REGLAS GENERALES

- Los grupos deben trabajar cada uno **sin ayuda de otro grupo**. Los grupos que presenten código sospechosamente parecido **suspenderán el proyecto y la asignatura en su totalidad**.
- Se debe subir a eGela la última versión que se tenga del trabajo por lo menos una vez por semana. Tened en cuenta de que hacer grandes cambios en poco tiempo es sospechoso para el profesorado. Para evitar problemas, subid el código a menudo para que se pueda ver la evolución de vuestro trabajo (ver sección *Subir el Proyecto*).

OBJETIVO DEL PROYECTO

El objetivo de este proyecto de la asignatura Fundamentos de Informática 2019/20 es procesar los datos recopilados y suministrados por el grupo de investigación Robotics and Perception Group (Institute of Neuroinformatics de la Universidad de Zurich UZH). Este sitio web (http://rpg.ifi.uzh.ch/zurichmavdataset.html) presenta los resultados del artículo científico titulado *The Zurich Urban Micro Aerial Vehicle Dataset* en el que presentan un método para minimizar el error en las medidas de un GPS tomadas por un dron entre edificios. En este paquete de datos se encuentran las medidas del GPS y la trayectoria teórica deducida por el método que presentan en el artículo.

En este trabajo analizaremos los datos y produciremos diferentes vistas de los datos y se calcularán otros valores como la distancia recorrida y la energía consumida por el dron al elevarse.

EVALUACIÓN POR GRUPOS

Cada grupo tendrá una calificación que será la base de la nota de cada alumno (que puede ser mayor o menor que la del grupo). La nota máxima del grupo dependerá del número de tareas finalizadas correctamente por el grupo:

- 100%: todas las tareas.
- 80%: tareas 1, 2, 3 y 4.

DIRECTRICES PARA LA PROGRAMACIÓN

- No imprimir mensajes por pantalla, crear gráficas o pedir entradas dentro de una función, a no ser que sea el objetivo de dicha función. La entrada y salida de datos de las funciones es mediante parámetros y resultado.
- Cuando sea posible, intenta reutilizar código mediante llamadas a funciones. En la mayor parte de las tareas se indicará cómo diseñar las funciones.
- Elige nombres con significativos y con la nomenclatura adecuada para crear variables/scripts/funciones, de modo que cualquiera que lea el código pueda

entenderlo y seguir el valor de las variables sin pasar tiempo averiguando qué es cada elemento.

- Usa ciclos (*for*/*while*) para iterar (repetir instrucciones) sobre un conjunto de valores en lugar de repetir código ya existente.
- Escribe comentarios (con **%**) para explicar el objetivo de cada parte de los programas y qué datos almacenan las variables.

SUBIR EL PROYECTO

Con la intención de facilitar la revisión de vuestro trabajo, os solicitamos que subáis el código a la tarea de eGela habilitada al respecto, **al menos una vez por semana**.

- Tan sólo un estudiante del grupo debe subir el código.
- Cada vez que se suba el código es obligatorio subir un fichero ZIP:
  - El nombre del fichero debe tener el formato: '*project-yyyy-mm-dd.zip*', donde *yyyy* es el año, *mm* el mes y *dd* el día.
  - El fichero *zip* debe contener todos los *scripts*/funciones que se hayan desarrollado hasta la fecha.
  - El fichero *zip* también debe contener todos los resultados solicitados en las tareas que hayáis podido desarrollar hasta ese punto (*plots*, ficheros de texto, salidas por pantalla).
  - Dentro del fichero también se incluirá un fichero único llamado '*log.txt*', donde se glosará lo realizado cada día de desarrollo del proyecto. A este fichero se le irá añadiendo las aportaciones del trabajo realizado cada semana por cada integrante del grupo.

