A Field Tool to Assess the Accuracy of Automated Drone Flights

Robert Lauder

Ulster University

Introduction

This script has been created to help drone mappers to check the accuracy of their flights whilst still in the field so that they know whether they need to redo part or all of the flight before they leave site. To describe the script fully and give context to the scenarios in which I will be describing, I need to give a short introduction into the workflow of drone mapping.

Creating orthophotos with a drone requires aerial photos to be taken at set intervals along a preplanned flight pattern, as seen in figure 1. The automated flight and image capture is guided by the drones inbuilt GPS unit, this will also geotag all images taken. The interval and flight pattern will vary depending on the drone's altitude, the camera specification, and the needs of the user. In general, the more complex a surface is, the more overlap is required to get an accurate representation of that surface, the user will also likely benefit from a double grid pattern, this provides more points that can be matched between images (tie points). There are many other factors to consider when creating a plan, like ground sample distance (the size of the pixels), speed, flight duration and topography but they are not in the scope of this project. The basic concept to be aware of this that the flight plan instructs the drone where to fly, how to fly and the exact positions in that flight that it must take photos. These positions can also be downloaded from the mission planner and saved as a KMZ file for preview in Google Earth (Fig. 1). It is this flight plan KMZ as well as the actual images captured by the drone that will be used to assess the accuracy of the flight, by comparing the distance between the flight plan coordinates with the geotagged image coordinates. For most of this project the flight plan coordinates (Fig. 2) will be called PreCoordinates because they are coordinates from before the flight, and the drone image coordinates (Fig. 3) will be called PostCoordinates because they are the result of the flight. In an ideal world there would be no difference between the PreCoordinates and PostCoordinates, but that is not the case as you will see. Drone flyers are always at the mercy of the weather, and even when conditions seem pleasant on the ground at your take-off location, the winds can be or can quickly change to much faster speeds at higher altitudes. Different drones have different capabilities when it comes to counteracting these winds, but it is common for them to deviate from the flight plan in such scenarios. If the drone deviates too far off the plan, it will exit the automated flight and ask the user to land the drone due to high winds.

The user can instruct the drone to continue the flight plan from the last checkpoint (last accurate interval where a photo was taken), or manually return the drone to land where they can either wait for weather to improve otherwise abandon the flight completely. If the flight is continued from the last checkpoint there is a high change of duplicate images being taken as the drone repositions back onto the flight plan and retakes the images along the portion of the flight where it had previously gone astray, before continuing with the rest of the plan as normal. If the flight was completely abandoned, there will be fewer images captured than were scheduled in the flight plan.

These different situations need to be highlighted as they guide and control the flow in the accuracy assessment portion of the script, in which the number of images and the number of points in the flight plan must align so that points [1, 2, 3] are compared against images [1, 2, 3] rather than [1, 3, 2], which would not give valid results.

Figure 1 - Preview of flight plan KMZ in Google Earth showing the flight path in blue and the points of photo capture in white.

