Dylan Brasil Santos

Project Title: **DroneVision: Facial Recognition and Keyboard Control System**

Contents

Analysis

1.1 Problem Identification

With the recent skyrocketing popularity of drones and technology as a whole for photography, surveying, and filming, these drones need operators. However, controlling a drone manually can be challenging, especially in complex environments or when the drone needs to follow a moving person. This is why I aim to implement facial recognition with an optional keyboard control detection, to autonomously follow a person and respond to keyboard inputs. Both facial recognition and keyboard control of a drone are complex tasks that require advanced computational techniques and processing power, which I believe I can provide. My drone could potentially be applied in fields such as film, photography, surveillance, or possibly even security where the drone can be used to track a subject without requiring manual control.

The problem without my solution (the drone) involves people having to analyse visual footage, but I can code an AI to analyse the footage picked up from the built-in Tello 720p camera, which may be difficult and time-consuming to code, considering that this will be the first AI I have coded. This is still the most logical method; facial recognition is well-suited when approaching this problem as it involves machine learning and recognising/analysing patterns in visual data. Since I will be sticking true to this, the problem can be solved more quickly and accurately than to the traditional manual methods.

Computational methods that the solution lends itself to:

Problem recognition:
The overall goal of my project is to automate drone movement for a variety of situations where a drone might need to be able to follow a person’s face, and to have the drone recognise different hand signals, responding accordingly. One roadblock I might face along the way is getting the drone to actually recognise what is and what is not a face, since it may not comprehend that regardless of the different shapes and sizes of the user's facial features, it is still a face that should be followed. The same applies for recognising hand signals. Once I have overcome this, the rest of my solution simply involves making sure that the drone responds accordingly to each hand signal or face that it detects, and in minimal time.

Problem decomposition
When a big problem is broken up into smaller, more manageable problems, it becomes a lot easier to navigate. This is called problem decomposition, and can be applied to my ambitious project. This is an initial idea of what these steps might consist of, but of course this may change during the development process.

1. Data Collection
2. Preprocessing
3. Developing the Keyboard Control Algorithm
4. Developing the Facial Recognition Algorithm
5. GUI Development

6. Integrating the three Algorithms
7. Testing and Refinement
8. Create User-Friendly Documentation
9. Iterative Improvement

As aforementioned, the steps above are subject to change and evolve when it comes to putting those same steps into action, but this gives an idea of what I am going to aim towards with this project. I also plan to do effective testing after each stage, entering all test data types (Normal, Boundary and Erroneous) to ensure that my program works as intended. This will be better than only doing testing at the end, as with my proposed method of doing things I will know exactly what section needs correcting, if any. Here is each step in more detail:

Data Collection
I will be gathering a diverse dataset of facial images (whether it be from publicly available images on the internet or taking pictures of myself or consenting parties). This will help in training the machine learning algorithm, helping the software to detect a wide range of facial variations to ensure accurate detection and recognition.

Preprocessing
I will be applying preprocessing techniques to these images like image resizing or normalisation to prepare the data effectively for the machine learning stage. I will either resize manually using Photoshop or use an AI down/upscaling tool to retain quality with each resize. When connecting the built-in Tello camera to my drone, I will use the resize function in Python to ensure that the output window is as smooth as possible, whilst still being able to make out what is going on in the video footage.

Developing the Keyboard Control Algorithm
In this stage I will be programming the other main part of my software - recognising and responding to keyboard inputs made by the user. Considering that this will be a main part of my program, it is also imperative that this is coded to perfection. I should allow for a wide range of commands, all to instinctive keys on the keyboard (e.g. if W is forward, then backward should not be Backspace).

Developing the Facial Recognition Algorithm
In this stage I will be programming one of the main parts of the software, which is the facial recognition algorithm itself. It is imperative that the code for this section is written flawlessly and can perform facial detection in real-time, especially if my software is used for security purposes.

GUI Development
My program will also need a GUI (Graphical User Interface) for the user to engage with. Mine will be for portable computer devices (laptops and tablets) as it is fairly common for people to have, and is more convenient than a desktop computer. In this GUI there will be a toggle button/switch to go from Keyboard Control to Facial Recognition Following and vice versa.

Integrating the three Algorithms
This section will be me putting the three algorithms together (facial recognition code, keyboard control code, GUI code) in one Main python file. Since these will have been all complete already, I can insert them in and not worry about them any longer than I have to.

Testing, Review and Refinement
As I will be doing each section of the project for multiple hours at a time with breaks potentially varying in time whether it be hours or days, it is imperative that whenever I start another session of getting back into the code I first test what I have so far, review what went well and what needs to be fixed and then rectify these issues, and taking advantage of a fresh mindset to even refine my code, improving overall efficiency.

Create User-Friendly Documentation
I will have to make sure to make and provide a simple but effective piece of short documentation which teaches the user how to use my system's GUI and unlock the full functionality of DroneVision.

Iterative Improvement
As mentioned previously with Refinement, I will be making sure that every time I go back to my project I make sure that there are no gaps in my code which affects the code negatively in any way, using different testing types to make sure that the code itself is a reflection of my capabilities. I also will ensure that I am making continuous improvements and comments as I go along, so that on the rare chance that I take a long break from coding I can come back to my project with complete understanding of where I am at and what my code does. I will also use the pre-installed desktop application 'Sticky Notes' to mark where I am and what I need to do next. This will ensure that I do not get lost in development.

Divide and conquer
I will be going into this project with a mindset of 'Divide and conquer'. What this means in terms of my project is that I will use a level of Problem Decomposition so that I have many different areas to conquer. With these separate areas, I plan to make different Python files in my one project tackling each issue. This way I can not only import modules at will, but when combining keyboard control with facial recognition I can simply copy and paste sections from each .py file instead of starting from scratch.

Abstraction
Abstraction is crucial in almost all things to do with programming. It involves filtering out unnecessary information in order to concentrate on the elements and characteristics of my project that are important and necessary.

1.2 Stakeholders

Primary stakeholders would be individuals or groups who require drone surveillance, filming, photography, monitoring etc. They will benefit from this by now having the ability to autonomously track a person using facial recognition, or doing smoother, more stable movements with a familiar WASD keyboard control system. This also means that they will not have to be experts in flying a drone.

Stakeholders in the surveillance field could be from security companies or organisations, or even the police. The same program I create could be applied to a different system such as a security camera which tracks the suspect but does not follow.

Videos can already be filmed, or pictures taken by using the normal means of controlling a drone manually, but my proposed solution will provide a more efficient and automated way of capturing footage. This can be particularly beneficial in scenarios where precise tracking and smooth camera movements are required, such as in film production or event coverage.

In the film industry, directors and cinematographers are often looking for innovative techniques to capture captivating shots. By implementing facial recognition and keyboard control on the drone, filmmakers can achieve dynamic shots without the need for manual drone piloting. This can save time, resources, and potentially reduce production costs.

Photographers (both amateur and professional) can also benefit from my solution. They can use the autonomous drone to follow subjects while maintaining a steady shot. This opens up new avenues for creative compositions and perspectives, improving the overall quality of their work.

Law enforcement agencies, including the police, can also benefit from the capabilities of the autonomous drone for surveillance and investigation purposes. The drone's ability to track individuals and respond to inputs on a keyboard can assist in covertly tracking suspects, gathering evidence, and enhancing situational awareness in potentially dangerous situations.

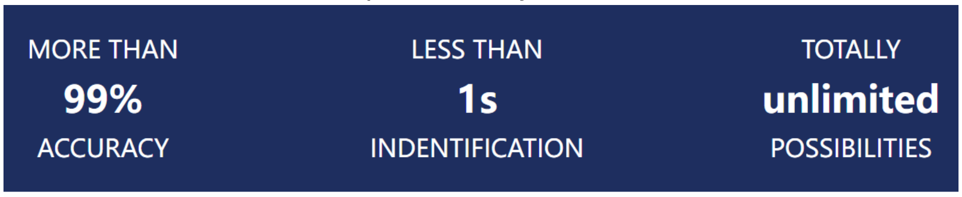
1.3 Researching The Problem

Existing similar solution:

FA6 Drone

The FA6 Drone is a facial recognition software made by Face-Six in 2017 that runs in real time and can identify criminals, missing people and/or civilians all from a drone’s camera (which is similar to my project). It can process recorded videos to find targets in crowds. This drone also has the capabilities to take snapshots from far away, which they flaunt on their website.

When searching for case studies where the FA6 Drone was used, I came up short. As of today, I could not find a single news article or anything showcasing how the FA6 Drone software was used to stop a crime or identify criminals. The only real proof that exists out there is on Face-Six's website, where they simply just state that the software CAN identify criminals and CAN take snapshots. They also flaunt this:

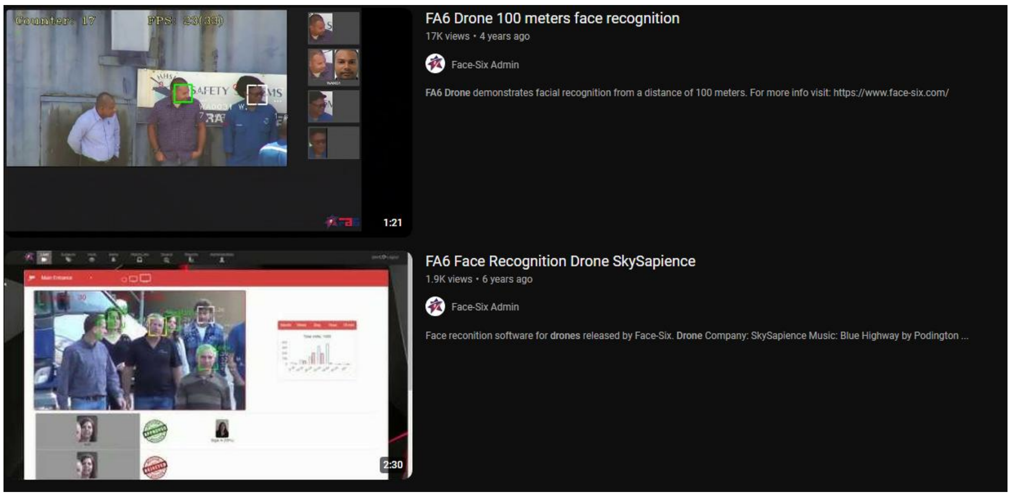


MORE THAN **99%** ACCURACY

LESS THAN **1s** INDENTIFICATION

TOTALLY **unlimited** POSSIBILITIES

But it is hard to verify since there exists no official coverage that proves it. I did manage to find two YouTube videos showcasing the features of the FA6 drone shown below;



FA6 Drone 100 meters face recognition

17K views • 4 years ago

Face-Six Admin

FA6 Drone demonstrates facial recognition from a distance of 100 meters. For more info visit: https://www.face-six.com/

1:21

FA6 Face Recognition Drone SkySapience

1.9K views • 6 years ago

Face-Six Admin

Face reconition software for **drones** released by Face-Six. **Drone** Company: SkySapience Music: Blue Highway by Podington ...

2:30

These videos are both from "Face-Six Admin", supposedly the official account for Face-Six with 34 subscribers despite the mass amount of views. The videos show the software being able to detect faces and matching them up to any face on a criminal record database if the faces match.

All in all, the FA6 drone does a great job on being able to detect the faces even if blurry due to the subjects being far away, which can prove to be useful in law enforcement. The efficiency that it has from scanning the face all the way up to matching it with the criminal records database is quite fast too.

However there does exist some privacy concerns, specifically regarding the unauthorised collection of personal data. The current system also seems to struggle with identifying people with face accessories on, like hats or sunglasses, which may not prove effective in a real-life crime situation.

<u>Why is it suited to a computational solution?</u>

The problem at hand is perfect for a computational solution using Artificial Intelligence. This is because AI works much faster than humans without an immense amount of external input. Using AI will be cheaper too, as purchases would most likely be a one-time purchase of the service, or at most a monthly/annually payment based subscription service for infinite access, rather than a contract or payment per job done.

In addition to this, AI is generally much faster, cheaper, and readily available than humans. This can assist in maximising time efficiency whilst minimising costs, as well as not using human resources so other members involved can be distributed elsewhere (in a large production for example). My solution of a facial/keyboard control drone can also mitigate casualties or injuries, being able to follow in risky situations from a distant device (i.e. chasing a criminal).

1.4 Specifying the proposed solution

Based on the research above, the proposed solution involves developing a software system that integrates facial recognition and keyboard control algorithms with the existing control system of a drone equipped with a built-in Tello 720p camera.

The software will be developed using a combination of Python and OpenCV, which is suitable for computer vision and machine learning tasks. The facial recognition module will utilise learning techniques to detect and identify faces in real-time, while the keyboard control module will require advanced coding and an understanding of computer systems.

The software system will be designed to run on a lightweight computer so that it can be used in any scenario, ensuring efficient processing and minimal latency. The drone's control system will enable the software to provide autonomous tracking capabilities based on facial recognition and respond to predefined keyboard inputs for commands such as takeoff, land, forward, backward, left, right, up, down or changing camera angles.

1.5 Meeting Questions

Frequently meeting with a client/stakeholder will always be the best way to gather information about what your project should look, feel and operate like. It builds the client-programmer relationship further with frequent meetings and also allows you to ask questions that may not have been asked prior to meeting. It is important to get the frequency of meeting correct, however, considering that meeting too frequently or not frequent enough can put the stakeholder off and lead to them losing interest in the project. When meeting with a client to discuss a project involving drone technology, facial recognition, keyboard input detection, and GUI development, it's essential to gather a clear understanding of their requirements and expectations. I have chosen to meet in person with stakeholder Tyler Beck, a 19 year-old Photographer and short film director/producer, and founder of *Lemon Shot Productions*, a Bristol-based small studio who films visualisers and music videos for musical artists, as well as short-film passion projects. Below are the questions that I will ask:

<u>Questions</u>

1) What exactly is your vision for the project?
2) What programming language would you like the program to be in?

3) What general colour scheme would you like for the GUI?
4) Is there a deadline for this project?
5) What mode of communication would you like to have for me to provide updates?

Answers

1) I would love a drone which can follow the musician as they walk backwards, facing the drone. That would make for some innovative close-up shots for the ethereal trance-like music videos we have planned for the future. It would also be nice to have wireless control over the drone via laptop keyboard or tablet touchscreen, so that we can squeeze through tight spaces and get high to low shots.
2) I understand that the facial recognition will involve a threshold which determines when the drone goes backwards, forwards, and staying still. I am not an absolute expert in programming but still remember a fair amount from secondary school. I would like to be able to alter this threshold easily, and Python is what I have most experience with.
3) To match the company's colours, It is crucial for the main colour scheme to be black and yellow.
4) Since we have a schedule planned out for the rest of the year, I can say that we would need this project by the end of June 2024. This is when we will be filming ethereal music videos and would need your drone's close up following shots.
5) I believe you have my phone number already, I'd like feedback via phone call whenever a big section of the project is done. 3 months before the deadline, I will happily come over to London for an ETA of the project and test it out myself to see if there are any changes or improvements to be made.