    Por ejemplo:

log.txt* dentro de *project-2019-11-12.zip

```
•  2019-11-10

   Tarea 1: Comenzamos a programar el script 'myscript.m', pero
   todavía no funciona. Aitor hace la parte de carga de
   ficheros, Blanca la selección de columnas y Jon la del plot.

•  2019-11-11

   Tarea 1: Ya lo hemos terminado de revisar, había un error en
   el bucle principal que ha encontrado Blanca.

   Task 2: Hemos comenzado a escribir la función 'myfunction'
```

FICHEROS DE ENTRADA

En este proyecto trabajaremos con un conjunto de ficheros **parciales**:

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

Los datos de cada fichero responden al siguiente formato:

Ejemplo*:* `GroundTruthAGL-00.csv`

```
imgid, x_gt, y_gt, z_gt, omega_gt, phi_gt, kappa_gt, x_gps, y_gps,
        z_gps,
1,465666.057548,5247973.646622,469.019496,95.538828,-
        73.283629,9.170787,465670.706847,5247978.033438,464.9100
        04,
31,465665.957153,5247973.611734,469.709290,96.242463,-
        74.376423,10.138757,465671.064464,5247977.231159,466.324
        005,
61,465665.909026,5247973.676972,469.997052,98.277619,-
        77.413693,12.546079,465671.570117,5247977.217092,467.265
        015,
91,465665.829212,5247973.715876,470.127735,98.665721,-
        76.989201,13.114321,465671.879377,5247977.181944,467.740
        997,
121,465665.782634,5247973.631864,470.199503,98.874849,-
        77.131451,13.434804,465672.410589,5247977.667845,467.507
        996,
...
```

Cada fichero de registro o *log* tiene una línea de cabecera que describe el contenido de cada columna. La primera columna contiene un identificador de la imagen asociada (no vamos a usar esta información). En las siguientes columnas tenemos las coordenadas teóricas, los ángulos de una cámara y las coordenadas leídas por el GPS. El fichero README.txt proporciona más detalles sobre los datos.

En*:* `readme.txt`

```
1: imgid (id of the MAV image), 2: x_gt (ground truth camera
        position x), 3: y_gt (ground truth camera position y),
        4: z_gt (ground truth camera position z), 5: omega_gt
        (degrees, ground truth camera orientation yaw), 6:
        phi_gt (degrees, ground truth camera orientation pith),
        7: kappa_gt (degrees, ground truth camera orientation
        roll), 8: x_gps (GPS camera position x), 9: y_gps (GPS
        camera position y), 10: z_gps (GPS camera position z)
All values regarding positions are in the WGS 84 / UTM zone 32N
        coordinate system use plotPath.m to visualize the data
        in matlab
```