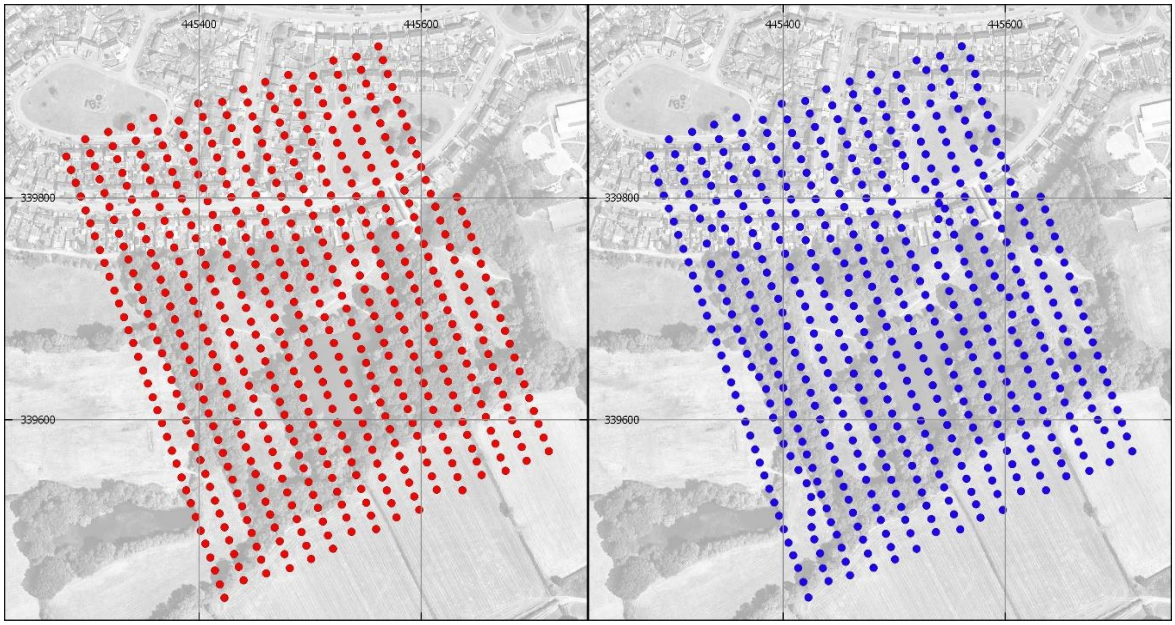

Figure 2 (left)- Flight plan coordinates.

Figure 3 (right) - Captured image coordinates from the drone.

Setup and Installation

The GitHub Repository that contains the scripts, data and environment can be found at: https://github.com/rdslauder/DroneMapping

Four sets of test data are included in the *Test_Images* folder. Each one represents a different scenario that can happen when mapping:

1_Standard: the number of images match the flight plan. The entire flight was completed as planned.

2_Duplicate: the number of images does not match the flight plan. Too many images, there are duplicate images.

3_AbandonedFlight: the number of images does not match the flight plan. Not enough images, the flight plan was not completed. There are no duplicate images.

4_DuplicateAbandonedFlight: the number of images does not match the flight plan. Not enough images, the flight plan was not completed. There are also duplicate images.

The virtual environment required to run the script can be imported with Anaconda using the *environment.yml* file (Table 1).

Channels
-conda-forge -defaults
Dependencies
- python - pandas - geopandas - shapely.geometry - pyproj - matplotlib.pyplot - cartopy>=0.21 - notebook - rasterio - pyepsg - folium - exifread

Table 1 - Required packages for this script to work.

Open Anaconda, activate the environment that came with the repository then open the python terminal via that environment. Navigate to the directory of the required test, within each folder there is already a python file. The only requirements for the script to work are that there are drone images in JPG/JPEG format, and there is a KMZ file containing the flight plan. When the script runs, it will use the images and KMZ file stored in the same directory as the script.

```
Cd [directory]
Python masterScript.py
```

Methods

The script is designed to give different output depending on the user requirements. Two flow charts can be found at the end of this section as Figures 4 and 5 that show the stages and direction of the script and will aid is visualisation during the subsequent descriptions. For in depth steps on how each function works, I have added descriptions into the code as I would not be enough words to include that here.

The first stage is to create standard initial outputs before asking the user for input (step one from Figures 5 & 6).

1) Create initial outputs | *createInitialOutputs()*

1.1) PreCoordinates (Flight Plan)

Convert compressed KMZ file to KML file, extract coordinates from the KML file and store in PreCoordinates.CSV with point number. Convert coordinates from EPSG:4326 (Latitude Longitude) to EPSG:27700 (British National Grid (BNG)) then convert to WKT representation, storing the two in new separate columns of the CSV. Create a shapefile from the WKT representation of BNG coordinates.