Review and takeaways from the meeting:

Overall, the meeting was a large success. What I can take away from this project are the following requirements:

- **Must have Facial Recognition with a toggle for Keyboard Control:** If not important enough before, Tyler has reiterated the importance of having both features readily available for their close-up shots and squeezing into tight spaces with the drone.
- **Programming language MUST be in Python:** Tyler has limited coding experience, and to ensure complete user satisfaction I will need to code my project in Python so that he can make suitable alterations to the code to whatever he sees fit. This is fine for me, as Python is the programming language I am most comfortable with by far. It will still not be a breeze, as this will be the most challenging and complex project I have tackled by far, and will still need to study and learn new functions and methods, especially with the OpenCV library.
- **Yellow and Black colour scheme:** This actually reduces stress for me when making the GUI, I plan to make custom graphics for the toggle switch and frame which the drone's output is displayed in. A black background with yellow text/graphics will work nicely visually.
- **Deadline - June 2024:** I will need to work hard, managing my time effectively so that I will have a finished product by the end of June 2024, and am close to finishing 3 months beforehand.
- **Make frequent phone calls to the client:** I do have Tyler's phone number, and I will take advantage of that by calling him with updates when each major section is complete. I need to make sure that I do not push it though, as excessive calls or calls

too late at night/ early in the morning could be overwhelming for him and lead to him not wanting anything to do with my project.

1.6 Requirements

In my project it's important to define the types of requirements that can assist its development. These can be split into three types of requirements; Functional, Non-Functional and Technical. Functional requirements are the specific functionalities or behaviours of the system, like the ability to recognise faces and respond to keyboard commands. On the other hand, non-functional requirements are criteria that judge how the system will run, such as performance efficiency and usability. Finally, technical requirements outline the specific technical aspects, including programming languages and hardware specifications. Henceforth, below is how I have decided to detail the requirements for my project.

Functional Requirement	**Solution**
Facial Recognition	I will develop a real-time facial recognition system using the Tello drone's 720p camera and the OpenCV library, which is capable of identifying faces under different (but reasonable) lighting conditions.
Keyboard Control	I will create a keyboard control interface for the drone, ensuring it responds smoothly to different commands/keystrokes and is easy to navigate.

Non-Functional Requirement	**Solution**
Performance	The goal is minimal latency in facial recognition and command response, which I aim to do by resizing the image to a smaller size, whilst still having the events occurring in the footage being visible.
Usability	The graphical user interface will be intuitive and adaptable for users, allowing easy toggling between control modes.

Technical Requirement	**Solution**
Programming Language	I will be using Python, taking advantage of its strengths in computer vision and drone control with the programmable Tello drone.
Colour Scheme	The GUI will be designed in black and yellow, aligning with Lemon Shot Productions main colours and brand identity.

Apart from this, there are other requirements which do not fall under any of these categories which are listed below:

Other Requirement	**Solution**
Stakeholder’s Requirements	**-The system will be coded in Python to allow Tyler Beck to seamlessly adjust facial recognition parameters for different environments, by having these parameters at the top of the main code.** -The prototype is planned for March 2024, with project completion by June 2024.
Testing and Validation	**-I will frequently conduct iterative testing with various data types to ensure my system stays working at all times, even with new additions to code.** -I will form contact with stakeholder Tyler Beck for regular validation, ensuring my project meets his needs and expectations.
Documentation	**-I will provide clear and concise documentation, making the system easy for Tyler to understand and operate.**
Hardware Requirements	-I will need access to a mouse (to select one or more words in my code for copying), a monitor (to see and review my code in real-time), a keyboard (for writing the actual code itself) and suitable RAM (for having multiple processes running whilst testing, for research). Luckily, I already have access to all of this in the comfort of my own home even with a Personal Computer with 32GB of RAM.
Software Requirements	**-I will need access to any Integrated Development Environment but my IDE of choice is PyCharm due to it being user friendly and being easy to install libraries. I have already downloaded and installed this in preparation for this project.**

Limitations

I recognise that whilst I may have extensive experience already, working on such an ambitious project like this with specified deadlines will not be perfect and may have its own flaws, whether it be due to human error or other elements out of my control. Below is a table containing the limitations that I believe this project may lend itself to.

Limitation	**Reason**
Facial Recognition Accuracy	-For the majority of facial recognition systems, the accuracy of facial recognition can vary depending on lighting conditions, angles, and distances. My DroneVision might struggle in low light, faraway targets or with fast-moving subjects.
Drone Battery Life	**-The operational time of the Tello drone is limited by its battery life. Its maximum possible flight time is 13 minutes when merely hovering in the air, with actual flight time having an average battery life of 8-10 minutes. This combined with prolonged and continuous use of facial recognition and uninterrupted flight might drain the battery faster.**
Environmental Constraints	-The system's performance could be affected by environmental factors like wind, rain, or other weather conditions, which might impact drone stability and camera visibility (e.g. a raindrop smudging the camera's vision).
Range and Signal Interference	**-The effective range of the drone's control and facial recognition might be limited, considering the Tello Drone only has access to the code by outputting a Wi-Fi signal to connect to the device hosting the code. Signal interference or obstacles could disrupt the communication between the drone and the controlling device.**
User Interface and Control Responsiveness	-There may be a learning curve for users to efficiently switch between and utilise both modes of control, the code's full potential not being realised. Additionally, responsiveness could vary depending on the processing power and latency.

Success Criteria

No.	**Requirements**	**Justification**	**What demand does it meet?**
1	Drone connects to control device	To enable user interaction and control over the drone.	Project Requirement
2	Live video feed display	Allows the user to see through the drone's camera in real-time.	Project Requirement
3	Keyboard control for manual operation	Provides precise manual control with Keyboard inputs when autonomous tracking is not needed.	Stakeholder Interview
4	Autonomous face tracking	This will allow the drone to follow a target's face without manual intervention.	Project Requirement
5	Toggle between control modes	Offers flexibility in how the drone can be controlled.	Stakeholder Interview
6	Emergency landing functionality	Ensures safety and quick response in case of issues, preventing potential damage to the drone itself or parties involved.	Safety Essentials
7	Smooth transition between modes of control	Provides a seamless user experience, with little-to-no wait time.	Enjoyable User Experience
8	Battery status message	Informs the user about the remaining battery life in the beginning of flight.	Project Requirement
9	Responsive control to input	Ensures that the drone reacts quickly to user commands, with each input giving its expected output.	Enjoyable User Experience
10	Stability during flight	Maintains steady flight throughout code execution for better video feed and no pull-back from control.	Enjoyable User Experience

11	Durability in various conditions	Drone can operate in a range of environments.	User Requirement
12	Easy-to-use interface	Users can intuitively control the drone and understand its status, even without the User Documentation.	Enjoyable User Experience
13	Adequate user documentation	Provides the user with necessary and adequate information for users to operate the drone in both modes.	Project Requirement
14	Compliance with safety regulations	Meets legal and safety standards for drone operation.	Legal Requirement
15	System testing	Confirms that all features work as intended, with no unforeseen errors.	Quality Assurance
16	Colour scheme / GUI Design	Enhances user experience and aligns with accessibility standards, as well as stakeholder requirements.	Enjoyable User Experience / Stakeholder Interview.
17	Error handling / User feedback	Informs users of potential errors and provides guidance for resolutions.	Enjoyable User Experience
18	Face tracking performance in various lighting conditions	Ensures the drone can still operate effectively even in different lighting.	User Requirement
19	Extensive control range (distance)	Allows the drone to be operated wirelessly from a large distance.	Technical Requirement
20	Low-latency video transmission	Provides real-time responsiveness, which leads to better control for the user.	Technical Requirement
21	Robustness against signal obstacles	Ensures consistent performance despite potential disruptions/ obstacles to the Tello Wi-Fi signal output.	Technical Requirement

Design

2.1 Decomposition List

DroneVision:

1. Control Systems
    a. Facial Recognition System
    b. Keyboard Control System
2. Core Functionalities
    a. Drone Navigation
    b. Camera Operation
3. User Interface
    a. Live Video Feed Display
    b. Colour Scheme
    c. Mode of Control Indicator
    d. Emergency Landing Button
4. Input Mechanisms
    a. Camera Input for Facial Recognition
    b. Keyboard Input for Keyboard Control
5. Software Components
    a. Face Detection Algorithm using OpenCV
    b. Key Press Module
6. Safety Features
    a. Emergency Land Functionality
7. Data Processing
    a. Image Processing and rectangle-drawing for Facial Recognition
    b. Keystroke Translation for Keyboard Control leading to Drone Response
8. Testing and Validation
    a. Functional Testing for each Mode of Control
    b. Safety and Compliance checks (getting too close/under/over)
    c. Creating User Documentation

Above is my decomposition list for my project. I believe that presenting it in this order will be beneficial when it comes to tackling my project, as I can work on separate issues one by one instead of all together. Below is a short justification for why I chose to include each one of these.

Point	**Justification**
Facial Recognition System	**Implemented to enable autonomous**

	drone tracking, which matches the stakeholders’ requirements for being used in photography and surveillance without manual control.
Keyboard Control System	My program should be able to toggle to manual Keyboard control with the press of a button for precise control for situation where autonomous following may not be the most feasible option.
Drone Navigation	**This is fundamental to the drone's operation, allowing for movement and the fulfilment of the project's core objective, which is to provide two different modes of movement for the Tello drone.**
Camera Operation	Central to both methods of movement (facial recognition and keyboard control), the camera serves as the primary input for visual data, which users can observe from the GUI.
Live Video Feed Display	**Provides real-time visual footage to the user, essential for both navigation and monitoring the environment.**
Colour Scheme	Enhances the user experience and aligns with accessibility standards, as well as Lemon Shot Productions’ request.
Mode of Control Indicator	**Provides clear visual cues for the current control mode (either Facial Tracking or Keyboard Control), improving user orientation, awareness and safety.**
Emergency Landing Button	Acts as a fail-safe mechanism to ensure immediate response in case of malfunction or emergency.
Camera Input for Facial Recognition	**Imperative to the autonomous tracking system, enabling the drone to detect and follow subjects.**
Keyboard Input for Keyboard Control	Make sure that the user can interact with the drone directly, commanding its movement when necessary with keyboard inputs.
Face Detection Algorithm using OpenCV	**Uses open-source computer vision tools for reliable facial detection.**
Key Press Module	What I will use as part of the Keyboard

	Control. it interprets user keystrokes into actionable commands.
Emergency Land Functionality	**Imperative for lowering risks and ensuring the drone can be safely recovered during unexpected events, like weather complications.**
Image Processing and rectangle-drawing for Facial Recognition	Processes visual data for tracking and provides visual cues on the interface, such as a red rectangle around the person's face with a green circle in the centre.
Keystroke Translation for Keyboard Control leading to Drone Response	**Will convert the user's input into drone movements, allowing for intuitive and familiar manual operation.**
Functional Testing for each Mode of Control	Verifies the effectiveness and reliability of both control modes.
Safety and Compliance checks (getting too close/under/over)	**Ensures the system adheres to regulatory standards and operates safely within the intended environment space.**
Creating User Documentation	User Documentation must be created to ensure that users know how to use and modify the code if needed.

2.2 Data Table

Below is a Data Table for all pieces of data across my 7 files:
basicMovements.py

Variable Name	**Data Type**	**Explanation**
drone	Tello Object	The drone object from the Tello SDK used for sending commands to the drone.
sleep	Function	Function from time module used to delay execution between commands.

faceTracking.py

Variable Name	**Data Type**	**Explanation**

cv2	Module	OpenCV library used for image processing.
np	Module	NumPy library used for numerical operations.
tello	Module	Tello library used for controlling the drone.
time	Module	Module used for time-related functions like sleep.
os	Module	Module used to interact with the operating system.
drone	Tello Object	The drone object from the Tello SDK used for sending commands to the drone.
width	Integer	Width of the image frame from the drone's camera.
height	Integer	Height of the image frame from the drone's camera.
forwardBackwardRange	List	Range values that define the 'sweet spot' for drone movement.
pid	List	PID controller coefficients.
pError	Integer	Previous error value used in PID control.
faceCascade	Object	Haar cascade classifier for face detection.
imgGrayscale	Image	Grayscale version of the image for face detection.
faces	Array	Array of detected faces from the cascade classifier.
faceListC	List	List of face centres detected in the drone’s Point of View.
faceListA	List	List of face areas detected in the drone’s Point of View.
cx	Integer	X-coordinate of the centre of the face.
cy	Integer	Y-coordinate of the centre of the face.
area	Integer	Area of the detected face.

error	Integer	Deviation of the face center from the image center.
speed	Integer	Speed calculated by the PID controller for drone movement.
forwardBackward	Integer	Forward or backward movement speed of the drone.
img	Image	Image frame captured from the drone's camera.
info	List	Information about the face center and area.

imageCapture.py

Variable Name	**Data Type**	**Explanation**
tello	Module	Tello library used for controlling the drone.
cv2	Module	OpenCV library used for image processing.
drone	Tello Object	The drone object from the Tello SDK used for sending commands to the drone.
img	Image	Image frame captured from the drone's camera.

keyboardControl.py

Variable Name	**Data Type**	**Explanation**
tello	Module	Tello library used for controlling the drone.
kpm	Module	Custom module made by me for detecting key presses.
drone	Tello Object	Instance of the drone from the Tello library.
leftRight	Integer	Lateral movement speed; negative for left, positive for

		right.
forwardBack	Integer	Forward/backward movement speed; positive for forward, negative for backward.
upDown	Integer	Vertical movement speed; positive for up, negative for down.
yawVelocity	Integer	Rotation speed; negative for left turn, positive for right turn.
speed	Integer	Constant value for the drone's movement speed.
values	List	List of speed values for drone movement in each direction.

keyPressModule.py

Variable Name	**Data Type**	**Explanation**
pygame	Module	Pygame library used for creating windows and handling keyboard events.
win	Pygame Object	Window object created by Pygame for display.
keyName	String	Name of the key for which the press status is being checked.
answer	Boolean	Indicates whether the specified key is pressed (True) or not (False).
ketInput	List	List of booleans representing the press status of all keys.
myKey	Integer	The key identifier for the specified key in Pygame.

mapping.py

Variable Name	**Data Type**	**Explanation**
math	Module	Module used for mathematical operations, such as cos and sin.
time	Module	Module used for time-related functions like sleep.
sleep	Function	The drone object from the Tello SDK used for sending commands to the drone.
tello	Module	Tello library used for controlling the drone.
kpm	Module	Custom module made by me for detecting key presses.
numpy	Module	NumPy library used for numerical operations and array handling.
cv2	Module	OpenCV library used for image processing.
fowardSpeed	Float	Forward speed of the drone in cm/s.
angularSpeed	Float	Angular speed of the drone in degrees/s.
interval	Float	Time interval for drone movement updates.
distanceInterval	Float	Distance covered by the drone in each interval.
angularInterval	Float	Angular movement of the drone in each interval.
x, y	Integers	Coordinates representing the drone's position.
angle	Integer	The current angle of the drone's movement in degrees.
yaw	Integer	The yaw movement of the drone.
points	List of Tuples	List of coordinate points representing the drone's path.

leftRight, forwardBack, upDown, yawVelocity	Integers	Movement variables for lateral, forward/backward, vertical, and rotational movement of the drone.
angularSpeed	Integer	Speed for angular movement.
speed	Integer	Constant value for the drone's movement speed.
distance	Float	Distance moved by the drone in the current interval.
img	NumPy Array	Image array used for drawing the drone's path visualisation.