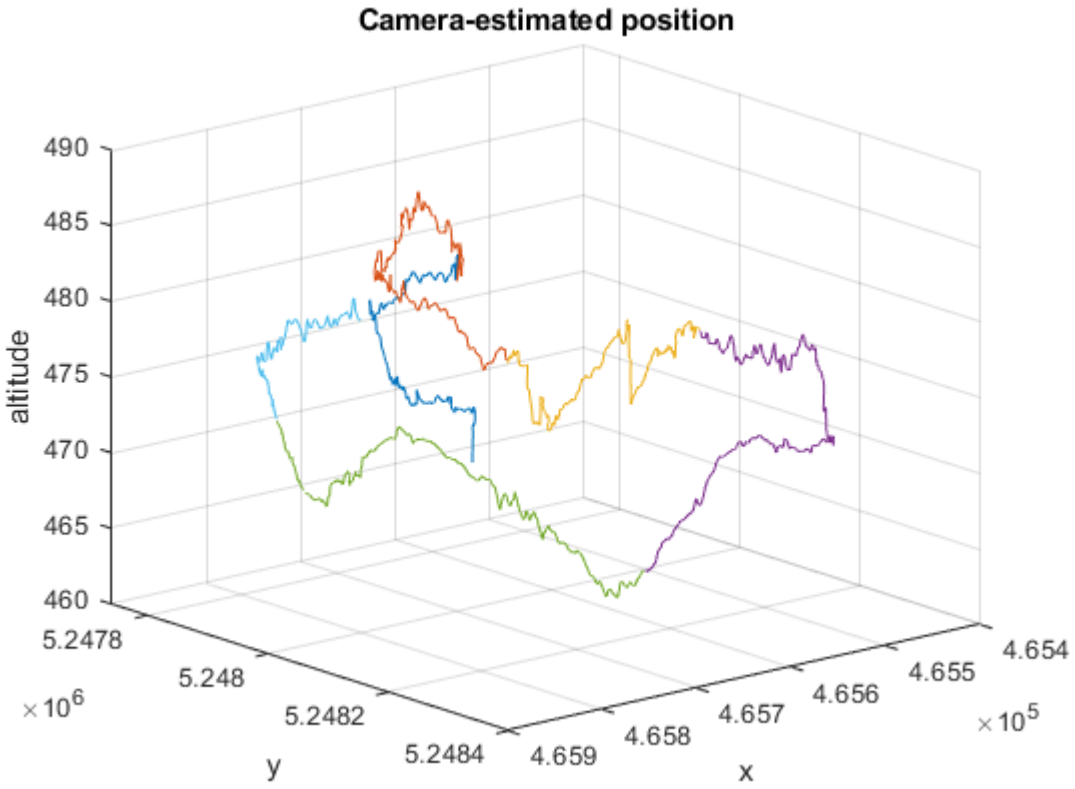
TAREAS

Tarea 1: Visualizar las trayectorias en una gráfica 3D

Programa un *script* para observar la trayectoria teórica del dron en una gráfica 3D. Llamaremos `T1.m` al *script*.

Repetir por cada uno de los seis ficheros de datos las siguientes fases:

- cargar el fichero n (comprobando que existe)
- seleccionar las columnas 2, 3 y 4 en sendas variables (latitud, longitud y altura)
- añadir a la gráfica mediante la instrucción `plot3(lat, lon, h, '.');`
- etiquetar los ejes y la gráfica



Tarea 2: Coordenadas y altitudes teóricas y medidas por GPS

En esta tarea vamos a comparar las trayectorias teóricas frente a las medidas con el GPS en 2D mediante *subplots*. El *script* se llamará `compareEstimations.m`.

En una única imagen se mostrarán dos *subplots* **para cada fichero**. En el de la izquierda se dibujará la trayectoria teórica del dron y en el de la derecha las diferentes altitudes de los mismos ficheros.

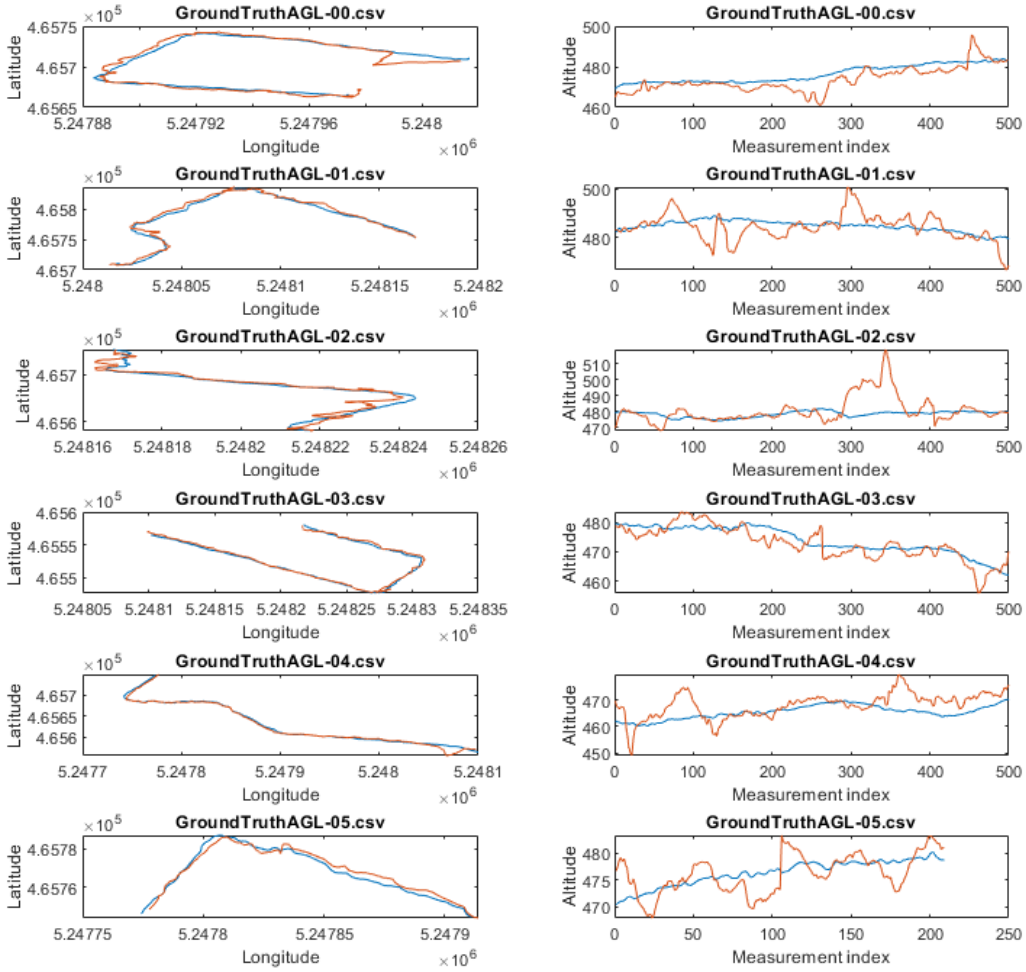
1. Para cada n (un número del 0 al 5) vamos a cargar los diferentes ficheros parciales `GroundTruthAGL-0n.csv`. Por cada fichero vamos a hacer dos *subplots* en una línea, en el de la izquierda las coordenadas teóricas (columnas 2 y 3) y las del GPS (columnas 8 y 9), y en el de la derecha las altitudes (columnas 4 y 10).

