Drone Controller with Computer Vision Capabilities for the Tello EDU Mini Drone

Jacob E. Pitsenberger

Division of Science, Mathematics and Technology, Governors State University

CPSC-4900: Senior Project and Seminar

Professor Freddie Kato

May 5, 2023

Table of Contents

[Abstract 5](#_Toc134272217)

[System Documentation 7](#_Toc134272218)

[System Overview 7](#_Toc134272219)

[System Diagram 9](#_Toc134272220)

[System Inputs: 11](#_Toc134272221)

[System Outputs: 14](#_Toc134272222)

[System Processing: 19](#_Toc134272223)

[System Requirements: 20](#_Toc134272224)

[Operating System Requirements: 20](#_Toc134272225)

[Program Requirements: 21](#_Toc134272226)

[System Constraints: 21](#_Toc134272227)

[System Implementation: 22](#_Toc134272228)

[Configuration and Launching the ManagementUI 22](#_Toc134272229)

[Adding a User and Logging in 22](#_Toc134272230)

[Preparing the Face Directories Data Files 22](#_Toc134272231)

[The Objects Frame and Adding Objects 23](#_Toc134272232)

[Launching and Using the Drone Controller 24](#_Toc134272233)

[Launching and Using the Mapping Controller 25](#_Toc134272234)

[Viewing Flight Logs and Logging Out 26](#_Toc134272235)

[Program Documentation 27](#_Toc134272236)

[Program Name: managementUI.py 27](#_Toc134272237)

[Program Flow Diagram 27](#_Toc134272238)

[Program Input: 28](#_Toc134272239)

[Program Output: 29](#_Toc134272240)

[Program Modules: 31](#_Toc134272241)

[Program Description: 35](#_Toc134272242)

[Program Name: droneController.py 38](#_Toc134272243)

[Program Flow Diagram 38](#_Toc134272244)

[Program Input: 39](#_Toc134272245)

[Program Output: 39](#_Toc134272246)

[Program Modules: 41](#_Toc134272247)

[Program Description: 43](#_Toc134272248)

[Program Name: mappingController.py 45](#_Toc134272249)

[Program Flow Diagram 45](#_Toc134272250)

[Program Input: 46](#_Toc134272251)

[Program Output: 46](#_Toc134272252)

[Program Modules: 47](#_Toc134272253)

[Program Description: 49](#_Toc134272254)

[Data Structure or Algorithm Analysis 50](#_Toc134272255)

[Data Structure or Algorithm Name: 50](#_Toc134272256)

[Description 50](#_Toc134272257)

[Analysis 53](#_Toc134272258)

[DB Documentation 55](#_Toc134272259)

[DB Description: 55](#_Toc134272260)

[DB Type: 55](#_Toc134272261)

[DB Tables 55](#_Toc134272262)

[DB Relationship Diagrams 56](#_Toc134272263)

[File Storage Documentation 57](#_Toc134272264)

[File Storage Description: 57](#_Toc134272265)

[File Storage Directories: 57](#_Toc134272266)

[Forms and Screens 58](#_Toc134272267)

[Use Cases 69](#_Toc134272268)

[Known Issues 70](#_Toc134272269)

[Future Enhancements 73](#_Toc134272270)

[Other Resources 75](#_Toc134272271)

# Abstract

***Introduction.*** The world of the future once dreamed about is upon us. The prevalence of technology and the internet we see today and how it has an effect on every facet of human life demonstrates this fact. This has allowed for many societal benefits yet there remains untapped potential in current technology that is underexploited. This revolutionary technology in discussion refers to robotics and artificial intelligence and from these areas drone technology and computer vision are two particular avenues needing to be explored. This project thus implements this underutilized technology by providing a drone controller program with computer vision capabilities for the Tello EDU mini drone. ***Problem.*** A drone such as this provides the solution to many problems that exist in the world today. For example, a drone with computer vision capabilities can be controlled to navigate through hazardous situations and identify people in distress so that a rescue team can then safely extract them. Similarly, a drone can be deployed as a mobile security system that can identify people as permitted personnel or trespassers. ***Interdisciplinary.*** From these examples, it can be understood that this project has an impact on the academic disciplines of disaster management and homeland security. Drone technology with computer vision capabilities mitigates risk in both sectors while at the same time increasing the efficiency of operations. ***Body.*** Implemented as a native computer application, this project provides a GUI in which users can control the drone using their computer keys as input and receive the live video feed from its camera. Additionally, users can utilize computer vision functionality such as object and facial recognition/detection through this interface. Other features provided through the GUI include 2D mapping of an area, the ability to capture and store images, video, and other data from the camera feed and drone, and the ability to add data for the drone to learn from. These functionalities are accomplished using the TelloEDU SDK and the OpenCV computer vision library. With all code written in the Python programming language, the Tkinter interface is readily available for GUI creation. This choice of programming language also allows for an SQLite relational database to be easily deployed as it is provided by default with Python. An SQLite relational database will allow for information from the drone to be stored and tracked easily through the use of the GUI. Advantages of using this database include its serverless architecture, open source nature, and is stored as a single binary disk file. The latter of these benefits allows for the database to be accessed remotely as well as for easy transport of the database as it can be stored on a flash drive. To accomplish these functionalities the system presented for this project is comprised of three primary modules; one database module, one TelloEDU module, and one GUI module. ***Results.*** Operation of this system mitigates risk and increases the safety of those involved in situations where areas must be searched or people must be identified by providing the capabilities to do so remotely. ***Conclusions.*** This project accomplishes the goal of providing a drone controller with computer vision capabilities. Being run natively and implementing a database with a serverless architecture allows for the controller to be accessed remotely and for any data collected to be easily transported. The system can be utilized for disaster management to search for and identify persons. Likewise, this system can act as a mobile security camera system providing those in homeland security the ability to secure areas and identify persons in these areas safely and remotely.

# System Documentation

## System Overview

This overall system integrates a data management system, drone controller with computer vision capabilities, and a drone mapping controller into one overall structure. Throughout this work the drone controller with computer vision capabilities will simply be referred to as the drone controller. The data management system is where user accounts can be added to the database and users can login to the overall system. Upon logging in with valid credentials this data management system provides users a way to view data created through use of the management system and drone controller. Users can also add face and object images in the data management system to be used in the drone controller to accomplish computer vision functionalities. This data management system also provides access to the drone controller and mapping controller. In addition to the data management system, the drone controller provides a GUI in which users can control the drone using their computer keys as input, receive the live video feed from the drones, utilize computer vision functionality such as object and facial recognition, and capture and store images, video, and other data from the camera feed and drone itself. Lastly, the mapping controller system provides the ability to perform 2D flight path mapping of an area and save this map to the Maps directory.

I approached creating this system by first dividing the features stated in my abstract into two programs; one to manage and provide access to all data and a second to control the drone and use computer vision functionalities. Next the system was designed from the point where data is first added to the system to where it is implemented. To do this the data management system was developed first. This was approached using a MVC design pattern so that the modules within the system could be created and tested independently for each data element. First dealing with how a user is added to the database and logs into the system the home frame and other data element related frames were procedurally designed and developed. To implement the drone controller and its functionality I referenced a compilation of Python scripts I have developed over the past year while learning to program the Tello Edu drone. An issue was encountered when working on this program relating to the number of threads the drone can manage. To overcome this challenge, the system was simplified to its most basic forms of the functionality required. In doing this the mapping controller needed to be created as a separate executable program from the drone controller with computer vison capabilities. In this way the three systems were developed to be accessible through individual executable files with the management system having entry points to the drone controller and mapping controller executables.

