GENERAL RULES

- Groups must work on their own without any help from any other group. Groups that present suspiciously similar code will not only fail the project, but also fail the subject.
- The last version of the code will be uploaded to eGela at least once a week. Bear in mind that big changes in little time may look suspicious from the lecturer's point of view. To avoid problems, upload your code as often as possible so that we can see the evolution of your work (see *Upload your code*).

PROJECT GOAL

The goal of this project is to process data recorded during a drone flight by the research group *Robotics and Perception Group* (Institute of Neuroinformatics – University of Zurich UZH). The results of the study they conducted (http://rpg.ifi.uzh.ch/zurichmavdataset.html) were presented in the scientific publication *The Zurich Urban Micro Aerial Vehicle Dataset.* In this publication, they present a method to minimize the error using a GPS receiver when driving a drone in an urban environment. The data recorded include the most accurate estimations done by using a camera (the -ground truth-) and the estimation done by using GPS.

In this project, you will read the data and generate different plots and statistics that will help you analyze the behavior of the drone.

GROUP GRADES

Each group will be given a group grade that will be used as the baseline to calculate each student's individual grade (it could be greater or less than the group grade). The maximum group grade will depend on the number of tasks finished by the group:

- 100%: all the tasks
- 80%: tasks 1, 2, 3 and 4

PROGRAMMING GUIDELINES

- Do not print or output any information (or plots) from inside a function.
- Print the results in the exact same format we use in this document.
- When possible, reuse code by using functions. In most of the tasks, we will tell you how to design the functions.
- When possible, use loops (*for*/*while*) to iterate over sets of values instead of replicating existing code.
- Use meaningful names for variables/scripts/functions so that anyone can understand the code without spending too much time reading it.
- Write comments to explain the goals of each part of the programs (start the line with '%').

UPLOAD YOUR CODE

1. For reviewing purposes, we ask you to upload your code inside the appropriate task in E*gela*, **at least once a week**. Do not erase old versions.

2. Only one student per group needs to upload the code.

3. You must add a ZIP file each time:

a. The name of the file must have the format: '*project-yyyy-mm-dd.zip*', where *yyyy* is the year, *mm* the month, and *dd* the day.

b. The zip must contain all the script/function files you have done

c. The zip must also contain all the input files required so that we can run the scripts using only what's inside the zip file.

d. It will also include a file named '*log.txt*', where you will add what you have done since the last time you uploaded your code.

For example:

log.txt inside *project-2017-11-22.zip*

```
2017-11-20
Task 1: We started working on 'myscript.m', but it
still doesn't work

2017-11-22
Task 1: We finally made it work, results look
reasonable
Task 2: Just started working on function 'myfunction'
```

INPUT FILES

In this project, you will process some log files, which were recorded while driving a drone. The whole log file is rather big, so we have it divided in several partial files. Any processing done to the full trajectory needs to be done for all the partial files:

- `GroundTruthAGL-00.csv`
- `GroundTruthAGL-01.csv`
- ...
- `GroundTruthAGL-05.csv`

The format of the log files is the following:

Example*:* `GroundTruthAGL-00.csv`

```
imgid, x_gt, y_gt, z_gt, omega_gt, phi_gt, kappa_gt, x_gps, y_gps, z_gps,
1,465666.0,5247973.6,469.0,95.5,-73.2,9.1,465670.7,5247978.0,464.9,
31,465665.9,5247973.6,469.7,96.2,-74.3,10.1,465671.0,5247977.2,466.3,
61,465665.9,5247973.6,469.9,98.2,-77.4,12.5,465671.5,5247977.2,467.2,
91,465665.8,5247973.7,470.1,98.6,-76.9,13.1,465671.8,5247977.1,467.7,
121,465665.7,5247973.6,470.1,98.8,-77.1,13.4,465672.4,5247977.6,467.5,
...
```

Each log file has a header in the first line that describes the content of each column:

- the first column is the identifier of an image (we are going to ignore this information)
- the 2nd to 4th columns are the position coordinates of the drone: x, y, and z (estimated using a camera)
- the 5th to 7th columns are the orientation angles of the drone: yaw, pitch, and roll (estimated using a camera)
- and the 8th to 10th columns are the position coordinates of the drone: x, y, and z (estimated using the GPS receiver)

You can find additional information in the README.txt file.

