UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE LORENA

Projeto de automação de drone

Gabriel Costa Oliveira
Kauan Ramos Lima
Mario Hideki Murata
Rafael Cândido Reis
Victor Gabriel Moreira da Silva

Lorena - SP
2023

RESUMO

O foco deste projeto está na automação de drones por meio de software, algoritmos e sensores avançados, utilizando a linguagem de programação Python. A metodologia detalha a implementação do controle de movimento de drone e de reconhecimento facial, destacando as bibliotecas e o código principal desenvolvido. Os resultados dos testes evidenciam o sucesso no controle de movimento de drone e a capacidade de reconhecimento facial, culminando na habilidade da aeronave de reconhecer o rosto do usuário. A escolha cuidadosa de módulos e bibliotecas e os testes bem-sucedidos validam a eficácia da abordagem adotada. Este estudo não apenas contribui para o campo de automação de drones, mas também busca democratizar o acesso a essa tecnologia ao elaborar um código em Python, tornando as ferramentas disponíveis acessíveis a qualquer indivíduo interessado.

Palavras-chave: Drone. Python. Voo. Reconhecimento facial.

SUMÁRIO

1. INTRODUÇÃO

Entende-se por drone uma aeronave que opera sem piloto a bordo. Também conhecido como Veículo Aéreo Não Tripulado (VANT) ou Veículo Aéreo Remotamente Pilotado (VARP), este dispositivo é controlado remotamente por um operador humano e segue um conjunto pré-programado de instruções. Eles podem variar em tamanho, desde pequenos modelos de brinquedo até grandes aeronaves utilizadas para fins militares, comerciais ou de segurança.

Consoante Ciriaco (2015), os anos 80 marcaram grande desenvolvimento para os drones, principalmente em cenários de guerra, como quando o exército da África do Sul usou drones israelenses em combates contra Angola em 1981. Apesar de ter sido criado para fins nada nobres, os drones — equipados com diferentes tipos de sensores e câmeras que podem capturar imagens e dados em tempo real — têm ampla variedade de aplicações em diversos setores, incluindo:

a) fotografia e videografia: são frequentemente utilizados para capturar imagens aéreas e vídeos de paisagens, eventos e atividades esportivas;
b) agricultura: na agricultura de precisão, drones podem ser usados para monitorar cultivos, avaliar a saúde das plantas e otimizar o uso de fertilizantes (ENTERPRISE, 2021) .
c) mapeamento e topografia: podem ser empregados para criar mapas tridimensionais de áreas geográficas, sendo úteis em projetos de construção, urbanismo e monitoramento ambiental (MAPPA, 2022);
d) entregas: algumas empresas exploram o uso de drones para entregas de pacotes, especialmente em áreas urbanas — como a Amazon em seu projeto Amazon Prime Air (AMAZON, 2022);
e) inspeção e manutenção: drones são usados para inspecionar estruturas, como linhas de transmissão, turbinas eólicas, pontes e dutos, proporcionando uma maneira eficiente e segura de avaliar condições. Na Petrobras, são utilizados em larga escala na inspeção de manutenção em refinarias e plataformas (PETROBRAS, 2021).

A automação de drones pode ser alcançada por meio de software, algoritmos e sensores avançados integrados a eles. Neste trabalho, adotou-se a linguagem de programação em Python para controle de movimento da aeronave e reconhecimento facial.

O objetivo principal é desenvolver um código capaz de realizar o controle de voo do dispositivo e, por meio da câmera acoplada a este, detectar rostos. A justificativa é a elaboração de um módulo em Python para que qualquer indivíduo possa acessar as ferramentas disponibilizadas pelo código.

2. METODOLOGIA

2.1 Controle de movimento de drone

As bibliotecas utilizadas na parte do código associada ao controle de movimento do drone são as seguintes:

a) *time*: empregada para lidar com operações relacionadas ao tempo. Ela fornece funções que permitem medir o tempo, realizar pausas no programa, formatar datas e horas, entre outras operações relacionadas ao tempo (LISBOA, 2023);
b) *pygame*: adiciona funcionalidades em cima do SDL, que é uma biblioteca multiplataforma que abstrai os componentes multimídia do computador como áudio e vídeo, permitindo o desenvolvimento com maior facilidade de programas que lidam com estes recursos (ROCHA, 2017).
c) *cv2*: também conhecida como *OpenCV*, trata-se da principal biblioteca de código aberto para visão computacional, processamento de imagem e aprendizagem de máquina (PASSARELLI, 2017). Neste caso, é aplicada para comunicação com a câmera do drone;
d) *djitellopy*: utilizada para comunicação com o drone;

2.1.1 Arquivo *test.py*

O arquivo utilizado pelo código principal de controle de drone foi nomeado de *test.py* e sua explicação é dada a seguir:

A figura 1 expõe as bibliotecas usadas no arquivo e a figura 2 a classe relacionada ao drone, a qual gera a maior parte da conexão e das atribuições principais.