Project-keyboardControlImageCapture.py

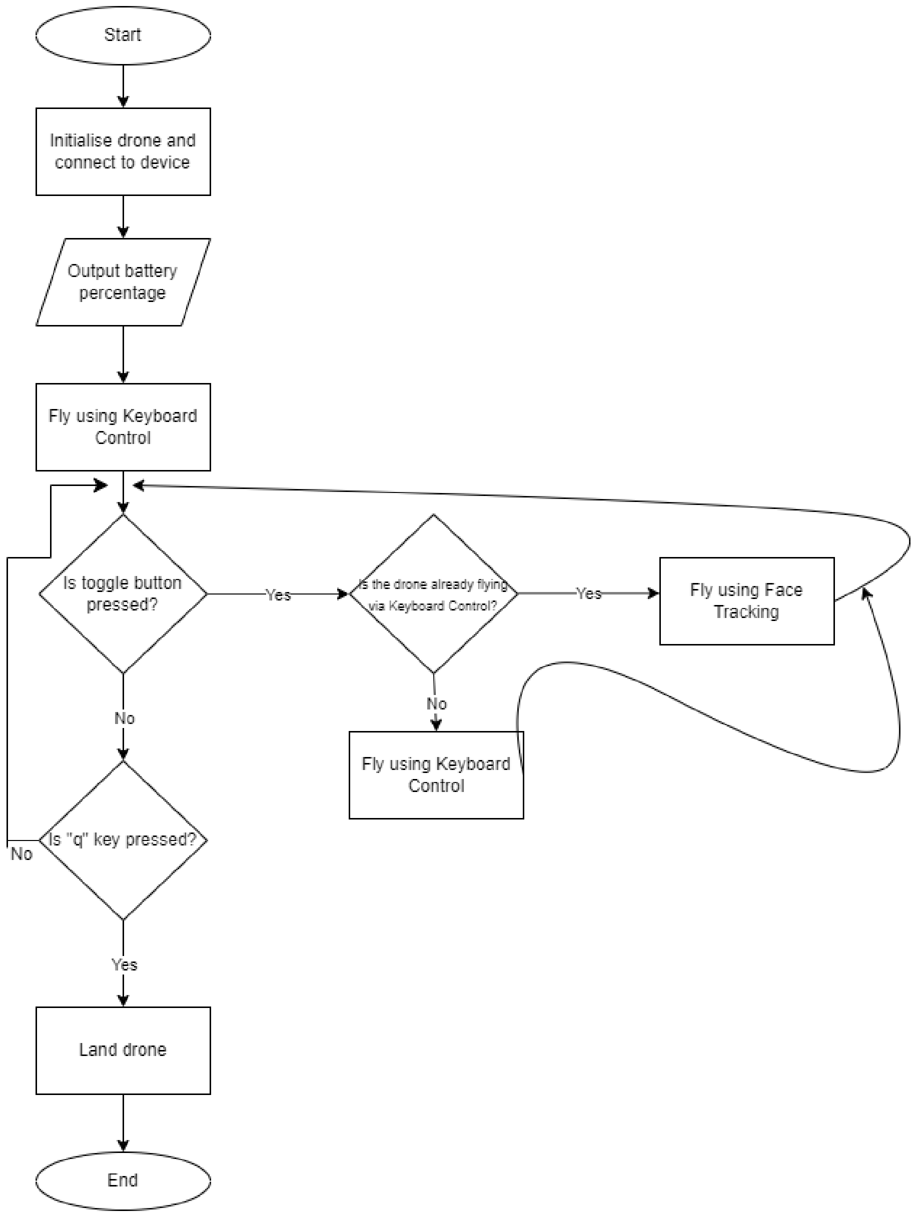
Variable Name	**Data Type**	**Explanation**
time	Module	Module used for time-related functions like sleep.
tello	Module	Tello library used for controlling the drone.
kpm	Module	Custom module made by me for detecting key presses.
cv2	Module	OpenCV library used for image processing.
drone	Tello Object	Instance of the drone from the Tello library.
img	Image	Image frame captured from the drone's camera.
leftRight	Integer	Lateral movement speed; negative for left, positive for right.
fowardBack	Integer	Forward/backward movement speed; positive for forward, negative for backward.
upDown	Integer	Vertical movement speed; positive for up, negative for down.

yawVelocity	Integer	Rotation speed; negative for left turn, positive for right turn.
speed	Integer	Constant value for the drone's movement speed.
values	List	List of speed values for drone movement in each direction.

2.3 Flowcharts

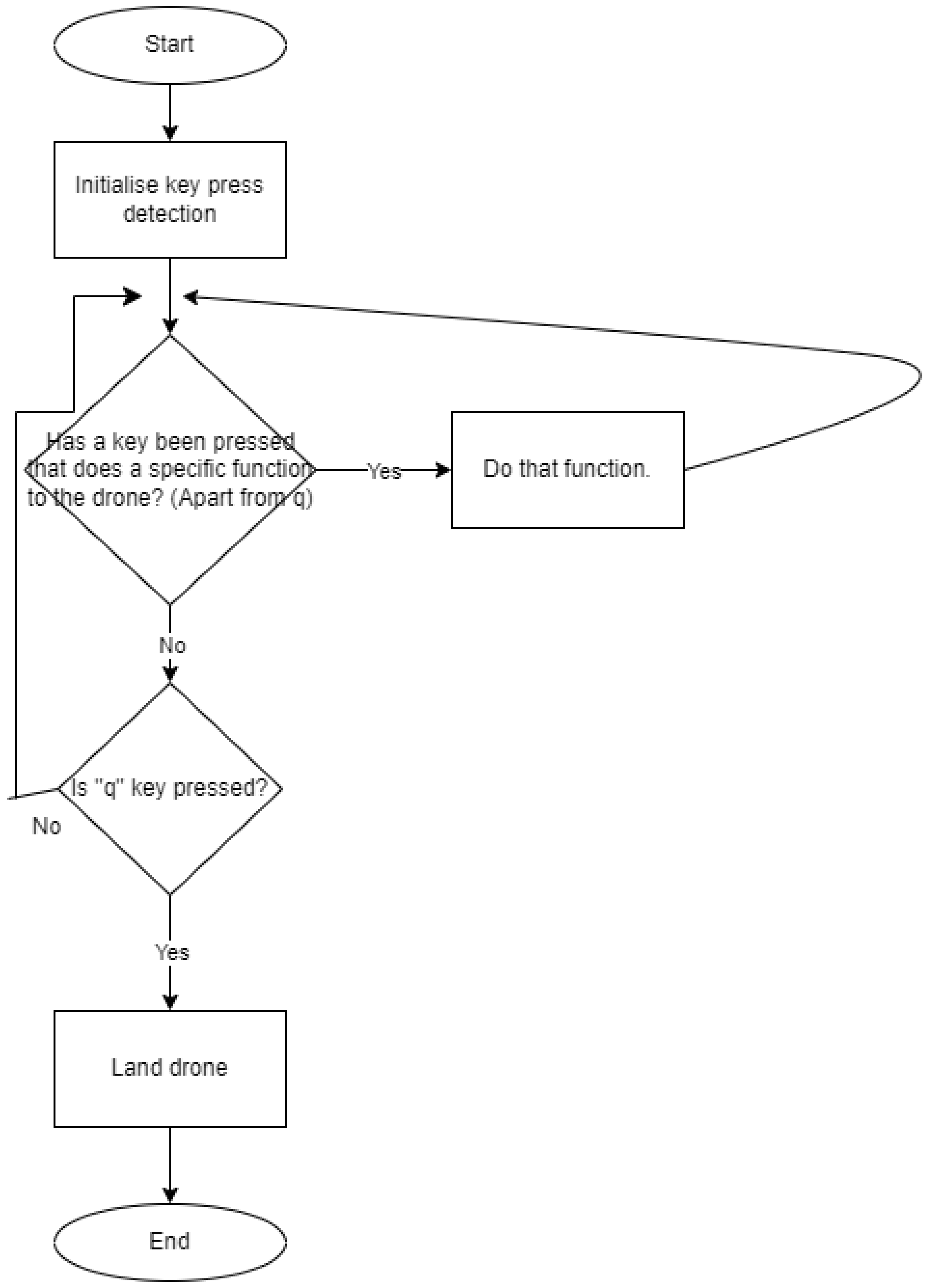
A flowchart is a diagram showing logical steps and the order that they are carried out. It is vital for me to make flowcharts for each and every main function of my system, so that when I am developing the software I can follow logical steps, avoiding confusion and increasing time efficiency. Below are my flowcharts that I, alone, have come up with.

1. **Overall System Flowchart**


Start
Initialise drone and connect to device
Output battery percentage
Fly using Keyboard Control
Is toggle button pressed?
Yes
Is the drone already flying via Keyboard Control?
Yes
Fly using Face Tracking
No
Fly using Keyboard Control
No
Is "q" key pressed?
No
Yes
Land drone
End

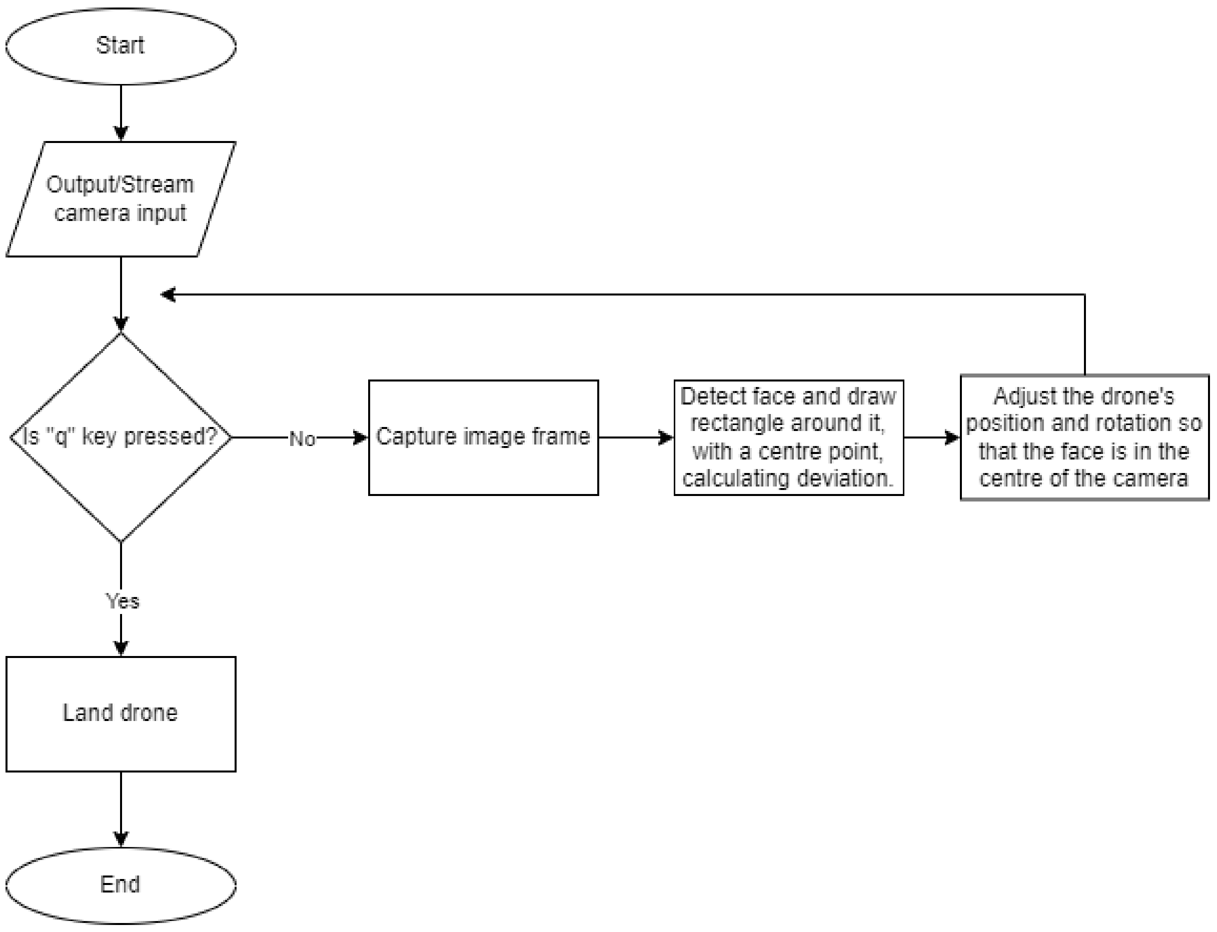
This flowchart presents the overarching logic of my project’s code. It starts with the initialisation of the drone and connecting to the device, followed by outputting the drone's battery percentage for the user to be wary of. It will then start with Keyboard Control, until toggled. The main decision point revolves around whether the toggle button is pressed, which determines if the system should fly using Keyboard Control or switch to Face Tracking mode. There is also a decision point for an emergency landing with the "q" key press, which is when the program will end. This flowchart is crucial for visualising the operation of the system, showing how it transitions between different modes of operation and how it responds to user inputs for control and emergency procedures.

2. Keyboard Control Flowchart


Start
Initialise key press detection
Has a key been pressed that does a specific function to the drone? (Apart from q)
Yes
Do that function.
No
Is "q" key pressed?
No
Yes
Land drone
End

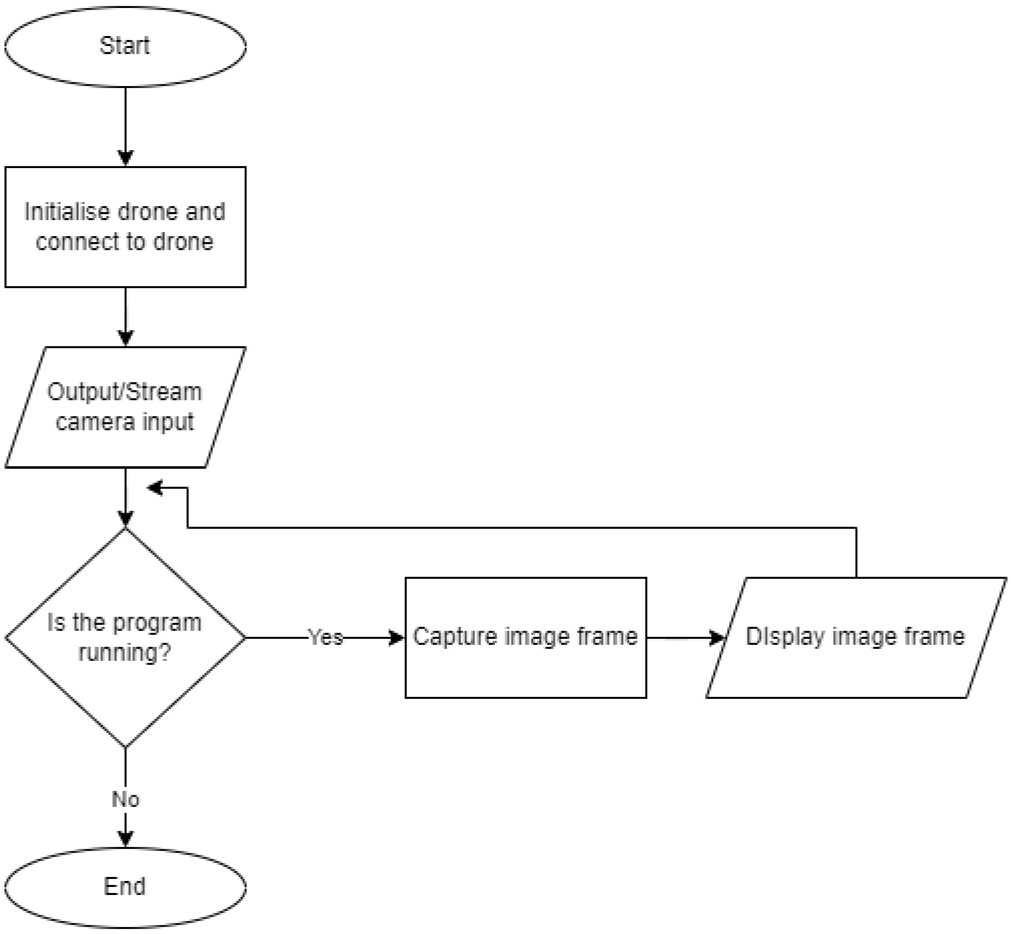
This flowchart shows the process of controlling the drone using keyboard inputs. It begins with initialising key press detection and enters a loop where it continually checks if a key related to a specific drone function has been pressed, apart from the "q" key which is reserved for landing. If a function-specific key is pressed (e.g. “w”, “a”, “s”, “d”, “e”, or any of the arrow keys), it performs the corresponding action (e.g., moving left, right, up, down). The program then is also continuously checking for the "q" key to be pressed and, if pressed, the drone lands and the program ends. This flowchart is essential for understanding the manual Keyboard Control aspect of the project and how user input translates into drone movements.

3. Face Tracking Flowchart


Start
Output/Stream camera input
Is "q" key pressed?
No
Capture image frame
Detect face and draw rectangle around it, with a centre point, calculating deviation.
Adjust the drone's position and rotation so that the face is in the centre of the camera
Yes
Land drone
End

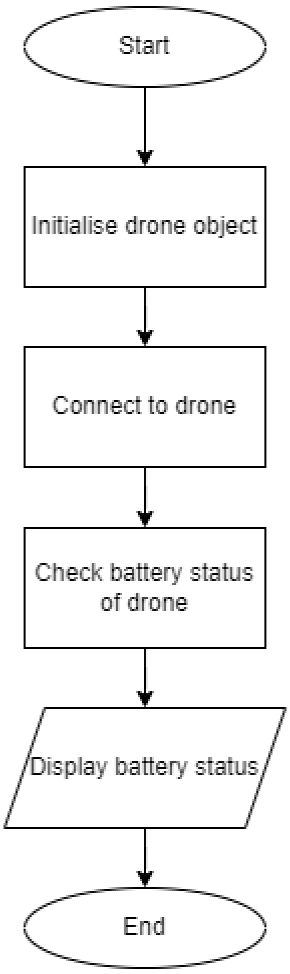
This flowchart shows the Face Tracking functionality of the project. It starts with outputting the camera input and enters a loop where it captures image frames from the drone for as long as the drone has not landed. The system then detects a face within the frame, draws a rectangle around it with a centre point, and calculates the deviation from the centre to adjust the drone's position and rotation accordingly. The flowchart includes a decision point for the "q" key press, which leads to the drone landing and the end of the program. This flowchart is important to demonstrate how the drone autonomously follows a subject using visual input, helping me when it comes to actual development.

4. Image Capture Flowchart



The image capture flowchart depicts the process of streaming camera input to the monitor. After initialising the drone and connecting, the system enters a loop checking if the program is running. It captures image frames and displays them, continually looping until the end of the program. This flowchart will be key for the project, being used in most of the Python files made. It represents the method that the system provides video footage from the drone to the user.

5. Initialisation and Connection Flowchart



Much talk has been had about initialising the drone in every flowchart. But this straightforward flowchart merely states the minimal steps taken to actually initialise the drone object, connect to the drone, check the battery status, and display it. There are no decision points here, just a sequence of initialisation and user experience tasks that prepare the drone for operation. This flowchart is the most important part of the setup phase as it must take place before any control actions can be performed.

2.4 Data Structures

As shown previously in the Data Table on page 16, here is the complete list of all carefully handpicked Data Structures that will be used among my project, and what they will be used for.

Variable Name	Data Type	Explanation
faceListC	List	List of face centre coordinates detected in the image.
faceListA	List	List of face areas detected in the image.
points	List of Tuples	List of tuples representing points (x,y) in the drone's path.

pid	List	PID controller coefficients as a list.
forwardBackwardRange	List	List containing the range values for forward and backward movement (aka. the ‘sweet spot’).

2.5 Usability Features

In my project, usability is not just an afterthought; it is a cornerstone of the design. With the integration of a Graphical User Interface (GUI), I aim to make the system accessible, intuitive, and tailored to user needs. Here's how I plan to achieve this:

Usability Feature	**Explanation**
Intuitive GUI Layout	With an intuitive GUI layout, the GUI will be laid out to naturally guide users through the drone's operation, through instinct alone. Key information like battery life and mode of control will be prominently displayed. The live video feed, which is central to the drone's operation, will take up the top middle of the screen to provide users with immediate visual footage.
Visible Touchscreen Controls	With touchscreen controls which are clearly visible, controls will be clearly labelled and organised in a logical manner within the GUI. Users will be able to fly the drone using either keyboard controls with an actual keyboard or touchscreen buttons that are displayed on the GUI, ensuring quick learning and ease of use.
Responsive Design	With keeping a responsive design in mind at all times when programming the GUI, I can ensure that as many stakeholders can use my product as possible, as I aim to create a GUI that will adapt to various screen resolutions. This will ensure that all interface elements are always visible, legible and accessible
Guidance with Errors	With an unexpected error, I will ensure that the User Documentation contains step by step information to resolve it. This approach minimises frustration and downtime, which would possibly lead to closing the GUI and program.
Tutorial Section	With a tutorial section in my GUI, This will encourage users to learn at their own pace

	and improve their proficiency with the system.

2.6 Test Data

In ensuring that my project meets all the requirements laid out in the Analysis, I have established a comprehensive testing plan. This plan includes various types of test data to validate the system's functionality under different scenarios. Each type of test data is designed to rigorously challenge the system and identify any potential issues.

Normal Test Data: These data points reflect typical user inputs and interactions that the system is expected to handle during regular operation. For instance, keyboard inputs for controlling the drone's movement will be tested to ensure they trigger the correct responses. (W for forward, S for backward etc.)

Boundary Test Data: This set of data will test the limits of the system's parameters, such as the maximum and minimum values for the drone's speed and altitude. The goal is to confirm that the system can handle edge cases without malfunctioning or crashing.

Invalid Test Data: Here, I will input data that the system should reject, such as incorrect key presses or commands that are out of context. These tests are crucial for verifying the robustness of the system's error handling and validation mechanisms.

Erroneous Test Data: These tests will intentionally introduce errors, such as commands to move the drone when it's not in flight, to ensure that the system can gracefully handle unexpected situations.

For each test, I will document the test case, the type of test, the expected outcome, the actual outcome, and whether it passed or failed the test. If the test fails, I will work on rectifying this. This documentation will not only facilitate troubleshooting but also serve as a record of the system's reliability and robustness.

<u>Keyboard Control Test Table</u>

Test Case	**Type of Test**	**Expected Outcome**
‘W’ key is pressed.	Normal	The drone should move forwards in a straight horizontal line with a constant velocity until the ‘W’ key is released.
‘A’ key is pressed.	Normal	The drone should rotate left with a

		constant yaw velocity until the ‘A’ key is released.
‘S’ key is pressed.	Normal	The drone should move backwards in a straight horizontal line with a constant velocity until the ‘S’ key is released.
‘D’ key is pressed.	Normal	The drone should rotate right with a constant yaw velocity until the ‘D’ key is released.
‘UP ARROW’ key is pressed.	Normal	The drone should move upwards in a straight vertical line with a constant velocity until the ‘UP ARROW’ key is released.
‘DOWN ARROW’ key is pressed.	Normal	The drone should move downwards in a straight vertical line with a constant velocity until the ‘DOWN ARROW’ key is released.
LEFT ARROW’ key is pressed.	Normal	The drone should move leftwards in a straight horizontal line with a constant velocity until the ‘LEFT ARROW’ key is released.
‘RIGHT ARROW’ key is pressed.	Normal	The drone should move rightwards in a straight horizontal line with a constant velocity until the ‘RIGHT ARROW’ key is released.

'E' key is pressed.	Normal	Takeoff sequence - The drone's propellers should slowly start spinning, lifting the drone off of the ground.
'Q' key is pressed.	Normal	Emergency landing - The drone slowly descends and the propellers power off.

Facial Recognition and Tracking Test Table

Test Case	**Type of Test**	**Expected Outcome**
No face is in the camera's vision / Out of frame.	Normal	The drone should stay still until it detects a face in its vision.
A face is in the centre of the drone's camera footage.	Normal	The drone should either move forwards or backwards, depending how close the person is to the drone
A face is to the *very* left of the drone's camera footage.	Boundary	The drone should quickly rotate to the left, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person's face in the centre.
A face is *a little* to the left of the drone's camera footage	Normal	The drone should slightly rotate to the left, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person's face in the centre.
A face is to the *very* right of the drone's camera footage.	Boundary	The drone should quickly rotate to the right, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person's face in the centre.
A face is *a little* to the right of the drone's camera footage.	Normal	The drone should slightly rotate to the left, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person's face in the centre.

The face detected is too close to the drone.	Erroneous	The drone should move backwards until the area of the face is within the specified range.
The face detected is too far from the drone.	Erroneous	The drone should move forwards until the area of the face is within the specified range.
The target detected moves backwards at a constant speed, whilst the drone can see their face.	Normal	The drone should move forwards to follow the target as the face area is decreasing and it should want to aim to get the area within the sweet spot
The target detected moves forwards at a constant speed, whilst the drone can see their face.	Normal	The drone should move backwards to gain distance from the target as the face area is increasing and it should want to aim to get the area within the sweet spot
The target's face is in the 'sweet spot'.	Normal	The drone should not move forwards or backwards, and should continue tracking the target and rotating if needed.
'Q' key is pressed.	Normal	Emergency landing - The drone slowly descends and the propellers power off.

General Test Table

Test Case	**Type of Test**	**Expected Outcome**
'T' key is pressed.	Normal	The system should toggle in between two modes of control - Facial Tracking and Keyboard Control.
faceTracking.py is ran.	Normal	The drone should automatically elevate to an average height for a human to start detecting faces.

2.7 GUI Design

Due to the fact that my project will be able to run on laptops and tablet computers, it will need a Graphical User Interface (GUI) to allow the user to intuitively interact with the drone by pressing certain keys on the keyboard or tapping on graphical components on the screen like buttons. This will also allow the user to see what the drone sees, making it easier to control rather than having to actually look at the physical drone to interact with it effectively.

<u>Requirements of the GUI:</u>

- **Movement buttons:** This includes the arrow keys for the tablet port and an on screen visual of the arrow keys on a keyboard, highlighting whenever an arrow key on the keyboard is pressed. This also includes additional movement features such as 360° spins and frontflips already included in the official Tello app, making it easier to implement using the djitellopy library.
- **Drone POV:** This will be available on both versions on the GUI. This will showcase the drone's point of view in real time, making frustrating moments when avoiding obstacles and traversing scenery with the drone a thing of the past.
- **Toggle button:** This button has to be in the final product in order for the user to tell whether the drone is in facial tracking mode or keyboard control mode. This also tells the drone whether to start tracking and following faces or start receiving inputs from the laptop's keyboard/ accept touchscreen inputs.
- **Tutorial Button:** This should be a small button in the top right corner, allowing users to click on it and then an element on the screen to get information on how it works and how to interact with it.
- **Black and Yellow colour scheme:**

In general, the tablet port will have to be more in depth, as there are no physical buttons/keys that they can press and will have to be guided into how to effectively operate the program. This overarching task only reminds me that my GUI must be intuitive and efficient.

Software Development

In order to be organised and efficient, I decided to write the code for each file as I went on and as I remained in a focused mindset, but conducting iterative and tireless testing in the midst of programming, giving myself time to rest and recover, which also gave the project a chance to grow beyond I thought was possible when initially choosing the project. I also made sure to leave comments on nearly every line as I went on, so that I can always understand what section of code I was on, even when it was getting more advanced than I have been used to.

3.1 basicMovements.py *(31/07/2023, 17:09)*

basicMovements.py was the first Python file I made for this project, and was made for the sole purpose to get me, the programmer, used to djitellopy and its library. I wrote a simple code which would do the following:

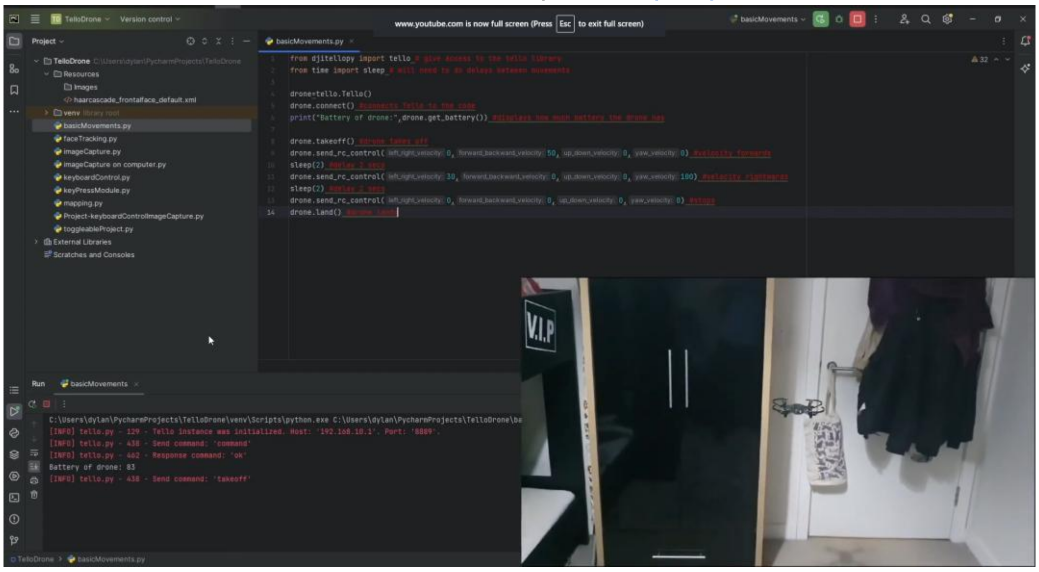
1) Connect to the drone,
2) Display its battery,
3) Takeoff,
4) Go forwards with a velocity of 50 cm/s,
5) Go rightwards with a velocity of 30 cm/s and rotate with a yaw velocity of 100 degrees/s simultaneously,
6) Stay stationary,

7) Land.

Below is the code with comments that explain the code line by line!

```
from djitellopy import tello # give access to the tello library
from time import sleep # will need to do delays between movements

drone=tello.Tello()
drone.connect() #connects Tello to the code
print("Battery of drone:",drone.get_battery()) #displays how much battery
the drone has

drone.takeoff() #drone takes off
drone.send_rc_control(0,50,0,0) #velocity forwards
sleep(2) #delay 2 secs
drone.send_rc_control(30,0,0,100) #velocity rightwards
sleep(2) #delay 2 secs
drone.send_rc_control(0,0,0,0) #stops
drone.land() #drone lands
```

And here is a video of it in action, taken by me: https://youtu.be/XJMxBtAYfbc



```
from djitellopy import tello # give access to the tello library
from time import sleep # will need to do delays between movements

drone=tello.Tello()
drone.connect() #connects Tello to the code
print("Battery of drone:",drone.get_battery()) #displays how much battery the drone has

drone.takeoff() #drone takes off
drone.send_rc_control( left_right_velocity: 0, forward_backward_velocity: 50, up_down_velocity: 0, yaw_velocity: 0) #velocity forwards
sleep(2) #delay 2 secs
drone.send_rc_control( left_right_velocity: 30, forward_backward_velocity: 0, up_down_velocity: 0, yaw_velocity: 100) #velocity rightwards
sleep(2) #delay 2 secs
drone.send_rc_control( left_right_velocity: 0, forward_backward_velocity: 0, up_down_velocity: 0, yaw_velocity: 0) #stops
drone.land() #drone lands
```

```
C:\Users\dylan\PycharmProjects\TelloDrone\venv\Scripts\python.exe C:\Users\dylan\PycharmProjects\TelloDrone\ba
[INFO] tello.py - 129 - Tello instance was initialized. Host: '192.168.10.1'. Port: '8889'.
[INFO] tello.py - 438 - Send command: 'command'
[INFO] tello.py - 462 - Response command: 'ok'
Battery of drone: 83
[INFO] tello.py - 438 - Send command: 'takeoff'
```

3.2 imageCapture.py *(31/07/2023, 19:10)*

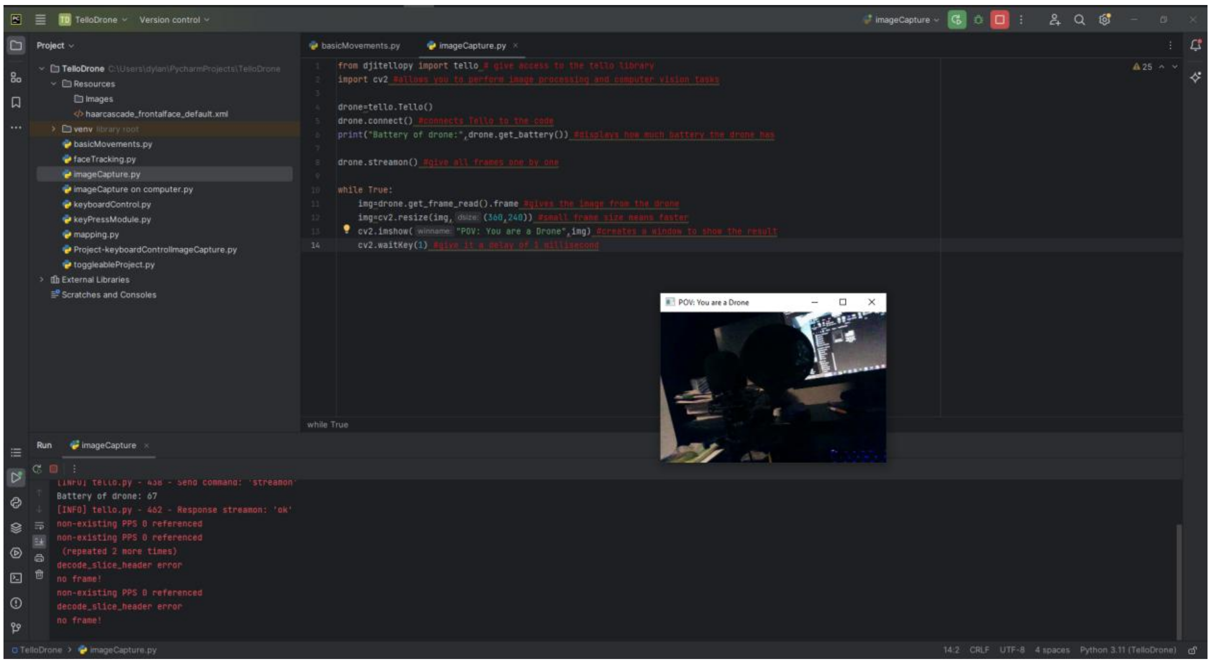
imageCapture.py was the second Python file I worked on, and as you can see by the time difference between this and the previous file, I was eager to begin work on this once again. This was mainly because the library was simple to understand so far and I wanted to experiment more with it. I realised that if I wanted to eventually code a Facial Recognition system, I would need some way to connect to the Tello Drone’s 720p camera. And so I did my own research and found how to connect and display the camera. imageCapture.py does the following:

1) Connects to the drone,
2) Displays the current battery of the drone,
3) Connects to the drone's camera,
4) Constantly gets frames from the drone's camera and saves it to a variable called img,
5) Resizes img to a smaller frame size,
6) Creates a window with img which shows the drone's Point of View.

Below is the code with comments that explain the code line by line!

```python
from djitellopy import tello # give access to the tello library
import cv2 #allows you to perform image processing and computer vision
tasks

drone=tello.Tello()
drone.connect()  #connects Tello to the code
print("Battery of drone:",drone.get_battery()) #displays how much battery
the drone has

drone.streamon() #give all frames one by one

while True:
    img=drone.get_frame_read().frame #gives the image from the drone
    img=cv2.resize(img,(360,240)) #small frame size means faster
    cv2.imshow("POV: You are a Drone",img) #creates a window to show the
result
    cv2.waitKey(1) #give it a delay of 1 millisecond
```

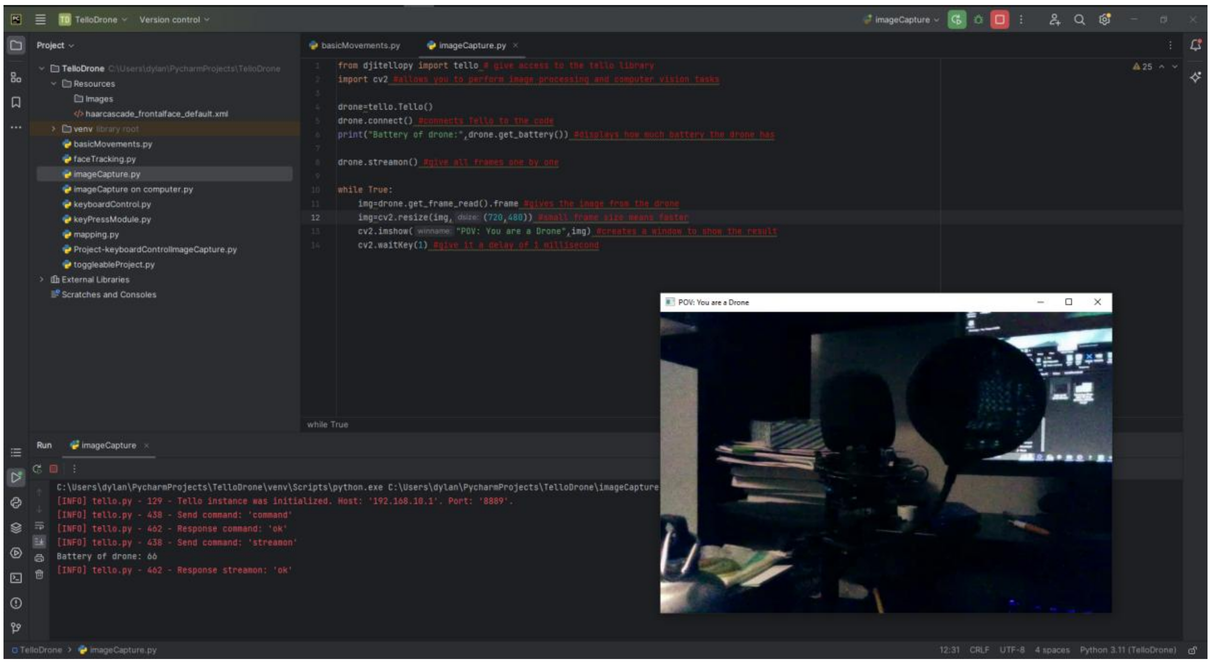
This is what the window looks like when ran:



```python
from djitellopy import tello # give access to the tello library
import cv2 #allows you to perform image processing and computer vision tasks

drone=tello.Tello()
drone.connect() #connects Tello to the code
print("Battery of drone:",drone.get_battery()) #displays how much battery the drone has

drone.streamon() #give all frames one by one

while True:
    img=drone.get_frame_read().frame #gives the image from the drone
    img=cv2.resize(img, dsize: (360,240)) #small frame size means faster
    cv2.imshow( winname: "POV: You are a Drone",img) #creates a window to show the result
    cv2.waitKey(1) #give it a delay of 1 millisecond
```

```
[INFO] tello.py - 438 - Send command: 'streamon'
Battery of drone: 67
[INFO] tello.py - 462 - Response streamon: 'ok'
non-existing PPS 0 referenced
non-existing PPS 0 referenced
 (repeated 2 more times)
decode_slice_header error
no frame!
non-existing PPS 0 referenced
decode_slice_header error
no frame!
```

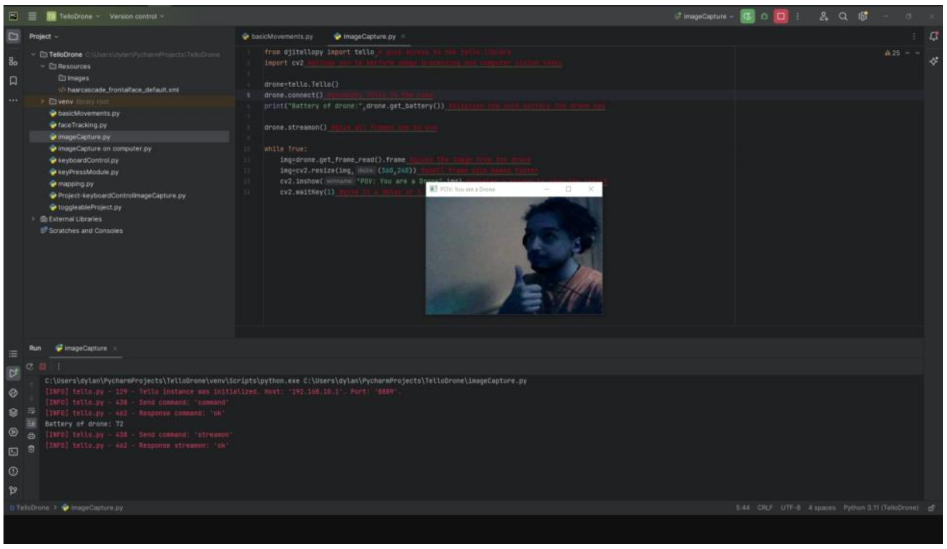
Resizing the frame size to double the size (720x480) makes it look like this, but at the cost of reduced performance:



```python
from djitellopy import tello # give access to the tello library
import cv2 #allows you to perform image processing and computer vision tasks

drone=tello.Tello()
drone.connect() #connects Tello to the code
print("Battery of drone:",drone.get_battery()) #displays how much battery the drone has

drone.streamon() #give all frames one by one

while True:
    img=drone.get_frame_read().frame #gives the image from the drone
    img=cv2.resize(img, dsize: (720,480)) #small frame size means faster
    cv2.imshow( winname: "POV: You are a Drone",img) #creates a window to show the result
    cv2.waitKey(1) #give it a delay of 1 millisecond
```

```
C:\Users\dylan\PycharmProjects\TelloDrone\venv\Scripts\python.exe C:\Users\dylan\PycharmProjects\TelloDrone\imageCapture
[INFO] tello.py - 129 - Tello instance was initialized. Host: '192.168.10.1'. Port: '8889'.
[INFO] tello.py - 438 - Send command: 'command'
[INFO] tello.py - 462 - Response command: 'ok'
[INFO] tello.py - 438 - Send command: 'streamon'
Battery of drone: 66
[INFO] tello.py - 462 - Response streamon: 'ok'
```

Along with a video of the code running: https://youtu.be/LSxXmzuh490



```python
from djitellopy import tello # give access to the tello library
import cv2 #allows you to perform image processing and computer vision tasks

drone=tello.Tello()
drone.connect() #connects Tello to the code
print("Battery of drone:",drone.get_battery()) #displays how much battery the drone has

drone.streamon() #give all frames one by one

while True:
    img=drone.get_frame_read().frame #gives the image from the drone
    img=cv2.resize(img, dsize: (360,240)) #small frame size means faster
    cv2.imshow( winname: "POV: You are a Drone",img)
    cv2.waitKey(1)
```

```
C:\Users\dylan\PycharmProjects\TelloDrone\venv\Scripts\python.exe C:\Users\dylan\PycharmProjects\TelloDrone\imageCapture.py
[INFO] tello.py - 129 - Tello instance was initialized. Host: '192.168.10.1'. Port: '8889'.
[INFO] tello.py - 438 - Send command: 'command'
[INFO] tello.py - 462 - Response command: 'ok'
Battery of drone: 72
[INFO] tello.py - 438 - Send command: 'streamon'
[INFO] tello.py - 462 - Response streamon: 'ok'
```

3.3 keyPressModule.py *(31/07/2023, 20:01)*

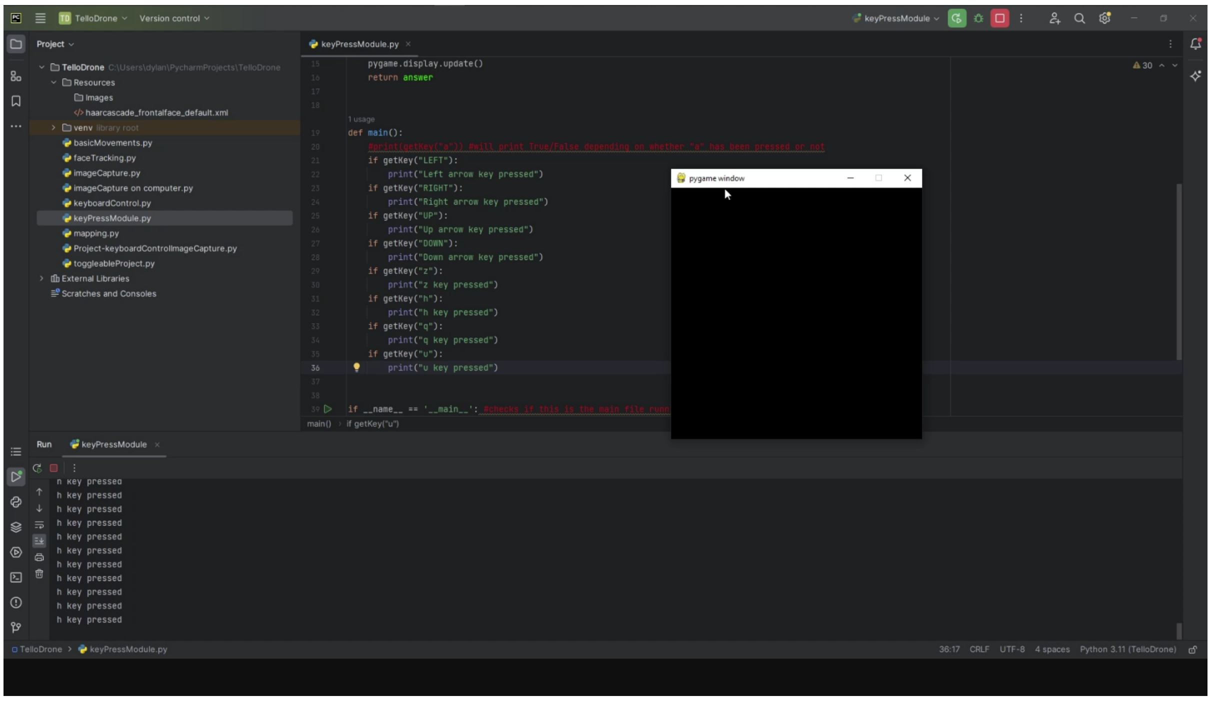
keyPressModule.py was the third Python file I developed in the sequence of this project. After having established basic drone movements and the ability to see what the drone sees, the next logical step was to enhance the interaction between the user and the drone as originally intended. The focus here was to implement a Keyboard Control system that could detect and respond to keyboard inputs, paving the way for more dynamic and interactive control of the drone. This module represents my first ever time delving into integrating user input into the drone's operation, setting the foundation for more complex control mechanisms (like the toggling system). The keyPressModule.py accomplishes the following:

1) Creates a pygame window where inputs will be detected,
2) Creates a getKey() function that returns whether a key has been pressed or not (either 'True' or 'False'),
3) Runs the main function that will always be active. Whenever any of the arrow keys are pressed, it continuously says "... arrow key pressed" until it has been released.