TASKS

Task 1: Visualize the trajectory followed by the drone using a 3D plot

Write a script (*visualizeTrajectory.m*) that visualizes the position of the drone along the trajectory in a 3D plot.

Since, we have the complete trajectory divided in several partial files, we need to load every partial file (after checking that they exist), and plot the [x,y,z] coordinates (those estimated by the camera) using the function:

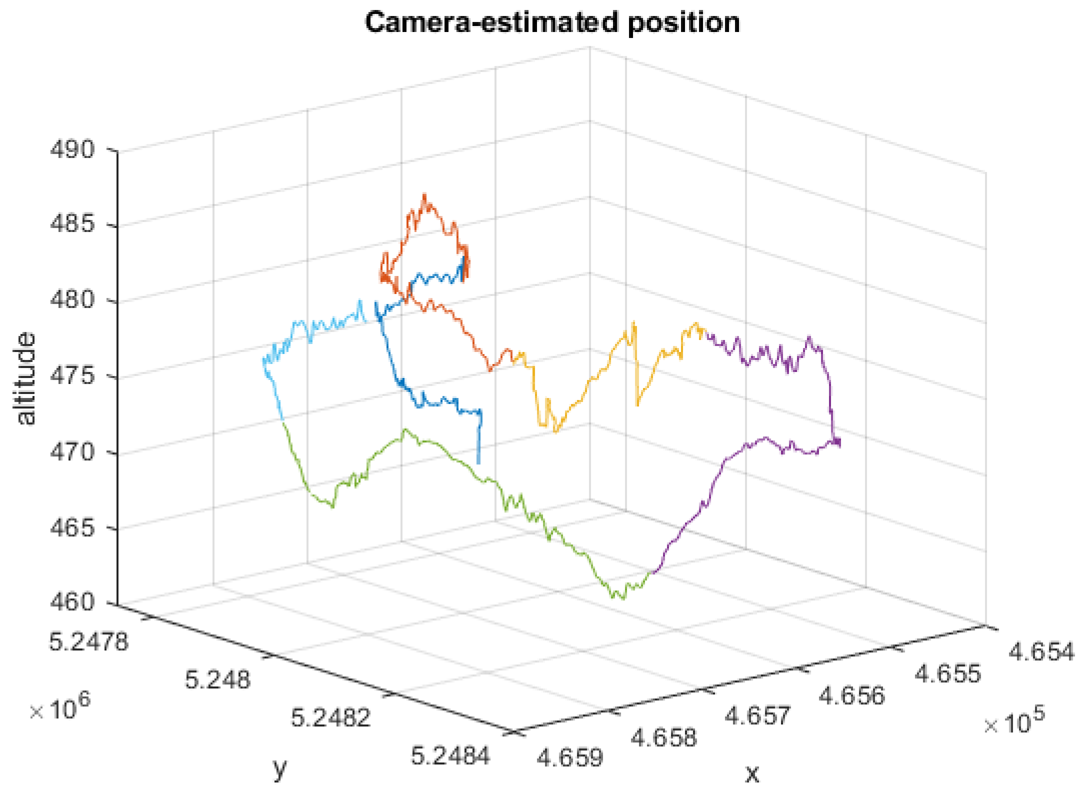
```
plot3(x,y,z);
```

Label the axes, and set the title of the figure as below. Also, set the camera view so that the azimuth angle (around the z axis) is 140 degrees and the elevation angle is 20 degrees using the function:

```
view([azimuth, elevation]);
```

Finally, show the grid using:

