Universidade Federal de Uberlândia

FEELT49081

Sistemas Digitais para Mecatrônica

Trabalho 1 – Simulação de Drone 2D

Gabriel Elisbão da Silva – 11711EMT005

Mateus Antônio Souza Silva – 11711EMT013

Tercio de Melo Alves Júnior – 11711EMT016

Thiago de Souza Alves – 11711EMT002

07 de outubro de 2021

1. Descrição de módulos/bibliotecas

Foram usadas as seguintes bibliotecas:

1.1. pygame

Pygame (a biblioteca) é uma biblioteca de linguagem de programação python de código livre e aberto para fazer aplicativos multimídia como jogos. Dessa forma, conceitos importantes no nosso trabalho como objetos surfaces foram obtidos por essa biblioteca.

1.2. copy

Como em python tudo é endereço, quando fazemos atribuição na verdade estamos linkando 2 endereços na memória, o que implica em alteração nas duas. Para copiar o valor é necessário o que chamamos de deep copy e essa ferramenta é conseguida através dessa biblioteca

1.3. sys

Essa biblioteca é responsável pela captura e tratamento de eventos como o keypress que usamos para mover o drone, mas explicando melhor, essa biblioteca serve para utilizarmos o sistema para as funções dentro do código.

1.4. math

Biblioteca responsável por funções matemáticas como integração, raiz quadrada entre outros, essa biblioteca se faz necessário para o nosso sistema de controle principalmente.

1.5. logging

Essa biblioteca tem classes que possibilitam a gravação de loggings para eventos do sistema entre outros. Através dela podemos debugar o sistema e entender onde há falhas corrigindo-as de forma mais eficaz.

1.6. time

Biblioteca que fornece funções de tempo, como o tempo em determinado instante, utilizamos ela para traduzir instantes em frames além de pontos como derivada entre outros.

1.7. numpy

Biblioteca para aceleração por gpu. Usamos no nosso programa grandes matrizes, operações com elas quando direcionadas a cpu demoram pela quantidade de n´núcleos. Por outro lado, na gpu temos (por mais que mais simples) mais n´núcleos, se tornando assim perfeitas para essas tarefas.

1.8. threading

Biblioteca para utilizarmos threads, com isso aumentarmos a performance do nosso programa pelo princípio da paralelização de processos.

1.9. dataclasses

Este módulo fornece um decorador e funções para adicionar automaticamente métodos especiais gerados, como e para classes definidas pelo usuário.

2. Descrição da integração do sistema

Para se desenvolver a simulação, é necessário que se descreva o comportamento da planta, para tal existe a função “x_dot”, onde nela se encontram os parâmetros da planta e as equações diferenciais que a descrevem.

Utilizando o método numérico de Runge-Kutta, a função “rk4” soluciona numericamente o sistema para se obter os próximos estados da planta, esses estados são calculados ao final de cada loop de simulação.

A classe “IteractiveSimulator” será a responsável por realizar os cálculos de simulação do controlador. Sendo a “fillTimeWindow” responsável por iniciar a simulação, com a intenção de gerar dados antes de se criar a interface do jogo, além é claro de se estabilizar a planta antes de se aplicar as posições de referência. A função “nextStep” será responsável por continuar a execução dos cálculos quando o jogo terminar de ser inicializado.

Os cálculos são realizados para simular a tempo contínuo, para isso, o passo de simulação de tempo contínuo é bem menor que o intervalor de atuação do controlador, de forma que a simular progride no loop e quando chegado no intervalo do controlador o mesmo age, realizado por meio dos ticks, ou passos de simulação, sendo contados durante a execução do programa, usando a biblioteca “time” para isso.

Quando executado o “main”, é estipulado um alvo inicial na posição (0, 0), mandando a “fillTimeWindow” simular os instantes iniciais com os parâmetros enviados para o “controler.IteractiveSimulator”, logo em seguida é criada uma thread para lidar com o “simulationHandle” que realiza um loop infinito com a função “nextStep”, essa thread tem o daemon true para ser independente e encerrar junto com o programa. Outra thread será iniciada, sendo responsável pelo “mainFrameHandle”, ou seja, o pygame. Enquanto isso o programa principal estará em um loop infinito buscando waypoitns para o controlador.

A função “mainFrameHandle” é responsável pela renderização da tela, ela é executada em uma thread separada, criada na função “main()” com o auxílio da biblioteca “threading” .

Essa função recebe como parâmetro o simulador instanciado na função “main()”.

Algumas variáveis são criadas, como o “frameconfig” que define o tamanho da screen, além de objetos gerados com o auxílio da biblioteca do “pygame”, janela (screen), player que é o drone, plano de fundo (background).

O método blit do objeto screen que posiciona o drone na tela.

Um objeto do tipo “Drone” é criado, além de variáveis de alvo, estado e tempo.

O loop principal do programa definido com while.

Com o auxílio das bibliotecas “copy” e “np” pegamos o último valor gerado pelo controlador do objeto “simulator”. Além de pegar os valores que estão sendo calculados pelo controlador.

Um laço de for é usado dentro do loop para mapear os eventos, com as condições de finalizar a thread caso um evento de QUIT do “pygame” ou executar o método que define as ações caso um evento do teclado.

No código, “memory_ocup” armazena o tamanho da memória ocupada pelo “state_vector”, que é usado para filtrar e analisar a qualidade dos valores armazenados através da análise da variação dos dados conforme as derivadas.

No fim do loop a posição do drone é atualizada assim com a tela.

A classe Drone define métodos que realizam a movimentação do drone, todos os métodos recebem “self”, indicando alteração da posição do próprio objeto. O método de construção “__init__” carrega as imagens, fundo e posição inicial do drone.

“Target” é uma classe usada pelo controlador, define alguns métodos para cálculo do próximo ponto de movimentação do drone de acordo com os parâmetros para simular a estabilidade. O método de construção “__init__” carrega parâmetros e configurações iniciais.

3. Conceitos extras utilizados

- Thread: Com o auxílio da biblioteca “threading” criamos 2 threads para a função de renderização e para os cálculos do controlador. Melhorando de forma significativa o desempenho do programa;
- Logs: A biblioteca “logging” utilizada para printar os logs durante a execução do programa no console e acompanhar as rotinas;
- Princípios de POO: Criação de classes e métodos, instancia de objetos;
- Programação Orientada a Função: Passagem de funções como variável.