Figura 1 - Bibliotecas usadas no arquivo *test.py*.

```
from djitellopy import tello
import pygame as pg
import time
import cv2
```

Fonte: Autores.

Figura 2 - Classe *Drone*.

```
def __init__(self,wth=100,hght=100):
    self.tello = tello.Tello()
    self.screen = pg.display.set_mode((wth, hght))
    print(self.tello.get_battery())
    self.tello.connect()
    self.tello.streamon()
```

Fonte: Autores.

A função *init* apresentada na figura 2 define as primeiras atividades a serem feitas ao chamar a classe *Drone*. A variável *self.tello* define a forma de se referir às funções do drone, como *connect* e *streamon*; a variável *self.screen* define a exibição do *pygame*, o usuário, quando chamar o objeto da classe, pode alterar o tamanho do display do *pygame* por meio de *wth* e *hght*. O *print(self.tello.get_battery())* imprime a bateria do drone. A variável *self.tello.connect()* refere-se à função do drone Tello que conecta o drone ao código, já a *self.tello.streamon()* refere-se à função do drone Tello que inicializa o modo de streaming, isto é, inicia a câmera do drone.

A figura 3 exibe a função *KeyBoardInput*, a qual permite que o usuário escolha os botões para controles básicos e para controle de velocidade do drone.

Figura 3 - Definição da função *KeyBoardInput*.

```
def KeyBoardInput(self, b1='LEFT', b2='RIGHT', b3='UP', b4='DOWN', b5='w', b6='s', b7='a', b8='d',b9='x', b10='z', b11='c', b12='v', speed=50):

    lr, fb, up, yv = 0, 0, 0, 0
    keys = pg.key.get_pressed()

    b1 = getattr(pg,'K_{}'.format(b1))
    b2 = getattr(pg,'K_{}'.format(b2))
    b3 = getattr(pg,'K_{}'.format(b3))
    b4 = getattr(pg,'K_{}'.format(b4))
    b5 = getattr(pg,'K_{}'.format(b5))
    b6 = getattr(pg,'K_{}'.format(b6))
    b7 = getattr(pg,'K_{}'.format(b7))
    b8 = getattr(pg,'K_{}'.format(b8))
    b9 = getattr(pg,'K_{}'.format(b9))
    b10 = getattr(pg,'K_{}'.format(b10))
    b11 = getattr(pg,'K_{}'.format(b11))
    b12 = getattr(pg,'K_{}'.format(b12))
```

Fonte: Autores.

Há doze parâmetros que vão de *b1* a *b12* e referem-se aos botões do teclado; todos já possuem teclas atribuídas, porém o usuário pode alterá-las chamando a função e digitando o parâmetro e a tecla de interesse. *b1* e *b2* são a velocidade esquerda e direita do drone, respectivamente; *b3* e *b4* foram criadas para a velocidade para frente e para trás, respectivamente; *b5* e *b6* são para a velocidade para cima e para baixo, respectivamente; *b7* e *b8* são para a velocidade de guinada; *b9* é para comandar o drone para pousar; *b10* é para comandar o drone para decolar; *b11* é para iniciar a transmissão; e *b12* é para encerrar a transmissão.

A figura 4 a seguir mostra o uso da sequência de *if*s na função *KeyBoardInput* para alterar o valor das velocidades. Trata-se da mudança da velocidade do drone quando um botão é pressionado, o qual também pode ser alterado pelo usuário.

Figura 4 - sequência de *if*s na função *KeyBoardInput*.

```
if keys[b1]:
    lr = -speed
if keys[b2]:
    lr = speed
if keys[b3]:
    fb = speed
if keys[b4]:
    fb = -speed
if keys[b5]:
    up = speed
if keys[b6]:
    up = -speed
if keys[b7]:
    yv = -speed
if keys[b8]:
    yv = speed
if keys[b9]:
    self.tello.land()
if keys[b10]:
    self.tello.takeoff()
if keys[b11]:
    self.tello.streamon()
if keys[b12]:
    self.tello.streamoff()
```

Fonte: Autores.

A figura 5 evidencia a definição do atraso quando um botão é pressionado e o retorno dos valores de *lr*, *fb*, *up* e *yv*.

Figura 5 - Atraso e retorno.

```
time.sleep(0.05)

return [lr, fb, up, yv]
```

Fonte: Autores.

A figura 6 expõe a função *cascata* utilizada por meio do método de Viola-Jones para detecção facial. A variável *faceCascade* recebe o caminho para encontrar o arquivo *haarcascade_frontalface_default.xml*, que é instalado junto com a biblioteca *OpenCV*. A variável *imgGray* transforma a imagem já processada pela função *get_frame_read().frame* de BGR (padrão do *OpenCV*) para escala cinza (escala que permite o reconhecimento pelo método de Viola-Jones). A variável *faces* executa o arquivo *haarcascade_frontalface_default.xml* para *imgGray* e é definido os parâmetros de precisão *f* e *p* do sistema, assim a função retorna as "faces" identificadas com coordenadas iniciais das faces (x, y) e finais (x+w, w+h).

Figura 6 - Definição da função cascata.

```
def cascata(self,img, f, p):

    faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(imgGray, f, p)
    return faces
```

Fonte: Autores.

A figura 7 mostra o método *trackFace*, o qual, ao receber os valores de *faces* e *img*, cria um retângulo ao redor da face e uma lista com o cálculo da área do retângulo e das coordenadas de seu centro (cx, cy).

Figura 7 - Definição da função *trackFace.*

```
def trackFace(self,faces,img):
    self.myFaceList = []
    self.myFaceArea = []

    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h),(0,0,255), 2)
        cx = x + w //2
        cy = y + h //2
        area = w * h
        cv2.circle(img,(cx,cy), 5, (0,255,0), cv2.FILLED)
        self.myFaceList.append([cx,cy])
        self.myFaceArea.append(area)
    if len(self.myFaceArea) != 0:
        i = self.myFaceArea.index(max(self.myFaceArea))
        return img, [self.myFaceList[i],self.myFaceArea[i]]
    else :
        return img, [[0,0], 0]
```

Fonte: Autores.

A função *main* presente na figura 8 a seguir executa os métodos anteriores de maneira que processa e lê a imagem capturada pelo drone, redimensiona a imagem processada e utiliza o método de Viola-Jones para retornar as faces. As faces são contornadas por um retângulo e, por fim, essa imagem é mostrada em uma janela. O usuário pode definir o nome e o tamanho da janela que mostrará a imagem, o delay de *frame* e os parâmetros de precisão da função *cascata*.

Figura 8 - Definição da função *main.*

```
def main(self, window_name="Frame", width=360, height=240, delay=1, f=1.2, p=5):

    self.cap = self.tello.get_frame_read().frame
    img = cv2.resize(self.cap, (width,height))
    faces = Drone.cascata(self,img,f,p)
    img = Drone.trackFace(self,faces,img)
    cv2.imshow(window_name, img[0])
    cv2.waitKey(delay)
```

Fonte: Autores.

2.2 Código principal

A figura 9 mostra o primeiro passo para construção do código principal de controle de drone:

Figura 9 - Importação da biblioteca *pygame* e do arquivo *test.py*.

```
import pygame as pg
import test
import cv2
```

Fonte: Autores.

A biblioteca *pygame* é importada para criar os eventos em que se clica e solta um botão, bem como para encerrar o código ao fechar o display do pygame. O arquivo test.py é importado para que o código vigente possa usufruir de suas ferramentas e a biblioteca *cv2* é importada para fechar o display da imagem da câmera do drone. A figura 10 destaca o objeto *dr* da classe Drone e uma variável *running* criada para dar o start no looping do código.

Figura 10 - Variável *running* e objeto *dr*.

```
running = True
while running:
    dr.main()

    for event in pg.event.get():

        if event.type == pg.QUIT:
            running = False
            cv2.destroyAllWindows()
            dr.tello.land()
        if event.type == pg.KEYDOWN:
            vals = dr.KeyBoardInput()
            dr.tello.send_rc_control(vals[0], vals[1], vals[2], vals[3])
        if event.type == pg.KEYUP:
            dr.tello.send_rc_control(0,0,0,0)
```

Fonte: Autores.

Os eventos de *pygame* no código servem para definir o que acontece caso sejam cumpridos seus requisitos. O evento *pg.QUIT* define que se o *display* do *pygame* for fechado, o looping será encerrado, a janela da câmera do drone será fechada e o drone receberá o comando de aterrissar. O evento *pg.KEYDOWN* define que se uma tecla for apertada, a variável *vals* rodará a função *KeyBoardInput()* (que retorna uma lista com as velocidades) e envia as velocidades para o drone. O evento *pg.KEYUP* define que, se uma tecla deixar de ser apertada, retornará para o drone as velocidades 0, 0, 0, 0.

dr.main() resgata a função *main* da classe *Drone* e abre a janela contendo a câmera do drone em tempo real (porém demora certo tempo até que ela se estabilize) e aplica as funções relacionadas a reconhecimento facial.