2. Para cada fichero parcial se mostrarán por pantalla las siguientes estadísticas **del GPS**: nombre del fichero, altitud máxima y mínima, y media. La salida puede tener el siguiente formato:

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

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

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



Tarea 3: Crear menú

Programa el script *mainMenu.m*, de manera que a la persona usuaria se le muestre el siguiente menú:

```
#######  MENU  ########:

1. Visualizar las trayectorias en una gráfica 3D

2. Coordenadas y altitudes teóricas y medidas por GPS

4. Calcular distancias y energías

5. Comprobar el exceso de desviación

0. Exit

Elige tarea:
```

Según la opción elegida, se ejecutará el *script* de la tarea correspondiente. Cada entrada del menú se corresponde con cada una de las tareas realizadas para el siguiente proyecto. Si el número introducido como entrada no es correcto, se mostrará un mensaje de error: *"La opción elegida no es correcta: elige una opción entre 0 y 5"*.

Antes de mostrar el menú, el *script* limpiará la consola. Al elegir una de las opciones, se mostrarán los resultados de la tarea correspondiente. A continuación se mostrará de nuevo el menú. Esta operación se repetirá mientras no se elija la opción *0. Exit*. No uses la función `exit()` ya que cierra Matlab/Octave.

Después de mostrar los resultados de una tarea y antes de mostrar el menú de nuevo, se preguntará a la persona usuaria que pulse cualquier tecla (*"Press a key to continue..."*). Ello nos dejará tiempo extra para leer los resultados, ya que si limpiamos la consola inmediatamente después de imprimir los resultados no sería posible leer la salida del *script* elegido.

Tarea 4: Calcular distancias y energías

En esta tarea vamos a calcular la distancia recorrida en dos las trayectorias, en la teórica y en la medida con GPS:

1. Crea una función para calcular la distancia recorrida mediante la siguiente fórmula tanto para las columnas 2, 3 y 4 como para las 8, 9 y 10. Ten en cuenta que en el sistema de coordenadas usado la latitud y longitud están en km y la altura en metros, así que se deben pasar todas a metros.

