# Drone Tracker

***Release 1.0.3***

**Hayley Eckert, Jonathan Westerfield, Donald Elrod, Ismael Rodrig**

**Nov 18, 2019**

# CONTENTS:

CHAPTER

ONE

# INTRO

Version 1.0.3

Written by: Hayley Eckert, Jonathan Westerfield, Donald Elrod, and Ismael Rodriguez

This application is the controller for the entire project. It interfaces with the cell phones and controls them in order to record footage of a drone flight. It will then open 2 OpenCV processes in order to analyze the footage from the cell phones.

This flight application relies on OpenCV and MatPlotLib.

**The main sections of the application are as follows:**

- **Program_Controller:** Starts up the entire Drone Tracker program the user interacts with
- **PhoneController:** Sends and receives network signals from the phone over TCP to control the phone.
- **OpenCV:** Takes the file of the video transferred from the phone and extracts the coordinates of the drone from the footage

CHAPTER

TWO

# PROGRAM CONTROLLER

**class** src.Controllers.Program_Controller.**Controller**(*phoneControl: Controllers.PhoneController.PhoneControl*)

Controller class for the application. Changes between application views based on user input. Run this file in order to begin the application.

**cleanup**() → None

Goes through our file structure and deletes all files (except anything in the Flight folder). This is for when we are done with the program and want to delete the raw footage from the phone, the output points from the opencv processing and any other files that were used during the program execution. Should also be called before any time the user wants to fly.

**Returns** None

**get_all_files**(*folderPath: str*) → list

Will get all of the file names in a folder. This will be used after the video footage is transferred from the phone to the laptop. We need to grab the 2 videos filenames in the folder in order to send them to the opencv analysis to have the coordinates extracted. This will also be used when we need to splice together all of the coordinates output by the OpenCVController json files.

**Parameters** **folderPath** – The directory that we need to get all of the files from.

**Returns** A list of all of the file names in that directory

**get_flight_info**(*pilotName: str*, *instructorName: str*, *flightInstructions: str*) → None

Saves the pilot name, instructor name, and flight instructions once confirmed by the user.

**Parameters**

- **pilotName** – String containing the pilot name
- **instructorName** – String containing the instructor name
- **flightInstructions** – String containing the flight instructions

**Returns** None

**get_in_order**(*files: list*) → list

Will return the file names in order. This means that phone-1 will be first in the list and phone-2 will be second. Need this to ensure that a specific phone's coordinates are being used for the Z, X axis and the other is used for Z, Y

**Parameters** **files** – The list of files we get from the get_all_files() function

**Returns** The list of file names starting with phone-1 first and phone-2 second

**import_flight**(*flightPath: str*) → None

Reads in a .flight file and displays the report view for it.

**Parameters** **flightPath** – String with file path of chosen file to import.

**Returns** None

**setupFileStructure**() → None
Sets up the folders that need to exist before we can transfer footage and analyze. The file structure should be: drone-tracker > FTP, opencv-output, Flights. This makes it easier to keep track of the files that we are working with during the application lifecycle.

**Returns** None

**show_home**() → None
Loads the home startup screen for the user.

**Returns** None

**show_loading_window**() → None
Loads the loading screen for the user.

**Returns** None

**show_report_window**(*previousFlight: str*, *usingPreviousFlight: bool*, *flightData: dict*) → None
Loads the report screen for the user.

**Parameters**

- **previousFlight** – String containing path to flight data. Should be .flight file if usingPreviousFlight is true, or empty if usingPreviousFlight is false.
- **usingPreviousFlight** – Boolean representing if the report view is for an existing .flight file or a new analysis.
- **flightData** – Dictionary containing the flight data. Should be populated with only coordinates if usingPreviousFlight is false, and empty if usingPreviousFlight is true.

**Returns** None

**show_tracking_window**() → None
Loads the tracking screen for the user.

**Returns** None

**show_verify_screen**() → None
Loads the verify setup screen for the user.

**Returns** None

**start_analysis**() → None
Spawns the sub processes that will analyze the footage of the drone footage. We will need to have the files before hand so we can pass them into each OpenCVController process.

**Returns** None

**transfer_complete**(*flightData: dict*) → None
Calls the report view using the flight data dictionary.

**Parameters** **flightData** – Dictionary of flight data

**Returns** none.

**transfer_footage**(*phoneControl: Controllers.PhoneController.PhoneControl*) → None
Transfers footage and calls DroneController to analyze the footage.

**Parameters** **phoneControl** – Phone Controller object for the active phone connection.

**Returns** none

**updateFlightStatus**() → None
Sets the status of the system verification test.

**Returns** none

**wait_for_analysis**() → None
Waits for the analysis of the footage to complete. Essentially is just a loop that checks to see if the file locks (*.lock) for our files are still in that directory. If they are, we wait, otherwise, we will exit the loop. From there, we need to get the output from those processes and splice the points together into a single 3D coordinate list.

**Returns** None

src.Controllers.Program_Controller.**close_conn**(*phoneControl: Controllers.PhoneController.PhoneControl*) → None
Closes the connection from the laptop to the phones.

**Parameters** **phoneControl** – PhoneControl object containing the active connection.

**Returns** None.

src.Controllers.Program_Controller.**createPhoneConnection**(*portNo*) → Controllers.PhoneController.PhoneControl
Creates a PhoneControl object used to commmunicate with the phones.

