UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Faculdade de Engenharia Mecânica

Graduação em Engenharia Mecatrônica

Sistemas digitais para Mecatrônica

Prof.: Éder Alves de Moura

Trabalho drone

Baltazar Alic Borges da Silva – 11711EMT022

Rodrigo Alves Prado – 11521EMT003

Fernando Rabelo Fernandes Junior – 11611EMT020

2022

1. DESCRIÇÃO DAS BIBLIOTECAS UTILIZADAS

1.1. Pygame

Pygame é uma biblioteca escrita em Python e baseada em SDL. Voltada para o desenvolvimento de games e interfaces gráficas, o Pygame fornece acesso a áudios, teclados, controles, mouses e hardwares gráficos via OpenGL e Direct3D. Por ser multiplataforma o Pygame pode rodar em qualquer sistema operacional com alterações mínimas de código no funcionamento de um ou outro. Os mais populares sistemas operacionais do mercado hoje são Windows, Mac OS X, Linux, Android e iOS.

1.2. Numpy

NumPy é uma abreviação para Numerical Python, o qual é uma poderosa biblioteca da linguagem de programação Python, que consiste em objetos chamados de arrays. Além disso, essa biblioteca vem com uma coleção de rotinas para processar arrays.

O NumPy fornece um grande conjunto de funções e operações de biblioteca que ajudam os programadores a executar facilmente cálculos numéricos. Ele é bastante útil para executar várias tarefas matemáticas como integração numérica, diferenciação, interpolação, extrapolação e muitas outras. Além disso também conta com constantes matemáticas como ‘π’ e ‘e’, e operação como seno, cosseno e logaritimo.

Os códigos referentes a essa biblioteca vem com ‘np.’ antes da função ou constante utilizada como por exemplo, “np.pi” e “np.sin(‘variavel’)”.

2. DESCRIÇÃO DA INTEGRAÇÃO DO SISTEMA

2.1. Arquivo game.py

Para a criação de um jogo a primeira parte a ser trabalhada deve ser o loop principal da aplicação, uma parte do programa que vai ser repetida varias vezes onde são chamadas as funções que definem o funcionamento do jogo.

Para o nosso jogo o loop principal está como na imagem a seguir:

```
while True:
    #taxa de clock
    clock.tick(FPS)

    if menu_open:
        menu_res = menu()
        if menu_res != 0:
            menu_open = 0
            comandar = menu_res - 1
            if comandar:
                pygame.time.wait(500)
                instru_open = 1

    else:

        # atualiza o background na tela
        screen.blit(background, (0, 0))

        if comandar and instru_open:
            instru_f()  #abre as intruções
        else:
            drone_f()   #mostra o drone na tela
```

No código acima é possível observar que existem 3 partes constituintes dessa aplicação, que são: o menu, representado pela função '$menu()$' que está localizada no arquivo interface.py que será mostrado depois, as instruções na parte de controle pelo teclado que está representado pela função '$instru\_f()$' e pela função que desenha, colhe os dados do teclado e calcula o movimento do drone na tela representada pela função '$drone\_f()$'.

Também foram introduzidas 3 variáveis logicas, '*menu_open*' que quando é verdadeira mostra o menu na tela, '*instru_open*' que mostra na tela as instruções e '*comandar*' que habilita o controle pelo teclado.

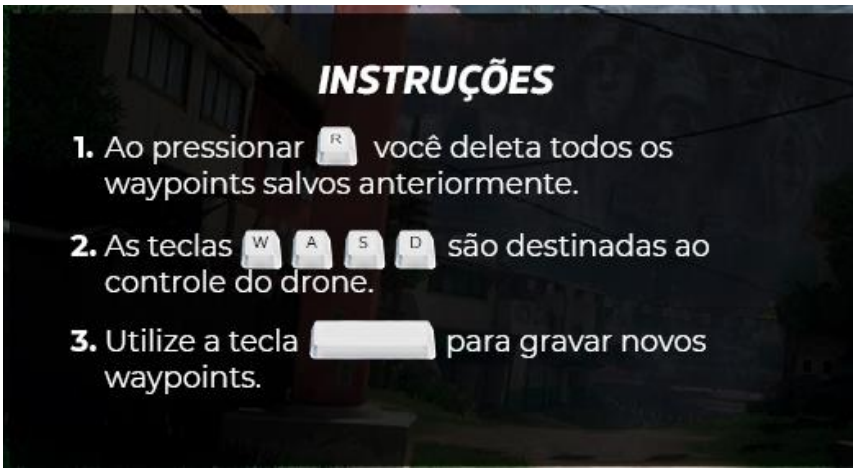
```
#interface grafica
def instru_f():
    global i, instru_open

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()

        if event.type == pygame.KEYDOWN:
            instru_open = 0

    #atualização do drone na tela
    i += 1
    if i >= 3:
        instru_button.draw()
        pygame.display.update()
        pygame.display.flip()
        i = 0
        if pygame.mouse.get_pressed()[0]:
            instru_open = 0
```

Dentro da função '*instru_f*()' e colhido os dados do teclado e do mouse, se qualquer um for utilizado as instruções somem da tela. A tela de instruções é como vista abaixo:



Dentro da função *'drone()'*, primeiramente o drone é desenhado na tela a uma taxa 4 vezes menor que que o loop principal do jogo que está a 100 FPS, quando o botão de menu *'menu_button'* é pressionado o programa volta pro menu e deixa de apresentar o drone na tela.

```
#modificação da frequencia de apresentação na tela
i += 1
if i >= 3:
    drone.update()
    drone.draw()
    menu_button.draw()
    if menu_button.click():
        menu_open = 1
    pygame.display.update()
    pygame.display.flip()
    i = 0
```

Ainda na função *'drone()'*, a função *'pygame.event.get()'* é chamada para atualizar os dados de teclado e mouse na aplicação, como também configurar as variáveis logicas responsáveis pelo movimento do drone.

```
#coletar dados dos perifericos
for event in pygame.event.get():
    if event.type == QUIT:
        pygame.quit()
    #controlador de posição pelo teclado

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_a:
            esquerda = True
        if event.key == pygame.K_d:
            direita = True
        if event.key == pygame.K_w:
            cima = True
        if event.key == pygame.K_s:
            baixo = True
        if comandar and not menu_open:
            if event.key == pygame.K_SPACE:
                gravar(True)
            if event.key == pygame.K_r:
                gravar(False)
```

Dentro da classe '*Drone*' estão contidas três funções, update, move e draw. Na função '*update*()' o drone recebe o efeito de animação (onde acontece a troca de frames do drone e onde ele recebe a transformação de rotação) como visto na imagem abaixo:

```
def update(self):
    #troca as imagens do drone para a animação das hélices
    self.flip += self.passo
    if self.flip >= len(self.animado)-1 or self.flip == 0:
        self.passo *= -1
    self.image = self.animado[int(self.flip)]
    self.image = pygame.transform.scale(self.image,
                                        (int(self.image.get_width() * self.scale),
                                         int(self.image.get_height() * self.scale)))
    #rotação do drone na tela
    self.image = pygame.transform.rotate(self.image, self.angle)
    # transformação para fazer o drone girar em relação ao centro
    size = self.image.get_size()
    self.rect.x = self.x - (size[0] - self.size[0])/2
    self.rect.y = self.y - (size[1] - self.size[1])/2
```

Na função move são colhidos os dados de movimentação do drone (posição e ângulo) e são passados os controles (obtidos do teclado) para a parte de simulação do movimento do drone, realizado pela função '*movimenta*()' que está descrita no arquivo '*fisica.py*'.

Também na função move, são feitas as transformações do ângulo de '*rad*' para '*graus*' e também as transformações dos sistemas de coordenadas da simulação para o sistema de coordenadas da tela do jogo.

```
def move(self, esquerda, direita, cima, baixo, comandar):

    self.pos, angle = movimenta(esquerda, direita, cima, baixo, comandar)
    self.angle = angle * 180 / np.pi

    #Transformação linear da posição
    self.x = rx*self.pos[0]        ############################
    self.y = rx*self.pos[1] - 30       ########################
```

2.2. **Arquivo física.py**

O arquivo física.py teve sua maior parte escrita pelo professor logo a lógica por trás deste não será abordada. Mas algumas modificações foram feitas, a primeira delas é a função '*gravar*()' que serve para substituir os waypoints do caminho do drone ou adicionar outros.

```
def gravar(reset):
    global r_, r_ID, r_IDN, r_points
    if not reset:
        r_ID = 0
        r_IDN = 0
    elif r_IDN == 0:
        r_points = np.array(x[2:4, k - 1], ndmin=2).transpose()
        r_IDN += 1
    elif r_IDN > 0:
        r_points = np.concatenate((r_points,
                                   np.array(x[2:4, k - 1], ndmin=2).transpose()), axis=1)
        r_IDN += 1
    r_ = r_points
```