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

   ```
   function d = calculateDistance(lat,lon,h)
   ```

   Para probar la función:

   ```
   >> lat=1:20;
   >> lon=1:20;
   >> h=0.01:0.01:0.2;
   >> recorrido=calculateDistance(lat,lon,h)
   recorrido =  26.8707
   ```

2. Escribe una función `function e = calculateEnergy(altitudes)` para saber la energía gastada en elevarse calculando cuántos metros ha subido el dron (sin tener en cuenta los que ha bajado) y multiplica por 98 mAh/m (miliamperios*hora por cada metro subido). Recuerda que la forma de Matlab para obtener los valores positivos de un vector es `vp=v(v>0);`

   Prueba la función con el ejemplo:

   ```
   >> energy= calculateEnergy([5.1 5.4 5.2 6])
   energy =  127.4000
   ```
   (|5.4-5.1|+|5.2-5.4|+|6-5.2|)*98= 127.4 mAh

3. Vamos a medir la cantidad de veces en las que la altura teórica y la medida por GPS difieren más de 10 metros. Llamaremos a esta diferencia con la expresión *medida dudosa*. Escribe una función que calcule la proporción que existe entre el número de estas *medida dudosa* frente al total de alturas.

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

   Para probar la función tienes este ejemplo que devuelve el 25% de diferencias mayores de 10 entre los valores de los dos vectores:

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

4. Crea un *script* llamado `calculateDistanceAndEnergy.m` en el que se carguen todos los ficheros y concatenen las matrices, y se apliquen las funciones anteriores.

   Inicia el valor de la matriz `data` al valor *matriz nula* con `data=[];`

   Carga el fichero con `dlmread` en la matriz `dataf`.

   Une la matriz cargada a la matriz final con la instrucción

   ```
   data = [data ; dataf];
   ```

   Usa las funciones `calculateDistance(lat,lon,h)` y `calculateEnergy(h)` para crear un mensaje como el siguiente.

   ```
   El recorrido planificado/teórico es 1915.63 m
   Metros subidos 139.56 m
   Energía consumida 26405.75 mAh
   El recorrido estimado por el GPS es 2722.26 m
   Metros subidos 660.81 m
   Energía consumida 127938.11 mAh
   ```

   Usa la función `doubtfulAltitudePercentage(h1, h2)` en el *script* para imprimir un mensaje de este tipo:

   ```
   La proporción de medidas dudosas (diferencias mayores de 10 entre
   alturas teóricas y medidas con GPS) es de 6.69%
   ```

Tarea 5: Calcular la variabilidad de la altitudes

Para observar la variabilidad de los vectores de altitud vamos a calcular la desviación típica de los vectores usando *ventanas*. Llamaremos ventana a un sub-vector compuesto por n valores anteriores y n valores posteriores a cada posición del vector resultado.

Por lo tanto, vamos a definir una función:

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

en la que vamos a calcular un vector de desviaciones a partir de cualquier vector `v` con una ventana de tamaño `tamVentana`. En la figura se representan los vectores de desviaciones resultado de aplicar la función a las dos altitudes con un `tamVentana` de 100.

Supongamos que la variable `tamVentana` contiene ese valor n, por ejemplo 50. El tamaño del vector resultado `desv` será la longitud de `v` dividido entre el `tamVentana`. Crearemos para ello un vector de ceros de dicho tamaño, indexado por la variable `d`.

Iremos creando ventanas desde el punto `d-100` hasta el `d+100` (si `tamVentana` vale 100) seleccionando esa parte del vector `v`, y después de aplicar la función `std()` a esa ventana anotaremos ese valor en la posición pos del vector `desv.` Como últimos pasos incrementaremos en 1 el valor de `pos` y moveremos la ventana 100 posiciones hacia la derecha.

Para comprobar que funciona correctamente la función, podemos ejecutar estas instrucciones:

```
>> h=1:20;
>> tamVentana=5;
>> desvtip=calculaDesv(h, tamVentana)
desvtip =
   1.8708    3.3166    3.3166    3.0277
```

También:

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

Crea el *script* `T5.m` que aplique la función a los dos vectores de altitud completos como en el anterior ejercicio, para obtener una gráfica como la siguiente: