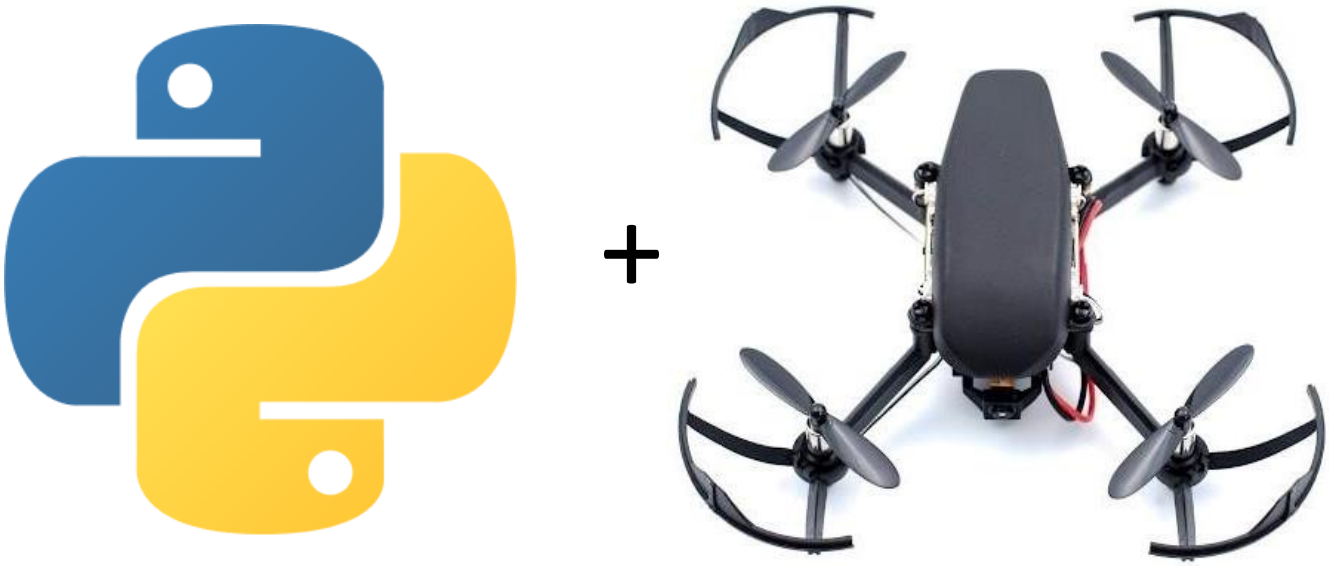
PROJECTS WITH PLUTO's CAMERA USING PYTHON

Version 1



python

Table of Contents

Introduction

Pluto is not just a drone, it is a development tool. It enables the user to develop numerous projects across different areas. With the Pluto Platform, you can create and innovate to bring your ideas into reality. From simply glowing LEDs, to understanding complicated scientific concepts, to developing fun games - everything is possible with the Pluto Python Platform. The sensor data and the remote-control data are all accessible with python, making it one of a kind.

With the platform being open source, tinkering is made a lot easier with the predefined python functions. Additionally, for those who prefer Python, the Pluto Platform offers extensive support for Python programming. This makes it easier to integrate various libraries and tools, providing a versatile environment for developers.

Some key features of the Pluto Platform are:

✓ Easy predefined functions in Python
✓ Real-time sensor data access
✓ Remote control data integration
✓ camera access and control
✓ opencv image processing

The Pluto Platform aims to motivate users to begin their journey into the universe of drones, making it accessible and enjoyable for both beginners and experienced programmers.

Camera Details

WIFI Camera Module

Camera attached to the drone

Features:

- Compatible with Pluto X and Pluto 1.2
- Video: 720p / 1080p
- Photo: 1MP / 2MP
- Range: 30m
- SD card supported
- Frame rate:
    - Live transmission video: ~18 FPS
    - On-board card: 25 FPS
- Weight: 8g (With casing)
- Minimum supply voltage (+V): 3.4V

Compatibility

The Projects with Pluto: Using Python contains projects that are built using python and executed using Pluto nano drone. The compatibility for both is given below.

Operating System:

- Windows
- Linux
- MacOS

Python Versions:

- Any version greater than 3.7.0

Drones:

- Pluto X nano drone
- Pluto 1.2 nano drone

Other:

- Rover
- Hover

Python Installation and Setup

For Windows Users:

1. **Download Python**: Visit the official Python website (python.org) and download the latest version of Python for Windows. Choose the executable installer for simplicity.