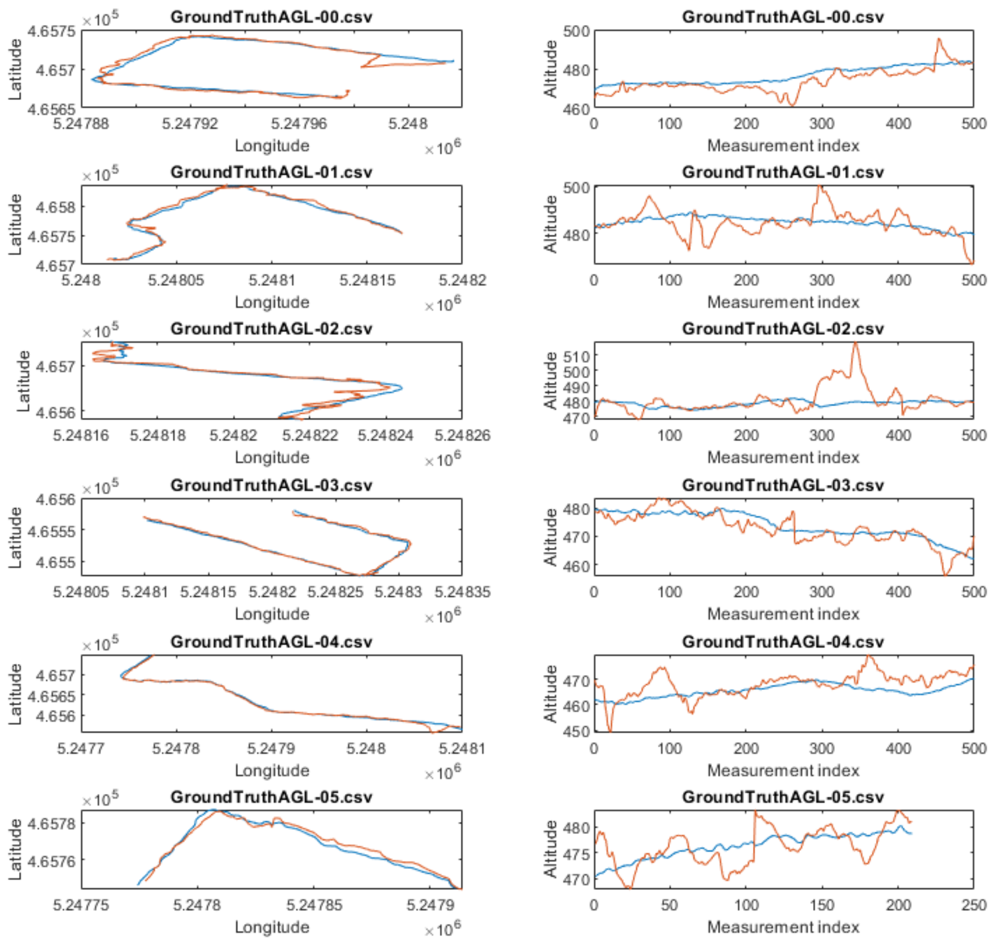
```
grid on;
```



Task 2: Estimated position coordinates (by camera and GPS)

In this task, you will write a script (*compareEstimations.m*) that compares both graphically and numerically the position coordinates estimated by the camera and those estimated by the GPS receiver.

1. The script will create an image with one row for every partial file and two columns. For each partial file, it will plot on the left column both estimations of the [x,y] position coordinates, and, on the right column, both estimations of the altitude [z]. Set the titles and labels, and resize the window until it looks as the one below.



2. For every partial file, the script will calculate and show on the screen the following data: filename, maximum and minimum altitude, and the average altitude. **The script will use the GPS-estimated positions**. The format of the output must be:

```
Log: GroundTruthAGL-00.csv
Max & min altitude: 495.65m, 461.07m
Avg. altitude: 473.75m

Log: GroundTruthAGL-01.csv
Max & min altitude: 500.64m, 466.77m
Avg. altitude: 484.27m

...............
Log: GroundTruthAGL-05.csv
Max & min altitude: 483.35m, 467.92m
Avg. altitude: 476.17m
```

Task 3: Main program menu

Write a script (*mainMenu.m)* that shows the following menu:

```
####### MENU ########:
1. Show the trajectory as a 3D plot
2. Compare the estimated position coordinates
4. Calculate the distance, energy and estimation error
5. Calculate the per-window altitude standard deviation
0. Exit
Select an option:
```

Depending on the option chosen, the program will run the selected script (to run a script from another script, we just need to write its name without the ".m" extension). Before running a script, whether or not user has introduced a valid option number will be checked. Whenever the userintroduces a wrong option, the message "*The selected option is invalid: please choose 1, 2, 4, 5 or 0*" will be shown.

