021.