O loop do programa original também foi transformado em na função '*movimenta*()', a qual é chamada pela função '*move*()' na classe drone, dentro dessa função foi acrescentado ao movimento a partir de waypoints uma condição que limita sua execução à apenas quando o drone não é controlado pelo teclado. Além disso foi acrescentado um mecanismo de caminho infinito onde os waypoints são percorridos em loop.

```
# Colocar os way points na referencia
if not comando:
    r_ = r_points
    ref = r_[:, r_ID]
    eP = ref - r_k
    if np.linalg.norm(eP) < .1 and r_ID < r_IDN:
        r_ID += 1
    if np.linalg.norm(eP) < .1 and r_ID >= r_IDN:
        r_ID = 0
    if r_ID == r_IDN:
        ref = r_[:, r_ID]
```

Quando a movimentação pelo teclado está ativa o drone recebe referencias da movimentação pelo teclado a partir das variáveis logicas de movimento, e ainda limita o controle para além dos limites da tela.

```
# setar as referencias pelo teclado
if comando:
    r_ID = 0
    if esquerda and ref[0] >= 0:
        ref += [-0.05, 0]
        if ref[0] < 0:
            ref[0] = 0
    if direita and ref[0] <= 6:
        ref += [0.05, 0]
        if ref[0] > 6:
            ref[0] = 6
    if cima and ref[1] >= -0.4:
        ref += [0, -0.05]
        if ref[1] < -0.4:
            ref[1] = -0.4
    if baixo and ref[1] <= 3.5:
        ref += [0, 0.05]
        if ref[1] > 3.5:
            ref[1] = 3.5
```

A ação de controle do drone também foi limitada para que o drone não receba sinal muito altos e desestabilize, a logica implementada está na imagem abaixo:

```
# limitar a ação de controle
if comando == 0:
    if abs(eP[0]) > 2:
        eP[0] = 2*np.sign(eP[0])
    if abs(eP[1]) > 2:
        eP[1] = 2*np.sign(eP[1])
else:
    if abs(eP[0]) > 1.5:
        ref[0] = r_k[0] + 1.5 * np.sign(eP[0])
    if abs(eP[1]) > 1.5:
        ref[1] = r_k[1] + 1.5 * np.sign(eP[1])
```

A simulação do drone e o controle são feitos sob tempos diferentes, com esse objetivo foi feito um loop onde a simulação do drone é efetuada 10 vezes a cada vez que o controle e calculado, para a adequação dos tempos de amostragem e controle.

```
#Ajuste de tempos entre simulação e controle
for j in range(10):
    x[:, k + 1 + j] = rk4(tc[k + j], h, x[:, k + j], w_[0, :])
k += 10
```

2.3. **Arquivo interface.py**

Neste arquivo foram escritos os códigos referentes aos elementos da interface da aplicação como os botões e o menu.

Os botões são uma classe assim como o drone, mas ao invés de receber uma função para movimento, recebe uma função para clique, a qual funciona armazenando a posição do cursor do mouse e avaliando se ele foi clicado enquanto estava em cima do botão, como mostrado no código abaixo:

```
def click(self):
    #get mouse position
    pos = pygame.mouse.get_pos()
    self.image = pygame.transform.scale(self.animado[0],
                                        (int(self.width * self.scale),
                                         int(self.height * self.scale)))
    #check mouseover and clicked conditions
    if self.rect.collidepoint(pos):
        self.image = pygame.transform.scale(self.animado[1],
                                            (int(self.width * self.scale),
                                             int(self.height * self.scale)))
        if pygame.mouse.get_pressed()[0] == 1:
            self.clicked = True
    res = self.clicked
    self.clicked = False
    return res
```

O menu é uma função com um loop próprio como um programa independente, este mostra na tela 2 botões referentes aos modos de waypoints e controlado pelo teclado e quando qualquer um desses é clicado o loop é encerrado e o programa passa a apresentar as outras partes da aplicação na tela.

```
while res == 0:
    clock_menu.tick(20)
    screen.blit(BG, (0, 0))
    track_button.draw()
    way_button.draw()
    if track_button.click():
        res = 2
        screen = pygame.display.set_mode((1000, 600))
    if way_button.click():
        res = 1
        screen = pygame.display.set_mode((1000, 600))
    pygame.display.update()
    #event handler
    for event in pygame.event.get():
        #quit game
        if event.type == pygame.QUIT:
            pygame.quit()
```

A aparência do menu é como mostrada abaixo:

3. REFERÊNCIAS SOBRE CONCEITOS EXTRAS UTILIZADOS

Pygame:

https://www.youtube.com/watch?v=xcfjgeYRkQM&list=PLJ8PYFcmwFOxtJS4EZTGEPxMEo4YdbxdQ&index=14

Numpy:

https://www.w3schools.com/python/numpy/default.asp

PyGame Beginner Tutorial in Python - Adding Buttons:

https://www.youtube.com/watch?v=G8MYGDf_9ho&t=973s