2.3 Tutorial para o usuário

Para a utilização do código por usuários externos, serão necessários os seguintes requisitos: As bibliotecas *djitellopy*, *time, pygame* e *cv2* devem estar instaladas no dispositivo a ser utilizado. O arquivo *test.py* está disponibilizado no github pelos autores e deve estar transferido do repositório online para o local pelo usuário. Conexão wifi disponível no dispositivo a ser controlado o drone.

Com todos esses requisitos, já é possível lançar o drone. Primeiramente, deve-se ligar o drone segurando o botão em seu lado até que luzes multicoloridas comecem a piscar, em seguida o coloque em uma superfície plana sem obstruções acima do drone. Em seguida, o dispositivo a ser utilizado para o controle deve ser conectado com o drone "**TELLO-ABF520**" por via wifi ou a conexão disponibilizada pelo drone utilizado. Feito isso, só é necessário abrir um novo arquivo e codificar.

Primeiramente, o usuário precisa importar as bibliotecas *pygame* e *cv2*, além do arquivo *test* para assim poder acessar os módulos de controle e imagem do módulo. Logo em

seguida, é necessário criar um objeto da classe *Drone* e, caso necessário, é possível mudar o tamanho da janela do *pygame* colocando dentro do argumento da classe os valores de *width* e *height da tela.*

Logo após, é necessário criar um looping a partir de uma variável usando o método *True/False,*ou seja, enquanto essa variável for verdadeira o código irá funcionar normalmente, porém quando for falsa, encerrará o código. Logo em seguida, o usuário, caso queira que o código mostre a imagem e ative o reconhecimento facial, precisará puxar a função *main* do objeto dentro do *while*.

Após isso, é necessário, com a ajuda da biblioteca *pygame* chamar um evento usando *for event in pygame.event.get():* e dentro desse *for* definir 3 eventos utilizando *ifs*. O primeiro sendo *pygame.QUIT,* que dentro possuirá *running=False* (para fechar o código), *cv2.destroyAllWindows()* (que fecha o display da imagem) e *dr.tello.land()* (que manda para o drone o comando de pousar). O segundo é *pygame.KEYDOWN* que dentro possuirá uma variável que chama a função *KeyBoardInput()* e logo em seguida chamar o comando do drone usando *dr.tello.send_rc_control()* e colocar os 4 itens da lista (que está contida em na variável) no argumento para que seja enviado para o drone. Por último, o evento *pygame.KEYUP* que retornará a velocidade 0,0,0,0 para o drone quando uma tecla for solta. Com isso, é só rodar o código e ele começará a processar os dados.

Quando o código processar os dados, uma guia será aberta na interface do dispositivo, em que o usuário deve clicar-lá para que seja possível se utilizar dos comandos do drone, podendo assim clicar na tecla **Z** que ativará o lançamento do drone e levantará voo, seguido de um estado estacionário. Para o seu controle, serão utilizadas as teclas **WASD** para a subida, rotação para esquerda, descida e rotação para a direita do drone, respectivamente, além das setas do teclado, sendo elas: **SETA PARA CIMA, SETA PARA ESQUERDA, SETA PARA BAIXO, SETA PARA DIREITA** utilizadas para o movimento para frente, esquerda, trás e direita do drone, respectivamente. É importante ressaltar que as teclas só precisarão ser seguradas para o comando, definido pelas funções *pg.KEYUP* e *pg.KEYDOWN.*

Além disso, o referencial para o controle do drone sempre virá da direção que sua câmera está apontando, ou seja, as definições de direita, esquerda, cima, baixo, entre outras, vêm do display que aparecerá na tela do dispositivo mostrando a imagem gravada da câmera do drone.

Esse display mostrará a imagem da câmera do drone em tempo real e mostrará também o reconhecimento facial caso algum rosto apareça na câmera.

Finalmente, caso o usuário queira terminar o programa de controle, será necessário clicar na tecla **X** para rodar o comando de *land* do drone e pousá-lo.

Além disso, o usuário poderá modificar as teclas e a velocidade do drone através da função *KeyBoardInput*. Modificando a string que segue os valores de *b1* a *b12*, o usuário pode definir quaisquer teclas para o controle do drone, incluindo os comandos de lançamento e pouso. Junto disso, o valor *speed* também pode ser modificado pelo usuário para definir a velocidade do drone, sendo o valor mínimo 1 e máximo 100, podendo chegar a 28.8 km/h.

3. RESULTADOS E DISCUSSÕES

3.1 Teste de controle de movimento

O primeiro teste para verificar a funcionalidade do código criado foi bem-sucedido. Utilizou-se um Drone Tello Boost para tal atividade. A velocidade, que anteriormente estava programada para 20, foi modificada para 50 para melhor rendimento de voo.

3.2 Teste para reconhecimento facial

As partes do código no que tange ao reconhecimento facial foram criadas separadamente e, posteriormente, integradas ao código de controle de movimento da aeronave. A priori, a câmera do notebook foi utilizada durante o período de criação da programação para detecção facial e, uma vez que demonstrou bons resultados, testes com a câmera do drone foram feitos. Vale ressaltar que houve sucesso em fazer com que o drone seguisse a face do usuário conforme este se movimentava.

4. CONCLUSÃO

Destarte, o desenvolvimento do código para controle de movimento de drones, utilizando a linguagem de programação Python e integrando funcionalidades de reconhecimento facial, representa um avanço significativo na automação deste tipo de tecnologia. A abordagem adotada neste trabalho permite não apenas o controle eficiente do voo do drone, mas também a implementação de recursos avançados, como o reconhecimento facial em tempo real.

A escolha de bibliotecas específicas, como *pygame*, *OpenCV* e *djitellopy* para implementação do código demonstra a abordagem prática e técnica adotada. Os testes bem-sucedidos de controle de movimento, incluindo a modificação da velocidade do drone e a realização de testes para reconhecimento facial, culminando na capacidade do drone de seguir o rosto do usuário, validam a eficácia do código desenvolvido.

A possibilidade de acesso às ferramentas disponibilizadas pelo código reforça a intenção de tornar essa solução acessível a qualquer indivíduo interessado. Assim, este trabalho não apenas contribui para a automação de drones, mas também para a democratização do uso dessa tecnologia, abrindo portas para diversas aplicações inovadoras e acessíveis a um público mais amplo.

REFERÊNCIAS

AMAZON. **How Amazon is building its drone delivery system**. 2022. Disponível em: https://www.aboutamazon.com/news/transportation/how-amazon-is-building-its-drone-delivery-system. Acesso em: 15 dez. 2023.

CIRIACO, Douglas. **O que é drone?** 2015. Disponível em: https://canaltech.com.br/produtos/o-que-e-drone/. Acesso em: 15 dez. 2023.

ENTERPRISE. **The Use of Drones in Agriculture Today**. 2021. Disponível em: https://enterprise-insights.dji.com/blog/drones-in-agriculture. Acesso em: 15 dez. 2023.

LISBOA, Paulo. **Biblioteca Time Python:** Aprenda a programar com a biblioteca Python. 2023. Disponível em: https://awari.com.br/biblioteca-time-python-aprenda-a-programar-com-a-biblioteca-python/?utm_source=blog&utm_campaign=projeto+blog&utm_medium=Biblioteca%20Time%20Python:%20Aprenda%20a%20programar%20com%20a%20biblioteca%20Python#:~:text=Com%20a%20biblioteca%20Time%20Python%2C%20%C3%A9%20poss%C3%ADvel%20medir%20o%20tempo,fun%C3%A7%C3%B5es%20ou%20blocos%20de%20c%C3%B3digo.. Acesso em: 16 dez. 2023.

MAPPA. **Topografia com drones:** o que é, por onde começar +12 aplicações. 2022. Disponível em: https://mappa.ag/blog/topografia-com-drones-como-onde-comecar/#:~:text=A%20topografia%20com%20drones%20nada,levantamento%20dos%20dados%20do%20terreno.. Acesso em: 15 dez. 2023.

PASSARELLI, Leandro. **Aplicação de visão computacional com OpenCV**. 2017. Disponível em: https://embarcados.com.br/aplicacao-de-visao-computacional-com-opencv/. Acesso em: 16 dez. 2023.

PETROBRAS. **Drones inspetores – como eles contribuem para a manutenção de equipamentos**. 2021. Disponível em:

https://nossaenergia.petrobras.com.br/energia/drones-inspetores-como-eles-contribuem-para-a-manutencao-de-equipamentos/. Acesso em: 15 dez. 2023.

ROCHA, Humberto. **Desbravando o pygame 1 - Conhecendo a Biblioteca**. 2017. Disponível em: https://humberto.io/pt-br/blog/desbravando-o-pygame-1-conhecendo-a-biblioteca/. Acesso em: 16 dez. 2023.