1.2) PostCoordinates (Drone Images)

Rename image based on creation date or modified date, whichever is earliest, to account for the fact that when files are copied, the created date becomes the modified date and the current date (when the file was copied) becomes the created date. Extract coordinates from image EXIF data, convert degrees/minutes/seconds to latitude longitude and store in PostCoordinates.CSV with image number. Convert coordinates from EPSG:4326 (Latitude Longitude) to EPSG:27700 (British National Grid (BNG)) then convert to WKT representation, storing the two in new separate columns of the CSV. Format the CSV to sort rows by image number to match the PreCoordinates CSV. Create a shapefile from the WKT representation of BNG coordinates.

1.3) Plot Coordinates

Plot PreCoordinates, with and without labels, save the extents for these plots and store them in a global variable so that the rest of the plots can match their extents and be in the exact same position. Plot PostCoordinates, with and without labels. Plot PreCoordinates and PostCoordinates on the same image, with and without labels. These are all saved as .PNG images (to differentiate from the JPEG drone images and prevent actions against them that were meant for the drone images) and allow easy comparison between the planned flight and the actual flight, this is why it is essential that the extent of the flight plan is used for all plots, as that is the baseline that all other plots are compared to. If an accuracy assessment is requested, the labelled versions will be essential in finding any errors.

2) Ask the user if they want to run an accuracy assessment | *askAccuracyAssessment()*

The accuracy assessment calculates the difference between the planned coordinates from the flight plan and the actual coordinates from the captured images. If the user enters no (“No”, “no”, “N”, “n”) then the accuracy assessment stage is skipped and the next stage is called. If the user enters yes (“Yes”, “yes”, “Y”, “y”) then *checkAccuracyAssessment()* is called. If the user does not enter any of the above valid inputs, then they will be reprompted until they do.

2.1) Check the data and direct the user accordingly | *checkAccuracyAssessment()*

This will automatically check whether the number of drone images matches the number of points in the flight plan by counting the rows in the PreCoordinates and PostCoordinates CSV. They need to match so that the correct rows are compared during the accuracy assessment.

2.1.1) Flight plan matches the number of images

If they match, the accuracy assessment is run.

2.1.2) More images than in the flight plan

If there are more images than in the flight plan, there are duplicate images. Ask the user to check the plots saved to the directory and identify the duplicate image(s), input the duplicate image number(s), if the input number is not present in the PreCoordinates CSV, the user is reprompted until valid input is given and then the images are deleted. Delete and recreate the PostCoordinates CSV, plots and shapefile to reflect the dataset without the duplicate(s). Run the accuracy assessment. Delete and recreate shapefile so it contains the accuracy assessment data.

2.1.3) Less images than in the flight plan | *runFlightPlanAccuracyAssessment()*

If there are less images than in the flight plan, the flight was not fully completed. This often happens door to poor weather conditions, in these scenarios there is a higher chance of duplicate images being taken as the drone tries to counteract high winds and repositions itself on the flight plan, so as standard the user is asked to check the plots for duplicate images and asked if there are any, if the user inputs no (“No”, “no”, “N”, “n”) then the scripts continues to the PreCoordinates deletion. If the user inputs yes (“Yes”, “yes”, “Y”, “y”) then the same process of duplicate deletion as above occurs before continuing.

Now that there are no duplicates, the PreCoordinates CSV can be amended to match the PostCoordinates CSV. The user is asked if they want to delete individual points or a range of points. Deleting individual points is useful if there are only a few points in the flight plan that were missed. Deleting a range of points is useful is large areas were missed, this saves the user inputting potentially hundreds of individual numbers. If the user asks for induvial points (“Individual”, “individual”, “I”, “i”), they are asked to input the numbers, them numbers are checked to against the PreCoordinates CSV, if not present then the user is reprompted for valid input until it is given. When valid input is given, those rows are deleted from the PreCoordinates CSV. If the user asks for a range of points (“Range”, “range”, “R”, “r”), they are asked to input the first number in the range which is checked to against the PreCoordinates CSV, if not present then the user is reprompted for valid

input until it is given. When valid input is given, the user is asked for the last number in the range, the same validation process occurs again. Once valid first and last numbers are given, the range is calculated and stored in a list before being deleted in the same manner as above.

