Module Seven

Henrik Frank, hefra13@student.sdu.dk
Christian Arentsen, chare13@student.sdu.dk
Vasileios Karvouniaris, vakar15@student.sdu.dk
Asbjørn Schou Müller, asmul10@student.sdu.dk

I. AQ Ground Control

I. Inspecting AutoQuad log files

UKF_POSD For the down position, as seen i Figure 1 it can be seen that the positions are actually positive, though it should be negative values, as we are measuring going down as positive. Regardless, we can see, after the GPS has gotten a fixed position, we get an altitude of approx 10 meters above sea level, and we can see how the drone takes off, and lands again a couple of times.

UKF_POSN, UKF_POSE and UKF_POSD For all positions in the coordinate system, Figure 2, it can be seen how the drone moves around.

I.1 Generating mission route plans

Use the `aqlogreader` Python script available in the module folder to import track data from the file `021-AQL.LOG`.

Create a Python class to generate a route plan consisting of waypoints (latitude, longitude, altitude, heading) based on the track data removing excessive track points.

The class should perform the conversion base on the different input parameters below:

1. maximum distance deviation from track (multirotor).
2. ~~maximum waypoints allowed (multirotor).~~
3. ~~maximum distance and bearing angle deviation from track (fixed wing).~~

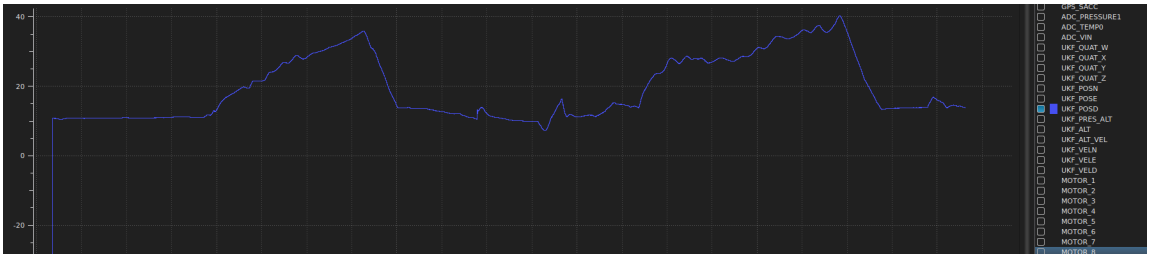

Figure 1: Graph showing UKF_POSD for down

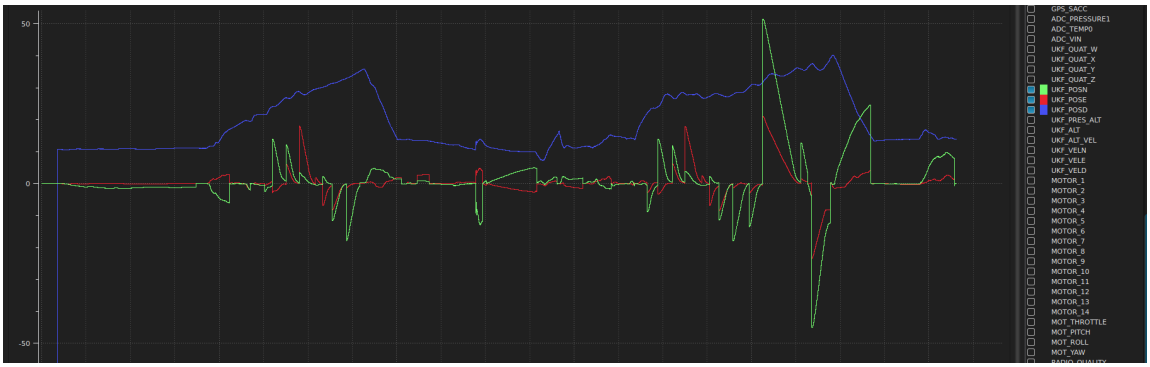

Figure 2: Graph showing UKF coordinates for North, East and Down

Please look into the literature to find existing algorithms or devise your own. Plot both the track log and generated route plan using Google Earth using the `export_kml.py` class available under course materials. Test your route plan class using as input the NMEA data (for which you developed an import class in module 1) from the file `nmea_trimble_gnss_eduquad_flight.txt`

Present the algorithm, your results and discuss advantages/disadvantages in the report.

Results In order to remove excessive points from the track we estimated the Euclidean distance from one point to consecutive points. Then we set as a checkpoint the one whose distance from the 'point of origin' was larger than a threshold. And we repeat the process each time with the new checkpoint as the point of origin for the following points.

```
from math import hypot
    ...
    dstnc = hypot(nextLat, nextLng)
    dstnc = hypot(dstnc, nextAlt)
    if dstnc >= threshold:
        initLat = oldLat
        initLng = oldLng
        initAlt = coords
        cpLat.append(oldLat)
        cpLng.append(oldLng)
        corAlt.append(coords)
    ...
```

By applying this method on the data in *021-AQL.LOG* file we managed to get a good fit from the original points with threshold=1 (Figure 3). The amount of checkpoints in this case was 382. Then we increased the threshold to 5 and the points dropped to 76! the fit however was less than optimal (Figure 4). The process was also applied on the Nmea data from *nmea_trimble_gnss_eduquad_flight.txt* with produced an impressive amount of 157 points out of the original 5500+ points (Figure 5).

The main advantage of this approach is its simplicity. It is very easy to implement and seems rather intuitive. On the other hand it lacks consistent performance across different kinds of data. For example a threshold of 0.3 is needed for the Nmea data to get a total of 157 out of 5531 points while a threshold of 1 is sufficient for 382 out of 2500 points in the AQL data.

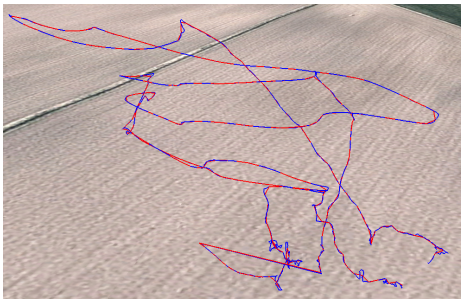

Figure 3: *Threshold 1*

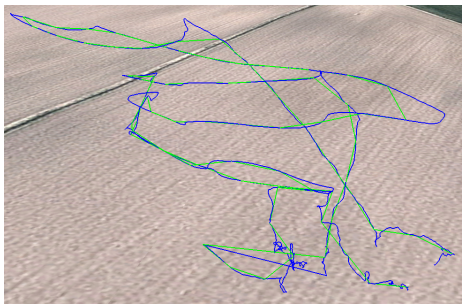

Figure 4: *Threshold 5*

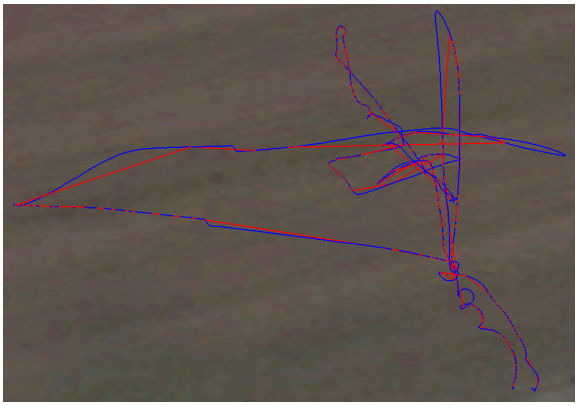

Figure 5: Nmea Data with Threshold 0.3