Below is the code with comments that explain the code:

```python
import pygame

def init():
   pygame.init()
   win=pygame.display.set_mode((400,400)) #set window size

def getKey(keyName): #tells whether a key has been pressed or not
   answer=False
   for i in pygame.event.get():
       pass #clears the pygame event queue without processing any events
   keyInput=pygame.key.get_pressed()
   myKey=getattr(pygame, 'K_{}'.format(keyName)) #puts keyName inside {}
   if keyInput[myKey]:
       answer=True
   pygame.display.update()
   return answer


def main():
   #print(getKey("a")) #will print True/False depending on whether "a" has
been pressed or not
   if getKey("LEFT"):
       print("Left arrow key pressed")
   if getKey("RIGHT"):
       print("Right arrow key pressed")
   if getKey("UP"):
       print("Up arrow key pressed")
   if getKey("DOWN"):
       print("Down arrow key pressed")


if __name__ == '__main__': #checks if this is the main file running
   init() #if so then it will run init function
   while True:
       main() #and then constantly runs main function
```

Below is also a video of it working in action, from launch to end. It’s also easily changeable, being able to do this for any key on the keyboard! This too is shown in the video: https://youtu.be/CDAhENPJxqk



```python
    pygame.display.update()
    return answer


def main():
    #print(getKey("a")) #will print True/False depending on whether "a" has been pressed or not
    if getKey("LEFT"):
        print("Left arrow key pressed")
    if getKey("RIGHT"):
        print("Right arrow key pressed")
    if getKey("UP"):
        print("Up arrow key pressed")
    if getKey("DOWN"):
        print("Down arrow key pressed")
    if getKey("z"):
        print("z key pressed")
    if getKey("h"):
        print("h key pressed")
    if getKey("q"):
        print("q key pressed")
    if getKey("u"):
        print("u key pressed")


if __name__ == '__main__': #checks if this is the main file runn
```

```
h key pressed
h key pressed
h key pressed
h key pressed
h key pressed
h key pressed
h key pressed
h key pressed
h key pressed
h key pressed
h key pressed
```

You may think that the fact that it says it so many times for a single tap of the key is an error. This is actually intentional. Whilst you may think a key has only two states (PRESSED and NOT PRESSED), this will be used for the Tello drone. Because of this, the Tello drone along with all drones will not go from 0 to 100 cm/s with the press of the forward button, but instead needs to build and lose momentum when it is pressed and not pressed respectively. This is why the constant detections of the up arrow being pressed is necessary.

3.4 keyboardControl.py *(31/07/2023, 20:36)*

keyboardControl.py marked a significant milestone in the development of my project, being the fourth Python file and the most ambitious project I had tackled thus far. This script was where the capabilities of keyPressModule.py were fully utilised, merging the user input functionality with the actual control of the drone. The excitement of bringing together the user interface with drone command execution was a key moment for me personally in the project, being able to showcase the synergy between different components. The implementation of keyboardControl.py was crucial, as it allowed for manual control of the drone, which is already half of the functionality. What this script did was the following:

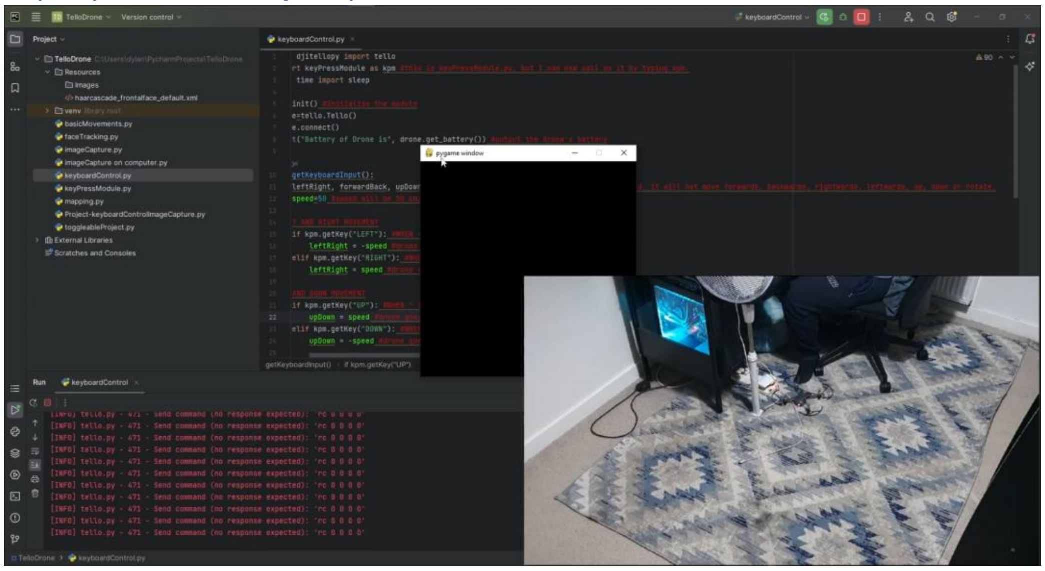
1) Imports the keyPressModule as kpm,
2) Connects to the drone,
3) Outputs its battery,
4) Creates a function which is designed to figure out which key is being pressed, and corresponds it to the appropriate movement option.

Below is the full code for it, with comments made by me to make it easier to understand:

```
from djitellopy import tello
import keyPressModule as kpm #this is keyPressModule.py, but I can now call
on it by typing kpm.
from time import sleep

kpm.init() #initialise the module
drone=tello.Tello()
drone.connect()
print("Battery of Drone is", drone.get_battery()) #output the drone's
battery

def getKeyboardInput():
   leftRight, forwardBack, upDown, yawVelocity = 0,0,0,0 #if no key is
being pressed, it will not move forwards, backwards, rightwards, leftwards,
up, down or rotate.
   speed=50 #speed will be 50 cm/s (but also 50 degrees/s)

#LEFT AND RIGHT MOVEMENT
   if kpm.getKey("LEFT"): #WHEN <-- IS PRESSED
       leftRight = -speed #drone strafes leftwards
   elif kpm.getKey("RIGHT"): #WHEN --> IS PRESSED
       leftRight = speed #drone strafes rightwards

#UP AND DOWN MOVEMENT
   if kpm.getKey("UP"): #WHEN ^ IS PRESSED
       upDown = speed #drone goes up
   elif kpm.getKey("DOWN"): #WHEN v IS PRESSED
       upDown = -speed #drone goes down

#FORWARD AND BACKWARD MOVEMENT
   if kpm.getKey("w"): #WHEN W IS PRESSED
       forwardBack = speed #drone goes forwards
   elif kpm.getKey("s"): #WHEN S IS PRESSED
       forwardBack = -speed #drone goes backwards

#ROTATION LEFT AND ROTATION RIGHT MOVEMENT
   if kpm.getKey("a"): #WHEN A IS PRESSED
       yawVelocity = -speed #drone rotates left
   elif kpm.getKey("d"): #WHEN D IS PRESSED
       yawVelocity = speed #drone rotates right

#LANDING AND TAKEOFF
   if kpm.getKey("q"): #WHEN Q IS PRESSED
       drone.land() #drone lands
   if kpm.getKey("e"): #WHEN E IS PRESSED
       drone.takeoff() #drone takes off

   return [leftRight, forwardBack, upDown, yawVelocity] #gives these values
to line 47


while True:
   values= getKeyboardInput() #gets the values from line 44
```

```
    drone.send_rc_control(values[0],values[1],values[2],values[3]) #sends
values for leftRight, forwardBack, upDown, yawVelocity to the drone's
movement controls
    sleep(0.05) #waits 0.05 milliseconds for next input (near instant)
```

Linked is a video of it working in action, along with my commentary:
https://youtu.be/QAi9gKMqcTc



```
 djitellopy import tello
rt keyPressModule as kpm #this is keyPressModule.py, but I can now call on it by typing kpm.
 time import sleep

init() #initialise the module
e=tello.Tello()
e.connect()
t("Battery of Drone is", drone.get_battery()) #output the drone's battery

getKeyboardInput():
leftRight, forwardBack, upDown
speed=50 #speed will be 50 cm/

T AND RIGHT MOVEMENT
if kpm.getKey("LEFT"):
    leftRight = -speed
elif kpm.getKey("RIGHT"):
    leftRight = speed

AND DOWN MOVEMENT
if kpm.getKey("UP"):
    upDown = speed
elif kpm.getKey("DOWN"):
    upDown = -speed
```

```
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
[INFO] tello.py - 471 - Send command (no response expected): 'rc 0 0 0 0'
```

3.5 Project-keyboardControlImageCapture.py (*01/08/2023, 19:09*)

Project-keyboardControlImageCapture.py was the next step in the evolution of my project, representing the first major combination in the development sequence. Building on the foundations laid by keyboardControl.py, and imageCapture.py, this script brought an additional layer of interaction to the drone's control system by integrating real-time image capture capabilities. Combining manual control with visual feedback was a motivating moment for me, pushing me onwards to conquer more impressive feats such as this one. It significantly enhanced the drone's usability and user experience. The key moments of Project-keyboardControlImageCapture.py are as follows:

1) Imports keyPressModule as kpm,
2) Connects to the drone,
3) Outputs the battery of the drone,
4) Activates the drone's camera
5) Uses getKeyboardInput, a function used to figure out which key is being pressed, and corresponds it to the appropriate movement option,
6) Also allows the user to take pictures of the drone's point of view by pressing the "z" key.

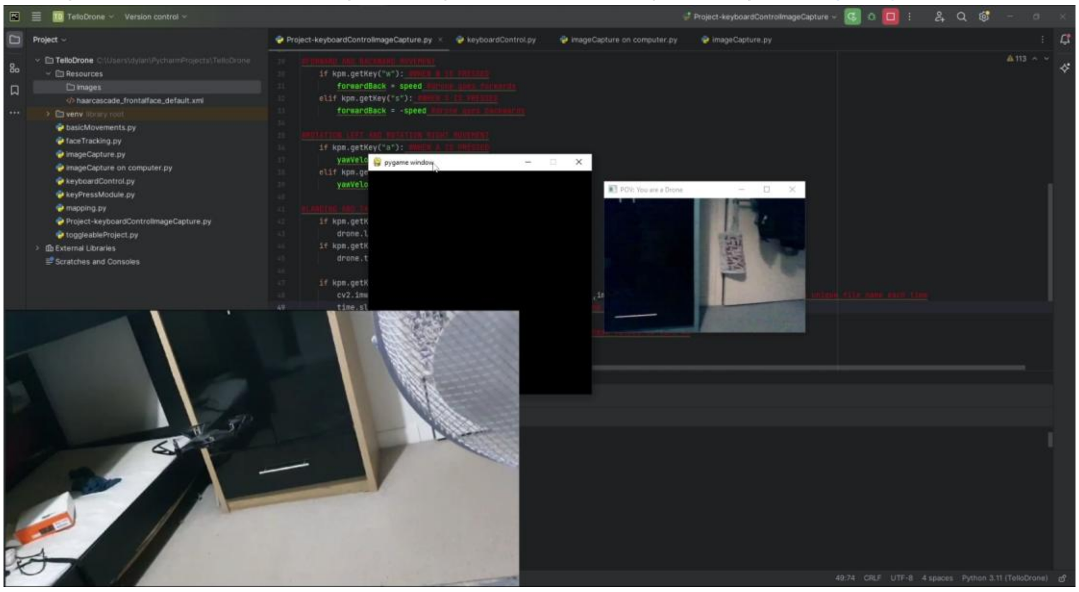
Below is a breakdown of how the code works!

```
import time
from djitellopy import tello
```

```
import keyPressModule as kpm #this is keyPressModule.py, but I can now call
on it by typing kpm.
import cv2

kpm.init()
drone=tello.Tello()
drone.connect() #connects to tello drone
print("Battery of Drone is", drone.get_battery()) #displays drone battery
global img #makes sure we can use the img on line 33
drone.streamon() #give all frames one by one

def getKeyboardInput():
   leftRight, forwardBack, upDown, yawVelocity = 0,0,0,0 #if no key is
being pressed, it will not move forwards, backwards, rightwards, leftwards,
up, down or rotate.
   speed=50 #speed will be 50 cm/s (but also 50 degrees/s)

#LEFT AND RIGHT MOVEMENT
   if kpm.getKey("LEFT"): #WHEN <-- IS PRESSED
       leftRight = -speed #drone strafes leftwards
   elif kpm.getKey("RIGHT"): #WHEN --> IS PRESSED
       leftRight = speed #drone strafes rightwards

#UP AND DOWN MOVEMENT
   if kpm.getKey("UP"): #WHEN ^ IS PRESSED
       upDown = speed #drone goes up
   elif kpm.getKey("DOWN"): #WHEN v IS PRESSED
       upDown = -speed #drone goes down

#FORWARD AND BACKWARD MOVEMENT
   if kpm.getKey("w"): #WHEN W IS PRESSED
       forwardBack = speed #drone goes forwards
   elif kpm.getKey("s"): #WHEN S IS PRESSED
       forwardBack = -speed #drone goes backwards

#ROTATION LEFT AND ROTATION RIGHT MOVEMENT
   if kpm.getKey("a"): #WHEN A IS PRESSED
       yawVelocity = -speed #drone rotates left
   elif kpm.getKey("d"): #WHEN D IS PRESSED
       yawVelocity = speed #drone rotates right

#LANDING AND TAKEOFF
   if kpm.getKey("q"): #WHEN Q IS PRESSED
       drone.land() #drone lands
   if kpm.getKey("e"): #WHEN E IS PRESSED
       drone.takeoff() #drone takes off

   if kpm.getKey("z"): #taking a picture! (image capture)
       cv2.imwrite(f'Resources/Images/{time.time()}.jpg',img) #stores the
frame that the drone sees as a unique file name each time
       time.sleep(0.3) #so i dont get a billion images of the same frame
```

```
    return [leftRight, forwardBack, upDown, yawVelocity] #gives these values to line 43


while True:
    values= getKeyboardInput() #gets these values from line 39
    drone.send_rc_control(values[0],values[1],values[2],values[3]) #sends values for leftRight, forwardBack, upDown, yawVelocity to the drone's movement controls

    img=drone.get_frame_read().frame #gives the image from the drone
    img=cv2.resize(img,(360,240)) #small frame size means faster
    cv2.imshow("POV: You are a Drone",img) #creates a window to show the result
    cv2.waitKey(1) #give it a delay of 1 millisecond
```