Now that the excess points are removed from the PreCoordinates CSV, it is formatted to sort rows by image number to match the PostCoordinates CSV. The PreCoordinates plots and shapefile is deleted and recreated to reflect this new change and then the accuracy assessment is run. Upon completion, the PostCoordinates shapefile is deleted and recreated so it contains the accuracy assessment data.

3) Ask the user if they want to batch rename the images | *askBatchRename()*

Adding batch numbers can be useful for keeping track of image origin. It can also be good when using the images for annotation, for example in training datasets for machine learning.

If no, this step is skipped. If yes, the user is asked to input a batch number, this will be added as a prefix to all the image names as well as a "#". For example, 1#1, 1#2, 1#3 etc. Originally a point was being used but this created problems later on if they were added to a CSV, with excel formatting the names incorrectly – 1.1, 1.10 and 1.100 would al be reduced to 1.1.

4) Print a summary of outputs | *printSummary()*

At each stage of the script where the direction could be changed by a conditional, a global variable was defined, and a value assigned to it. These variables are global so they can be accessed outside of the functions they are defined in, unlike local variables. There are two types of global variables used: Boolean and non-Boolean. The Boolean variables have a True or False value, these are used to track the direction of the code so that messages can be printed that are specific to the location the user was taken. Non-Boolean variables are used to store user input, such as duplicate images and CSV rows to delete, or the batch number, so that they can be printed at the end in this summary message.