## System Diagram

Graphical user interface

Description automatically generated

A picture containing text, sign, different, several

Description automatically generated

Timeline

Description automatically generated

## System Inputs:

* Data Management System (Program 1)
  + Login Frame
    - Username
    - Password
    - Login Button
    - Add User Button
  + New User Frame
    - Username
    - Password
    - First name
    - Last name
    - Add user button
  + Home Frame
    - Faces Frame Button
    - Objects Frame Button
    - Logs Frame Button
    - Drone Controller Button
    - Mapping Controller Button
    - Logout Button
  + Faces Frame
    - Open Button
    - New Button
    - Close Button
  + Add Face Frame
    - First Name entry box
    - Last Name entry box
    - search button
    - Add button
    - Close Button
  + Objects Frame
    - Open Button
    - New Button
    - Close Button
  + Add Object Frame
    - Object Name entry box
    - search button
    - Add button
    - Close Button
  + Logs Frame
    - Close Button
* Drone Controller (Program 2)
  + Takeoff/Land button
  + Key Presses (a, w, s, d, up arrow, down arrow, right arrow, and left arrow keys)
  + Key Releases (a, w, s, d, up arrow, down arrow, right arrow, and left arrow keys)
  + Picture Button
  + Video Button
  + Recognize Face Button
  + Recognize Object Button
  + Set Speed Scale Bar
  + Reset Speed Button
  + Tello EDU WIFI connection
* Mapping Controller (Program 3)
  + Key Presses (a, w, s, d, and up arrow keys)
  + Takeoff Button
  + Land Button
  + Write Point Button
  + Set Distance Scale Bar
  + Reset Distance Button
  + Save Map Button

## System Outputs:

* Data Management System (Program 1)
  + Login Frame
    - Login Button – Check that the username and password entered exist in the database and if true then log them into the system as the current user. If logged in as the current user the home frame is displayed.
    - Add User Button – If clicked display the New User Frame.
  + New User Frame
    - If there isn’t already a user in the database with the entered Username and Password then add the user to the database with these entries and as well as the entries for the users first name and last name for the records column values.
  + Home Frame
    - Faces Frame Button – display the faces frame.
    - Objects Frame Button – display the objects frame.
    - Logs Frame Button – Query the logs table and display its entries in the logs frame.
    - Drone Controller Button – Launch the drone controller executable.
    - Mapping Controller Button – Launch the mapping controller executable.
    - Logout Button – Set the current user to none and display the login frame.
  + Faces Frame
    - Open Button – Display the open file dialogue box in the Faces directory and allows the user to either double click one of these image files to display it or select it and then press the open button to do so.
    - New Button – Display the add face frame.
    - Close Button – Display the home frame.
  + Add Face Frame
    - search button – Display the open file dialogue box allowing users to either double click an image files to copy its file path or select it and then press the open button to do so.
    - Add button – Add the image searched to the Faces directory with the base file name set to the values entered in the first and last names entry boxes with a space between them.
    - Close Button – Display the faces frame.
  + Objects Frame
    - Open Button – Display the open file dialogue box in the Objects directory and allows the user to either double click one of these image files to display it or select it and then press the open button to do so.
    - New Button – Display the add objects frame.
    - Close Button – Display the home frame.
  + Add Object Frame
    - search button – Display the open file dialogue box allowing users to either double click an image files to copy its file path or select it and then press the open button to do so.
    - Add button – Add the image searched to the Objects directory with the base file name set to the value entered in the object name entry box.
    - Close Button – Display the Objects frame.
  + Logs Frame
    - Close Button – Display the home frame.
* Drone Controller (Program 2)
  + Takeoff/Land Button – Send the takeoff command to the drone if its flying state is false. Send the land command to the drone if its flying state is true.
  + Key Presses (a, w, s, d, up arrow, down arrow, right arrow, and left arrow keys)
    - a – While pressed rotate the drone counter clock wise.
    - w – While pressed move the drone upward.
    - s – While pressed rotate the drone clock wise.
    - d – While pressed move the drone downward.
    - up arrow – While pressed move the drone forward.
    - down arrow – While pressed move the drone backward.
    - right arrow - While pressed move the drone right.
    - left arrow – While pressed move the drone left.
  + Key Releases (a, w, s, d, up arrow, down arrow, right arrow, and left arrow keys) – If a key pressed is release send RC controls to stop flying (hover in place) to the Drone
  + Picture Button - Save the current frame to the images directory with the file name being the time the picture was taken.
  + Video Button - Record video for the duration of the flight by constantly getting the current frame read and writing it to a video file. This button is made inactive when pressed.
  + Recognize Face Button – Display “Unknown” in the Face Recognized label if no faces in the current frames match the encoded ones from the faces directory. If a match is found then display the file name of the face image matched to.
  + Recognize Object Button – Display “Unknown” in the Object Recognized label if no objects in the current frames match the encoded ones from the Objects directory. If a match is found then display the file name of the object image matched to.
  + Set Speed Scale Bar – Slider value moved to indicates current speed drones default speed will be reset to if reset speed button is pressed.
  + Reset Speed Button – Set the drones flight speed to the value set in the set speed scale bar.
  + Tello EDU WIFI connection – Open a connection with the drone to send commands and turn its video stream on and show it in the video stream label.
  + Face Count label – After turning on the drones video stream a frontal face cascade is used to dynamically get and update the count of the faces currently in the drones camera feed by writing this number in the face count label
* Mapping Controller (Program 3)
  + Key Presses (a, w, s, d, and up arrow keys)
    - a – Rotate the drone counter clockwise 90 degrees and set the turtles heading respectively.
    - w – Move the drone up the set distance.
    - s – Move the drone down the set distance.
    - d – Rotate the drone clockwise 90 degrees and set the turtles heading respectively.
    - up arrow – Move the drone forward the set distance and move the turtle respectively.
  + Takeoff Button – Send the takeoff command to the drone.
  + Land Button – Send the land command to the drone.
  + Write Point Button – Write a point with the drones current position next to the turtles current position on the canvas.
  + Set Distance Scale Bar – Slider value moved to indicates current distance the drone will move by when reset by pressing the reset distance button.
  + Reset Distance Button – Set the default movement distance to the value set in the set distance scale bar.
  + Save Map Button – Save the map made on the turtle canvas to the maps directory as a .png file.
  + Tello EDU WIFI connection – Open a connection with the drone to send commands

## System Processing:

This system starts with the login frame where a user enters their username and password and click logins to gain further access. This button causes the system to check the databases Users table for a record with a username and password matching those entered in the login frame and if a match is found this user with those credentials is logged into the system as the current user. If a user is logs in as such then the home frame is displayed. If this is the first time the system is being used then the user must instead click the new user button on the login frame to display the new user frame. After entering values in this form for username, password, first name, last name, and clicking the add user button the system will check that there are no records in the Users table matching the entered username and password and if none are found then a record is added to the table with the entered values. From here the user can login as previously described to display the home frame.

From the home frame a user can select the mapping controller button or the drone controller button to launch these programs respective executable files and terminate the management program. The home frames logs button can be pressed to display the logs frame. When pressed the flight logs table in the database is queried to get all of the records and show them in the logs frame. If the close button is clicked on the logs frame then the home frame is again displayed. The home frames faces and object buttons can be pressed to display the faces and objects frames respectively. Both of these frames have open, new, and close buttons with the open buttons displaying the open file dialogue box in the frames respective directory when pressed, the new button displaying the add face frame or the add object frame respectively, and the close button displaying the home frame. The last button on the home frame is the logout button and when pressed the current user is set to none and the login frame is displayed. In both the add face and add objects frame, a user must complete all fields and then click the add button to add the forms associated image/information to the frames respective file directory.

If either the mapping controller or drone controller executables are launched from the home frame the management program is terminated and is replaced by the executable selected. For both of these programs to launch successfully a user must first connect their computer to the Tello EDUs Wi-Fi signal otherwise the program will not launch but instead an error message will be displayed. In the drone controller program, all processes described in the inputs/outputs are functional until the End Flight button is pressed (excluding recording video as this can only be executed once be running of the program). When pressed the databases flight logs table is queried and the flight log information for that use is inserted into the table. From here commands are sent to the drone to land, turn off its video stream, and disconnect from the program. Lastly, the program ends all running process and the window is closed. In the mapping controller program, all process described in the inputs/outputs are functional until the window is closed by clicking the “X” button in the windows top right corner. When this occurs all processes are ended and the window is closed. From here a user must re-launch the management program to see any data captured or utilize any of the different programs functionality again.

## System Requirements:

### Operating System Requirements:

* Windows 8.1 or newer

### Program Requirements:

* Python 3.10
* Pillow~=9.4.0
* opencv-python~=4.7.0.68
* numpy~=1.24.1
* djitellopy~=2.4.0

## System Constraints:

This system was developed to be used in both remote and populated locations. Given this, an elementary approach was taken so that the system includes and requires the least amount of resources possible and can be executed on any windows computer with little to no setup. This was accomplished by using the python standard library to implement as much of the project as possible. Deciding this, the Tkinter package was the most sensible and readily available toolkit for creating the GUI. This led to issues when implementing the computer vision functionality in the project. These features I had previously worked with using the OpenCV window to process video streams. Thinking this could translate directly to Tkinter, it was found that there is a lot of behind the scenes processing that OpenCV does that would need to be accounted for if displaying images in Tkinter. Many approaches were taken to implement these processes but all resulted in to many processes needing synchronized between the video stream processes and the ones for controlling the drone itself. To overcome this, computer vison was implemented in such a way that user events trigger the computer vision methods to run once and return a value instead of dynamically running as the video stream is received from the drone. While this was not the original intention it allowed for the required functionality for the system to be implemented. From this it can be understood that the largest constraint for this system was implementing complex processes as simplicity is a foundational aspect of the way this project was designed.