The menu script will start by clearing the console and, when an option is selected, it will show the results. Next, the user will be shown the message "*Press a key to continue...*", and when the user presses a key, the menu script will again clear the console and show the options. This will allow the user to read the results before clearing the console.

The menu will be shown until the user selects 0. Do not use the function *exit()* to end the script (it closes Matlab), just exit from the iterative structure you are using to show the menu.

Task 4: Calculate distance and energy consumption

In this task, you will calculate the distance traveled by the drone according to both estimations of the position (camera and GPS). To this end, you must carry out the following tasks:

1. Write a function that, given three vectors with the position coordinates (x, y, and z) recorded during a trajectory, calculates the total travelled distance using the formula:

   $$d = \sqrt[2]{(lat_i - lat_{i-1})^2 + (lon_i - lon_{i-1})^2 + (h_i - h_{i-1})^2}$$

   The function will be defined as:

   ```
   function d = calculateDistance(latitudes, longitudes, altitudes)
   ```

   Test the function with the following example:

   ```
   >> distance=calculateDistance(1:20,1:20,0.01:0.01:0.2)
   distance = 26.8707
   ```

2. Write a function that, given a series of altitudes, calculates the accumulated positive altitude (the sum of the positive altitude differences).

   ```
   function alt = calculateAccPosAltitude(altitudes)
   ```

   Test the function with the following example:

   ```
   >> accPosAlt= calculateAccPosAltitude([5.1 5.4 5.2 6])
   accPosAlt = 1.1
   ```

   In this example, the accumulated positive altitude is calculated: *(5.4-5.1)+(6-5.2)= 1.1*

3. Write a function that, given a series of altitudes, calculates the energy consumed by the drone by multiplying the absolute altitude increases by 98. The result will be the energy consumption in mAh.

   ```
   function e = calculateEnergy(altitudes)
   ```

   Test the function with the following example:

   ```
   >> energy= calculateEnergy([5.1 5.4 5.2 6])
   energy = 127.4000
   ```

In this example, the energy is calculated: *(|5.4-5.1|+|5.2-5.4|+|6-5.2|)*98= 127.4 mAh*

In Matlab, the absolute value is not calculated using |...|, but using function abs(...).

4. Write a function that, given two vectors of altitudes estimated by two different means during the same trajectory (for example: camera and GPS), calculates the percentage of values with a difference greater than 10 meters between both estimations. The function will be defined:

```
function percentage = doubtfulAltitudePercentage(h1, h2)
```

   Test the function with the following example:

```
>> percentage=doubtfulAltitudePercentage([5 4 5 3],[6 15 3 6])
percentage =  25
```

   The function will return that *25%* of the altitudes differ in more than *10* meters in both estimations: only one out of four (the second value is *4* in the first estimation, and *15* in the second).

5. Write a script called *statsFullTrajectory.m* that uses the three functions you just created to calculate the following statistics about the full trajectory.

```
  1) According to the camera-estimated positions:
  The drone traveled 1915.63 m
  The drone climbed 139.56 m
  The energy consumed was 26405.75 mAh
  2) According to the GPS-estimated positions:
  The drone traveled 2722.26 m
  The drone climbed 660.81 m
  The energy consumed was 127938.11 mAh


  6.68% of the estimated altitudes differ in more than 10 meters
between estimations
```

   To this end, you will load the data of all the files in a single matrix in the following way:

   a) Initialize the matrix (*data*) that will contain the final data as an empty one:

```
data=[];
```

2. Then, one by one, read all the partial files in a matrix (for instance, m), and concatenate it to the complete data matrix doing

```
data = [data ; m];
```

Once the data has been loaded and stored in the matrix, the matrix can be processed to generated the statistics mentioned above.