Along with a video showing exactly how it works by example: https://youtu.be/Dx9f2y-YNXg


TelloDrone
Version control
Project-keyboardControlImageCapture
Project
Project-keyboardControlImageCapture.py
keyboardControl.py
imageCapture on computer.py
imageCapture.py
TelloDrone C:\Users\dylan\PycharmProjects\TelloDrone
Resources
images
haarcascade_frontalface_default.xml
venv library root
basicMovements.py
faceTracking.py
imageCapture.py
imageCapture on computer.py
keyboardControl.py
keyPressModule.py
mapping.py
Project-keyboardControlImageCapture.py
toggleableProject.py
External Libraries
Scratches and Consoles
pygame window
POV: You are a Drone
49:74 CRLF UTF-8 4 spaces Python 3.11 (TelloDrone)

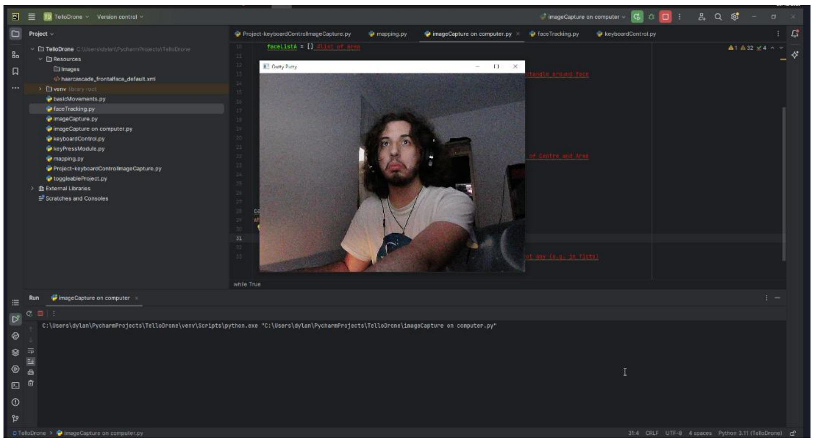
```
    if kpm.getKey("w"):
        forwardBack = speed
    elif kpm.getKey("s"):
        forwardBack = -speed

    if kpm.getKey("a"):
        yawVelo
    elif kpm.ge
        yawVelo

    if kpm.getK
        drone.l
    if kpm.getK
        drone.t

    if kpm.getK
        cv2.imw
        time.sl
```

And here are the three pictures that the drone took!

3.6 faceTracking.py *(05/08/2023, 02:19)*

"faceTracking.py stands as the final and one of the most sophisticated Python scripts I have ever developed, let alone for the DroneVision project. This script was pivotal in realising the project's ambition of integrating autonomous features, specifically through real-time facial recognition and tracking using OpenCV. The development of faceTracking.py was a blend of complex image processing techniques and dynamic drone control logic. The key features of this script are too much to summarise, so I will show a step-by-step process, with the final code at the end!

First, I had to test it on my Desktop PC's webcam. This is because the drone's battery life is relatively short, and it would die even whilst I was just tweaking accuracy settings. I used `cap = cv2.VideoCapture(0)` to capture the footage from my webcam and have it display. Here's what it looked like:
https://youtu.be/_ca9AHwSOY0 (See the performance here)

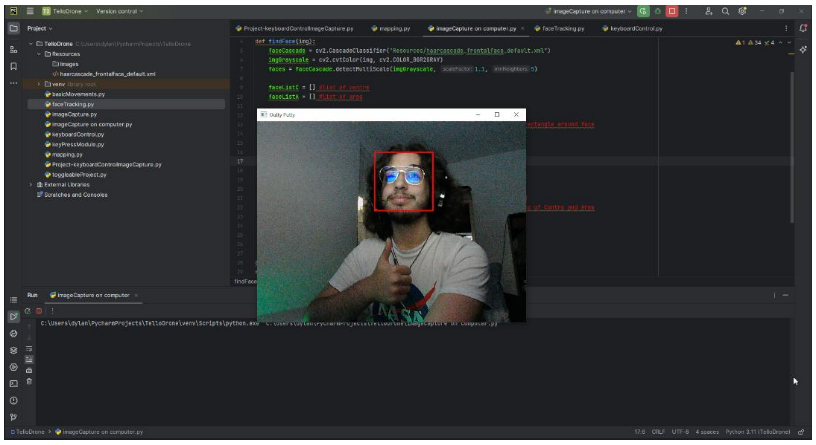


```
faceListA = [] #list of area
```

```
C:\Users\dylan\PycharmProjects\TelloDrone\venv\Scripts\python.exe "C:\Users\dylan\PycharmProjects\TelloDrone\imageCapture on computer.py"
```

I was watching my webcam's footage, which was great. But that wasn't all. Next, I had to draw a rectangle around my face so that the drone can use the area of that rectangle to decide whether to get closer or further away. I did this and also tested it using glasses to see if the AI would still recognise their faces even with the glasses obstruction. I was happy to see that people wearing glasses were still considered people! With the addition of the rectangle though, performance slightly tanked. This is understandable, since the rectangle is constantly updating with every tick. It should not affect the drone to such a degree that it does not do its main functions.

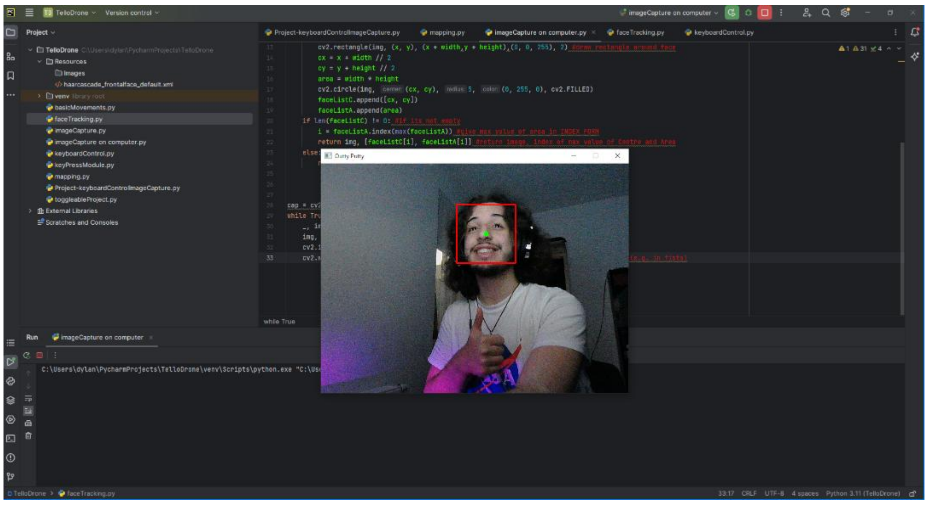
Watch it here: https://youtu.be/kY8SkDz-ZKY



```
def findFace(img):
    faceCascade = cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")
    imgGrayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(imgGrayscale, scaleFactor: 1.1, minNeighbors: 5)

    faceListC = [] #list of centre
    faceListA = [] #list of area
```

Next, I had to put the centre point in. This and the area of the rectangle actually determine the drone's movement. The area has been explained above, but the green circle in the centre of the webcam output is crucial for telling the drone when to rotate. It should rotate when the Centre point is in the middle of the screen. How will we know when this is? That is the next section.

But for now, here is a video of me moving around to showcase the great Facial Tracking Artificial Intelligence: https://youtu.be/c_vmVwpVu6g



```
        cv2.rectangle(img, (x, y), (x + width,y + height),(0, 0, 255), 2) #draw rectangle around face
        cx = x + width // 2
        cy = y + height // 2
        area = width * height
        cv2.circle(img, center (cx, cy), radius 5, color (0, 255, 0), cv2.FILLED)
        faceListC.append([cx, cy])
        faceListA.append(area)
    if len(faceListC) != 0: #if its not empty
        i = faceListA.index(max(faceListA)) #give max value of area in INDEX FORM
        return img, [faceListC[i], faceListA[i]] #return image, index of max value of Centre and Area
    else:

cap = cv2
while Tru
    _, im
    img,
    cv2.i
    cv2.
```

```
C:\Users\dylan\PycharmProjects\TelloDrone\venv\Scripts\python.exe "C:\Us
```

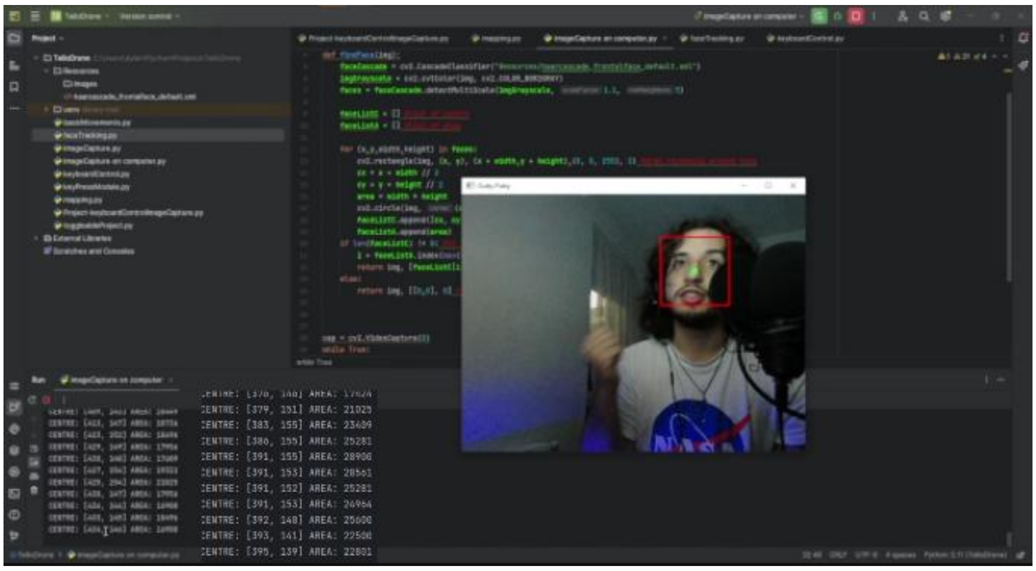
I created this function to help me not only draw the red rectangle and green centre point, but to help me make all of this code possible. I call it findFace().

```
def findFace(img):
   faceCascade =
cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")
   imgGrayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   faces = faceCascade.detectMultiScale(imgGrayscale, 1.1, 5)

   faceListC = [] #list of centre
   faceListA = [] #list of area

   for (x,y,width,height) in faces:
       cv2.rectangle(img, (x, y), (x + width,y + height),(0, 0, 255), 2)
#draw rectangle around face
       cx = x + width // 2
       cy = y + height // 2
       area = width * height
       cv2.circle(img, (cx, cy), 5, (0, 255, 0), cv2.FILLED)
       faceListC.append([cx, cy])
       faceListA.append(area)
   if len(faceListC) != 0: #if its not empty
       i = faceListA.index(max(faceListA)) #give max value of area in INDEX
FORM
       return img, [faceListC[i], faceListA[i]] #return image, index of max
value of Centre and Area
   else:
       return img, [[0,0], 0] #if its empty return nothing
```

The third line uses something called `haarcascade_frontalface_default.xml`. This is a file that has all the parameters and information of a model that helps to detect different objects. It is available for free on the official OpenCV website, so this is fair use. In this case, it helps to detect faces.

If the rest of the function is too complicated to understand, I have recorded a video explaining what it does.

Here is the video!: https://youtu.be/ocfc6hOOFpo

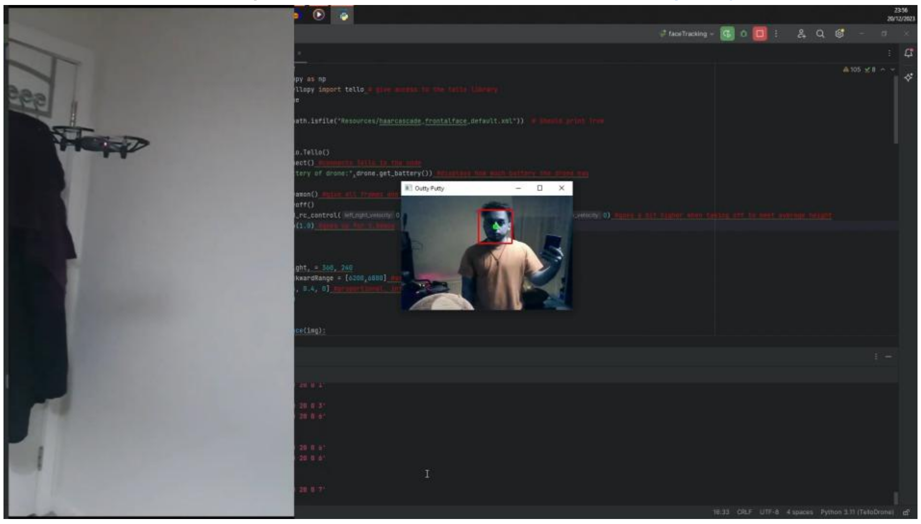


```
CENTRE: [379, 151] AREA: 21025
CENTRE: [383, 155] AREA: 23409
CENTRE: [386, 155] AREA: 25281
CENTRE: [391, 155] AREA: 28900
CENTRE: [391, 153] AREA: 28561
CENTRE: [391, 152] AREA: 25281
CENTRE: [391, 153] AREA: 24964
CENTRE: [392, 148] AREA: 25600
CENTRE: [393, 141] AREA: 22500
CENTRE: [395, 139] AREA: 22801
```