**Parameters** **portNo** – Port number we are going to be listening for signals from the phone over.

**Returns** PhoneControl object.

src.Controllers.Program_Controller.**main**() → None
Begins the main application.

**Returns** None

CHAPTER

# THREE

# OPENCV

**class** src.Controllers.OpenCVThreadedController.**DroneTracker**(*videoFile*)

This class is intended to track sUASs in video recorded from the smartphone app All containing code to track the drone and output the coordinates extracted from the recorded video is contained within this class, and this file is intended to be run as a separate process so that both of the recordings can be processed in parallel (don't try this on Windows though).

**is_light_on**(*frame*) → bool

Takes in a video frame and returns the frame at which the light turns on.

**Parameters** **frame** – A single video frame to see if the light is on.

**Returns** True if the light is on, false otherwise.

**read_video**() → None

This function is to be threaded, and its purpose is to read in the video file all at once to improve performance.

**rescale_frame**(*frame*, *percent=50*)

Resizes a frame as a percentage of the original frame size

**Parameters**

- **frame** – the frame to be resized
- **percent** – the percent value the frame needs to be rescaled to

**resize_bbox**(*bbox: tuple*, *factor=2*) → tuple

Resizes the bounding box for translating it to the full size video. In order to be able to see enough of the footage on screen to draw the box around the drone, the video frame must be resized, so the drawn bounding box must be translated back into the coordinate system the full size video uses. For example, if the 4k footage is shrunk by 50% (to 1080p), the scale factor here must be 2 so the coordinates chosen in the 1080p frame will match up with the actual drone coordinates in the 4k frame.

**Parameters**

- **bbox** – bounding box of selected drone, which is (x, y, box_width, box_height)
- **factor** – the factor by which to scale the bounding box

**Returns** tuple

**trackDrone**() → list

Function that contains all code to track the drone, and is to be run as a thread. Will run much slower if 2 processes running this method are started and run on different videos at the same time.

**Returns** List of tuples of the extracted coordinates of the footage, in the format [(time, x_coord, y_coord, z_coord)].

**exception** src.Controllers.OpenCVThreadedController.**VideoCorruptedException**(*message: str*)

This error is raised if the video being read is corrupted, or if the frames cannot be successfully extracted from the video files

**exception** src.Controllers.OpenCVThreadedController.**VideoNotPresentException**(*message: str*)

This error is raised when the video for processing is not there, or if an incorrect path is given

src.Controllers.OpenCVThreadedController.**get_phone_id**(*filename: str*) → str

Gets the phone Id from the end of the file name so we can keep track of the json and lock files.

**Parameters** **filename** – The file name of the video file. Should have "phone-#.mp4" file names.

**Returns** The ID of the phone from the file name

src.Controllers.OpenCVThreadedController.**main**(*filename: str*) → None

Will take the filename passed in and analyze the footage. All coordinates of the drone in the footage will be output to a json file.

**Returns** None

src.Controllers.OpenCVThreadedController.**merge_data_points**(*phone1Points: list*, *phone2Points: list*) → dict

Takes the points outputted by the opencv analysis and merges the points together to create the 3D coordinates needed to output the visual flight path.

**Parameters**

- **phone1Points** – The opencv datapoints created from the main method of this class for the first phone
- **phone2Points** – The opencv datapoints created from the main method of this class for the second phone

**Returns** List of tuples of coordinates and time values that represent the flight path of the drone in the format [(time, x_coord, y_coord, z_coord)]

CHAPTER

FOUR

# EXCEPTIONS

**exception** src.Controllers.Exceptions.**FailedDisconnectException**(*message: str*)
This error is for telling us that something went wrong when we tried to disconnect from the RPI.

**exception** src.Controllers.Exceptions.**FailedRPIFlashException**(*message: str*)
This error is for letting us know that the RPI did not flash the light.

**exception** src.Controllers.Exceptions.**PhonesNotSyncedException**(*message: str*)
This exception is for when the user tries to perform an operation with the phones without actually syncing the phones first.

**exception** src.Controllers.Exceptions.**RPINotConnectedException**(*message: str*)
This class is for letting us know that we had an issue connecting to the raspberry pi.

**exception** src.Controllers.Exceptions.**RecordingNotStartedException**(*message: str*)
This exception class is for alerting the user that the recording has not started and they are trying to access a function that requires the phone cameras to be rolling.

**exception** src.Controllers.Exceptions.**TransferNotStartedException**(*message: str*)
This exception class is for alerting the user that the file transfer process has not been started and any actions that depend on it will fail.

**exception** src.Controllers.Exceptions.**VideoCorruptedException**(*message: str*)
This error is raised if the video being read is corrupted, or if the frames cannot be successfully extracted from the video files

**exception** src.Controllers.Exceptions.**VideoNotPresentException**(*message: str*)
This error is raised when the video for processing is not there, or if an incorrect path is given

CHAPTER

FIVE

# PHONECONTROLLER

**class** src.Controllers.PhoneController.**PhoneControl**(*portNum: int*)

This class is for communicating with and controlling the phones out in the field. It uses simple TCP connections with each phone in order to control them.

**closeConn**() → None

Closes all of the connections and the socket.

**Returns** None

**isTransferring**() → bool

A flag for us to access to see if the system is still waiting for the video to finish the file transfer of the videos it recorded.

**Returns** True if the video has finished transferring, false otherwise.

**setupSocket**() → None

Creates the socket that we will use to listen for incoming connections.

**Returns** None

**startFileTransfer**(*filepath: str*) → None

Will send a signal to the phone that tells it to transfer the video files it recorded over to the laptop by opening an FTP connection.

**Parameters** **filepath** – The file path on our laptop that the phones will need to send their videos to over FTP.

**Returns** None

**startRecording**() → None

Call this in order to send a signal to the phones that they need to start recording.

**Returns** None

**stopRecording**() → None

Call this in order to send a signal to the phones that they need to stop tracking. Sends both phones a stop signal and the name of the file path that they will need to send their videos to over FTP.

**Returns** None

**sync**() → None

This function will wait until both phones have been connected to this app.

**Returns** None

**synced**() → bool

The getter function for seeing if the phones synced or not.

**Returns** True if they have been synced, false otherwise

**threadSendSignal**(*conn: socket.socket*, *signal: str*, *sigMessage: str*, *sigAck: str*) → None
This function takes a signal and the expected output so that we don't have to rewrite the same code for every action we have with the phones.

**Parameters**

- **conn** – A socket connection to a phone that has already been opened.
- **signal** – The Signal we want to send to the phone. Valid options are: START, STOP, and START_FTP
- **sigMessage** – A message that we want to send alongside the signal for the phone to use.
- **sigAck** – The Signal we expect to get back from the phone in response to our signal. Valid options are: START_ACKNOWLEDGE, STOP_ACKNOWLEDGE, START_FTP_ACKNOWLEDGE.

**Returns** None

**threadWaitForFileTransfer**(*conn: socket.socket*) → None
This is a thread for waiting for the signal from the phone that the file transfer to the filepath specified in the startFileTransfer() function arguments.

**Parameters** **conn** – A socket connection to a phone that has already been opened.

**Returns** None

**waitForFileTransfer**() → None
Spawns threads that will wait for both phones to send a signal saying that the file transfer of the videos is complete.

**Returns** None

CHAPTER

SIX

# EXPORT FLIGHT

src.Export.ExportFile.**export_data**(*pilotName*, *instructorName*, *flightDate*, *flightLength*, *flightInstructions*, *xCoordinates*, *yCoordinates*, *zCoordinates*, *velocityValues*, *outPath*) → None

Exports the flight data to a JSON file stored with a ".flight" extension.

**Parameters**

- **pilotName** – String containing the pilot name
- **instructorName** – String containing the instructor name
- **flightDate** – String containing the flight date
- **flightLength** – String containing the flight length
- **flightInstructions** – String containing the flight instructions
- **xCoordinates** – Array of x coordinates
- **yCoordinates** – Array of y coordinates
- **zCoordinates** – Array of z coordinates
- **velocityValues** – Array of velocity values
- **outPath** – String containing the path to save the file. Should end in ".flight".

**Returns** none

CHAPTER

# SEVEN

# IMPORT FLIGHT

src.Export.ImportFile.**importData**(*inPath*) → dict

Imports the flight data from a JSON file stored with a ".flight" extension.

**Parameters** **inPath** – String containing the pilot name.

**Returns** Flight dictionary

CHAPTER

# EIGHT

# LOADING SCREEN

**class** src.Views.View_LoadingScreen.**LoadingWindow**

The view for the loading page that is shown when the user presses the "Stop Tracking" button on the tracking window page.

**Variables** **__btnHome** – The class property for the 'Return to Home' button.

**property BtnHome**

The home for the view. Is used to return to home screen.

**Returns** None

**property BtnTestReport**

The test report for the view. Is used to switch to the test report screen.

**Returns** The reference to the test report button.

**property LblStatus**

Getter property for the timer label. We need to attach a QTimer to it so it can count the time the application has been tracking the drone.

**Returns** The timer label

**property del_BtnHome**

The home for the view. Is used to return to home screen.

**Returns** None

**property del_BtnTestReport**

The test report for the view. Is used to switch to the test report screen.

**Returns** The reference to the test report button.

**property del_LblStatus**

Getter property for the timer label. We need to attach a QTimer to it so it can count the time the application has been tracking the drone.

**Returns** The timer label

**initView**() → None

Sets up the view and lays out all of the components.

**Returns** None

**returnHome**() → None

Sends a signal to the main controller that the Cancel and Return to Home button was pushed.

**Returns** none

**setSubtitle**() → PyQt5.QtWidgets.QLabel

Sets up the subtitle label.

**Returns** The subtitle label

**setTitle**() → PyQt5.QtWidgets.QLabel
Sets up the title with the application title on top and the name of the screen just below it.

**Returns** Layout with the application title and screen title labels

**property set_BtnHome**
The home for the view. Is used to return to home screen.

**Returns** None

**property set_BtnTestReport**
The test report for the view. Is used to switch to the test report screen.

**Returns** The reference to the test report button.

**property set_LblStatus**
Getter property for the timer label. We need to attach a QTimer to it so it can count the time the application has been tracking the drone.

**Returns** The timer label

**setupLoadingIcon**() → PyQt5.QtWidgets.QLabel
Used for configuring the loading icon on the loading screen. Loading icon is a gif, so QMovie is used to animate the icon.

**Returns** The icon containing the loading label.

**signalTestReport**() → None
Sends a signal to the main controller that the Test Report button was pushed. NOTE: ONLY USED FOR TESTING PURPOSES

**Returns** none

**signalTransferFootage**() → None
Sends a signal to the main controller that the button to transfer footage was pressed.

**Returns** none

CHAPTER

NINE

# REPORT SCREEN

**class** src.Views.View_ReportScreen.**ReportWindow**(*pilotName: str*, *instructorName: str*, *flightInstructions: str*, *previousFlight: str*, *usingPreviousFlight: bool*, *flightData: dict*)

The view for the report page that is shown when the user opens the application.

**Variables**

- **__btnExport** – The class property for the 'Export Results' button.
- **__btnFlyAgain** – The class property for the 'Fly Again' button.
- **__btnHome** – The class property for the 'Return to Home' button.
- **__btnViewGraphVelocity** – The class property for the 'View Flight Path' button.
- **__btnViewGraphNoVelocity** – The class property for the 'View Flight Path with Velocity Changes' button.
- **__btnViewInstructions** – The class property for the 'View Flight Instructions' button.

**property BtnExport**

The export button for the view.

**Returns** None

**property BtnFlyAgain**

The fly again button for the view.

**Returns** None

**property BtnHome**

The home for the view. Is used to return to home screen.

**Returns** None

**property BtnViewGraphNoVelocity**

The home for the view graph without velocity button.

**Returns** None

**property BtnViewGraphVelocity**

The home for the view graph with velocity button.

**Returns** None

**property LblFlightDate**

Getter for the flight date label.

**Returns** Reference to the flight date label.

**property LblFlightInstructions**
Getter for the flight instructions label.

**Returns** Reference to the flight length label.

**property LblFlightLength**
Getter for the flight length label.

**Returns** Reference to the flight length label.

**property LblInstructor**
Getter for the instructor label.

**Returns** Reference to the instructor label.

**property LblPilot**
Getter for the Pilot label so we can set who the pilot is for the flight in child class.

**Returns** Reference to the pilot label.

**analyzeFlight**(*flightDict: dict*) → dict
Analyzes the flight data to extract coordinates, velocity values, and statistics.

**Parameters** **flightDict** – Dictionary of flight data, with only coordinates populated.

**Returns** Updated dictionary, with legal points and flight statistics included.

**createStatisticsTable**() → PyQt5.QtWidgets.QTableWidget
Creates a table containing flight statistics.

**Returns** QTableWidget containing flight statistics.

**property del_BtnExport**
The export button for the view.

**Returns** None

**property del_BtnFlyAgain**
The fly again button for the view.

**Returns** None

**property del_BtnHome**
The home for the view. Is used to return to home screen.

**Returns** None

**property del_BtnViewGraphNoVelocity**
The home for the view graph without velocity button.

**Returns** None

**property del_BtnViewGraphVelocity**
The home for the view graph with velocity button.

**Returns** None

**property del_LblFlightDate**
Getter for the flight date label.

**Returns** Reference to the flight date label.

**property del_LblFlightInstructions**
Getter for the flight instructions label.

**Returns** Reference to the flight length label.

**property del_LblFlightLength**
Getter for the flight length label.

**Returns** Reference to the flight length label.

**property del_LblInstructor**
Getter for the instructor label.

**Returns** Reference to the instructor label.

**property del_LblPilot**
Getter for the Pilot label so we can set who the pilot is for the flight in child class.

**Returns** Reference to the pilot label.

**handleEndSliderValueChange**(*value*) → None
Listener that will updated the stop time label when the user moves the stop time slider.

**Parameters** **value** – The value that the stop slider has been moved to horizontally.

**Returns** None

**handleStartSliderValueChange**(*value*) → None
Listener that will updated the start time label when the user moves the start time slider.

**Parameters** **value** – The value that the start slider has been moved to horizontally.

**Returns** None

**initView**(*pilotName: str*, *instructorName: str*, *flightInstructions: str*, *previousFlight: str*, *usingPreviousFlight: bool*, *flightData: dict*) → None
Sets up the view and lays out all of the components.

**Parameters**

- **pilotName** – String containing pilot name
- **instructorName** – String containing instructor name
- **flightInstructions** – String containing flight instructions.
- **previousFlight** – String containing path to flight data. Should be .flight file if usingPreviousFlight is true, or blank if usingPreviousFlight is false.
- **usingPreviousFlight** – Boolean denoting if the report should be populated from the same file or a different one.
- **flightDict** – Dictionary containing flight data. Should be empty if usingPreviousFlight is true.

**Returns** None

**setButtonLayout**() → PyQt5.QtWidgets.QHBoxLayout
Lays out the 'Export Results', 'Fly Again' and 'Import Previous Flight' buttons into a horizontal layout to be put on screen.

**Returns** The horizontal layout containing the 3 buttons

**setSubTitle**(*text*) → PyQt5.QtWidgets.QLabel
Sets up a subtitle label for the window

**Parameters** **text** – String as name for label

**Returns** Subtitle label

**property set_BtnExport**
The export button for the view.

**Returns** None

**property set_BtnFlyAgain**
The fly again button for the view.

**Returns** None

**property set_BtnHome**
The home for the view. Is used to return to home screen.

**Returns** None

**property set_BtnViewGraphNoVelocity**
The home for the view graph without velocity button.

**Returns** None

**property set_BtnViewGraphVelocity**
The home for the view graph with velocity button.

**Returns** None

**property set_LblFlightDate**
Getter for the flight date label.

**Returns** Reference to the flight date label.

**property set_LblFlightInstructions**
Getter for the flight instructions label.

**Returns** Reference to the flight length label.

**property set_LblFlightLength**
Getter for the flight length label.

**Returns** Reference to the flight length label.

**property set_LblInstructor**
Getter for the instructor label.

**Returns** Reference to the instructor label.

**property set_LblPilot**
Getter for the Pilot label so we can set who the pilot is for the flight in child class.

**Returns** Reference to the pilot label.

**setupFlightInfo**() → PyQt5.QtWidgets.QGridLayout
Sets up the flight info (pilot, instructor, date, length, and smoothness score) in a grid.

**Returns** Grid layout of the flight information

**setupGraph**(*flightData: dict*, *displayVelocity: bool*) → None
Sets up the 3d plot for viewing upon click of button. displayVelocity is a boolean denoting if the graph should display colored segments for velocity.

**Parameters**

- **flightData** – Dictionary of flight data
- **displayVelocity** – Bool denoting if velocity should be plotted or not.

**Returns** None

**setupSlider**() → PyQt5.QtWidgets.QVBoxLayout
Setups the slider that will be used to adjust the times of the flight path that will be displayed on the graph. This lets us control the beginning and ending bounds of the time of the flight that we want to view. We

have to use 2 sliders. One for the start time and one for the end. We originally wanted to make this such that both sliders were on top of each other to make it more intuitive, but we ran into a bug that prevented us from doing that.

**Returns** A horizontal layout containing both sliders and the labels displaying the time that they represent.

**setupTitle**() → PyQt5.QtWidgets.QVBoxLayout
Sets up the title with the application title on top and the name of the screen just below it.

**Returns** Layout with the application title and screen title labels

**showWindow**() → None
Takes all of the elements from the view and displays the window.

**Returns** None

**signalExportResults**() → None
Sends a signal to the main controller that the Export Results button was pushed.

**Returns** none

**signalReturnHome**() → None
Sends a signal to the main controller that the Return Home button was pushed.

**Returns** none

**signalStartTracking**() → None
Sends a signal to the main controller that the Fly Again button was pushed.

**Returns** none

CHAPTER

# TEN

# STARTUP SCREEN

**class** src.Views.View_StartupScreen.**StartupWindow**(*flightModeEnabled: bool*)

The view for the home Startup page that is shown when the user opens the application.

**Variables**

- **__btnVerifySetup** – The class property for the 'Verify Setup' button.
- **__btnStart** – The class property for the 'Start Tracking' button.
- **__btnImport** – The class property for the 'Import Previous Flight' button.

**property BtnImport**

Getter for the Import Previous Flight button. Is used to import past flight files. Use to attach functionality.

**Returns** None

**property BtnStart**

Getter for the startTracking button. Use to attach functionality.

**Returns** None

**property BtnVerifySetup**

Getter for the verifySetup button. Use to attach functionality.

**Returns** The reference to the verifySetup button

**property del_BtnImport**

Getter for the Import Previous Flight button. Is used to import past flight files. Use to attach functionality.

**Returns** None

**property del_BtnStart**

Getter for the startTracking button. Use to attach functionality.

**Returns** None

**property del_BtnVerifySetup**

Getter for the verifySetup button. Use to attach functionality.

**Returns** The reference to the verifySetup button

**initView**() → None

Sets up the view and lays out all of the components.

**Returns** None

**openFileNameDialog**() → None

Allows user to select a .flight file from a file dialog window.

**Returns** Path to selected file as a string.

**setButtonLayout**() → PyQt5.QtWidgets.QHBoxLayout
Lays out the 'Test Config', 'Start' and 'Import' buttons into a horizontal layout to be put on screen.

**Returns** The horizontal layout containing the 3 buttons

**setTeamMembers**() → PyQt5.QtWidgets.QVBoxLayout
Sets up the team members label for the window

**Returns** Team members label of the application

**setTitle**() → PyQt5.QtWidgets.QVBoxLayout
Sets up the title with the application title on top and the name of the screen just below it.

**Returns** Layout with the application title and screen title labels

**property set_BtnImport**
Getter for the Import Previous Flight button. Is used to import past flight files. Use to attach functionality.

**Returns** None

**property set_BtnStart**
Getter for the startTracking button. Use to attach functionality.

**Returns** None

**property set_BtnVerifySetup**
Getter for the verifySetup button. Use to attach functionality.

**Returns** The reference to the verifySetup button

**setupAMLogo**() → None
Used for configuring the display for the A&M logo on the startup screen.

**Returns** None

**setupPicture**() → None
Used for configuring the display for the logo on the startup screen.

**Returns** None

**signalImportFlight**() → None
Calls function to allow user to select a file for import. Sends a signal to the main controller that the Import Previous Flight button was pushed.

**Returns** None.

**signalStartTracking**() → None
Sends a signal to the main controller that the Start Tracking button was pushed.

**Returns** none

**signalVerifySetup**() → None
Sends a signal to the main controller that the Verify Setup button was pushed.

**Returns** none

CHAPTER

# ELEVEN

# TRACKING SCREEN

**class** src.Views.View_TrackingScreen.**TrackingWindow**(*phoneControl: Controllers.PhoneController.PhoneControl*)

The view for the tracking view page that is shown when the user presses the "Start Tracking" button on the home page. Allows the user to enter in flight information and begin tracking the drone.

**Variables**

- **__btnConfirm** – The class property for the 'Confirm' button.
- **__btnClear** – The class property for the 'Clear' button.
- **__btnStart** – The class property for the 'Start Tracking' button.
- **__btnStop** – The class property for the 'Stop Tracking' button.

**property BtnClear**

Getter for the Clear button so we can attach functionality to it later.

**Returns** Reference to the clear button

**property BtnConfirm**

Getter for the Confirm button so we can attach functionality to it later.

**Returns** Reference to the confirm button

**property BtnStart**

Getter for the Start button

**Returns** Reference to the start button

**property BtnStop**

Getter for the Stop button

**Returns** Reference to the stop button

**property LblInstructor**

Getter for the Instructor label so we can attach functionality to it later.

**Returns** The instructor label

**property LblPilot**

Getter for the Pilot label so we can attach functionality to it

**Returns** The pilot label

**property LblTimer**

Getter property for the timer label. We need to attach a QTimer to it so it can count the time the application has been tracking the drone.

**Returns** The timer label

**property TBInstructor**
Getter for the Instructor textbox so we can attach functionality to it later.

**Returns** The instructor textbox

**property TBPilot**
Getter for the Pilot Textbox so we can attach functionality to it

**Returns** The pilot textbox

**property TEInstructions**
Getter for the instructions text edit box

**Returns** Reference to the instructions text edit box

**clearValues**() → None
Clears the values in the text boxes.

**Returns** None

**confirmValues**() → None
Confirms the values in the textboxes by displaying a pop up message of the values.

**Returns** None

**property del_BtnClear**
Getter for the Clear button so we can attach functionality to it later.

**Returns** Reference to the clear button

**property del_BtnConfirm**
Getter for the Confirm button so we can attach functionality to it later.

**Returns** Reference to the confirm button

**property del_BtnStart**
Getter for the Start button

**Returns** Reference to the start button

**property del_BtnStop**
Getter for the Stop button

**Returns** Reference to the stop button

**property del_LblInstructor**
Getter for the Instructor label so we can attach functionality to it later.

**Returns** The instructor label

**property del_LblPilot**
Getter for the Pilot label so we can attach functionality to it

**Returns** The pilot label

**property del_LblTimer**
Getter property for the timer label. We need to attach a QTimer to it so it can count the time the application has been tracking the drone.

**Returns** The timer label

**property del_TBInstructor**
Getter for the Instructor textbox so we can attach functionality to it later.

**Returns** The instructor textbox

**property del_TBPilot**
Getter for the Pilot Textbox so we can attach functionality to it

**Returns** The pilot textbox

**property del_TEInstructions**
Getter for the instructions text edit box

**Returns** Reference to the instructions text edit box

**initView**() → None
Initializes and lays out all of the controls and elements on the view.

**Returns** None

**returnHome**() → None
Sends a signal to the main controller that the Return Home button was pushed.

**Returns** none

**setClrConfirmBtns**() → PyQt5.QtWidgets.QHBoxLayout
Sets the buttons for clearing and confirming the pilot, instructor, and flight instruction information.

**Returns** The confirmation button

**setFlightInstructions**() → PyQt5.QtWidgets.QVBoxLayout
Sets the textbox that will allow the instructor to type in the flight instructions for the pilot to try to match.

**Returns** A vertical layout with the Instructions label on top of the text box

**setInstructor**() → PyQt5.QtWidgets.QVBoxLayout
Sets up the instructor label and the textbox that will be used to set the instructor flying during this session.

**Returns** Returns a vertical layout with the instructor label over the instructor textbox

**setPilot**() → PyQt5.QtWidgets.QVBoxLayout
Sets up the Pilot label and the textbox that will be used to set the pilot flying during this session.

**Returns** Returns a vertical layout with the pilot label over the pilot textbox

**setStartAndStopBtns**() → PyQt5.QtWidgets.QHBoxLayout
Sets up the start and stop buttons for tracking the drones.

**Returns**

**setStatusLabel**(*text*) → PyQt5.QtWidgets.QLabel
Sets up a status label for the window

**Returns** Label of the application taken from the "text" parameter

**setSubTitle**(*text*) → PyQt5.QtWidgets.QLabel
Sets up a subtitle label for the window

**Returns** Subtitle of the application taken from the "text" parameter

**setTimerLabel**() → PyQt5.QtWidgets.QVBoxLayout
Sets the label that will continuously update and display the time that the application has been actively tracking. Need to attach a QTimer() to it.

**Returns** The label that will contain the timer.

**setTitle**() → PyQt5.QtWidgets.QVBoxLayout
Sets up the title with the application title on top and the name of the screen just below it.

**Returns** Layout with the application title and screen title labels

**property set_BtnClear**
Getter for the Clear button so we can attach functionality to it later.

**Returns** Reference to the clear button

**property set_BtnConfirm**
Getter for the Confirm button so we can attach functionality to it later.

**Returns** Reference to the confirm button

**property set_BtnStart**
Getter for the Start button

**Returns** Reference to the start button

**property set_BtnStop**
Getter for the Stop button

**Returns** Reference to the stop button

**property set_LblInstructor**
Getter for the Instructor label so we can attach functionality to it later.

**Returns** The instructor label

**property set_LblPilot**
Getter for the Pilot label so we can attach functionality to it

**Returns** The pilot label

**property set_LblTimer**
Getter property for the timer label. We need to attach a QTimer to it so it can count the time the application has been tracking the drone.

**Returns** The timer label

**property set_TBInstructor**
Getter for the Instructor textbox so we can attach functionality to it later.

**Returns** The instructor textbox

**property set_TBPilot**
Getter for the Pilot Textbox so we can attach functionality to it

**Returns** The pilot textbox

**property set_TEInstructions**
Getter for the instructions text edit box

**Returns** Reference to the instructions text edit box

**startTracking**() → None
Sends a signal to the main controller that the Start Tracking button was pushed.

**Returns** none

**stopTracking**() → None
Sends a signal to the main controller that the Stop Tracking button was pushed.

**Returns** none

CHAPTER

# TWELVE

# VERIFY SETUP SCREEN

**class** src.Views.View_VerifySetupScreen.**VerifySetupWindow**(*phoneControl: Controllers.PhoneController.PhoneControl*)

The view for the verify setup page that is shown when the user presses the "Verify Setup" button on the home page.

**Variables**

- **__btnPhoneSync** – The class property for the 'Phone Sync' button.
- **__btnTestLight** – The class property for the 'Test Light' button.
- **__btnTestFull** – The class property for the 'Test Full Setup' button.
- **__btnCheck** – The class property for the 'Check Status' button.
- **__btnHome** – The class property for the 'Return to Home' button.

**property BtnCheck**

The check status button for the view so we can attach functionality to it later on. Is used to check the status of the set up procedures.

**Returns** None

**property BtnHome**

The home for the view. Is used to return to home screen.

**Returns** None

**property BtnPhoneSync**

The phone sync button so we can attach functionality to it later on.

**Returns** The reference to the phoneSync button

**property BtnTestFull**

The test full setup button for the view so we can attach functionality to it later on. Is used to import past flight files.

**Returns** None

**property BtnTestLight**

The test light button so we can attach functionality to it later on.

**Returns** None

**checkStatus**() → None

Shows the status of the system setup.

**Returns** None

**property del_BtnCheck**
The check status button for the view so we can attach functionality to it later on. Is used to check the status of the set up procedures.

**Returns** None

**property del_BtnHome**
The home for the view. Is used to return to home screen.

**Returns** None

**property del_BtnPhoneSync**
The phone sync button so we can attach functionality to it later on.

**Returns** The reference to the phoneSync button

**property del_BtnTestFull**
The test full setup button for the view so we can attach functionality to it later on. Is used to import past flight files.

**Returns** None

**property del_BtnTestLight**
The test light button so we can attach functionality to it later on.

**Returns** None

**initView**() → None
Sets up the view and lays out all of the components.

**Returns** None

**returnHome**() → None
Sends a signal to the main controller that the Return Home button was pushed.

**Returns** none

**setButtonLayout**() → PyQt5.QtWidgets.QHBoxLayout
Lays out the 'Phone Sync', 'Test Light' and 'Test Full Setup' buttons into a horizontal layout to be put on screen.

**Returns** The horizontal layout containing the 3 buttons

**setTitle**() → PyQt5.QtWidgets.QVBoxLayout
Sets up the title with the application title on top and the name of the screen just below it.

**Returns** Layout with the application title and screen title labels

**property set_BtnCheck**
The check status button for the view so we can attach functionality to it later on. Is used to check the status of the set up procedures.

**Returns** None

**property set_BtnHome**
The home for the view. Is used to return to home screen.

**Returns** None

**property set_BtnPhoneSync**
The phone sync button so we can attach functionality to it later on.

**Returns** The reference to the phoneSync button

**property set_BtnTestFull**
The test full setup button for the view so we can attach functionality to it later on. Is used to import past flight files.

**Returns** None

**property set_BtnTestLight**
The test light button so we can attach functionality to it later on.

**Returns** None

**setupPicture**() → PyQt5.QtWidgets.QLabel
Used for configuring the display for the logo on the screen.

**Returns** Returns the label that contains our logo so it can be put on the main panel.

**syncPhone**() → None
Runs phone sync test.

**Returns** None

**testFull**() → None
Runs full system test.

**Returns** None

**testLight**() → None
Runs light test.

**Returns** None

CHAPTER

# THIRTEEN

# MATPLOT GRAPH

src.Views.Graph.**checkCoordinates**(*flightDict: dict*)
Reads the input dictionary of flight data from a .flight file.

**Parameters** **flightDict** – Dictionary containing flight data.

**Returns** Dictionary of flight data.

src.Views.Graph.**checkLegalInput**(*x*, *y*, *z*)
Checks if the inputted 3D coordinate is within legal bounds. Legal bounds are: x, y, z > 0 and x < 15 and y < 15 and z < 10.

**Parameters**

- **x** – x value to check.
- **y** – y value to check.
- **z** – z value to check.

**Returns** A boolean denoting if legal or not (true if legal, false if outside bounds).

src.Views.Graph.**computeVelocity**(*x1*, *y1*, *z1*, *x2*, *y2*, *z2*, *t1*, *t2*)
Computes the velocity of the drone between two points (x1,y1,z1) and (x2,y2,z2) at respective times t1 and t2.

**Returns** A float value representing the velocity of the drone.

src.Views.Graph.**computeVelocityStatistics**(*flightDict: dict*)
Computes statistics on the velocity points.

**Parameters** **flightDict** – Dictionary of flight data

**Returns** Updated dictionary.

src.Views.Graph.**dimensionless_jerk**(*movement: list*, *fs: int*) → float
Calculates the dimensionless jerk for a 1 dimensional array of points.

**Parameters**

- **movement** – The numpy array of points to calculate the jerk for. The array containing the movement speed profile. Doesn't need to be numpy array but it MUST at least be a 1 dimensional list.
- **fs** – The sampling frequency of the data points.

**Returns** The dimensionless jerk estimate of the given movement's smoothness.

src.Views.Graph.**generateGraph**(*flightData: dict*, *displayVelocity: bool*, *t1: float*, *t2: float*)
Driver function for generating the 3d graph of drone coordinates.

**Parameters**

- **flightData** – Dictionary containing flight data.
- **displayVelocity** – Boolean saying if velocity changes should be displayed on the graph.
- **t1** – Minimum time bound for plotting coordinates.
- **t2** – Maximum time bound for plotting coordinates.

**Returns** The figure to display as the 3d graph.

src.Views.Graph.**log_dimensionless_jerk**(*movement: list*, *fs: int*) → float

Calculates the smoothness metric for the given speed profile using the log dimensionless jerk metric.

**Parameters**

- **movement** – The numpy array of points to calculate the jerk for. The array containing the movement speed profile. Doesn't need to be numpy array but it MUST at least be a 1 dimensional list.
- **fs** – The sampling frequency of the data points.

**Returns** The dimensionless jerk estimate of the given movement's smoothness.

src.Views.Graph.**velocityColors**(*flightDict: dict*)

Determines the color of the line segment between two points based on velocity values. The line color is determined by the change in velocity of the drone between two points. The color of the line should be green if the velocity point is greater than the previous velocity point, yellow if within 0.5 m/s, and red if less.

**Parameters** **flightDict** – Dictionary of flight data.

**Returns** An array of color values to use when plotting line segments between points.

src.Views.Graph.**velocityPoints**(*flightData: dict*)

Calculates the velocity of the drone between consecutive points for the entire flight.

**Parameters** **flightData** – Dictionary of flight Data.

**Returns** Modified flightData dictionary.

CHAPTER

# FOURTEEN

# GRAPH TEST

**class** src.Tests.Graph_Test.**Graph_Test**(*methodName='runTest'*)

This class is for testing the graphing functions that we are using to display the flight path onto the UI. We want to make sure that what is being displayed is correct.

**test_checkLegalInput**() → None

Test that legal and illegal coordinate points can be detected.

**Returns** None

**test_computeVelocity**() → None

Test that the velocity is computed as expected between two points (1,1,1) and (3,3,1) with timeDiff = 1.

**Returns** None

**test_graphShows_noError**() → None

Test that graph generates correctly with and with velocity changes shown for data set of 100, then 200, then 800, then 1200 data points. For each size, two tests are run. One test contains all legal inputs. Another test contains 80% legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_size100_allLegal**() → None

Test that file containing 100 (x,y,z) points is read in correctly. This test contains all legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_size100_someLegal**() → None

Test that file containing 100 (x,y,z) points is read in correctly. This test contains 80% legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_size1200_allLegal**() → None

Test that file containing 1200 (x,y,z) points is read in correctly. This contains all legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_size1200_someLegal**() → None

Test that file containing 1200 (x,y,z) points is read in correctly. This contains 80% legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_size200_allLegal**() → None
Test that file containing 200 (x,y,z) points is read in correctly. This contains all legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_size200_someLegal**() → None
Test that file containing 200 (x,y,z) points is read in correctly. This contains 80% legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_size600_allLegal**() → None
Test that file containing 600 (x,y,z) points is read in correctly. This contains all legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_size600_someLegal**() → None
Test that file containing 600 (x,y,z) points is read in correctly. This contains 80% legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_readCoordinates_small_illegal**() → None
Test that coordinates from a small data file are read in correctly.

**Returns** None

**test_readCoordinates_small_legal**() → None
Test that coordinates from a small data file are read in correctly.

**Returns** None

**test_smoothnessComputes**() → None
Test that smoothness function returns a number when inputted data set of 100, then 200, then 800, then 1200 data points. For each size, two tests are run. One test contains all legal inputs. Another test contains 80% legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_smoothnessValues**() → None
Test that smoothness function returns expected number.

**Returns** None

**test_velocityColors**() → None
Test that the correct color is assigned to the graph segment between consecutive velocity values.

**Returns** None

**test_velocityColorsComputes_correctSize**() → None
Test that velocityColors returns array of correct size when inputted data set of 100, then 200, then 800, then 1200 data points. For each size, two tests are run. One test contains all legal inputs. Another test contains 80% legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_velocityComputes_correctSize**() → None
Test that velocityPoints returns array of correct size when inputted data set of 100, 200, then 800, then 1200 data points. For each size, two tests are run. One test contains all legal inputs. Another test contains 80% legal inputs. Illegal coordinate points in file should not be included in "legalPoints" list in dictionary.

**Returns** None

**test_velocityPoints**() → None
Test that velocity between consecutive points is computed as expected.

**Returns** None

CHAPTER

FIFTEEN

# IMPORT FILE TEST

**class** src.Tests.ImportFile_Test.**ImportFileTests**(*methodName='runTest'*)

This test class is used the test the import and export functions to ensure we can successfully save and reload past flights.

**test_import**() → None

Test that .flight file generated from exporting flight data can be imported successfully. All data members should exist, and no extra keys should be in the file.

**Returns** None

CHAPTER

SIXTEEN

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## S

# INDEX

## E



## F



## G



## H



## I

## L



## M



## O



## P



## R



## S

## T

## U



## V



## W