Student s1833951 ILP course work 2

Software architecture:

There are totally 7 class used for the course work2.

App: Used for creating the txt and geojson file when the path is determined, it is also responsible to add the start point before the route is complete and move the drone.

Drone: Planning the route when get the sensor and words data from webserver

move: when there is a no fly zone up front, it will work out a route to get around the no fly zone without going in to it.

noflyzone: a no fly zone object

sensor: a sensor object, contains battery, reading and a words object.

Webserver: used to get the data from local host

words: word object, contains coordinate

overview:

The procedure of the algorithm works:

It first get the sensor data, and used sensor data to get a list of what3words data, we process the what3words data in the drone.prediction, which will planned a route to get to every position that each of the object in the words list contained. This is done by dividing the map in to 4 zone according to the center point given by the no fly zone object. Then it will calculate the shortest path in each zone and

connect them together. After the prediction route is get, the drone will use the code inside App to fly.

Class documentation:

WebServer class:

This class contains 4 function:

readweb

is use to get to the localhost and drag data from it using a string url given.

getMap:

this is use to drag the data from the sensor, it will read the input string and form a address, then use the address to drag the data from the localhost by calling readweb, then return a arraylist of sensor object.

getWords:

this is used to get the arraylist of words from the localhost by having calling getmap to get list of sensor and use the list to get the words data for each of the sensor. For each sensor, it divide the string “location” in it by “.”, and get the list of words from localhost.

getbuilding:

it get the polygon list, which is 4 polygon in total.

Most of the code in this class is copied from the PPT given

Drone class:

This class takes a list of words, which returns by the getWords in webserver, and two double which is the input longitude and latitude value as the starting point. The last object is a noflyzone, which contains a list pf polygon for determine if the route is crossing the noflyzone or not.

dis:

used to find distance of the two point given, return the distance.

prediction:

This class first initial 4 arraylist of words, represent the zone, then we get the center point from the noflyzone object, the center point is get by using the longitude and latitude of the meadows and KFC add together and divide by two, this can be seen in the noflyzone class. This function will retuned the whole route for the drone.

The first for loop is used to allocate the list of words to the 4 different zone, by using the longitude and latitude value. The first zone is in the up-right, second zone is in the up-left, third zone is in the down left, forth zone is in the down right.

In the next for loop, I find the nearest point to the start point in zone3, and add it to the “result”(which is the return value of the prediction function, means the sequence of words it should reach), this point will be used as the first words to start. After we add the words, we delete the words in the zone3, means it is

reached.

The next 4 for loop is used for getting a list of words by using the “nearest” function and add to the “result”. When each time result add a words, the zone will delete one in itself, and the next time the zone list will be called to the “nearest” with smaller size.

nearest:

this function takes 4 arguments, the list of words ”a” is the remaining words in the zone, the words ”b” is the start words, list of words “result” is the list of words exists in the route already, and a noflyzone object. It will resutn a word that is nearest to the given words “b”.

The for loop, will calculate anc compare the distance of the “b” to all the other words in the “a” to find a shortest one, and if the b to a route is crossing the no fly zone, it will call the move.getnear function that returns a list of point that can get to the target point without going in the no fly zone, then I use for loop to calculate the total distance of the list of point returned by the move.getnear. if the route from b to a is not crossing no fly zone, it will calculate the distance directly.

The “crossing” used here is for determined if b to a point a is crossing no fly zone, see below the description.

The “if” below will compare the stored near and the next words in the zone list, if the next one is nearer, it will replace the current “near” words.

It return the “near” words in the end

Iscrossing:

Take 4 point see if the line between point1 and point2 is crossing with the line between point3 and point4. Return true is crossed.

Crossing:

Main function to determine if crossed the no fly zone or not. It get two point see if from point1 to point2 is intersect with the no fly zone.

It first initial the 4 outer limit from the noflyzone object, and see if the route is intersect, if it is, it means drone is going out the area. I store the Boolean for this check.

Next for loop I check if the route is intersect with any of the 4 polygon, this is done by calling two point from the each polygon and see If they intersect by calling the iscrossing function.

findzone:

similar to the “crossing”, instead, it returns a list of point not a Boolean. This function will first check if the route is intersect with no fly zone, then it will record the two point the route intersect with, and return the whole polygon it intersect with. It will add all the polygon point in the list and return the list of point.

For example, if the route is intersecting with david hume tower, this function will return all the point in the david hum tower no fly zone polygon list. This

function is use to find which nofly zone it is intersect with.

Move class:

compare:

if the two double value is equal, it will return true.

getnear:

this function is supposed to return a list of point that help from point a to get to point b while not getting in to the no fly zone.

first for loop, it first calculate the difference of the two point “diffx” “diffy”, x is longitude y is latitude for rest of the argument name. then it tries to approach the point2 by adding 0.1 * diffx to longitude and 0.1 * latitude each round to the point1 position. If it find itself encounter the no fly zone(if “store” to “next”), it store the last available point and calling the “findnearestp” to find rest of the point.

findnearestp:

it first use the route from a to b(which encounter no fly zone) to calling “findzone” to return a list of point representing the no fly zone it encounter with, then it use the list of point to find exactly which two point created the line the route encounter, and record the two point, pass to the “findroute”.

findroute:

since it get the two point and the whole noflyzone list of point it encounter with, this function create two possible route and pass it to the “findroute2”.

The two for loop, each find a route in different direction. For example:

David hume tower got 4 point, a,b,c,d, the route is encounter the line ab, so this function will return two list of point: “a,d,c,b” and “b,c,d,a”, and pass the two route to the next function.

findroute2:

Following the two route pass to this function, it will process each move and see if the target point can reach the final point directly, if it is achievable, it will store the last point and the final point, break. If it is not, it will keep adding each move. The two for loop is responsible for the two route.

For example “a,d,c,target” “b,c,target” is get for each route, it will calculate the total distance for each choice and use the nearer one and return it.

fix:

this function add “mistake” on both x and y value of the point, since when the drone fly, it can’t precisely follow the prediction path.

APP class:

output and file:

this get a list of point, which represent the route, and a list of feature, which is the sensor value. It will call the “file” function and produce a geojson output file.

symbol:

it will get a feature and a reading and battery for the sensor, then fill in the symbol for the sensor. If the battery is too low or the sensor gives reading “NaN”

or “null” it add cross and break. For rest of the reading it will add the symbol according to the needs.

marker:

it return list of feature represent the sensor reading. By calling symbol to each sensor object and add to the list.

filetxt:

produce the txt file.

Main:

It get the route from “prediction” and transfer the list of words to list of point. When one of the move is crossing the noflyzone, it will call the move.getnear and add all the point in the returned list of point.

After this, it will add the start point to the end of the list of point, means the route is completer.

filetxt:

it is the main function responsible to control the drone to move, and produce a geojson file and a txt file in the end.

newsensor:

this function is trying to identify if new sensor is detected. If it does, it will call a break in “filetxt”, which will end the loop and looking for the next point.

gett:

this function helps to avoid the no fly zone, but when I added this function in the “txtfile” it returns error “outofmemory heap”, and I failed to fix it in the end.

record:

this helps to identify and record the feature of sensor and returns a list of feature.

findsender:

this helps to record the string of the sensor when one is in range, and return the location string back.

angel:

this function will record two point and give back the degree in multiples of 10.

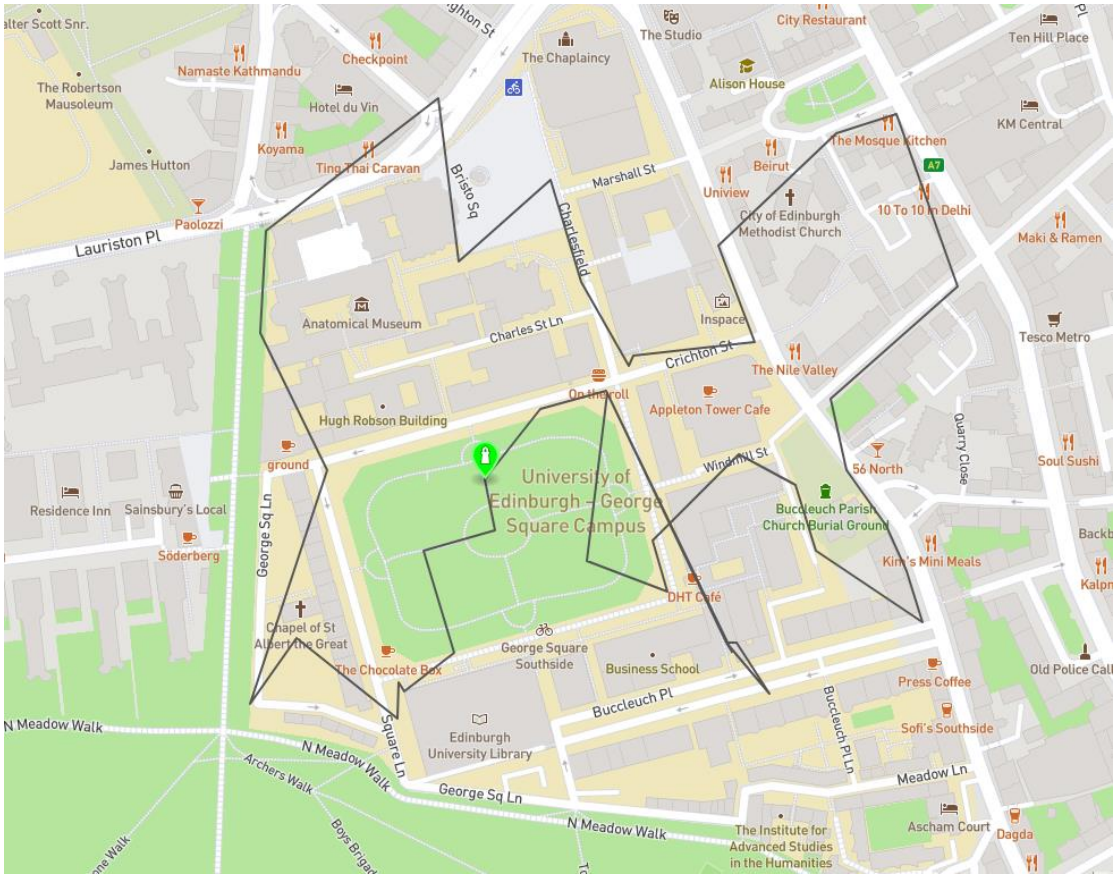
move1:

