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

Sistemas Digitais
Professor: Éder Alves de Moura

Trabalho final: Simulação do drone 2D

Augusto Perez de Andrade – 11611EMT009
Pedro Henrique Silva Oliveira – 11611EMT007
Matheus Santana de Araújo – 11611EMT026
Larissa Teodoro da Silva – 11321EMT021
Luiz da Silva Moura – 11611EMT028

Uberlândia, 03 de outubro de 2021

SUMÁRIO

1. Módulos e bibliotecas adotados no desenvolvimento

1.1 – Pygame

Pygame é um conjunto de módulos de Python desenvolvidos para a programação de videogames. Esse módulo adiciona funcionalidades acrescidas da biblioteca SDL, permitindo desenvolvimento de jogos e programas multimídia em Python.

O módulo é compatível com diversas plataformas embarcadas e capaz de ser executado em diversos sistemas operacionais.

Desta forma, o Pygame se destaca em diversos pontos:

- Facilidade de utilização em CPUs multicore
- Controle do loop principal
- Código enxuto
- Rápida resposta para identificação de bugs
- Número de downloads na ordem de milhões
- Diversos jogos desenvolvidos já foram publicados

No escopo do projeto, este módulo foi utilizado para desenvolver a interface gráfica do usuário e programação dos objetos de cena.

1.2 – Interfaces de sistema operacional diversos

Este módulo dispõe de um caminho prático para a funcionalidade de sistemas operacionais. Dentre as suas principais funções, vale destacar a manipulação de caminhos de diretórios.

Na atual aplicação, as bibliotecas são utilizadas para efetuar a navegação de arquivos e obter acessos a imagens.

1.3 – Numpy

O Numpy é um projeto de domínio público focado em computação numérica por meio da linguagem Python. O módulo dispõe de um conjunto de bibliotecas contendo funções matemáticas que englobam as áreas de cálculo, geometria, estatística, etc.

Para o nosso projeto, o Numpy foi utilizado principalmente para operações matemáticas vetoriais e trigonométricas (para obter o arco tangente de um ângulo, por exemplo).

2. Descrição da integração do sistema

O sistema foi desenvolvido por meio do ambiente de desenvolvimento integrado PyCharm, plataforma que dispõe de um *layout* simples e organizado para o desenvolvimento do projeto como apresentado na Figura 1.

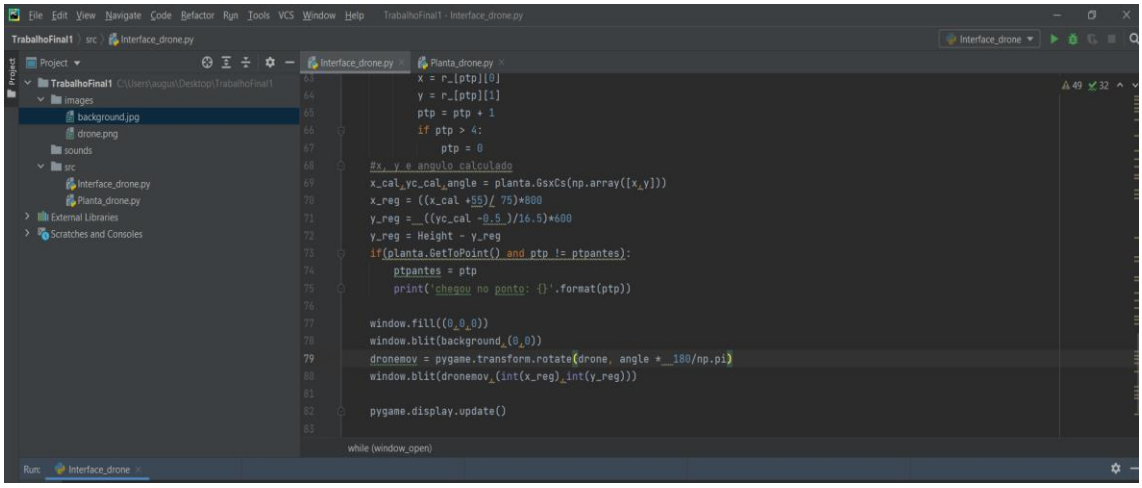

Figura 1 – Ambiente de desenvolvimento integrado PyCharm

Utilizando o módulo Pygame, o programa piloto do projeto gera uma interface gráfica de 800x600 pixels. A janela é preenchida por uma imagem de fundo e um objeto de drone como mostrado na Figura 2.

Figura 2 – Interface gráfica

Além disso, o código de interface é responsável por interpretar interrupções das teclas do usuário, obter posições e ângulos do drone e atualizar a janela a partir dos cálculos fornecidos pelo programa da planta.

O movimento do drone é definido por *waypoints*, de forma que o objeto deve se movimentar para atingir os pontos destinos de acordo com o sistema de controle adotado. A tecla ESPAÇO foi utilizada como comando para alternar os pontos previamente configurados.

Todos os passos da simulação para se obter o comportamento real do drone são realizados pelo código referente à planta. Desta forma, o programa é responsável por definir parâmetros físicos (massa, aceleração da gravidade, momento de inércia, etc.), calcular o módulo e direção da força necessária (proveniente do motor simulado) sobre o objeto em um determinado tempo, e carregar as posições e ângulos do estado a partir do código de interface.

Vale destacar o método *Runge-Kutta* utilizado para resolução de EDO's referentes à posição do drone em cada instante. A Figura 3 apresenta a seção do código utilizada no cálculo das constantes para a aproximação de quarta ordem do método em questão.

```
def rk4(self, tk, h, xk, uk):
    k1 = self.x_dot(tk, xk, uk)
    k2 = self.x_dot(tk + h / 2.0, xk + h * k1 / 2.0, uk)
    k3 = self.x_dot(tk + h / 2.0, xk + h * k2 / 2.0, uk)
    k4 = self.x_dot(tk + h, xk + h * k3, uk)

    return xk + (h / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
```

Figura 3 – Função *Runge-Kutta* 4

Além disso, é importante comentar sobre o controle P&D (Proporcional e derivativo) adotado para gerar a ação de controle. A função de transferência G(s)*C(s) é calculada a partir do erro entre a posição atual e a posição de referência definida pelo *waypoint*.

Desta forma, a partir das constantes Kp e Kd adotadas, é possível calcular as forças horizontais e verticais de cada instante. A seção do código responsável pelo cálculo é mostrada na Figura 4.

```python
def GsxCs(self, r_):#G(s)*c(s)
    if (self.k % (self.Ts / self.h)) == 0:
        r_k = self.x[2:4]
        v_k = self.x[4:6]
        phi_k = self.x[6]
        ome_k = self.x[7]

        v_ = np.array([0, 0])

        # Controle de posição
        kpP = np.array([0.075])
        kdP = np.array([0.25])

        eP = r_ - r_k
        eV = v_ - v_k

        if np.linalg.norm(eP) < .1:
            self.correct = True

        Fx = kpP * eP[0] + kdP * eV[0]
        Fy = kpP * eP[1] + kdP * eV[1] - self.Fg[1]
        Fy = np.maximum(0.2 * self.Fc_max, np.minimum(Fy, 0.8 * self.Fc_max))
```

Figura 4 – Seção responsável pelo controle do sistema

3. Referências

[1] https://www.pygame.org/wiki/about

[2] https://numpy.org/

[3] https://docs.python.org/3/library/os.html

[4] Curso de criação de jogos com Python – Pygame, disponível em https://www.cursou.com.br/informatica/criacao-jogos-python-pygame/

[5] KATSUHIKO OGATA, Engenharia de controle moderno, 5ª edição, 2014.