2. **Run the Installer**: Double-click the downloaded file to start the installation. Ensure to check the box that says "Add Python 3.x to PATH" to make Python accessible from the command line.

3. **Verify Installation**: Open Command Prompt and type python --version. If Python is successfully installed, you'll see the version number displayed.

4. **Note:** Avoid using the Microsoft store version its not very efficient for development.

For Linux Users:

1. **Check for Pre-installed Python**: Most Linux distributions come with Python pre-installed. Open a terminal and type python --version or python3 --version.

2. **Install Python**: If Python is not installed or you need a different version, use your distribution's package manager. For Ubuntu or Debian-based systems, use sudo apt-get update followed by sudo apt-get install python3.

3. **Verify Installation**: After installation, type python3 --version in the terminal to confirm the Python version.

For MacOS Users:

1. **Check for Pre-installed Python**: macOS comes with Python pre-installed. Open Terminal and type python --version or python3 --version to check the installed version.

2. **Install Latest Python (optional)**: If you need the latest version, visit python.org to download the macOS installer. Follow the installation prompts.

3. **Use Homebrew (alternative)**: Install Homebrew (a package manager for macOS) by pasting /bin/bash-c"$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)" in Terminal. Then, install Python by typing brew install python.

Chromebooks: With newer Chromebooks that support Linux apps, you can install Python via the Linux terminal using standard Linux commands to install Python.

Why Use Camera?

- **Aerial Photography and Videography:** Drones equipped with cameras can capture high-quality photos and videos from unique aerial perspectives, which are otherwise difficult or impossible to achieve with conventional cameras.

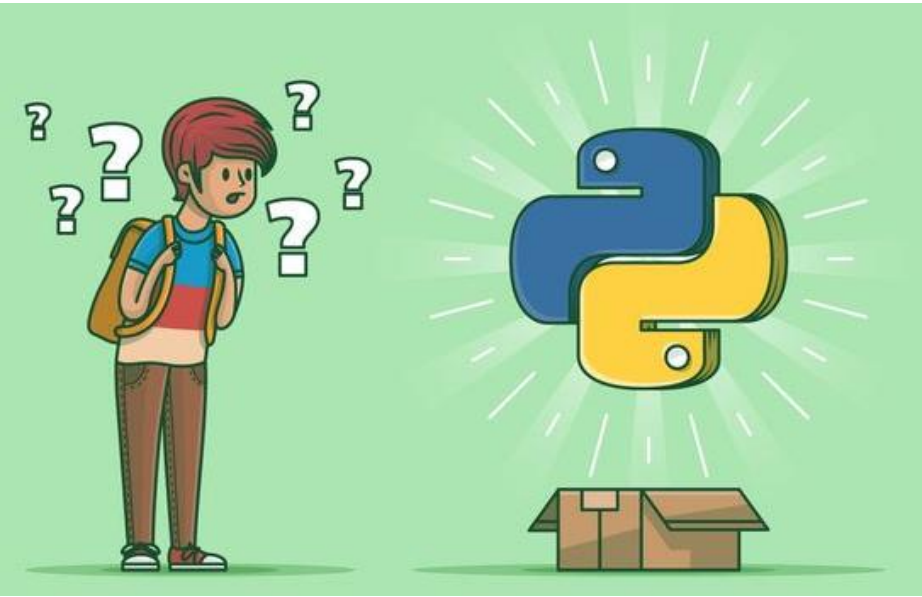
- **Surveillance and Security:** Drones with cameras are used for monitoring areas from above, enhancing security surveillance in both public and private sectors.

- **Search and Rescue Operations:** Cameras on drone's aid in search missions by providing aerial views of large areas, helping rescuers locate missing persons or objects more efficiently.

- **Monitoring and Inspections:** Drones enable close-up inspections of infrastructure such as bridges, power lines, and pipelines, allowing for visual assessment without the need for scaffolding or heavy equipment.

- **Personalized Tracking:** Drones can track individuals based on facial recognition, useful for security surveillance or following subjects during filming or photography.

- **Search and Rescue:** Identifying and tracking individuals in remote or disaster-stricken areas can aid rescue operations by pinpointing their location.