Task 5: Calculate the per-window altitude standard deviation

We want to analyze the standard deviation of the estimated altitudes, but, instead of calculating this statistic for a complete file, you will analyse the data in blocks (*windows*) of fixed size. For example, if a trajectory has *100* points and the window size is *10*, you will calculate the standard deviation of each window. To get each window, the central element is taken, and the window is completed with the preceding 10 elements and the following ten elements. To build the first window the 11th elements is taken as the central element and the window is completed with elements *1* to *10* and 12 to 22. The following windows will contain elements from *23* to *44*, and so on...

1. You will define a function, that given a vector (it could be any kind of data) and the window size returns a vector with the standard deviation of each of the windows used:

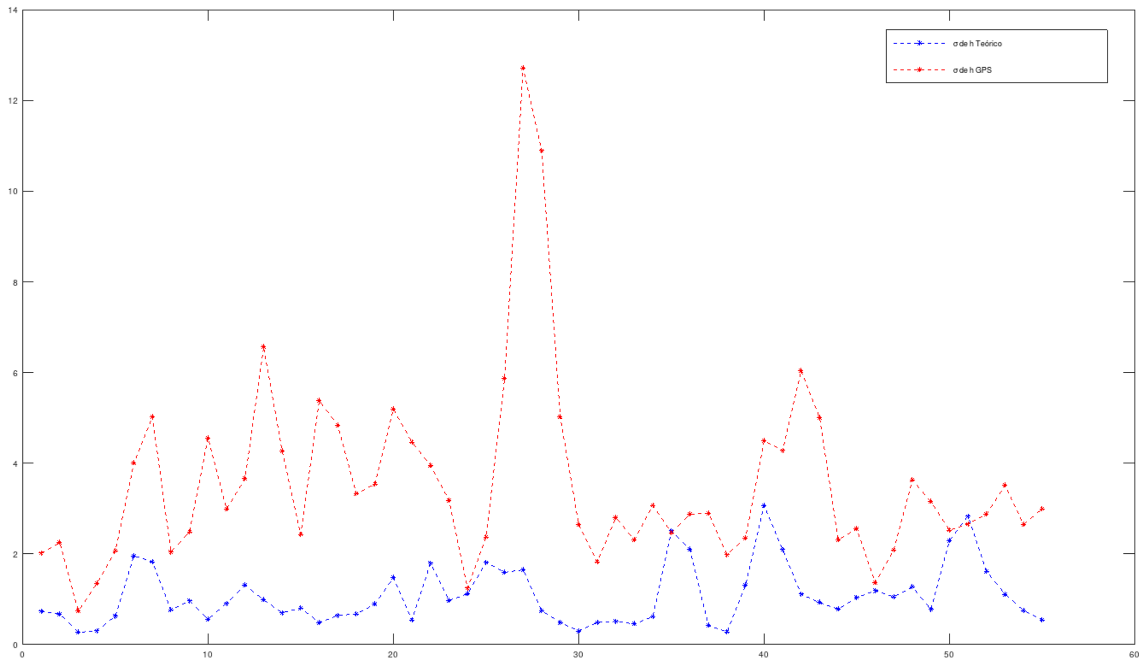
   ```
   function desv = calculateDev(v, windowSize)
   ```

   Test the function with the following example

   ```
   >> calculateDev([2 2.1 3 4.2 3 2.3 4.3 4.2 4.5],2)
   ans =     0.8877     1.0243
   ```

   The first value in the output vector is the standard deviation of the first 5 elements [2, 2.1, 3, 4.2, 3], and the second the standard deviation of the last 4 elements. [2.3, 4.3, 4.2, 4.5]

2. Write a script (*perWindowStdDev.m*) that, using the complete trajectory as in the previous exercise, calculates the standard deviation of both estimated altitudes using windows of size 10.


σ de h Teórico
σ de h GPS