## System Implementation:

### Configuration and Launching the ManagementUI

To implement this project one must first download the Project folder to a usb flash drive so that the program code can access the needed data folders in the correct way. From here, go into the execution folder and launch the managementUI executable file.

### Adding a User and Logging in

Upon the program load, the login frame will be displayed. In this frame, press the new user button to display the add user form. To add a user account enter a username, password, first name, and last name is the entry boxes on this form and press the submit button to query the database and insert a record to the users table with these values. If the entered username and password do not already exist in the database then a validation message is displayed at the bottom of the add user form stating, “User added to the system”. If this message is displayed press the cancel button to display the login frame and enter your newly added username and password in the entry boxes and press the login button to gain entry to the system and be brought to the home frame. If an incorrect username or password is entered in the login frame a error message will be displayed at the bottom of the frame stating, “Wrong Username or Password”.

### Preparing the Face Directories Data Files

If you want faces to be recognized when using the drone controller then these files must be added prior to launching it. To do this, press the Faces button on the home frame to display the faces frame. In this frame press the New button to display the Add Face form. In this form, press the search button to display a file system search box. In this, find the face image you want to add from your computer and select it to display its file path next to the search button in the add face frame. Next enter a first name and a last name in the first and last name entry boxes and press the add button to save this image with the entered first and last name to the faces directory. Upon successfully adding a face a verification message is displayed at the bottom of the add face frame stating, “face successfully added”. Repeat this process to add more face image files. Next, click the close button on the add face frame to display the faces frame. In this frame, the open button can be used to display a file search box in the faces directory. Selecting a file will display the image in the faces frame. Pressing the close button on the faces frame will return the you to the home frame.

### The Objects Frame and Adding Objects

If you want objects to be recognized when using the drone controller then these files must be added prior to launching it. To do this, press the objects button on the managementUI’s home frame to display the objects frame. In this frame press the New button to display the Add Object form. In this form, press the search button to display a file system search box. In this, find the object image you want to add from your computer and select it to display its file path next to the search button in the add object frame. Next enter object name in the entry box and press the add button to save this image with the entered object name to the objects directory. Upon successfully adding an object a verification message is displayed at the bottom of the add face frame stating, “object successfully added”. Repeat this process to add more object image files. Press the close button on the add object frame to display the objects frame. In this frame, the open button can be used to display a file search box in the objects directory. Selecting a file will display the image in the objects frame. Pressing the close button on the objects frame will return the you to the home frame.

### Launching and Using the Drone Controller

Once you have added face or object images you want the drone to be able to recognize the drone controller program can be used. Note, the drone controller can also be used without adding any face or object images to the respective directories but these recognition capabilities will not be functional. To use the drone controller, power on you Tello EDU drone then under your computers available Wi-Fi sources connect to the one with text “TELLO-FDCD##”. Note that the numbers here differ depending on the Tello EDU drone being used. Establishing a connection with the drone is indicated when your computers Wi-Fi states that it is connected to this source without internet. Once this is done, the drone controller button can be pressed on the managementUI’s home frame. This causes a command prompt to appear and upon exiting out of it the managementUI’s window will close. It may appear that nothing is happening from this point but the face encodings and being performed and once done the drone controller screen will be displayed for use.

At this point the facial and object recognition buttons can be pressed to identify known faces and objects in the drones feed and display the known face/objects filename in the labels located above each button. Recording, image capturing, and setting the drones flight speed can also be done prior to takeoff. Pressing the record button causes the program to record the drones video stream for the remainder of the programs life. Pressing the image capture button saves the drones current frame as an image file. Adjusting the scale bar changes the value the drones flight speed will be set to when the reset speed button is pressed and similarly the reset speed button sets the drones flight speed to the value indicated by the set speed scale bar.

Press the takeoff/land button for the first time sends the takeoff command to the drone, sets its flying status to true, and allow flight commands to the drone through key presses. Move the drone by pressing a movement key indicated in the control help image. Release the key last pressed to cause the drone to stop flying and hover at its current position. Press the takeoff/land button a second time to send the land command to the drone and set its flying status to false. Press the end flight button to query the databases flight logs table and insert a record with this flights recorded values. After inserting this record the database connection is closed and all processes under this program are terminated. The managementUI program must be launched again to access any captured data or utilize program functionality further. Note, once you are done using this program power off your drone so that it does not sit running as this can cause it to overheat and hardware to be damaged.

### Launching and Using the Mapping Controller

To use this program, power on you Tello EDU drone then under your computers available Wi-Fi sources connect to the one with text “TELLO-FDCD##”. Note that the numbers here differ depending on the Tello EDU drone being used. Establishing a connection with the drone is indicated when your computers Wi-Fi states that it is connected to this source without internet. Once this is done, the mapping controller button can be pressed on the managementUI’s home frame. This causes a command prompt to appear and upon exiting out of it the managementUI’s window will close and the mapping controller screen is displayed.

Here, the turtle canvas is displayed at the center of the screen with the turtles current position (zero point) marked next to the arrow on this canvas. An image is shown on the left side of the screen instructing one on the computer keys used for controlling the drone and how to use the other buttons available. Adjust the scale bar to change the distance the drone will move by default for each key press. Set this as the movement value by pressing the reset distance button. To complete a mapping mission first use the takeoff button to send the takeoff command to the drone. Move the drone using the movement keys to draw the path moved onto the turtle canvas. To draw a point with the drones current position next to the turtles on the canvas press the write point button. If done mapping an area press the save map button to save the map made on the turtle canvas to the maps directory as a .png file. Use the land button to send the land command to the drone. This can be done prior to saving the map or after saving the map but ensure to land the drone prior to closing out of the program completely. This is done by clicking the x button in the windows top right corner. From here a user must re-launch the managementUI program to see any data captured or utilize any of the different programs functionality again. Note, once you are done using this program power off your drone so that it does not sit running as this can cause it to overheat and hardware to be damaged.

### Viewing Flight Logs and Logging Out

After you have completed at least one flight using the drone controller program you can relaunch the managementUI from the execution folder and login as done previously. In the home frame, press the logs button to display the logs frame. In this frame, a list of all flight logs made using the drone controller is shown and a scroll bar can be used to scroll through the list if it is very long. Pressing the close button from this frame returns one to the home frame where the previous functionalities described can continue to be used. To logout of the system, press the logout button on the home frame to be logged out of the system as the current user and display the login frame. You can then login or add a user again or press the red x button in the top right corner of the window to close the program.

# Program Documentation

## Program Name: managementUI.py

### Program Flow Diagram

Graphical user interface

Description automatically generated

### Program Input:

* Login Frame
  + Username
  + Password
  + Login Button
  + Add User Button
* New User Frame
  + Username
  + Password
  + First name
  + Last name
  + Add user button
* Home Frame
  + Faces Frame Button
  + Objects Frame Button
  + Logs Frame Button
  + Drone Controller Button
  + Mapping Controller Button
  + Logout Button
* Faces Frame
  + Open Button
  + New Button
  + Close Button
* Add Face Frame
  + First Name entry box
  + Last Name entry box
  + search button
  + Add button
  + Close Button
* Objects Frame
  + Open Button
  + New Button
  + Close Button
* Add Object Frame
  + Object Name entry box
  + search button
  + Add button
  + Close Button
* Logs Frame
  + Close Button

### Program Output:

* Login Frame
  + Login Button – Check that the username and password entered exist in the database and if true then log them into the system as the current user. If logged in as the current user the home frame is displayed.
  + Add User Button – If clicked display the New User Frame.
* New User Frame
  + If there isn’t already a user in the database with the entered Username and Password then add the user to the database with these entries and as well as the entries for the users first name and last name for the records column values.
* Home Frame
  + Faces Frame Button – display the faces frame.
  + Objects Frame Button – display the objects frame.
  + Logs Frame Button – Query the logs table and display its entries in the logs frame.
  + Drone Controller Button – Launch the drone controller executable.
  + Mapping Controller Button – Launch the mapping controller executable.
  + Logout Button – Set the current user to none and display the login frame.
* Faces Frame
  + Open Button – Display the open file dialogue box in the Faces directory and allows the user to either double click one of these image files to display it or select it and then press the open button to do so.
  + New Button – Display the add face frame.
  + Close Button – Display the home frame.
* Add Face Frame
  + search button – Display the open file dialogue box allowing users to either double click an image files to copy its file path or select it and then press the open button to do so.
  + Add button – Add the image searched to the Faces directory with the base file name set to the values entered in the first and last names entry boxes with a space between them.
  + Close Button – Display the faces frame.
* Objects Frame
  + Open Button – Display the open file dialogue box in the Objects directory and allows the user to either double click one of these image files to display it or select it and then press the open button to do so.
  + New Button – Display the add objects frame.
  + Close Button – Display the home frame.
* Add Object Frame
  + search button – Display the open file dialogue box allowing users to either double click an image files to copy its file path or select it and then press the open button to do so.
  + Add button – Add the image searched to the Objects directory with the base file name set to the value entered in the object name entry box.
  + Close Button – Display the Objects frame.
* Logs Frame
  + Close Button – Display the home frame.

### Program Modules:

* Program Main Module
  + managementUI.py – This module contains and executes the main function for this program. In this function the model, view, and controller classes are initialized and the controllers method to launch the program is called.
* Database Module
  + database.py – This module contains variables that store the create table statements for the three database tables used by the system. In addition to this, a login function that is used to validate user login credentials and a function to get the name of the current user logged in are contained in this module. This module is imported for use in the models packages logs module, and users module as well as in the controllers packages login.py module.
* Models Package Modules
  + base.py – This module contains the Observable Model class which is used to register the systems event listeners by storing all event listeners for the system in a dictionary. This is done using a method that takes in an event name and a callback function as input and storing them in the dictionary with the events name as the key and with each key holding a list of the callback functions for that event. The output of this method is a callable function that removes the event listener when called. In addition to a method to add events to the dictionary, a second method is contained in this model to take an event name as input and check if the input has any callback functions registered to it. If this is found to be true, then all of the callback functions registered to this event are called.
  + faces.py – This model contains the Faces model class which interacts with the Faces directory. This handles the data logic of a user request regarding getting data from and writing data to the Faces directory. This module also contains a function for encoding the face images in the faces directory which is not used in the management system but is used to create the face encodings when the drone controller is launched.
  + logs.py – This module contains the Logs model class which interacts with the FlightLogs database table. This handles the data logic of a user request regarding getting data from and adding data to the flightLogs table but for this program, data is only taken from the table. Data is added through use of the drone controller program.
  + main.py – This module contains the Model class which initializes the Faces, Objects, Users, and Logs model classes for use in this program.
  + objects.py – This module contains the Object model class which interacts with the Objects directory. This handles the data logic of a user request regarding getting data from and writing data to the Objects directory. This module also contains a function for getting the descriptors for the object images in the objects directory which is not used in the management system but is used to create the object descriptors when the drone controller is launched.
  + users.py – This module defines the Users class model which is used to interact with the User database table. This handles the data logic of a user request regarding writing data to, getting data from, and validating data from the Users table.
* Views Package Modules
  + add\_face.py – This view defines the addFace frame class and is displayed in the Main window when the user selects the new button on the faces frame.
  + add\_object.py – This view defines the addObject frame class and is displayed in the Main window when the user selects the new button on the objects frame.
  + add\_user.py – This view defines the addUser frame class and is displayed in the main window when the user selects the New User button on the login frame.
  + faces.py – This view defines the faces frame class and is displayed in the Main window when the user selects the Faces menu item.
  + flight\_logs.py – This view defines the logs frame class and is displayed in the Main window when the user selects the logs menu item.
  + home.py – This view defines the home frame class and is displayed in the Main window on valid login.
  + login.py – This view defines the login frame class and is displayed in the main window on program start.
  + main.py – This view defines the view class which is used to display and switch between the other views. This is done by first initializing the class to store all views in a dictionary. Three methods are then contained in the class with one used to initialize and store the frames, the second used to switch between the frames by calling the tkraise() method, and the third being the a mainloop() method that starts the GUI.
  + objects.py – This view defines the objects frame class and is displayed in the Main window when the user selects the Objects menu item.
  + root.py – This view defines the Root class which inherits the Tk class. This acts as the root window for the system.
* Controllers Package Modules
  + add\_face.py – This controller controls the elements in the addFace frame view.
  + add\_object.py – This controller controls the elements in the addObject frame view.
  + add\_user.py – This controller controls the elements in the addUser frame view.
  + faces.py – This controller controls the elements in the Faces frame view.
  + flight\_logs.py – This controller controls the elements in the Logs frame view.
  + home.py – This controller controls the elements in the Home frame view.
  + login.py – This controller controls the elements in the Login frame view.
  + main.py – This controller initializes all other controllers, contains event listener functions to switch the between the views when the different menu items are pressed, and has a method that displays the login view on starting the GUI and launches the GUI by calling the method in the main.py view that runs the Tkinter event loop.
  + objects.py – This controller controls the elements in the Objects frame view.

### Program Description:

This program is the entry point for all of the systems functionality. Upon launching the application the login form is shown in the main window. A first time user must then click the new user button to display the add user form. Once all entries are completed in this form a user clicks the submit button to add a record to the databases Users table for that user. A message is displayed at the bottom of this form if the entered information already exists in the database. If the record is successfully added a message is displayed at the bottom of this form stating as such and the user can press the cancel button to return to the login form. Having a record in the database a user can then enter their username and password in the login form and click the login button to be entered into the system as the current user and have the home screen displayed. If either of these fields are not found in the users table an error message is displayed at the bottom of the login form stating as such.

Upon valid entry through the login form the home screen is displayed with the current users first and last names, the drone name, and the drones sdk version shown in a gray box under the navigation buttons. From here, a user can utilize any of the navigation buttons to utilize further parts of this program. Two of these buttons, the “Drone Controller” and “Mapping Controller” buttons, launch their associated program executables when pressed. In addition to this, the managementUI program and its associated processes end upon pressing these two buttons. The Faces button displays the faces frame when pressed. A user can then open a face image by clicking the open button in this frame. This causes an open file dialogue box to be displayed in the faces directory for the user to select a image t from. Once a image is selected, the faces frame is resized to display it in the area above the buttons with the images file name shown under the image itself. The cancel button can be pressed to return the user to the home screen or the new button can be pressed to display the add face frame. The Objects frame operates in the same manner as this when the objects button is pressed from the home screen but instead of using the faces directory to open files, the objects directory is used.

The add face frame is displayed when a user selects the new button on the faces frame. This form allows users to enter a first name, last name, and search for an image file through pressing the search button. Upon completing these actions, a user can select the add button to rename the image file searched to the first and last name entered and save the file in the faces directory. A message is displayed if the face is successfully added and the user can then press the close button to return to the faces frame. The add object frame operates in the same manner as this when the new button is pressed on the objects frame. The difference between these is that the add objects form only takes a file and a object name to add a file and it does so in the objects directory.

Back in the home screen, the logs button can be pressed to display the logs frame. This frame queries the databases flight logs table and displays all of its records in a scrollable panel in the frame. A user can use the scroll bar to move down the list of records or they can press the cancel button to return to the home screen. Lastly, the home screens logout button can be pressed to return the user to the login screen as log them out of the system by setting the active user to none. From here the managementUI can be utilized in the ways previously stated.