Prerequisites

Before setting up the Pluto camera live stream, ensure you have the following prerequisites installed on your system:

Python (if not installed already)
Plutocam Python package
FFmpeg (for your operating system)

Installation Steps
Step 1: Download FFmpeg
Download FFmpeg from one of the following sources:

Official FFmpeg Website
FFmpeg Builds on GitHub

Step 2: Install Ffmpeg
Follow the installation instructions based on your operating system:

Windows: FFmpeg Installation Guide for Windows
Linux: Install FFmpeg on Ubuntu
Mac: FFmpeg Installation Guide for Mac

Step 3: Check if correctly installed
Open up the terminal/ command line and type the command “ffmpeg”, if you get the following output, it means its correctly installed

```
C:\Users\giant>ffmpeg
ffmpeg version N-106757-geef652ca9c-20220430 Copyright (c) 2000-2022 the FFmpeg developers
  built with gcc 11.2.0 (crosstool-NG 1.24.0.533_681aaef)
  configuration: --prefix=/ffbuild/prefix --pkg-config-flags=--static --pkg-config=pkg-config --cross-prefix=x86_64-w64-
mingw32- --arch=x86_64 --target-os=mingw32 --enable-gpl --enable-version3 --disable-debug --disable-w32threads --enable-
pthreads --enable-iconv --enable-libxml2 --enable-zlib --enable-libfreetype --enable-libfribidi --enable-gmp --enable-lz
ma --enable-fontconfig --enable-libvorbis --enable-opencl --disable-libpulse --enable-libvmaf --disable-libxcb --disable
-xlib --enable-amf --enable-libaom --enable-libaribb24 --enable-avisynth --enable-libdav1d --enable-libdavs2 --disable-l
ibfdk-aac --enable-ffnvcodec --enable-cuda-llvm --enable-frei0r --enable-libgme --enable-libass --enable-libbluray --ena
ble-libjxl --enable-libmp3lame --enable-libopus --enable-librist --enable-libtheora --enable-libvpx --enable-libwebp --e
nable-lv2 --enable-libmfx --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libopenh264 --enable-libopenjpe
g --enable-libopenmpt --enable-librav1e --enable-librubberband --enable-schannel --enable-sdl2 --enable-libsoxr --enable
-libsrt --enable-libsvtav1 --enable-libtwolame --enable-libuavs3d --disable-libdrm --disable-vaapi --enable-libvidstab -
-enable-vulkan --enable-libshaderc --enable-libplacebo --enable-libx264 --enable-libx265 --enable-libxavs2 --enable-libx
vid --enable-libzimg --enable-libzvbi --extra-cflags=-DLIBTWOLAME_STATIC --extra-cxxflags= --extra-ldflags=-pthread --ex
tra-ldexeflags= --extra-libs=-lgomp --extra-version=20220430
  libavutil      57. 24.101 / 57. 24.101
  libavcodec     59. 27.100 / 59. 27.100
  libavformat    59. 23.100 / 59. 23.100
  libavdevice    59.  6.100 / 59.  6.100
  libavfilter     8. 37.100 /  8. 37.100
  libswscale      6.  6.100 /  6.  6.100
  libswresample   4.  6.100 /  4.  6.100
  libpostproc    56.  5.100 / 56.  5.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options] outfile}...

Use -h to get full help or, even better, run 'man ffmpeg'
```

Prerequisites

Step 4: Install Python

If Python is not installed on your system, download and install it from the official Python website.

Step 4: Install plutocam
Install the plutocam library using pip:

“ pip install plutocam “

Step 5: Connect to Pluto Camera
Ensure you are connected to the Pluto camera’s wifi before proceeding.

Step 6: Start the Stream
Open a terminal and run the following command:

“plutocam stream start --out-file - | ffplay -i -fflags nobuffer -flags low_delay -probesize 32 -sync ext –“

This command initiates the live stream from the Pluto camera using plutocam and FFmpeg.

Pluto Python Package - Plutocam

PlutoCam is a Python package designed to facilitate seamless communication with Pluto's integrated camera module, enhancing the capabilities of drones equipped with this feature.
This package enables users to capture images and video streams from the drone's camera in real-time.
By integrating PlutoCam with other libraries such as OpenCV, developers can perform advanced image processing tasks like object detection and tracking.

Key Features of PlutoCam:

- **Real-time Image and Video Capture:** PlutoCam enables users to capture images and stream video in real-time from Pluto drones, providing immediate visual feedback from aerial perspectives.
- **Integration with OpenCV:** By integrating PlutoCam with OpenCV, developers can leverage powerful image processing capabilities. This includes tasks such as object detection, facial recognition, and motion tracking, enhancing the drone's functionality beyond basic imaging.
- **Support for Drone-based Projects**: Whether used in research, commercial applications, or educational projects, PlutoCam facilitates the integration of drone-captured visual data into broader systems and workflows. This includes autonomous navigation systems, remote sensing applications, and interactive multimedia projects.