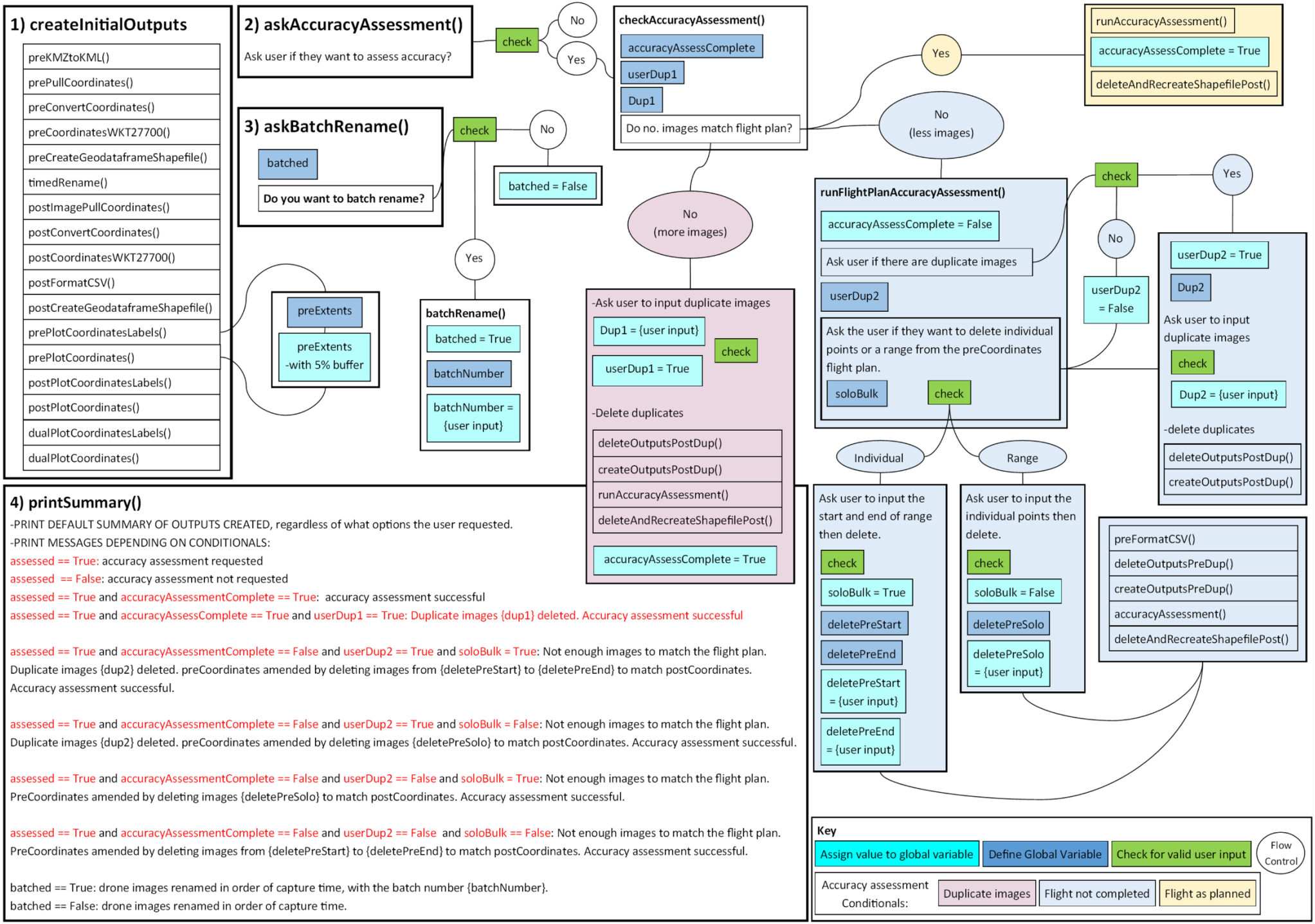

Figure 4 – Detailed flow chart of the script.