## Program Name: droneController.py

### Program Flow Diagram

A picture containing text, sign, different, several

Description automatically generated

### Program Input:

* Takeoff/Land button
* Key Presses (a, w, s, d, up arrow, down arrow, right arrow, and left arrow keys)
* Key Releases (a, w, s, d, up arrow, down arrow, right arrow, and left arrow keys)
* Picture Button
* Video Button
* Recognize Face Button
* Recognize Object Button
* Set Speed Scale Bar
* Reset Speed Button
* Tello EDU WIFI connection

### Program Output:

* Takeoff/Land Button – Send the takeoff command to the drone if its flying state is false. Send the land command to the drone if its flying state is true.
* Key Presses (a, w, s, d, up arrow, down arrow, right arrow, and left arrow keys)
  + a – While pressed rotate the drone counter clock wise.
  + w – While pressed move the drone upward.
  + s – While pressed rotate the drone clock wise.
  + d – While pressed move the drone downward.
  + up arrow – While pressed move the drone forward.
  + down arrow – While pressed move the drone backward.
  + right arrow - While pressed move the drone right.
  + left arrow – While pressed move the drone left.
* Key Releases (a, w, s, d, up arrow, down arrow, right arrow, and left arrow keys) – If a key pressed is release send RC controls to stop flying (hover in place) to the Drone
* Picture Button - Save the current frame to the images directory with the file name being the time the picture was taken.
* Video Button - Record video for the duration of the flight by constantly getting the current frame read and writing it to a video file. This button is made inactive when pressed.
* Recognize Face Button – Display “Unknown” in the Face Recognized label if no faces in the current frames match the encoded ones from the faces directory. If a match is found then display the file name of the face image matched to.
* Recognize Object Button – Display “Unknown” in the Object Recognized label if no objects in the current frames match the encoded ones from the Objects directory. If a match is found then display the file name of the object image matched to.
* Set Speed Scale Bar – Slider value moved to indicates current speed drones default speed will be reset to if reset speed button is pressed.
* Reset Speed Button – Set the drones flight speed to the value set in the set speed scale bar.
* Tello EDU WIFI connection – Open a connection with the drone to send commands and turn its video stream on and show it in the video stream label.
* Face Count label – After turning on the drones video stream a frontal face cascade is used to dynamically get and update the count of the faces currently in the drones camera feed by writing this number in the face count label

### Program Modules:

* Program Main Module
  + droneController.py – This module contains the GUI class and executes the main function for this program in which the GUI class is initialized and its run application method is called. In its definition, the GUI class initializes the root window, the Faces, Objects, and Logs models, the drone object, its video stream, and its other attribute variables. The GUI class then contains methods to takeoff/land the drone, take a picture, record video, end the flight, add the flight log to the database, update the drones default flight speed, run the app, count the faces in the video stream, recognize faces in the video stream, recognize objects in the video stream, and receive the video stream from the drone. In the GUIs run application method, the TKinter widgets and the video stream are added to the window, key/button presses are bound to their respective commands, and the TKinter main loop is started. Key press commands are bound to those functions contained in the flight\_commands.py module which is imported for use by this module.
* Database Module
  + database.py – This module contains variables that store the create table statements for the three database tables used by the system. In addition to this, a login function that is used to validate user login credentials and a function to get the name of the current user logged in are contained in this module but are only used in the management program. For the drone controller program this module is imported for use in the models packages logs module.
* Models Package Modules
  + faces.py – This model contains the Faces model class which interacts with the Faces directory. This module contains a function for encoding the face images in the faces directory which is used to create the face encodings when this program is launched. The Face model class also handles the data logic of a user request regarding getting data from and writing data to the Faces directory but for this program data is not written to the Faces directory but only read from it.
  + logs.py – This module contains the Logs model class which interacts with the FlightLogs database table. This handles the data logic of a user request regarding getting data from and adding data to the flightLogs table but for this program, data is only added to the table.
  + objects.py –This module contains the Object model class which interacts with the Objects directory. This module contains a function for getting the descriptors for the object images in the objects directory which is used to create the list of object descriptors when this program is launched. The Object model class also handles the data logic of a user request regarding getting data from and writing data to the Objects directory but for this program data is not written to the Objects directory but only read from it.
* Flight Commands Module
  + flight\_commands.py – This module contains three functions, one to fly the drone in a certain direction, another to start flying a given direction at a given speed when an event occurs, and a third to stop flying if an event occurs. The keys press events are bound to the start flying function in the droneController.py module with a series of checks in the function used to determine the direction to send to the fly function depending on which key is pressed. For any key that is pressed, its associated direction is sent to the fly function which then sends a RC control command to the drone to fly that way. All keys with press events have release events that are bound to the stop flying function which sends a RC control command to the drone to stop flying and hover in place.

### Program Description:

This program is launched by pressing the “Drone Controller” button in the managementUI programs home screen after turning on the drone and connecting to it as a WIFI source. Upon launch the managementUI program ends completely the GUI class is instantiated causing for a slight time delay between when the managementUI’s home screen is closed and the drone controller programs screen is displayed. During this time, the faces and objects model are instantiated and a list of encodings is created for the images in the faces directory and a list of descriptors is created for the images in the objects directory. This can take more or less time depending on the number of images in each directory needing processed. Next the drone is connected to, its stream is turned on, and its default flight speed is set to 25. The logs model is next instantiated and the pre-takeoff variables for a flight log record are initialized. Having initialized all variables and methods, the GUI classes run application method is called and is where the programs functionality used by the user takes place. Once the above processes occur, the drone controller screen is shown and the user can utilize any of the programs described inputs.

The screen includes a frame displaying the drones video feed in addition to an image with instructions of how to control the drone using key presses. Within the video feeds programming, each time a frame is received from the drone it is passed to the count faces method before updating the video stream label. This method uses a frontal face cascade to identify human faces in the video frame. Using this, the method tracks the number of faces in each frame by looping through the faces in the video stream and if one is detected it is added to a list. Once a single frame has been checked, the length of this list is used to get the count of the faces in the frame and updated the face count label to display this number.

At this point the facial and object recognition buttons can be used to identify known faces and objects in the drones feed and display the known face/objects filename in the labels located above each button. Recording, image capturing, and setting the drones flight speed can also be done prior to takeoff. Pressing the record button causes the program to record the drones video stream for the remainder of the programs life. Pressing the image capture button saves the drones current frame as an image file. Adjusting the scale bar changes the value the drones flight speed will be set to when the reset speed button is pressed and similarly the reset speed button sets the drones flight speed to the value indicated by the set speed scale bar.

Pressing the takeoff/land button for the first time sends the takeoff command to the drone, sets its flying status to true, and allows the user to send flight commands to the drone through key presses on their computer. When a key is pressed its associated flight command is send to the drone as an RC control value. When the key is released a flight command is sent to the drone to stop flying and remain hovering at its current position. When the takeoff/land button is pressed a second time, the land command is sent to the drone and its flying status is set to false. As stated, when launching the program the current date, time, and drone battery status are captured as variables to be utilized once the user clicks the end flight button. When the end flight button is pressed, the time the user pressed the button, the drones current battery level, and the time the drones motors ran during the programs life are captured as program variables. After this the these are used with the variables captured on program launch to query the databases flight logs table and insert the values these variables hold as a record in this table. After inserting this record the database connection is closed and all processes under this program are terminated. The managementUI program must be launched again to access any captured data or utilize program functionality further.

## Program Name: mappingController.py

### Program Flow Diagram

Timeline

Description automatically generated

### Program Input:

* Key Presses (a, w, s, d, and up arrow keys)
* Takeoff Button
* Land Button
* Write Point Button
* Set Distance Scale Bar
* Reset Distance Button
* Save Map Button
* Tello EDU WIFI connection

### Program Output:

* Key Presses (a, w, s, d, and up arrow keys)
  + a – Rotate the drone counter clockwise 90 degrees and set the turtles heading respectively.
  + w – Move the drone up the set distance.
  + s – Move the drone down the set distance.
  + d – Rotate the drone clockwise 90 degrees and set the turtles heading respectively.
  + up arrow – Move the drone forward the set distance and move the turtle respectively.
* Takeoff Button – Send the takeoff command to the drone.
* Land Button – Send the land command to the drone.
* Write Point Button – Write a point with the drones current position next to the turtles current position on the canvas.
* Set Distance Scale Bar – Slider value moved to indicates current distance the drone will move by when reset by pressing the reset distance button.
* Reset Distance Button – Set the default movement distance to the value set in the set distance scale bar.
* Save Map Button – Save the map made on the turtle canvas to the maps directory as a .png file.
* Tello EDU WIFI connection – Open a connection with the drone to send commands