Installation:
pip install plutocam

https://pypi.org/project/plutocam/

```
C:\Users\giant>pip install plutocam
Collecting plutocam
  Using cached plutocam-0.5.0-py3-none-any.whl.metadata (2.0 kB)
Using cached plutocam-0.5.0-py3-none-any.whl (24 kB)
Installing collected packages: plutocam
Successfully installed plutocam-0.5.0
```

Usage:
Now as you have also installed ffmpeg, open terminal and type the following command:

```
plutocam stream start --out-file - | ffplay -i -fflags nobuffer -flags low_delay -probesize 32 -sync ext -
```

Camera Operations

1. Live Camera feed:

Requirements:

pip install opencv-python

pip install numpy

```
import cv2
import numpy as np
import subprocess

# Starting the drone video stream and setting up ffmpeg to convert the video stream
process = subprocess.Popen(
    ["plutocam", "stream", "start", "--out-file", "-"],
    stdout=subprocess.PIPE
)
ffmpeg_process = subprocess.Popen(
    ["ffmpeg", "-i", "-", "-f", "rawvideo", "-pix_fmt", "bgr24", "-"],
    stdin=process.stdout,
    stdout=subprocess.PIPE
)

while True:
    # Read the frame from ffmpeg output
    raw_frame = ffmpeg_process.stdout.read(2048 * 1152 * 3)
    # Adjust dimensions based on your camera's resolution
    if not raw_frame:
        break
    # Convert the byte data to a numpy array
    frame = np.frombuffer(raw_frame, dtype=np.uint8).reshape((1152, 2048, 3))

    # Display the frame using OpenCV
    cv2.imshow('Video Stream', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Clean up: terminate the process and close all OpenCV windows
process.terminate()
ffmpeg_process.terminate()
cv2.destroyAllWindows()
```

Camera Operations

2. Image Capture

```
#press C to capture images
import cv2
import numpy as np
import subprocess

process = subprocess.Popen(
    ["plutocam", "stream", "start", "--out-file", "-"],
    stdout=subprocess.PIPE
)
ffmpeg_process = subprocess.Popen(
    ["ffmpeg", "-i", "-", "-f", "rawvideo", "-pix_fmt", "bgr24", "-"],
    stdin=process.stdout,
    stdout=subprocess.PIPE
)

image_counter = 0

while True:
    raw_frame = ffmpeg_process.stdout.read(2048 * 1152 * 3)
    if not raw_frame:
        break

    frame = np.frombuffer(raw_frame, dtype=np.uint8).reshape((1152, 2048, 3))
    cv2.imshow('Video Stream', frame)

    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break
    elif key == ord('c'):
        image_filename = f'pluto{image_counter}.png'
        cv2.imwrite(image_filename, frame)
        print(f'Image captured and saved as {image_filename}')
        image_counter += 1

process.terminate()
ffmpeg_process.terminate()
cv2.destroyAllWindows()
```

Camera Operations

3. Video Capture

```
import cv2
import numpy as np
import subprocess
import plutocam

capturing = False # Global variable to track capture state

def start_video_stream():
    process = subprocess.Popen(
        ["plutocam", "stream", "start", "--out-file", "-"],
        stdout=subprocess.PIPE
    )
    ffmpeg_process = subprocess.Popen(
        ["ffmpeg", "-i", "-", "-f", "rawvideo", "-pix_fmt", "bgr24", "-"],
        stdin=process.stdout,
        stdout=subprocess.PIPE
    )
    return ffmpeg_process

# Function to capture video frames to a file
def capture_video(drone):
    global capturing
    capturing = True
    with open('out.h264', 'wb') as fp:
        for frame in drone.start_video_stream():
            if not capturing:
                break
            fp.write(frame.frame_bytes)
    capturing = False

# Main function to handle streaming and capture logic
def main():
    global capturing
    # Initialize drone and start streaming
    drone = plutocam.LWDrone()
    drone.set_time()
    ffmpeg_process = start_video_stream()
```

Camera Operations

```
    while True:
        # Read the frame from ffmpeg output
        raw_frame = ffmpeg_process.stdout.read(2048 * 1152 * 3)
        if not raw_frame:
            break
        frame = np.frombuffer(raw_frame, dtype=np.uint8).reshape((1152, 2048, 3))

        # Display the frame using OpenCV
        cv2.imshow('Video Stream', frame)
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break
        elif key == ord('c'):  # Toggle 'c' for capture
            if capturing:
                capturing = False
            else:
                # Start capturing if not already capturing
                capture_video(drone)

    # Clean up: terminate the process and close all OpenCV windows
    ffmpeg_process.terminate()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

Camera URL

- Imagine having direct access to your drone's perspective in real-time. With Flask, we've set up a URL that dynamically streams live video from the camera attached to my drone.
- Similar to how an IP camera works, this URL allows seamless access to the drone's camera feed, enabling monitoring and control remotely.
- For generating a URL for the camera, we will be using flask
    - Flask is a lightweight Python web framework used for building web applications.
    - We are using it to create a server that streams live video from Pluto's camera.

You can build a flask app on your own or you can download the files from the following link:
https://github.com/DronaAviation/PROJECTS_WITH_PYTHON/tree/main/PlutoCam/Flask-App

Step 1: Install required libraries:

- pip install flask
- pip install opencv-python

Step 2: Setup Flask Application:

- Create a new Python file (e.g., app.py) and import Flask.
- Initialize a Flask application.
- Define a route (/video_feed) that returns a video stream from the drone's camera

App.py:

```
import cv2
import numpy as np
import subprocess
import threading
from flask import Flask, render_template, Response

#cap = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")
# Use the URL as the video source

app = Flask(__name__)

frame = None  # Global variable to store the current frame
```

Camera URL

```
def capture_video():
    global frame
    process = subprocess.Popen(
        ["plutocam", "stream", "start", "--out-file", "-"],
        stdout=subprocess.PIPE
    )

    ffmpeg_process = subprocess.Popen(
        ["ffmpeg", "-i", "-", "-f", "rawvideo", "-pix_fmt", "bgr24", "-"],
        stdin=process.stdout,
        stdout=subprocess.PIPE
    )

    while True:
        # Read the frame from ffmpeg output
        raw_frame = ffmpeg_process.stdout.read(2048 * 1152 * 3)
        # Adjust dimensions based on your camera's resolution
        if not raw_frame:
            break

        # Convert the byte data to a numpy array
        frame = np.frombuffer(raw_frame, dtype=np.uint8).reshape((1152, 2048, 3))

        # Display the frame using OpenCV
        #cv2.imshow('Video Stream', frame)

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

    # Clean up: terminate the process and close all OpenCV windows
    process.terminate()
    ffmpeg_process.terminate()
    cv2.destroyAllWindows()
```

Camera URL

```
# Flask route to serve the video feed
@app.route('/video_feed')
def video_feed():
    def generate():
        global frame
        while True:
            if frame is None:
                continue
            _, jpeg = cv2.imencode('.jpg', frame)
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + jpeg.tobytes() + b'\r\n\r\n')
    return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    # Start video capture thread
    video_thread = threading.Thread(target=capture_video)
    video_thread.daemon = True
    video_thread.start()

    # Start Flask app
    app.run(host='0.0.0.0', port=5000)
```

Step 3: create an html file

- Create a new folder with name “templates” in the directory where you stored app.py
- Now in “templates” create a new file named index.html
- This should be the structure of your flask app

```
flask-app
├── app.py
└── templates
    └── Index.html
```

Camera URL

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Video Feed</title>
</head>
<body>
  <h1>Live Video Feed</h1>
  <img src="{{ url_for('video_feed') }}" alt="Video Feed">
</body>
</html>
```

Step 4: Testing the link

- First connect to the camera's wifi
- Now run the app.py in a separate terminal
  (you can use the command: "python app.py" to run it using the command line)
- Now lets write a small code to capture the feed using url and displaying using opencv:

```
import cv2

vid = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")

while True:
  ret, frame = vid.read()

  cv2.imshow('frame', frame)

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

vid.release()
cv2.destroyAllWindows()
```

Camera Projects

We will be using the following libraries for doing image processing projects using pluto

Requirements:

- pip install cvzone
- pip install opencv-python
- pip install plutocam
- pip install numpy
- pip install pyzbar

Index:

You can refer the following link for the projects:

https://github.com/DronaAviation/PROJECTS_WITH_PYTHON/tree/main/PlutoCam

1: Face Detection

```
import cvzone
from cvzone.FaceDetectionModule import FaceDetector
import cv2

cap = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")
#cap = cv2.VideoCapture(0)

# minDetectionCon: Minimum detection confidence threshold
# modelSelection: 0 for short-range detection (2 meters), 1 for long-range detection (5 meters)
detector = FaceDetector(minDetectionCon=0.5, modelSelection=0)

# Run the loop to continually get frames from the webcam
while True:
    success, img = cap.read()
    img, bboxs = detector.findFaces(img, draw=False)
    if bboxs:
        for bbox in bboxs:
            center = bbox["center"]
            x, y, w, h = bbox['bbox']
            score = int(bbox['score'][0] * 100)

            cv2.circle(img, center, 5, (255, 0, 255), cv2.FILLED)
            cvzone.putTextRect(img, f'{score}%', (x, y - 10))
            cvzone.cornerRect(img, (x, y, w, h))

    cv2.imshow("Image", img)
    cv2.waitKey(1)

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

cap.release()
cv2.destroyAllWindows()
```

2: Face Tracking

```
import cv2
from cvzone.FaceDetectionModule import FaceDetector
import numpy as np

cap = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")
ws, hs = 1280, 720
cap.set(3, ws)
cap.set(4, hs)
if not cap.isOpened():
    print("Camera couldn't Access!!!")
    exit()
detector = FaceDetector()
while True:
    success, img = cap.read()
    img, bboxs = detector.findFaces(img, draw=False)

    if bboxs:
        #get the coordinate
        fx, fy = bboxs[0]["center"][0], bboxs[0]["center"][1]
        pos = [fx, fy]
        cv2.circle(img, (fx, fy), 80, (0, 0, 255), 2)
        cv2.putText(img, str(pos), (fx+15, fy-15), cv2.FONT_HERSHEY_PLAIN, 2, (255, 0, 0), 2 )
        cv2.line(img, (0, fy), (ws, fy), (0, 0, 0), 2)  # x line
        cv2.line(img, (fx, hs), (fx, 0), (0, 0, 0), 2)  # y line
        cv2.circle(img, (fx, fy), 15, (0, 0, 255), cv2.FILLED)
        cv2.putText(img, "TARGET LOCKED", (850, 50), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 255), 3 )
    else:
        cv2.putText(img, "NO TARGET", (880, 50), cv2.FONT_HERSHEY_PLAIN, 3, (0, 0, 255), 3)
        cv2.circle(img, (640, 360), 80, (0, 0, 255), 2)
        cv2.circle(img, (640, 360), 15, (0, 0, 255), cv2.FILLED)
        cv2.line(img, (0, 360), (ws, 360), (0, 0, 0), 2)  # x line
        cv2.line(img, (640, hs), (640, 0), (0, 0, 0), 2)  # y line
    cv2.imshow("Image", img)
    cv2.waitKey(1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

3: Hand Tracking

```
import cv2
from cvzone.HandTrackingModule import HandDetector

cap = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")
detector = HandDetector(detectionCon=0.8, maxHands=2)

while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)  # With Draw

    if hands:
        # Hand 1
        hand1 = hands[0]
        lmList1 = hand1["lmList"]  # List of 21 Landmarks points
        bbox1 = hand1["bbox"]  # Bounding Box info x,y,w,h
        centerPoint1 = hand1["center"]  # center of the hand cx,cy
        handType1 = hand1["type"]  # Hand Type Left or Right

        fingers1 = detector.fingersUp(hand1)

        if len(hands) == 2:
            hand2 = hands[1]
            lmList2 = hand2["lmList"]  # List of 21 Landmarks points
            bbox2 = hand2["bbox"]  # Bounding Box info x,y,w,h
            centerPoint2 = hand2["center"]  # center of the hand cx,cy
            handType2 = hand2["type"]  # Hand Type Left or Right

            fingers2 = detector.fingersUp(hand2)
            length, info, img = detector.findDistance(centerPoint1, centerPoint2, img)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

4: Image Segmentation

Image segmentation is the process of partitioning an image into distinct regions, often to isolate objects or backgrounds. In the provided code, the SelfiSegmentation module from cvzone is used to remove the background from a live video feed, replacing it with a magenta color.

```
import cvzone
from cvzone.SelfiSegmentationModule import SelfiSegmentation
import cv2

cap = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")

cap.set(3, 640)
cap.set(4, 480)

# Initialize the SelfiSegmentation class. It will be used for background removal.
# model is 0 or 1 - 0 is general 1 is landscape(faster)
segmentor = SelfiSegmentation(model=0)

while True:
    success, img = cap.read()

    # Use the SelfiSegmentation class to remove the background
    # Replace it with a magenta background (255, 0, 255)
    imgOut = segmentor.removeBG(img, imgBg=(255, 0, 255), cutThreshold=0.1)

    cv2.imshow("Image", imgOut)

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

cap.release()
cv2.destroyAllWindows()
```

5: QR Code Detection

Requirements:

pip install pyzbar

```
import cv2
import numpy as np
from pyzbar.pyzbar import decode

cap = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")
cap.set(3,640)
cap.set(4,480)

while True:

    success, img = cap.read()
    for barcode in decode(img):
        myData = barcode.data.decode('utf-8')
        print(myData)
        pts = np.array([barcode.polygon],np.int32)
        pts = pts.reshape((-1,1,2))
        cv2.polylines(img,[pts],True,(255,0,255),5)
        pts2 = barcode.rect
        cv2.putText(img,myData,(pts2[0],pts2[1]),cv2.FONT_HERSHEY_SIMPLEX,  0.9,(255,0,255),2)

    cv2.imshow('Result',img)
    cv2.waitKey(1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

6: Color Detection

```
# Press 'R' for Red color
# Press 'G' for Green color
# # Press 'Y' for Yellow color
# # Press 'B' for Blue color

import cv2
import numpy as np
import subprocess
import threading
import os

# Start streaming from the drone
process = subprocess.Popen(
    ["pylwdrone", "stream", "start", "--out-file", "-"],
    stdout=subprocess.PIPE
)

# Use ffmpeg to process the stream
ffmpeg_process = subprocess.Popen(
    ["ffmpeg", "-i", "-", "-f", "rawvideo", "-pix_fmt", "bgr24", "-"],
    stdin=process.stdout,
    stdout=subprocess.PIPE
)

def process_frame():
    color_ranges = {
        'R': ([0, 120, 70], [10, 255, 255], [170, 120, 70], [180, 255, 255], [0, 0, 255]),
        'G': ([36, 25, 25], [86, 255, 255], [0, 0, 0], [0, 0, 0], [0, 255, 0]),
        'B': ([94, 80, 2], [126, 255, 255], [0, 0, 0], [0, 0, 0], [255, 0, 0]),
        'Y': ([25, 150, 150], [35, 255, 255], [0, 0, 0], [0, 0, 0], [0, 255, 255])
    }

    current_color = 'R'
```

6: Color Detection

```
while True:
    # Read a raw frame from the ffmpeg output
    raw_frame = ffmpeg_process.stdout.read(2048 * 1152 * 3)
    if not raw_frame:
        break

    # Convert the raw frame to a NumPy array and reshape
    frame = np.frombuffer(raw_frame, dtype=np.uint8).reshape((1152, 2048, 3))

    # Resize frame to a smaller resolution to increase processing speed
    frame_resized = cv2.resize(frame, (1024, 576))

    # Apply Gaussian Blur to reduce noise
    blurred = cv2.GaussianBlur(frame_resized, (11, 11), 0)

    # Convert frame to HSV color space for color detection
    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

    # Get the current color range
    lower1, upper1, lower2, upper2, highlight_color = color_ranges[current_color]

    # Create masks for the selected color
    mask1 = cv2.inRange(hsv, np.array(lower1), np.array(upper1))
    mask2 = cv2.inRange(hsv, np.array(lower2), np.array(upper2))
    mask = mask1 + mask2

    # Apply morphological transformations to refine the mask
    kernel = np.ones((5, 5), np.uint8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

    # Highlight the detected color in the frame
    result = frame_resized.copy()
    result[mask > 0] = highlight_color  # Highlight detected color

    # Count the number of non-zero pixels in the mask
    num_detected_pixels = cv2.countNonZero(mask)
    text = f"{current_color} color pixels detected: {num_detected_pixels}"
    cv2.putText(result, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
```

7: Face Mesh

```
from cvzone.FaceMeshModule import FaceMeshDetector
import cv2

cap = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")

detector = FaceMeshDetector(staticMode=False, maxFaces=2, minDetectionCon=0.5, minTrackCon=0.5)

while True:
    success, img = cap.read()

    img, faces = detector.findFaceMesh(img, draw=True)

    if faces:
        for face in faces:
            leftEyeUpPoint = face[159]
            leftEyeDownPoint = face[23]
            leftEyeVerticalDistance, info = detector.findDistance(leftEyeUpPoint, leftEyeDownPoint)
            print(leftEyeVerticalDistance)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

8: Face Mask

```
from cvzone.FaceMeshModule import FaceMeshDetector
import cv2

def overlay_mask(mask_img, img, face_points):
    x_min = min(point[0] for point in face_points)
    y_min = min(point[1] for point in face_points)
    x_max = max(point[0] for point in face_points)
    y_max = max(point[1] for point in face_points)
    width = x_max - x_min
    height = y_max - y_min

    mask_resized = cv2.resize(mask_img, (width, height))
    mask_alpha = mask_resized[:, :, 3] / 255.0
    mask_rgb = mask_resized[:, :, :3]
    for c in range(0, 3):
        img[y_min:y_max, x_min:x_max, c] = mask_alpha * mask_rgb[:, :, c] + (1 - mask_alpha) *
img[y_min:y_max, x_min:x_max, c]
    return img
mask_img = cv2.imread('ironman.png', cv2.IMREAD_UNCHANGED)

cap = cv2.VideoCapture(0)  # Change to 0 or 1 if 2 doesn't work

detector = FaceMeshDetector(staticMode=False, maxFaces=2, minDetectionCon=0.5, minTrackCon=0.5)
while True:
    success, img = cap.read()
    if not success:
        print("Failed to capture image")
        break

    img, faces = detector.findFaceMesh(img, draw=True)

    if faces:
        for face in faces:
            img = overlay_mask(mask_img, img, face)
    cv2.imshow("Image", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

9: GestureCapture

The camera should be capable of capturing videos and photos based on hand gestures:

- Showing a "V" sign with the right hand will take a picture immediately.
- Showing a "V" sign with the left hand will take a picture after 3 seconds.
- Showing a "thumbs up" sign with the right hand will start recording a video.
- Showing a "thumbs up" sign with the left hand will stop recording the video.

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import time
import os

# Initialize the camera
#cap = cv2.VideoCapture(0)
cap = cv2.VideoCapture("http://127.0.0.1:5000/video_feed")

detector = HandDetector(detectionCon=0.8, maxHands=2)

# Directory to save captured images and videos
output_dir = "captured_media"
os.makedirs(output_dir, exist_ok=True)

# Variables to manage timing and recording
start_time = None
timer_started = False
capture_flag = False
cooldown_time = 3 # Cooldown period to prevent continuous captures
last_capture_time = 0

# Video recording variables
recording = False
video_writer = None
```

9: GestureCapture

```
while True:
    success, img = cap.read()
    if not success:
        break

    # Store a copy of the raw image before drawing landmarks
    raw_img = img.copy()
    hands, img = detector.findHands(img)  # With Draw
    for hand in hands:
        hand_type = hand['type']  # 'Left' or 'Right'
        fingers = detector.fingersUp(hand)

        if hand_type == 'Left' and not timer_started:
            if fingers == [0, 1, 1, 0, 0]:  # V sign
                # Start the 3-second timer
                start_time = time.time()
                timer_started = True

        elif hand_type == 'Right' and not capture_flag and (time.time() - last_capture_time > cooldown_time):
            if fingers == [0, 1, 1, 0, 0]:  # V sign
                # Capture the image immediately
                timestamp = time.strftime("%Y%m%d-%H%M%S")
                filename = f"{output_dir}/image_{timestamp}.png"
                cv2.imwrite(filename, raw_img)
                print(f"Captured image: {filename}")
                capture_flag = True
                last_capture_time = time.time()  # Update last capture time

        # Check for right thumb up to start recording
        if hand_type == 'Right' and fingers == [1, 0, 0, 0, 0] and not recording:  # Right thumb up
            timestamp = time.strftime("%Y%m%d-%H%M%S")
            video_filename = f"{output_dir}/video_{timestamp}.avi"
            fourcc = cv2.VideoWriter_fourcc(*'XVID')
            video_writer = cv2.VideoWriter(video_filename, fourcc, 20.0, (640, 480))
            recording = True
            print(f"Started recording: {video_filename}")
```

9: GestureCapture

```
            # Check for left thumb up to stop recording
            if hand_type == 'Left' and fingers == [1, 0, 0, 0, 0] and recording:  # Left thumb up
            recording = False
            video_writer.release()
            video_writer = None
            print("Stopped recording")

    # Check if the timer has started and 3 seconds have passed
    if timer_started and (time.time() - start_time) > 3:
        # Capture the image
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        filename = f"{output_dir}/image_{timestamp}.png"
        cv2.imwrite(filename, raw_img)
        print(f"Captured image: {filename}")
        timer_started = False # Reset timer
        capture_flag = True

    if capture_flag:
        capture_flag = False # Reset capture flag after image is taken
    # Display the timer on the screen if it's running
    if timer_started:
        elapsed_time = time.time() - start_time
        remaining_time = 3 - int(elapsed_time)
        cv2.putText(img, f'Timer: {remaining_time}s', (10, 70), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 3)

    # Write frames to the video file if recording
    if recording and video_writer is not None:
        video_writer.write(raw_img)
    cv2.imshow("Image", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release video writer if still recording
if recording and video_writer is not None:
    video_writer.release()
cap.release()
cv2.destroyAllWindows()
```