School of Mechanical & Manufacturing Engineering (SMME),

National University of Science and Technology (NUST),

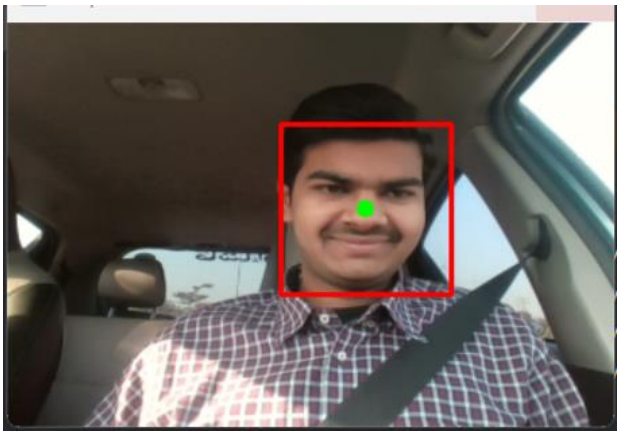
Sector H-12, Islamabad

FOP-II LAB Project

Program: BE-Aerospace Section: AE-01
Session: Fall 2024 Semester: 3rd

AUTONOMOUS DRONE PROGRAMMING IN PYTHON

FEATURE DETECTION AND TRACKING



Name	CMS ID
Ikhlas Jamshaid	454122

TABLE OF CONTENTS:

I. ABSTRACT

The objective of this work is to develop a proof of concept of an Autonomous drone able to carry out simple tasks consisting of detecting specific feature (face, body…), track the detected feature, and take pictures in real-time, additionally and for operational and safety measures the described system must be able to recognize and avoid obstacles in the environment, map the exact coordinates of the drone in real-time and switch to keyboard control when requested by the user. Besides the design and implementation of a simple, exhaustive and operational system, the contribution of this work is to identify a worthwhile trade-off between accuracy and speed (response time), imposed by the limitations of the commercial hardware (Tello Drone) used in the context of this project. Additionally, for replicability purposes, a set of quality measures, limitations, and tests are described.

KEYWORDS

UAV, Artificial Intelligence, Computer Vision, Autonomous Drones, Machine Learning, Recognition.

II. LIBRARIES AND IMPLEMENTATION:

DJI Tello Library: djitellopy version 2.3.1, developed by Damià Fuentes Escoté under the MIT License, with the following features:

- Tello drone commands
- Tello drone camera video streaming
- Tello drone state packet parsing.
- Tello drone keyboard control https://github.com/damiafuentes/DJITelloPy.git

Computer vision Library: OpenCV-python version 4.5.2.54, an open-source library designed by Alexander Mordvintsev and Abid Rahman K, to solve computer vision tasks in python:

- Image processing
- Object detection
- Machine learning https://github.com/opencv/opencv-python

PyGame library: Open-Source python programming language library for multimedia applications and keyboard control.
https://github.com/pygame/

Numerical Computing Library: **NumPy** version 1.24.3, an open-source library developed by **Travis Oliphant** and contributors, widely used for numerical computations and data manipulation in Python:

- Multidimensional array operations
- Mathematical functions and linear algebra
- Fast Fourier transforms (FFT)
- Random number generation and statistics

III. INTRODUCTION

Autonomous drones are unmanned aerial vehicles (UAVs), able to operate with a certain degree of autonomy, from taking off and navigating until landing. To pick the most profitable decisions, UAVs require the use of Artificial Intelligence (AI) solutions, defined as "the collection of methods that enable a computer to solve tasks that would require intelligence if they were solved by human beings". Thanks to their potential to carry out complex tasks in different sectors such as Logistics, agriculture, surveillance, filmmaking, and many more.

Research question:

How can real-time facial recognition and tracking algorithms be optimized for efficient and reliable drone-based face following in dynamic environments?

This research question focuses on developing and optimizing a **drone face follower system** that can reliably track and follow a person's face in real time. The explanation breaks down as follows:

1. **Real-Time Facial Recognition and Tracking**:
   The research emphasizes implementing or improving facial recognition and tracking algorithms that operate in real-time. This ensures the drone can detect and track a face without noticeable delays.
2. **Efficient and Reliable Operation**:
   Efficiency relates to minimizing computational load and power consumption, critical for drones with limited onboard resources. Reliability ensures the system can maintain accurate tracking under various conditions, including changes in lighting, occlusions, or rapid movements of the subject.
3. **Dynamic Environments**:
   The drone must function effectively in diverse and unpredictable environments, such as indoors, outdoors, or in crowded spaces, where background objects and other faces may interfere.