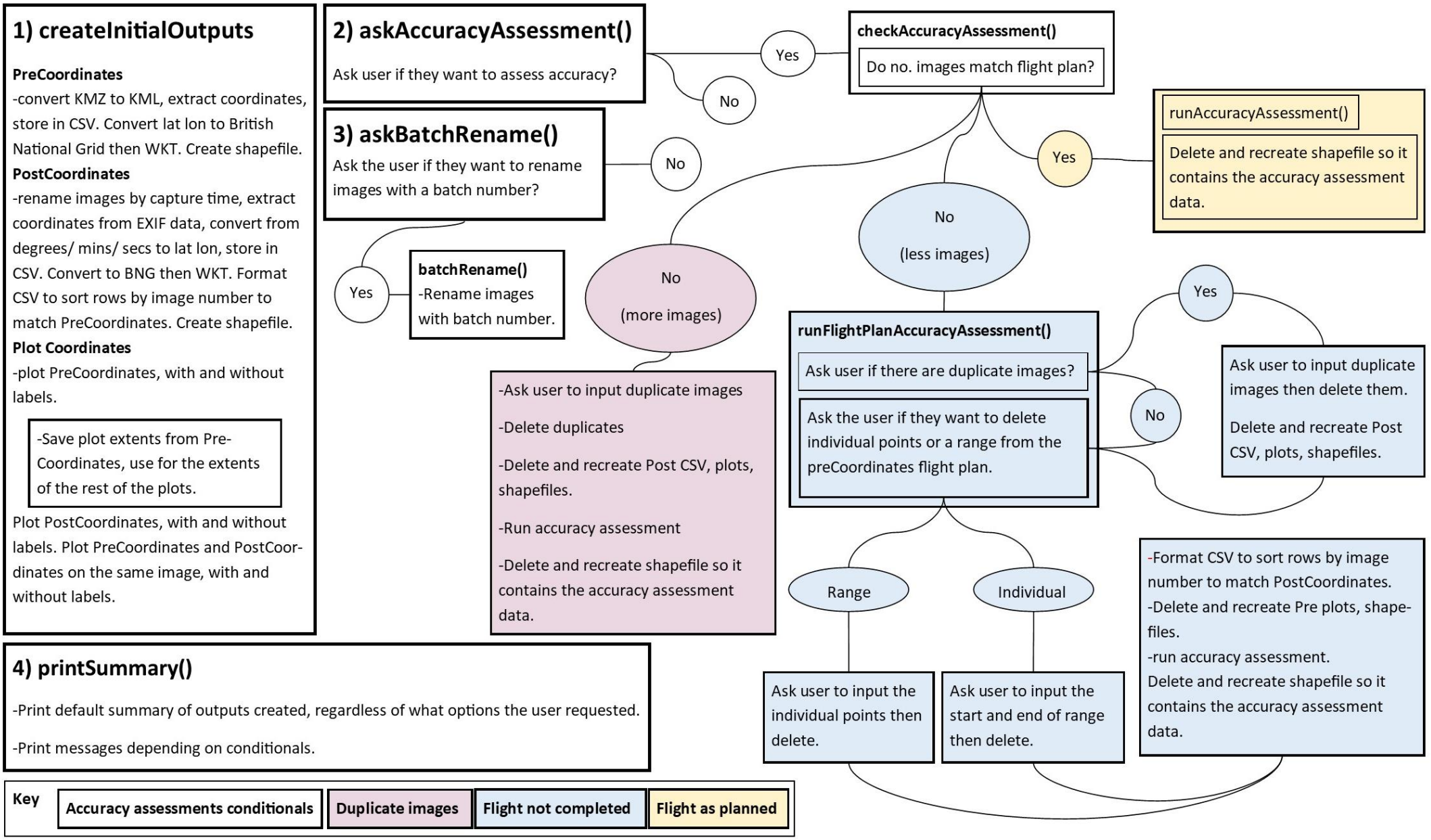

Figure 5 – Simplified flow chart of the script.

Results

This will depend on the inputs given and the outputs requested. Initial outputs will create the following and a backup will be saved in a new folder in the script directory in case the user amends the outputs throughout the accuracy assessment and wants to refer to the original:

a) Rename all images by their capture time to match the flight plan.

b) KML file (uncompressed KMZ)

c) PreCoordinates CSV - a CSV with the coordinates of each capture point from the flight plan with the point number.

d) PreCoordinates shapefile

e) PostCoordinates CSV – a CSV with the coordinates of each image with image number.

f) PostCoordinates shapefile

g) Six different plots of the coordinates

-Labelled PreCoordinates

-Unlabelled PreCoordinates

-Labelled PostCoordinates

-Unlabelled PostCoordinates

-Labelled Pre and Post Coordinates combined

-Unlabelled Pre and Post Coordinates combined

If the user requests an accuracy assessment, the PreCoordinates and PostCoordinates will be compared and if any errors are found, the user will be guided accordingly to fix them before the accuracy assessment is done. The accuracy assessment is provided in a new column in the PostCoordinates CSV.
If the user requests the images to be renamed with a batch number, they will be, and all the previously created outputs will remain unchanged.
A summary message is printed to inform the user what has been created and what has been deleted if anything.

Troubleshooting

A section that provides some troubleshooting advice in case things go wrong.

If there are problems with modules, open the script to check which have been imported in the first lines and manually install them to the environment if needed. It is recommended that the sets of test data are duplicated before running the script, so that if any errors occur, the original data set if available to retry. Where users give input, if/ elif/ else loops have been included to reduce the chances of errors occurring, but when deleting images or CSV rows, the user may accidently type the wrong number, and if that number is still a valid number it will be accepted and the user must carry on knowing the wrong data will be deleted, I will keep working on the script to account for situations like this. Do not change the name of any outputs and keep the CSV files closed until the script is completed.

Word count: 2591