(This also carries over the exact same way, but with the drone's camera. Here is the full code):

```
import cv2
import numpy as np
from djitellopy import tello # give access to the tello library
import time
import os
print(os.path.isfile("Resources/haarcascade_frontalface_default.xml"))  #
Should print True


drone=tello.Tello()
drone.connect() #connects Tello to the code
print("Battery of drone:",drone.get_battery()) #displays how much battery
the drone has

drone.streamon() #give all frames one by one
drone.takeoff()
drone.send_rc_control(0,0,25,0) #goes a bit higher when taking off to meet
average height
time.sleep(1.0) #goes up for 1.0secs



width, height, = 360, 240
forwardBackwardRange = [6200,6800] #sweet spot range
pid = [0.4, 0.4, 0] #proportional, integral, derivative
pError = 0

def findFace(img):
    faceCascade =
cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")
    imgGrayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(imgGrayscale, 1.1, 5)

    faceListC = [] #list of centre
    faceListA = [] #list of area

    for (x,y,width,height) in faces:
        cv2.rectangle(img, (x, y), (x + width,y + height),(0, 0, 255), 2)
#draw rectangle around face
        cx = x + width // 2
        cy = y + height // 2
        area = width * height
        cv2.circle(img, (cx, cy), 5, (0, 255, 0), cv2.FILLED)
        faceListC.append([cx, cy])
        faceListA.append(area)
    if len(faceListC) != 0: #if its not empty
        i = faceListA.index(max(faceListA)) #give max value of area in INDEX
FORM
        return img, [faceListC[i], faceListA[i]] #return image, index of max
value of Centre and Area
    else:
        return img, [[0,0], 0] #if its empty return nothing



def trackFace(info, width, pid, pError):
    area = info[1]
    x,y = info[0]
    forwardBackward=0

    error = x - width//2 #x is the face, width//2 is the centre of the
image. this finds how far it is (the deviation)
    speed = pid[0]*error + pid[1]*(error-pError)
    speed = int(np.clip(speed,-100,100)) #make sure it does not go below -
100 and above 100

    if area > forwardBackwardRange[0] and area < forwardBackwardRange[1]:
#the sweet spot where it stays still
        forwardBackward = 0 #don't move forward OR back
        print("staying still")
    if area > forwardBackwardRange[1]:
        forwardBackward = -20 #go back
        print("going back")
    elif area < forwardBackwardRange[0] and area != 0: #make sure if a face
is NOT detected it doesnt just go forward
        forwardBackward = 20 #go forward
        print("going forward")




    if x == 0: #if nothing is detected then dont move speed wise
        speed = 0
        error = 0

    #print(speed, forwardBackward)
    drone.send_rc_control(0,forwardBackward,0,speed) #sends forward/Backward
and Yaw velocity
    return error

#cap = cv2.VideoCapture(0)
while True:
    #_, img = cap.read()
    img=drone.get_frame_read().frame #gives the image from the drone
    img=cv2.resize(img,(width,height)) #resizes to w and h in the top
    img, info = findFace(img)
    pError = trackFace(info, width, pid, pError)
    #print("Center", info[0],"Area", info[1]) #center used to rotate the
drone, area used to move forwards/backwards
    cv2.imshow("Outty Putty", img)
    if cv2.waitKey(1) & 0xFF == ord('q'): #if q key is pressed then land
drone
        drone.land()
        break
```

Using this code, I now had my Facial Tracking mode of control fully functioning! The video where I test and explain lines of the code is here:https://youtu.be/_KWGzQo3YUs



```
py as np
llopy import tello
e
ath.isfile("Resources/haarcascade_frontalface_default.xml"))
o.Tello()
ect()
tery of drone:",drone.get_battery())
amon()
off()
_rc_control(
(1.0)
ght, = 360, 240
kwardRange = [6200,6800]
, 0.4, 0]
ce(img):
```

3.7 User Documentation

I made sure that my User Documentation was as clear and as concise as possible. It reads as follows:

User Documentation for DroneVision: Drone Control System

Introduction

Welcome to the DroneVision User Guide. This document provides instructions on how to operate your drone using our custom-designed control interface. Ensure you read all the safety instructions before flight to ensure a wonderful user experience!

Getting Started

Pre-Flight Checklist

- Ensure the drone's battery is fully charged.
- Check for any physical damage to the drone (like dust or hairs stuck in propellers).
- Ensure the drone's firmware is up to date.
- Confirm that the drone's camera is clear and unobstructed.
- Establish a clear flight area free of obstacles.

Connecting to the Drone

1. Turn on your drone.
2. Launch the DroneVision control software on your computer.
3. Wait for the drone to connect to your device.
4. Once connected, the DroneVision software will connect to your drone and display its battery percentage.

Basic Controls

- Take-off: Press the ‘E’ key to initiate the take-off sequence. The drone will lift off the ground and hover, waiting for your input.
- Landing: Press the ‘Q’ key to land the drone at its current location.
- Movement: W/S key = Forwards/Backwards --- A/D key = Rotate Left/Rotate Right --- UP ARROW/DOWN ARROW key = Up/Down --- LEFT ARROW/RIGHT ARROW = Move left/ Move right

Image Capture

- To capture an image from the drone’s camera, press the ‘Z’ key.

- Captured images are saved in the 'Images' folder with a unique timestamp as the file name to mitigate duplicates.

Facial Recognition Tracking

- Once in Facial Tracking mode, the drone will automatically fly up to detect a face. Ensure that you stand in front of it so that you are seen by the camera.
- The drone will adjust its position to keep a detected face at the centre of the frame.
- If no face is detected, the drone will hover in place.

Emergency Procedures

- To perform an emergency landing, tap the 'Q'key.

Post-Flight

- Turn off the drone and disconnect it from the control software.
- Recharge the drone's battery.
- Store the drone in a safe and secure place.

Troubleshooting

- If the drone does not respond to commands, check the connection status in the software.
- For issues with image capture, ensure sufficient lighting and clear camera lens.

Testing

4.1 Testing Data

Test Case	Type of Test	Expected Outcome	Actual Outcome	Pass/Fail
'W' key is pressed.	Normal	The drone should move forwards in a straight horizontal line with a constant	The drone does not move forwards at all due to a minor error with detecting a	Fail (Fixed in **FIX 1**)

		velocity until the ‘W’ key is released.	capital “W” instead of a lowercase “w”..	
‘A’ key is pressed.	Normal	The drone should rotate left with a constant yaw velocity until the ‘A’ key is released.	The drone does not rotate because the yaw control command was incorrectly mapped in the code.	Fail (Fixed in **FIX 2**)
‘S’ key is pressed.	Normal	The drone should move backwards in a straight horizontal line with a constant velocity until the ‘S’ key is released.	The drone should move backwards in a straight horizontal line with a constant velocity until the ‘S’ key is released.	Pass
‘D’ key is pressed.	Normal	The drone should rotate right with a constant yaw velocity until the ‘D’ key is released.	The drone should rotate right with a constant yaw velocity until the ‘D’ key is released.	Pass
‘UP ARROW’ key is pressed.	Normal	The drone should move upwards in a straight vertical line with a constant velocity until the ‘UP ARROW’ key is released.	The drone should move upwards in a straight vertical line with a constant velocity until the ‘UP ARROW’ key is released.	Pass
‘DOWN ARROW’ key is pressed.	Normal	The drone should move downwards in a straight vertical line with a constant velocity until the ‘DOWN ARROW’ key is released.	The drone should move downwards in a straight vertical line with a constant velocity until the ‘DOWN ARROW’ key is released.	Pass
LEFT ARROW’ key is pressed.	Normal	The drone should move leftwards in a	The drone should move leftwards in a	Pass

		straight horizontal line with a constant velocity until the 'LEFT ARROW' key is released.	straight horizontal line with a constant velocity until the 'LEFT ARROW' key is released.	
'RIGHT ARROW' key is pressed.	Normal	The drone should move rightwards in a straight horizontal line with a constant velocity until the 'RIGHT ARROW' key is released.	The drone should move rightwards in a straight horizontal line with a constant velocity until the 'RIGHT ARROW' key is released.	Pass
'E' key is pressed.	Normal	Takeoff sequence - The drone's propellers should slowly start spinning, lifting the drone off of the ground.	Takeoff sequence - The drone's propellers should slowly start spinning, lifting the drone off of the ground.	Pass
'Q' key is pressed.	Normal	Emergency landing - The drone slowly descends and the propellers power off.	Emergency landing - The drone slowly descends and the propellers power off.	Pass

<u>Facial Recognition and Tracking Test Table</u>

Test Case	**Type of Test**	**Expected Outcome**	**Actual Outcome**	**Pass/Fail**
No face is in the camera's vision / Out of frame.	Normal	The drone should stay still until it detects a face in its vision.	The drone drifts slowly due to an unhandled case where no input is received from the face detection algorithm.	Fail (Fixed in **FIX 3**)
A face is in the centre of the drone's camera	Normal	The drone should either move forwards	The drone should either move forwards	Pass

footage.		or backwards, depending how close the person is to the drone	or backwards, depending how close the person is to the drone	
A face is to the *very* left of the drone’s camera footage.	Boundary	The drone should quickly rotate to the left, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person’s face in the centre.	The drone should quickly rotate to the left, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person’s face in the centre.	Pass
A face is *a little* to the left of the drone’s camera footage	Normal	The drone should slightly rotate to the left, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person’s face in the centre.	The drone should slightly rotate to the left, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person’s face in the centre.	Pass
A face is to the *very* right of the drone’s camera footage.	Boundary	The drone should quickly rotate to the right, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person’s face in the centre.	The drone should quickly rotate to the right, lowering the yaw velocity as the centre of the face gets to the centre of the screen. It should end with the person’s face in the centre.	Pass
A face is *a little* to the right of the drone’s camera footage.	Normal	The drone should slightly rotate to the left, lowering the yaw velocity as the centre of the	The drone should slightly rotate to the left, lowering the yaw velocity as the centre of the	Pass

		face gets to the centre of the screen. It should end with the person’s face in the centre.	face gets to the centre of the screen. It should end with the person’s face in the centre.	
The face detected is too close to the drone.	Erroneous	The drone should move backwards until the area of the face is within the specified range.	The drone should move backwards until the area of the face is within the specified range.	Pass
The face detected is too far from the drone.	Erroneous	The drone should move forwards until the area of the face is within the specified range.	The drone should move forwards until the area of the face is within the specified range.	Pass
The target detected moves backwards at a constant speed, whilst the drone can see their face.	Normal	The drone should move forwards to follow the target as the face area is decreasing and it should want to aim to get the area within the sweet spot	The drone should move forwards to follow the target as the face area is decreasing and it should want to aim to get the area within the sweet spot	Pass
The target detected moves forwards at a constant speed, whilst the drone can see their face.	Normal	The drone should move backwards to gain distance from the target as the face area is increasing and it should want to aim to get the area within the sweet spot	The drone should move backwards to gain distance from the target as the face area is increasing and it should want to aim to get the area within the sweet spot	Pass
The target’s face is in the ‘sweet spot’.	Normal	The drone should not move forwards or backwards,	The drone oscillates forward and backward	Fail (Fixed in **FIX 4**)

		and should continue tracking the target and rotating if needed.	slightly due to too small of a sweet spot range.	
‘Q’ key is pressed.	Normal	Emergency landing - The drone slowly descends and the propellers power off.	Emergency landing - The drone slowly descends and the propellers power off.	Pass

General Test Table

Test Case	**Type of Test**	**Expected Outcome**	**Actual Outcome**	**Pass/Fail**
‘T’ key is pressed.	Normal	The system should toggle in between two modes of control - Facial Tracking and Keyboard Control.	The ‘T’ key does not alternate between facialTracking and keyboardControl. Both codes work separately, but cannot be toggled between each other.	Fail (Could not fix, may have some logical errors in my code).
faceTracking.py is ran.	Normal	The drone should automatically elevate to an average height for a human to start detecting faces.	The drone should automatically elevate to an average height for a human to start detecting faces.	Pass

FIXES:

1. Replaced `if kpm.getKey("W"):` with `if kpm.getKey("w"):`
2. Replaced `yawVelocity = 0` with `yawVelocity = -speed`
3. The Areas and Centre coordinates were not added to the faceListA and faceListC variables. Added the following:
   ```
   faceListC.append([cx, cy])
   faceListA.append(area)
   ```
4. Increased the range for a sweet spot so that the area has to be in between 6200 and 6800 (`forwardBackwardRange = [6200,6800]`)

Evaluation

5.1 Quantitative Testing

In the quantitative analysis of my project, I focused on the following key performance indicators:

1. Response Time: The latency between command input and drone response was consistently measured between 0.1 and 0.2 seconds, well within the acceptable range of 0.5 seconds for real-time control systems. This rapid response time ensured smooth and predictable handling of the drone during navigation and operations
2. Facial Recognition Accuracy: Over multiple testing sessions, the facial recognition system achieved a high accuracy rate of around 85%, surprisingly surpassing the initial target of 80%. This high accuracy level was crucial in ensuring reliable tracking and interaction, although it is noted that darker lighting conditions did impact performance.
3. Battery Efficiency: When the drone is constantly being used, even idly, its battery life averaged 15 minutes, aligning closely with the manufacturer's specification of 13 minutes online.

5.2 Qualitative Testing

The qualitative assessment of my project involved gathering subjective feedback from users to gauge the system's usability and overall experience. I got family members around my house to test it for quality, and the results obtained are as follows:

1. User Experience: The testers reported a seamless experience with intuitive controls for both Keyboard Control and Facial Tracking, with positive visual feedback from the drone's camera feed. The ease of use mentioned amongst all candidates, especially when regarding the straightforward keyboard control scheme, thanks to the User Documentation.
2. Engagement: The interactive nature of the Facial Tracking and Keyboard Control options greatly affected user engagement in a positive way. Users expressed high satisfaction with the drone's ability to maintain focus on a subject, which added an immersive dimension to the control experience. They also enjoyed flying the drone around the house from one room and being able to see it all unfold.

5.3 Incomplete Systems

Unfortunately, I did not get to add everything I wanted to in my final code. Some of it was accredited to logical errors too deep into the code, lack of understanding or time constraints. Unfortunately, I did not get to implement a Graphical User Interface into my project. I was saving this for last, once I had the “toggle” feature between Keyboard Controls and Face Tracking in one Python file. When trying desperately to implement the Toggle feature, I noticed that I was spending far too much time on that alone, even more than some pieces of code. It was a big learning curve and ended up being a burden instead of what I envisioned. Regardless, I am happy with the fact that both modes of control at least work in their separate files. I also was not able to design it for laptops and tablets like previously mentioned due to the lack of a formal GUI. I am disappointed that I did not get to implement these features I strived for, but am overall happy with how it all turned out.

5.5 Maintenance

In order to make sure that competitors do not surpass DroneVision, and to make sure that it will still work even with new drone firmware updates, I will employ code review to ensure that all scripts are running at an optimal level, checking for updates in libraries such as ‘djitellopy’. I will also conduct physical inspections of the drone to check for damage or wear to propellers, motors or the camera. They are essentials for my system, so it is essential that they are always at optimal quality.
I could enhance my system by optimising my algorithms for greater accuracy or efficiency, or even by implementing a Graphical User Interface or any of the incomplete systems.

5.6 Further Development

Overall, this project has been really fun and enjoyable for me. I pushed myself into new areas that I did not even attempt to touch before, despite having an interest in it for a while and I am overall very happy with what I managed to achieve in such a short amount of time. I made a fully functioning Facial Recognition system, complete with a drone that follows a target based off of the area of their face, even repositioning itself if the target is not in a centre line. Not only that, but in such a short amount of time from beginning the project I managed to learn a new library and was able to make my own Key Press Module which led to me being able to fully control my drone via only Keyboard inputs, being able to see what the drone sees from the comfort of my own Desktop PC. I have to admit, that was most likely what I spent the most time on. Flying around my house from the comfort of my own room, going into my parents’ and siblings’ rooms catching them by surprise brought so much joy to this project, and reignited the passion I felt for programming all those years ago, watching everything unfold from behind a screen.

If I were to continue this project further, I would add numerous elements. I would try to make a GUI (Graphical User Interface) for my drone, and fix the toggle feature which caused me the most stress. I would add different buttons on the keyboard to correlate to different movements. For example, “f” could do a frontflip. “8” could do a Figure-8 motion. “r” could do a full 360 degree rotation, and I have many more ideas. I would also like to expand it from Facial Tracking and Keyboard Control to Facial Tracking, Keyboard Control and Hand Gesture Detection. This looks very interesting and like a fun feature to add, but I would need to dedicate the majority of my time solely learning the mechanics of that alone. Making it be able to follow lines from afar would be very ambitious too and something that I would be interested in!

I think in the future I would like to continue working on this as a passion project, without the constraint of deadlines. It has truly been eye opening to work on this throughout the year, and am excited to take it further!