By addressing this research question, the study aims to contribute to advancements in autonomous drone systems, enabling applications like security surveillance, interactive robotics, or personal photography assistants. It also involves challenges like managing computational constraints, ensuring safety, and optimizing the drone's control system for smooth tracking.

IV. TELLO DRONE

I decided on Tello because it is meant to be programmed and can send streaming video. Perfect!

The **DJI Ryze Tello** is a compact, user-friendly drone designed for beginners, educators, and hobbyists. It offers an affordable entry point into drone piloting and programming, making it popular for recreational use and STEM education. Below is a detailed breakdown of the Tello drone's features, specifications, and potential applications:

1. Design and Build

- **Compact and Lightweight:**
  The Tello weighs approximately 80 grams (with propellers and battery) and is highly portable.
- **Durable Frame:**
  Made of robust plastic, it withstands minor crashes and is equipped with propeller guards for added safety.
- **Aesthetic:**
  Sleek and minimalistic, with a clean design suitable for indoor and outdoor flying.

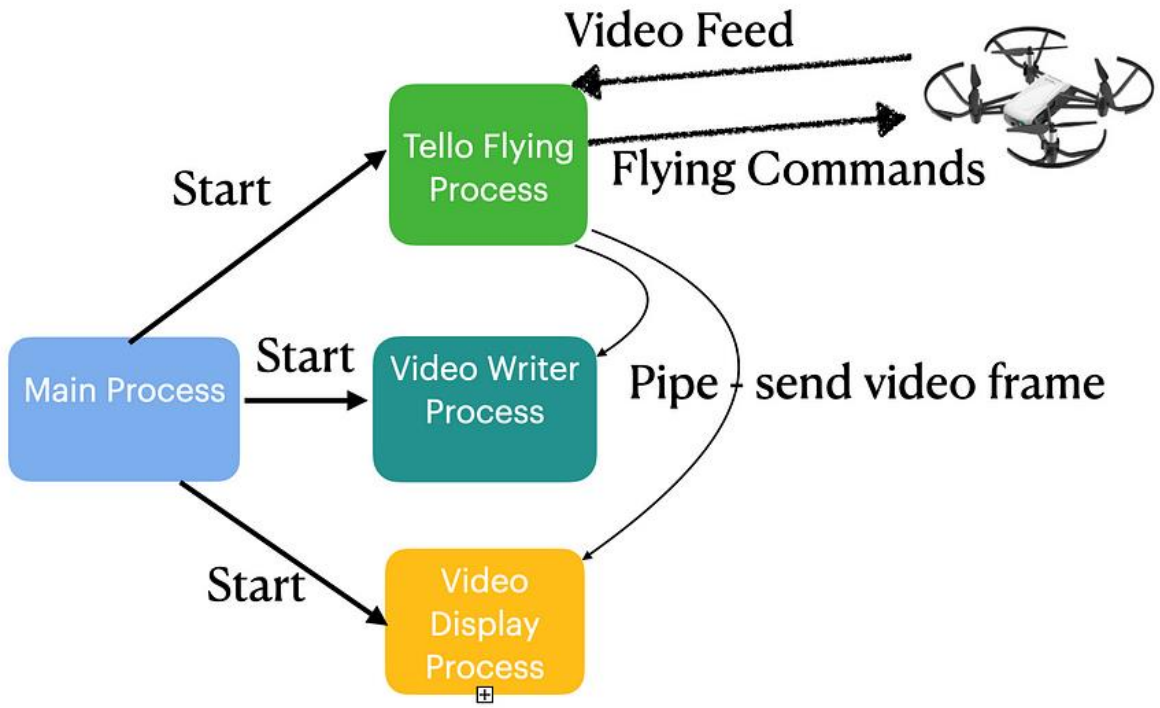
2. Technical Specifications

- **Camera:**
  - 5 MP (2592x1936 resolution) still photos.
  - HD video recording at 720p at 30fps.
  - Electronic image stabilization reduces shakiness for smoother videos.
- **Flight Performance:**
  - Maximum Flight Time: ~13 minutes on a single charge (depending on conditions).
  - Range: 100 meters via Wi-Fi control.
  - Top Speed: ~8 m/s (18 mph) in optimal conditions.
- **Sensors:**
  - Vision Positioning System (VPS): Uses a downward-facing camera and sensors for precise indoor hovering and stability.
  - Barometer: Ensures steady altitude control.
  - Inertial Measurement Unit (IMU): Tracks orientation and motion.
- **Battery:**
  - 3.8V 1100mAh Li-Po battery.
  - Modular design allows quick swapping.

V. APPLICATION ARCHITECTURE

I decided to break up the different parts of the application into different processes.
There is the main script which starts all the sub-processes.