### Program Modules:

* Program Main Module
  + mappingController.py – This module contains the MappingUI class and executes the main function for this program in which the Tello class is initialized, MappingUI class is initialized with this Tello class instance passed as its parameter, and the TKinter mainloop function is then called on this MappingUI object. The MappingUI class takes in the Tello drone as a parameter and in its definition initializes the drone object, the TKinter mainloop thread, the stop event set to None, the event flag for receiving responses from the drone, the starting coordinates for the drone, the root window, the mapping canvas, and the turtle object (mapper). After these but still in the definition, the event object for the stop event is initialized, a callback is set on the root window for when it is closed, a thread is created for sending commands to the drone by setting its target to this classes sending commands method, and the MappingUI’s run application method is called. The sending command method uses a while loop to then send commands to the drone over 5 second intervals. The callback set for the root window is used in this classes on close method which sets the stop event, deletes the drone object, and closes all other processes. This is called when the root window is closed by pressing the red x button on the top right corner of the screen. Some other methods of this class are individual movement methods for each movement command used from the Tello class for controlling the drone. Of these, the methods used to control the drones forward, backward, and rotation movements also control the mappers movement on the map canvas. Also for controlling the drone, there is a class method for updating the drones default movement distance which is used when the reset distance button is pressed. The other methods of this class contain functionality pertaining to the map itself. These include a method to write the mapper/drones current position on the map as an (x, y) coordinate, a method to save the map to the maps directory, and a method used in the movement methods to update the mappers coordinates relative to the drone. This last method updates coordinates by determining the drones heading in relation to past rotation commands executed and setting the mappers heading respectively. The last method of this class is the run application method in which the TKinter widgets are added to the window and the key/button presses are bound to their respective commands.
* Drone Commands Module
  + tello.py – This module was adapted from a TelloEDU wrapper class from the education online source One-Off Coder (<One-Off Coder > (<2019>) < Python, Manual Control> (<Version N/A>) [<Source code>]. <https://tello.oneoffcoder.com/python-manual-control.html> ). This class is used to interact with the TelloEDU drone and was adapted by only including the methods from the original source code used in the project for controlling the drones movements and receiving responses from it. All other were deleted to reduce the amount of code factors relating to this class in particular. Aside from this, all code for this module/class is directly from the source so all credit should be given as such.

### Program Description:

This program is launched by pressing the “Mapping Controller” button in the managementUI programs home screen after turning on the drone and connecting to it as a WIFI source. Upon launch the managementUI program ends completely the GUI class is instantiated. In doing this, the drone, its default movement and rotation distances, starting coordinates, turtle, and turtle canvas are initialized. . Having initialized all variables and methods, the GUI classes run application method is called and is where the programs functionality used by the user takes place. Once the above processes occur, the mapping controller screen is shown and the user can utilize any of the programs described inputs.

Upon launching the program, the turtle canvas is displayed at the center of the screen with the turtles current position (zero point) marked next to the arrow on this canvas. An image is shown on the left side of the screen instructing users on the computer keys used for controlling the drone and how to use the other buttons available. The takeoff button can be used to send the takeoff command to the drone and the land button can be used to send the land command to the drone. Adjusting the scale bar changes the distance the drone will move by default for each key press. This is set when the reset distance button is pressed. Similarly the reset distance button sets the drones default movement value to that indicated by the scale bar when the button is pressed. When pressed, the write point button draws a point with the drones current position next to the turtles on the canvas. When pressed, the save map button saves the map made on the turtle canvas to the maps directory as a .png file. The program ends by clicking the “X” button in the windows top right corner. When this occurs all processes are ended and the window is closed. From here a user must re-launch the managementUI program to see any data captured or utilize any of the different programs functionality again.

# Data Structure or Algorithm Analysis

Add Log

## Data Structure or Algorithm Name:

### Description

This Algorithm is used to add records to the databases flight logs table each time the drone controller program is used to fly the drone and the end flight button is pressed. These records include values for flight date, start time, end time, start battery level, end battery level, and motor run time. The start time and start battery level values are class variables instantiated when the GUI class is initialized on program launch. The other values are obtained when the user clicks the end flight button on the drone controller screen. With this event triggering the algorithm, the current date, end time (time in which the user presses the button), end battery level, and motor run time are captured and with the start time and battery variables held by the GUI class, the databases flight logs table is queried by calling the logs models add\_log method and these values are inserted into it as a new record. Note that the logs models add\_log method takes in the column attributes for a flight log record and uses an INSERT INTO… VALUES… statement to insert the values into the table, and then commits the changes to the database before updating the Drones tables flight logs column with the current logs identifier value which is always the largest id when a log is added. After this, these changes are committed and the database connection is closed.

The drone controllers add log method is a class method that only takes the class itself in as a parameter and performs 7 operations that can be considered to take the same constant time t. After performing these 7 operations this method calls the logs models add\_log method which takes in 6 parameters and executes and commits two database queries with two different tables in the database. The logs model makes a connection with the database when the program is launched so this doesn’t need to be considered to analyze the add log algorithm in particular. Given the inputs in this algorithms processing are all constant and never grow, this algorithm is in a constant time complexity. From this is can be said that the memory requirements of this algorithm are the number of operations summed being 7 in the controllers add log method, and 4 in the log models add log method. This makes for the total memory requirements for the algorithm based on the general number of elements to be O(11).

To get the memory requirements in bytes the data type for each variable used is accounted for. For the variables in the drone controllers add log method, the FlightDate variable is of the datetime datatype and stores the month, day, and year for this object. According to the documentation for the datetime object, these fields are packed as successive bytes with these three taking 2, 1, and 1 bytes of memory respectively. This results in the FlightDate variable taking a total of 4 bytes of memory. The StartBattery, EndBattery, and MotorTime are all of type int each taking 4 bytes in memory. The time\_now variable is also of the datetime datatype but unlike FlightDate includes all of this objects field resulting in it to take up 10 bytes in memory. The other variables here are the start\_time and endTime which are both strings that contain 8 characters so each require 8 bytes in memory. Taking the memory requirements for these variables and summing them results in this method taking up 42 bytes in memory total. Since the Logs models add log method solely involves an insert statement for adding these values as a record to the database it is not to be included in the overall memory requirements in bytes for this algorithm as it is considered to be a memory requirement of the databases insert algorithm. It is also important to note that this memory requirement of 42 bytes is solely considering the bytes explicitly held by the variables in the method and does not include those taken up by the python garbage collector.

### Analysis

1. What are the memory requirements, based off of the general number of elements used?

42 bytes

1. What are the pros and/or cons associated with the element?

The pros of this algorithm are that flight log records are able to be inserted into the flight logs table while at the same time updating the value for the number of flights a drone has taken in the drones table. Another pro of this algorithm is that it has a constant time complexity entirely dependent on the number of columns a record in the flight logs table has. With this, more columns can be included in these records if desired without noticeably effecting performance.

1. What is an alternative program element that could have been used?

An alternative program element that could have been used is a list to store the column attributes for a flight log record instead of individual variables. This would make for less operations in the add log method of the drone controller program without making more operation in the log models method. In this way, the list could be passed as the only parameter to the log models method and the columns could be referenced by their index position in this list when inserting.

1. Why is the element appropriate for your project’s use?

Using individual variables for the column attributes is appropriate because these values are not needing to be changed for any future enhancements. If more columns were to be added to the flight logs table it would be just as easy to get a value for it by using another variable in the add log method of the GUI class. With this algorithm not requiring much of the computer’s memory the complexity of the program as a whole is reduced and more easily interpreted.

# DB Documentation

## DB Description:

The name of the database is droneControllerDB. This stores data on the systems users, drones, and flight logs. User information tracked includes a user’s username, password, first name, last name, and drone used. Drone information tracked is the drone’s name, the number of flights taken with the drone, and the SDK used for programming the drone. Flight log information includes the flight logs id number, the date the flight was taken on, the time the drone started and ended flying, the drones battery at the start and end of the flight, and the time that the drones motors we running during the flight.

## DB Type:

SQLite Database

## DB Tables

* Users Table: This table holds user account information.
  + (Username, Password) – Composite Primary Key
  + FirstName
  + LastName
  + Drone
* Drones Table: This table holds information for the drone used by the user.
  + DroneName – Primary Key, FK Drone (Users Table)
  + Flights
  + SDK
* FlightLogs Table: This table holds information on flights taken with the drone.
  + LogID: Primary Key, FK Flights (Drone table)
  + FlightDate
  + StartTime
  + EndTime
  + StartBattery
  + EndBattery
  + MotorTime

## DB Relationship Diagrams

Diagram

Description automatically generated

# File Storage Documentation

## File Storage Description:

While a database is good for storing most types of data it is not adequate nor efficient for storing and retrieving image and video files. To overcome this issue, a collection of directories are created for and used by the system when images or videos of any type are needing stored/accessed. These directories are initially empty until the system is used to populate them.

## File Storage Directories:

* Faces Directory – This directory stores all image files for face recognition. These images should be a frontal shot of a person’s face with the file name being the persons first name and last name separated by a space.
* Objects Directory – This directory stores all image files for object recognition. These images should be a frontal shot of an object with the file name being the objects name. These images should be the best possible 2D representation of the object.
* Images Directory – This directory stores all image files taken from the drone's camera. The filename of these images will be set by default when captured by the drone.
* Videos Directory – This directory stores all video files taken from the drone's camera. The filename of these videos will be set by default when captured by the drone.
* Maps Directory – This directory stores all map files created with the mapping controller. The filename of these maps will be set by default when the save map button is pressed and the map is saved to the directory.

# Forms and Screens

**Login Form**

**Graphical user interface, application, website

Description automatically generated**

* A user first launches the managementUI and is shown the root window with the login form centered on it.
* After entering a valid username and password (exists as a record in the databases Users table) the login button brings the user to the home screen when selected and logs the user as the current user.
* The New user button can be used to display the add user form in the root window.
* If a user enters an invalid username or password, an error message is displayed in red text at the bottom of this form stating as such.

**Add User Form**

**Graphical user interface, text, application

Description automatically generated**

* This form allows a user to create a record in the databases Users table to be used as login to the managementUI.
* All fields are required for a record to be successfully added and include a unique username and password not already in the Users table, and a user’s first name and last name.
* Once all required fields have been completed the submit button can be used to query the Users table and add the record to the databases with the entered information. In doing this, the Drones table is also queried and a record is inserted with the drones name inserted as “Tello EDU” and its sdk as “3.0”. These are hardcoded and added by default when a record is added to the users table.
* The cancel button returns a user to the login form.

**Home Screen**

**Graphical user interface, text, application, chat or text message

Description automatically generated**

* Upon successful login the home screen is displayed.
* The gray box displays the current users first name and last name as well as the drones name and sdk. This data is not retrieved from the database but is held when a user logs in to the system and a class instance with these attributes is created.
* The home screens navigation bar contains button to access other program functionality.
  + The “Drone Controller” button launches the drone controller program as an executable file and closes the managementUI program and its associated processes.
  + The “Mapping Controller” button launches the mapping controller program as an executable file and closes the managementUI program and its associated processes.
  + The “Logs” button displays the logs frame in the root window.
  + The “Faces” button displays the faces frame in the root window.
  + The “Objects” button displays the objects frame in the root window.
  + The “Logout” button sets the active user to none and displays the login frame in the root window.

**Logs Frame**

Graphical user interface, table

Description automatically generated

* This frame uses a scrollable panel to display the contents of the databases flight logs table by querying its records and displaying them in a table format.
* The close button returns the user to the home screen.

**Faces Frame**

Graphical user interface, application, website

Description automatically generated

* The open button displays the open file dialogue box in the Faces directory and allows the user to either double click one of these image files to display it or select it and then press the open button to do so.
  + When an image is opened in this way the frame expands to display the image in the area above the buttons and the file name for the image just below the image.
* The new button displays the add face frame in the root window
* The close button returns the user to the home screen.

**Add Face Form**

Graphical user interface, application, website

Description automatically generated

* This frame is used to add an image of a person’s face to the faces directory by specifying the image to add through use of the search button and entering the first and last name of the person the face belongs to in the entry boxes.
* The search button displays an open file dialogue box in the users local disk allowing them to either double click an image files to copy its file path or select it and then press the open button to do so. When done in this way the images file path is displayed next to the “Image:” label.
* If an image to add has been specified and both the first and last name entry boxes are completed then the add button can be used to rename the file to be the first and last name entered and save this newly named file in the faces directory.
* The close button displays the faces frame in the root window.

**Objects Frame**

Graphical user interface, application

Description automatically generated

* The open button displays the open file dialogue box in the Objects directory and allows the user to either double click one of these image files to display it or select it and then press the open button to do so.
* When an image is opened in this way the frame expands to display the image in the area above the buttons and the file name for the image just below the image.
* The new button displays the add object frame in the root window
* The close button returns the user to the home screen.

**Add Object Frame**

Graphical user interface, application, website

Description automatically generated

* This frame is used to add an image of an object to the objects directory by specifying the image to add through use of the search button and entering the objects name in the entry box.
* The search button displays an open file dialogue box in the users local disk allowing them to either double click an image files to copy its file path or select it and then press the open button to do so. When done in this way the images file path is displayed next to the “Image:” label.
* If an image to add has been specified and the object name entry box is completed then the add button can be used to rename the file to be the name entered and save this newly named file in the objects directory.
* The close button displays the objects frame in the root window.

**Drone Controller Screen**



* This is the screen for the drone controller program which is accessible through clicking the “Drone Controller” button on the home screen of the ManagementUI program.
* A user must first turn on the drone and connect to it through their computers Wi-Fi prior to launching the program.
* A face image must also be included in the faces directory prior to launching the program otherwise an error will occur and this screen will never be displayed.
* Once displayed, the drones live camera feed is displayed at the center of the screen. There is also an image on the left side of the screen displaying instructions on how to control the drone through key presses. The image is static and the user can not interact with it.
* At this point the facial and object recognition buttons can be used to detect known faces and objects in the drones feed and display the known face/objects filename in the labels located above each button.
* Recording, image capturing, and setting the drones flight speed can also be done prior to takeoff
  + Pressing the record button causes the program to record the drones video stream for the remainder of the programs life.
  + Pressing the image capture button saves the drones current frame as an image file.
  + Adjusting the scale bar changes the value the drones flight speed will be set to when the reset speed button is pressed and similarly the reset speed button sets the drones flight speed to the value indicated by the set speed scale bar.
* Pressing the takeoff/land button for the first time sends the takeoff command to the drone, sets its flying status to true, and allows the user to send flight commands to the drone through key presses on their computer.
  + When a key is pressed its associated flight command is send to the drone
  + When the key is released a flight command is sent to the drone to stop flying and remain hovering at its current position.
  + When this button is pressed a second time, the land command is sent to the drone and its flying status is set to false.
* Upon launching the program the current date, time, and drone battery status are captured as program variables to be utilized once the use clicks the end flight button. When the end flight button is pressed, the time the user pressed the button, the drones current battery level, and the time the drones motors ran during the programs life are captured as program variables. After this the these are used with the variables captured on program launch to query the databases flight logs table and insert the values these variables hold as a record in this table.
  + - After inserting this record the database connection is closed and all processes under this program are terminated.
    - The managementUI program must be launched again to access any captured data or utilize program functionality further.

**Mapping Controller Screen**

Graphical user interface

Description automatically generated with medium confidence

* This is the screen for the mapping controller program which is accessible through clicking the “Mapping Controller” button on the home screen of the ManagementUI program.
* A user must first turn on the drone and connect to it through their computers Wi-Fi prior to launching the program.
* Upon launching the program, a turtle canvas is displayed at the center of the screen with the turtles current position (zero point) marked next to the arrow on this canvas.
* An image is shown on the left side of the screen instructing users on the computer keys used for controlling the drone and how to use the other buttons available.
* The takeoff button send the takeoff command to the drone and the land button send the land command to the drone.
* Adjusting the scale bar changes the distance the drone will move by default for each key press. This is set when the reset distance button is pressed. Similarly the reset distance button sets the drones default movement value to that indicated by the scale bar when the button is pressed.
* When pressed, the write point button draws a point with the drones current position next to the turtles on the canvas.
* When pressed, the save map button saves the map made on the turtle canvas to the maps directory as a .png file.
* The program ends by clicking the “X” button in the windows top right corner. When this occurs all processes are ended and the window is closed. From here a user must re-launch the managementUI program to see any data captured or utilize any of the different programs functionality again.

# Use Cases

1. The systems mapping controller can be used to map a path that is either unsafe or is too small for a person to travel. This aids in search and rescue operations if a path to save an individual is unknown by a person. The ability to map the path prior to sending a live person down it can prevent them from discovering that the path is untraversable after they have already entered the hazardous area.
2. The drone controllers facial recognition capabilities can be used by a security guard for a business charged with monitoring the properties security cameras and investigating any individuals seen on them that may not be permitted to be on the property. If the guard sees an individual in the security feed in which they do not immediately recognize they can avoid putting themselves in harm’s way by flying the drone to the unknown person to get a closer look and identify whether or not they are registered in the system as a person that is permitted to be on the property.
3. The drone controllers facial recognition capabilities can be utilized in search and rescue operations to identify individuals located amongst rubble that is untraversable by a human. The small drone being capable of flying through small spaces and unpredictable terrain can identify these individuals by their facial features, which is something that cannot be accomplished through photogrammetry and other such techniques. In this way a drone can be utilized for capturing more than just aerial data alone in the event of a natural disaster that requires search and rescue operations.
4. The drone controllers object recognition capabilities can be utilized to recognize specialized directional signs so that a user can determine which direction they need to move the drone during a specific mission.

# Known Issues

1. The logs frame in the managementUI displays all of the records from the FlightLogs table and not just those for the particular user logged into the system. The reason for this is that when launching the drone controller the name of the drone for that users record in the database is not passed as a parameter so there is no way to insert this value into the database when adding the other parameters for a flight log record. Since the DroneController is launched as an executable, the easiest way to get the drones name is to have a input box display on launching the drone controller so that the user can input the name of their drone. From this the name can be captured as a variable and added to the database when adding the flight log record. Another approach would be to not launch the drone controller as an executable but launch it in the same mainloop that the managementUI runs in.
2. Face encodings are done every time the management system or computer vision drone controller are launched. This makes for a slight delay in start up time for encoding only a few faces and could cause problems if a large number of faces are in the directory. This issue should be resolved by encoding a face image each time it is added in the management system and storing the encoding in a database table or file with the other added face encodings.
3. GUI for both drone controller and mapping controller do not hold layout on computers with a smaller screen size. For the drone controller this is less noticeable but for the mapping controller the ability to save the map as an image file is effected as the map area is not in the same location on the screen as is expected by the algorithm used to save the image. For the drone controller, this only effects the face/object recognition labels as they are slightly offset from their intended position.
4. The mapping controller is not effective for creating accurate flight path maps. This is a hardware problem as the Tello EDU drone is not capable of maintaining accurate positioning and as a result can flies inaccurately in comparison to the precise movements the python turtle makes. These features can be implemented in a similar way for other mini drones with more advanced hardware so this issue can be overcome by utilizing the program with a compatible drone that has a more advanced hardware regarding its positioning system.
5. The mapping controller uses key presses to move both the drone and the controller at the same time. This requires the user to wait approximately 5 seconds in between using key presses for moving the drone otherwise the turtle will move even if the drone hasn’t resulting in inaccurate point data. To overcome this issue, the mapping control could be enhanced to include a separate thread for controlling the python turtles movements separate from the drones. In this way, the user would not need to wait between moving the drone with different key presses as the thread that moves the turtle would only start after the drones thread has finished sending the move command for the set distance and receiving a response from the drone that it has moved.
6. Currently the drone controller does not utilize a thread manager for synchronization but instead utilizes threads in a way such that the threads run processes that are unlikely to or rarely interfere with one another. To implement this, I would utilize the tello.py module used for the mapping controller and implement the drone object for the drone controller as the tello class contained in this module. In this way the class already provides a means to manage the threads as is seen in the mapping controller and would be implemented as such given this.
7. There is currently no way to specify the sdk for a record in the Drones table as the record is created when the user creates an account and specifies the drone name in the add user form. The sdk is then hardcoded to be set to 3.0 as this is the sdk I used to implement the drones programming. This is the best way to insert the sdk as the user does not interact with it anyway since it is used in the programming that makes up the system. Given more Tello drones are released in the future with newer sdk versions this project would need to be restructured so that the functionality is implemented in a drone specific way with regards to which drones are capable of using which sdk and their contained functions.

# Future Enhancements

1. The drone controllers object recognition capabilities can be utilized to recognize specialized directional signs so that the user can determine which direction they need to move the drone on a given mission. This can serve as a future enhancement in which the drone uses these signs to autonomously fly a planned route. To implement this, descriptors and names for each unique directional sign would would be used so that the drone can identify the sign during flight from the descriptors and interpret the proper movement command using the name for each sign.
2. The system mapping controller can be utilized to map a path or an area that the drone needs to monitor and then have the drone fly this path autonomously using the map as a reference. To implement this, the points written when making the map can be used to by the drone to interpret which direction it needs to move and how far. For example, a map with a point for the starting position (0, 0) and another point at (0, 1) would be interpreted by the drone as the move forward command with the distance to move forward being one meter.
3. Computer vision functionality can be implemented dynamically instead of statically with a button press triggering their execution as it is implemented currently. To do this, the video stream would be displayed in an OpenCV window instead of how it is currently displayed as a Tkinter compatible photo image. This allows for computer vision to be performed dynamically as OpenCV handles all of the image processing for displaying and updating the video stream. This would greatly enhance the performance of the drone controller in addition to allowing for dynamic computer vision functionality and the ability to implement more if desired without impacting the systems processing. In addition to these benefits, the video stream could also be implemented for the mapping controller as the video stream window could be easily minimized and maximized for clear viewing. This would also prevent any issues from arising with the systems current threading procedures as would occur if displaying the stream as a Tkinter compatible photo image.

# Other Resources

1. “How To Organize Multi-frame TKinter Application With MVC Pattern”
   1. Nazmul Ahsan
   2. <https://nazmul-ahsan.medium.com/how-to-organize-multi-frame-tkinter-application-with-mvc-pattern-79247efbb02b>
   3. This resource was used/references to structure the managementUI.py program in an MVC design pattern.
      1. Using the code from this reference I reworked the modules to fit my program’s needs. For this I completely reused some of the modules/methods from the resource (changing different names and such to fit my projects purpose) and then used the rest as a reference to structure and create my own modules that were different from the ones discussed in this resource.
      2. The base.py module from the models package is the only code from this reference that is 100% unchanged. Some methods and class definitions from different modules also came directly from the resource but in all of these cases the data structures are to simple to implement differently or variable names have been changed.
      3. All parts from this resource are cited with those code sections I adapted from the resource stating as such.
2. “Feature Detection and Matching + Image Classifier Project | OPENCV PYTHON”
   1. Murtaza's Workshop - Robotics and AI
   2. <https://www.youtube.com/watch?v=nnH55-zD38I>
   3. The recognize object method used in DroneController.py was adapted from this reference and as was the object models loadLists method.
   4. The objects models findDes method came directly from this resource
   5. All parts from this resource are cited with those code sections I adapted from the resource stating as such.
3. “Drone Programming With Python Course | 3 Hours | Including x4 Projects | Computer Vision”
   1. Murtaza's Workshop - Robotics and AI
   2. <https://www.youtube.com/watch?v=LmEcyQnfpDA&t=6736s>
   3. The DroneController.py programs count faces method was adapted from this resources detect face function and is cited as such in the program code.
4. “FACE RECOGNITION + ATTENDANCE PROJECT | OpenCV Python | Computer Vision”
   1. Murtaza's Workshop - Robotics and AI
   2. <https://www.youtube.com/watch?v=sz25xxF_AVE&t=957s>
   3. The recognize face method used in DroneController.py was adapted from this reference and as was the face models loadLists method.
   4. The faces models find encodings method came directly from this resource.
   5. All parts from this resource are cited with those code sections I adapted from the resource stating as such.
5. flydo - commands.py (github)
   1. Charles Yuan
   2. <https://github.com/Chubbyman2/flydo/blob/main/commands.py>
   3. The flight commands module was adapted from this resource and is cited as such in the program code.
6. flydo - app.py (github)
   1. Charles Yuan
   2. <https://github.com/Chubbyman2/flydo/blob/main/commands.py>
   3. The droneController.py GUI class definition, video stream method, and run app method were adapted from this resource.
   4. The takeoff/land method was taken directly from this resource.
   5. All parts from this resource are cited with those code sections I adapted from the resource stating as such.
7. “Python, Manual Control”
   1. One-Off Coder
   2. <https://tello.oneoffcoder.com/python-manual-control.html>
   3. The tello.py module was taken from this resources tello.py module and adapted to include only the commands needed for this projects functionality. This is cited as such in the program code.
   4. The mappingController.py module also was adapted from this resources user interface module with methods directly from the resource and those sections adapted from the resource cited as such in the program code.
8. “How can I convert canvas content to an image?” - (stack overflow thread)
   1. B. Jenkins
   2. <https://stackoverflow.com/questions/9886274/how-can-i-convert-canvas-content-to-an-image>
   3. The mappingController.py modules save map method was adapted from this resource and is cited as such in the program code.