The ‘Tello Flying Process’ handles all the communication with the Tello drone and receives the video stream. This process will use the PID controllers to determine which direction (Up/Down and Left/Right) the drone should move to keep the centre of the face rectangle in the centre of the video frame. The Tello Flying Process will adorn the video stream with a face box, some information about X,Y error or difference from the centres, and an arrow showing the direction of movement. The process will then use Pipes to send the video frame to the other two processes.



My Tello Face Following Application Architecture

The Video Writer process will read the adorned video frame from the Pipe and write them to an mp4 file for later offline viewing. The Video Display Process will read the adorned video frame and display those frames using OpenCV so you can have real-time feedback and see what is going on.

VI. FEATURE TRACKING:

Before implementing the feature tracking function, an exhaustive understanding of the (DJI Tello) axis and degrees of freedom is required. Thus, as shown in figure 4, the movement of (DJI Tello) drone can be projected on the three-axis (x,y,z), with (3) translations and (3) rotations as follows: -Translation (x): Forward (+x) and Backward (-x) ,

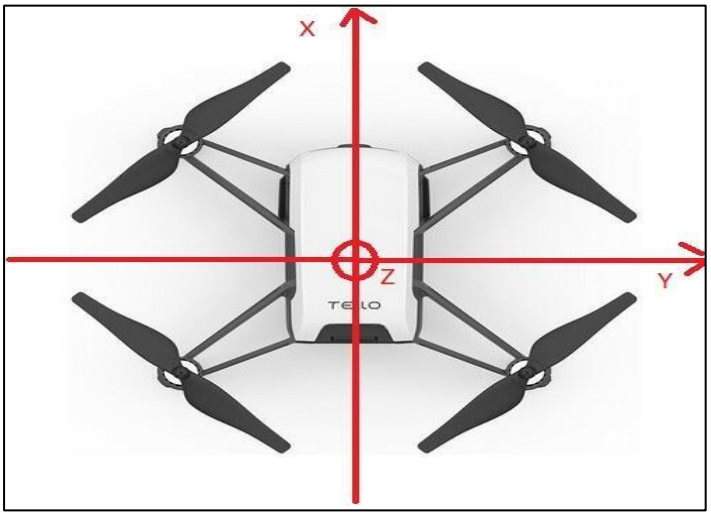


Figure : DJI Tello drone axis

-Translation (y): Right (+y) and left (-y),

-Translation (z): up (+z) and down (-z)

-Rotation (x): roll (+/-)

-Rotation (y): Pitch (+/-)

To enable the drone to track the detected feature, ensure the smoothness of the movement, and avoid sudden commands that could lead to overshooting, a control loop mechanism should be implemented. In this context, the role of a control loop is to monitor and regulate the output.

VII. PID TUNING:

There are multiple classifications for PID parameters tuning methods. Adopting one approach over others depends mainly on its complexity and the ability to monitor the system output. One variety consists of three categories:

- Trial and error tuning method,
- Rule-based tuning method,
- Model-based tuning method,

For this project and seeing the complexity of elaborating an accurate mathematical model for the drone steering control, which is one of the requirements of rule and model-based methods the trial-and-error tuning method was adopted. In this context, a set of tests were conducted by changing the controller's parameters (P and D) and observing the drone's response until the convergence. The objective is to define the trade-off point between stability (no overshooting due to momentum) and fast reaction (fast-tracking for moving faces). The findings of the experiment are shown in the following table:

Test Num	Value P	Value D	Response
1	0.3	0	Overshooting
2	0.1	0	Slow
3	0.15	0	Slow
4	0.2	0	Slow
5	0.25	0	A little overshooting
6	0.25	0.1	A little overshooting
7	0.25	0.2	A little overshooting
8	0.23	0.2	A little overshooting
9	0.22	0.1	A little overshooting
9	0.22	0.08	Converge to 0

PD controller tuning test

VIII. FACE DETECTION:

Face detection accuracy is highly affected by the quality of the camera (5MP (2592×1936)) and the level of lighting in the environment. Therefore, three experiments with different lighting conditions were conducted to quantify the accuracy (see Appendix C):

First experiment: The system accuracy is measured in an environment with relatively medium contrast lighting conditions.

Second experiment: The system accuracy is measured in an environment with relatively low contrast lighting conditions.

Third experiment: The system accuracy is measured in an environment with relatively high contrast lighting conditions.

For the three categorical values of contrast in the three experiments: Low, Medium, and high, the obtained accuracy values are: 0%, 83%, and 95%, respectively.

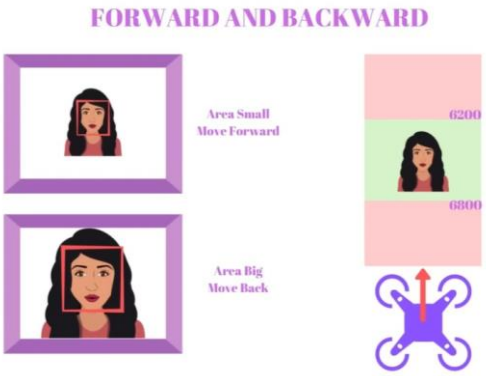
IX. THE CODE:

```
X.    import numpy as np
      import cv2
      #from djitellopy import tello

      # import time

      #me = tello.Tello()
      #me.connect()
      #print(me.get_battery())

      #me.streamon()
      #me.takeoff()
      #me.send_rc_control(0,0,25,0)
      #time.sleep(2.2)

      w, h = 360,240
      fbRange =[6200,6800]
      pid = [0.4,0.4,0]
      pError = 0


      def findFace(img):
          # Load the face detection model
          faceCascade =
      cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.x
      ml")
          if faceCascade.empty():
              print("Error loading Haar cascade file.")
              return img  # Return the original image if cascade file is
      not loaded

          # Convert the image to grayscale for face detection
          imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

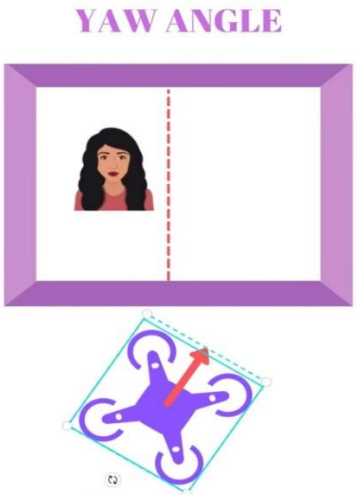
```
        faces = faceCascade.detectMultiScale(imgGray, 1.2, 8)
        myFaceListC = []
        myFaceListArea = []

        for (x, y, w, h) in faces:
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
    # Draw rectangles on detected faces
            cx = x + w // 2 #Get Centre Points
            cy = y + h // 2 #Get Centre Points
            area = w * h
            cv2.circle(img, (cx, cy), 5, (0, 255, 0), cv2.FILLED)
            myFaceListC.append([cx,cy])
            myFaceListArea.append(area)
        if len(myFaceListArea) !=0:
            i= myFaceListArea.index(max(myFaceListArea))
            return img, [myFaceListC[i], myFaceListArea[i]]
        else:
            return img, [[0,0],0]

    def trackFace(info, w, pid, pError):
        area = info[1]
        x,y = info[0]
        error = x - w//2
        fb = 0
        speed = pid[0]*error +pid[1]*(error-pError)
        speed = int(np.clip(speed, -100, 100))

        if area > fbRange[0] and area < fbRange[1]:
            fb = 0
        elif area > fbRange[1]:
            fb=-20
        elif area < fbRange[0] and area !=0:
            fb = 20
        print(speed, fb)
        if x ==0:
            speed = 0
            error = 0
        #me.send_rc_control(0, fb, 0, speed)
        return error
    # Initialize video capture (use 0 for the default camera)
    cap = cv2.VideoCapture(0)  # Use 0 if there's only one camera

    if not cap.isOpened():
        print("Error: Cannot access the camera.")
        exit()
    while True:
        ret, img = cap.read()
        if not ret:
           print("Failed to capture frame.")
           break
        #img = me.get_frame_read().frame
        img = cv2.resize(img, (w, h))
        img, info = findFace(img)
        pError = trackFace( info, w, pid, pError)
        print("Centre" , info[0], "Area", info[1])# Process the frame
    to detect faces
        cv2.imshow("Output", img)
        # Exit on pressing 'q'
        if cv2.waitKey(1) & 0xFF == ord('q'):
            #me.land()
            break
```

XI. EXPERIMENTATION OF THE CODE:

The algorithmic function used for the facial tracing and implementing the trajectory of the Drone was designed such that, we added Two threshold values an upper limit and a lower limit for the pixel's area covered by the bounding box of the CV detected facial region, the code comprised of a general if-else logic that processes the area and implements 3 logical functions, Move forward, Hover at a fixed location, and Move backwards.



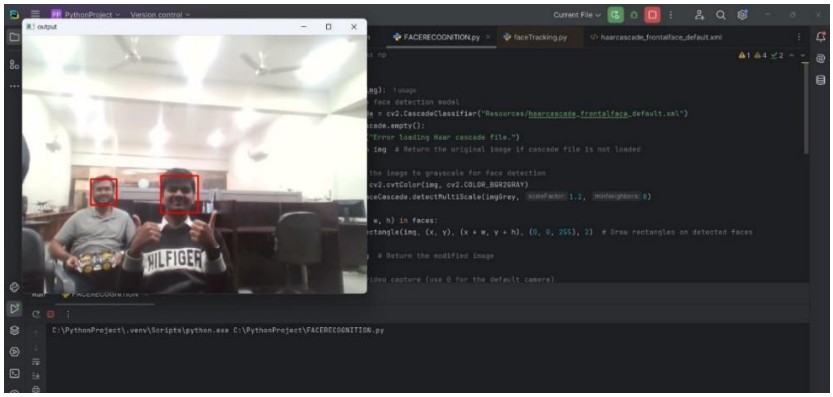
Another control functionality that was necessary for our mission plan was that we had to align the facial box at the center of focus so that the Horizontal movement of the drone may not cause a problem of losing the sight of face for this purpose another if else logic was used that controlled the yaw angle in such a way that the box may always get aligned in the center line of the video feed.



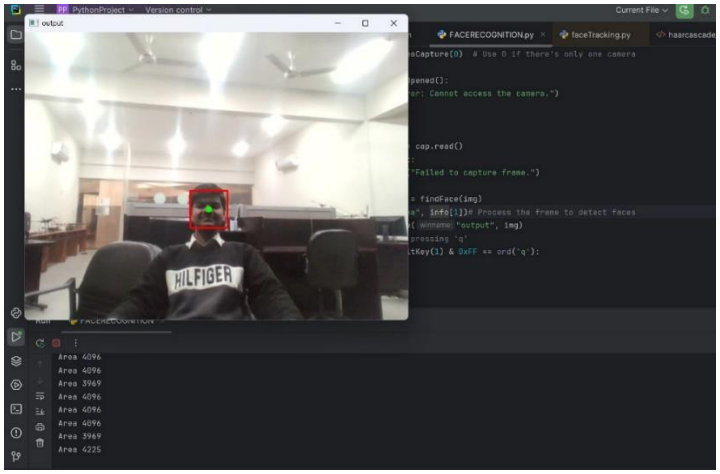
Below are a few examples that illustrate the code functionality tested by us, as the distance changes continuous feedback is generated for the Area by the serial reader, that will be further used by the algorithm to provide motor action to the flying Drone.

Haar cascade:

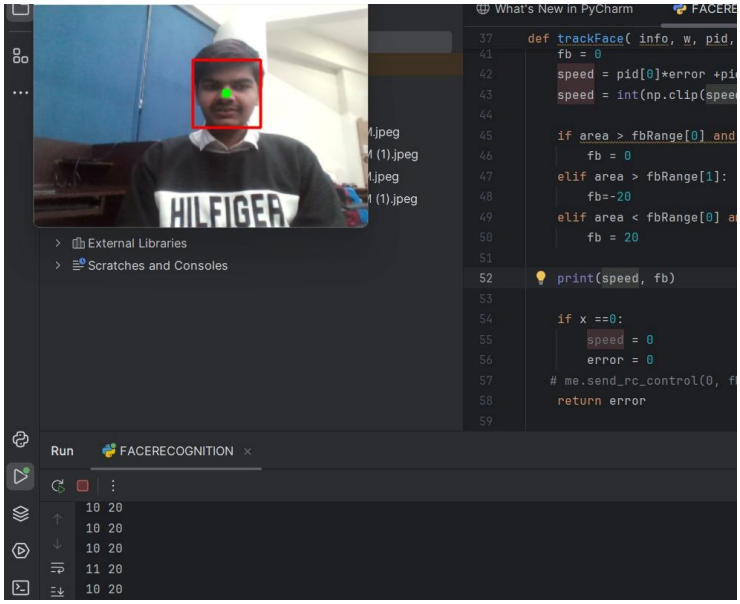
Haar cascade is an algorithm that can detect objects in images, irrespective of their scale in image and location. This algorithm is not so complex and can run in real-time. We can train a haar-cascade detector to detect various objects like cars, bikes, buildings, fruits, etc



Below is the functional algorithm that also applies a center marker that specifies the mean for the location of bounding box that will further help in the Yaw control algorithm, by aligning the point along the central X-axis.



The Serial monitor currently displaying the functional algoritm, with the YAW feedback printed on the Left (10,10,10,11,10….), and the Speed feedback for forward motion as (20,20,20,…) on the right column of the serial monitor, with respect to the following Video feedback, we did this calculation and feedback for a test case separately on our PC camera.



XII. FUNCTIONS USED IN DJI TELLO PY LIBRARY:

By combining these methods, the program initializes and controls a DJI Tello drone, performs image processing on the video feed, and ensures safe operation with error handling.

1. tello.Tello()
Description:
Creates an instance of the Tello drone, enabling access to its methods for control and interaction.
Syntax:
me = tello.Tello()
Purpose:
Initializes the Tello drone object.

---

2. connect()
Description:
Establishes a connection between the Python program and the Tello drone.
Syntax:
me.connect()
Purpose:
Ensures the drone is connected and ready for commands.

---

3. get_battery()
Description:
Fetches the current battery level of the Tello drone.
Syntax:
battery = me.get_battery()
Purpose:
Checks whether the drone has sufficient battery for operation.

---

4. streamon()
Description:
Starts the video stream from the Tello drone's camera.
Syntax:
me.streamon()
Purpose:
Allows video feed to be captured and processed in real-time.

---

5. takeoff()
Description:
Instructs the Tello drone to take off and hover at a safe altitude.
Syntax:
me.takeoff()
Purpose:
Initiates flight.

---

6. send_rc_control(left_right_velocity, forward_backward_velocity, up_down_velocity, yaw_velocity)

Description:

Sends manual control commands to the drone, specifying velocities in four directions.

Syntax:

me.send_rc_control(left_right, forward_backward, up_down, yaw)

Parameters:

- left_right_velocity: Horizontal movement speed (left/right).
- forward_backward_velocity: Forward/backward movement speed.
- up_down_velocity: Vertical movement speed (up/down).
- yaw_velocity: Rotation speed.

Purpose:

Controls drone movement in the specified directions.

---

7. get_frame_read()

Description:

Captures the current video frame from the drone's camera feed.

Syntax:

frame = me.get_frame_read().frame

Purpose:

Provides access to the current video feed for image processing.

---

8. land()

Description:

Commands the drone to safely land.

Syntax:

me.land()

Purpose:

Ends the flight and lands the drone.

---

9. streamoff()

Description:

Stops the video stream from the drone's camera.

Syntax:

me.streamoff()

Purpose:

Releases resources and halts the video feed.

---

10. end()

Description:

Closes the connection with the Tello drone and releases all associated resources.

Syntax:

me.end()

Purpose:

Ensures a clean exit from the program, preventing lingering connections.

---

11. cv2.imshow() (OpenCV Function)
Description:
Displays the processed video frame in a window.
Syntax:
cv2.imshow("Window_Name", img)
Purpose:
Visualizes the processed video stream with overlays.

---

12. cv2.waitKey() (OpenCV Function)
Description:
Waits for a key press for a specified duration, allowing smooth video playback.
Syntax:

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    me.land()
    break
```

Purpose:
Provides a mechanism to terminate the program by pressing 'q'.

---

13. cv2.destroyAllWindows() (OpenCV Function)
Description:
Closes all OpenCV-created windows.
Syntax:
cv2.destroyAllWindows()
Purpose:
Cleans up display windows upon program termination.

XIII. EXPLANATION OF THE CODE:

Data Set for:

faceCascade = cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")

Step-by-Step Code Explanation

---

1. Import Libraries

- **Libraries:**
  - numpy and cv2 for image processing.
  - tello from djitellopy for controlling the Tello drone.
  - time for delays and timing.

---

2. Initialize Tello Drone

- Create a Tello object to connect and communicate with the drone.

```
me = tello.Tello()
me.connect()
```

---

3. Check Battery Level

- Retrieve the drone's battery level.
- Exit if the battery is below 20%.

```
battery = me.get_battery()
if battery < 20:
    print(f"Battery too low ({battery}%). Please charge the drone.")
    exit()
```

---

4. Start Video Stream

- Begin streaming video from the drone's camera.
- Handle exceptions if the stream fails to start.

```
try:
    me.streamon()
except Exception as e:
    print(f"Error starting video stream: {e}")
    exit()
```

---

5. Takeoff

- Instruct the drone to take off and hover.
- Handle exceptions if the takeoff fails.

```
try:
    me.takeoff()
except Exception as e:
    print(f"Error during takeoff: {e}")
    me.end()
    exit()
```

---

6. Stabilize Drone

- Send control commands to move the drone upwards slightly for stabilization.
- Introduce a delay for consistent movement.

```
me.send_rc_control(0, 0, 25, 0)
time.sleep(2.2)
```

7. Define Constants

- Set the frame size (w, h), acceptable face area range (fbRange), and PID control parameters (pid and pError).

```
w, h = 360, 240
fbRange = [6200, 6800]
pid = [0.4, 0.4, 0]
pError = 0
```

---

8. findFace Function

- Detects faces using a Haar cascade classifier.
- Identifies the largest face and its position/area.
- Returns the processed image and face information.

```
def findFace(img):
    faceCascade = cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")
    # Detect faces and draw rectangles/circles.
    # Return coordinates of largest face.
```

---

9. trackFace Function

- Controls the drone to follow the detected face.
- Adjusts speed and forward/backward movement based on face position and size.
- Sends control commands to the drone.

```
def trackFace(me, info, w, pid, pError):
    area = info[1]
    x, y = info[0]
    # Calculate speed and adjustments using PID control.
    me.send_rc_control(0, fb, 0, speed)
```

---

10. Main Loop

- Continuously:
    1. Capture a video frame from the drone.
    2. Resize the frame.
    3. Detect and track the face.
    4. Display the processed video feed.
    5. Exit and land when 'q' is pressed.

```
while True:
    img = me.get_frame_read().frame
    img = cv2.resize(img, (w, h))
    img, info = findFace(img)
    pError = trackFace(me, info, w, pid, pError)
    cv2.imshow("Output", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        me.land()
        break
```

---

11. Cleanup

- Stop video streaming, disconnect the drone, and close display windows.

```
finally:
    me.streamoff()
    me.end()
    cv2.destroyAllWindows()
```

Purpose of the Code

This program:

1. Connects to a DJI Tello drone.
2. Streams video detects a face using OpenCV and tracks it.
3. Adjusts the drone's movement to follow the detected face.
4. Ensures safe operation and cleanup upon exit.

XIV. CONCLUSION:

We successfully implemented and tested our face detection and tracking project using the DJI Tello drone. The test was conducted at 10 PM at SINES, and Alhamdulillah, it worked as intended. This project demonstrates the integration of computer vision and drone technology to achieve real-time face detection and autonomous tracking.

I am grateful for the success of this project and pray that Allah blesses me with even more success in my future endeavours.