This will get a degree and a point returns the point it will end up with.

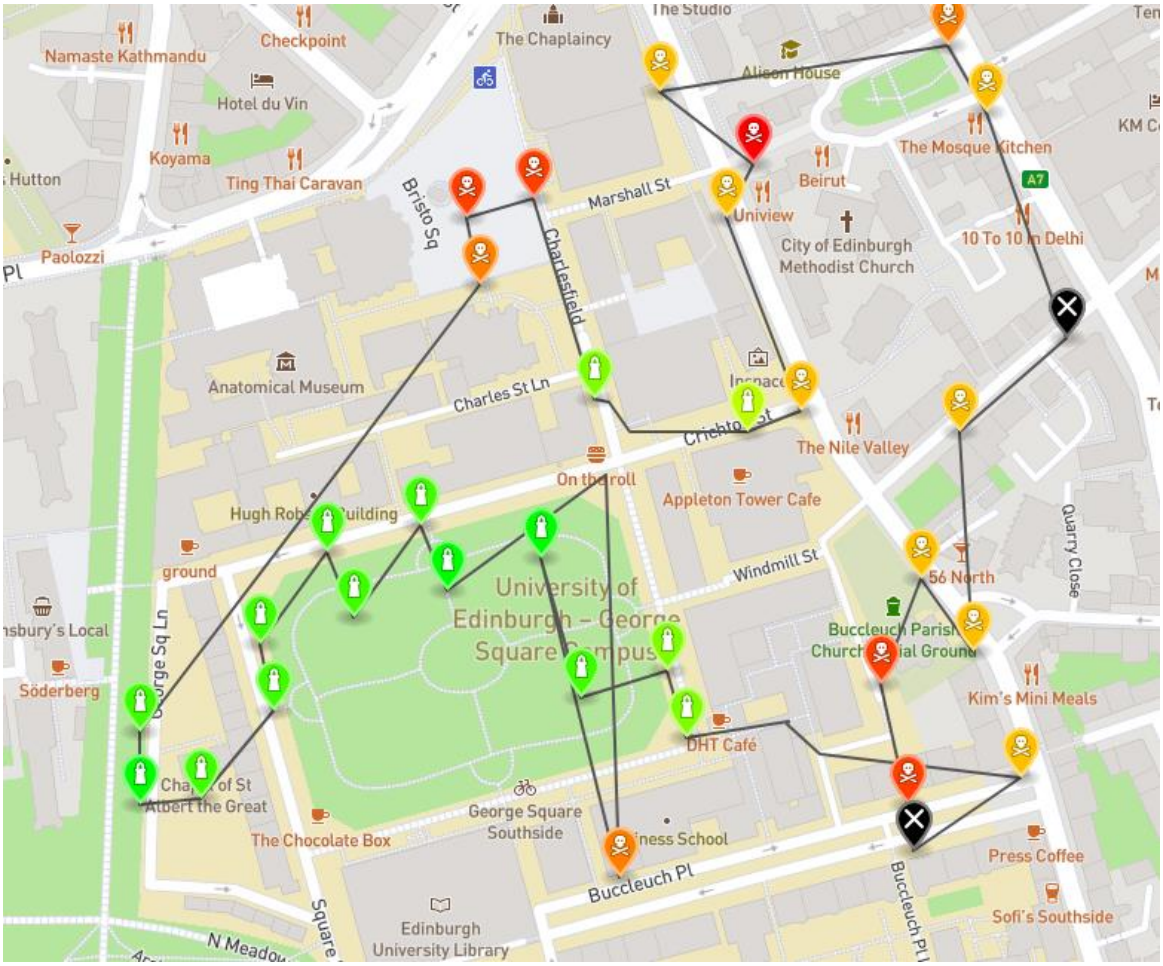
Drone control algorithm:

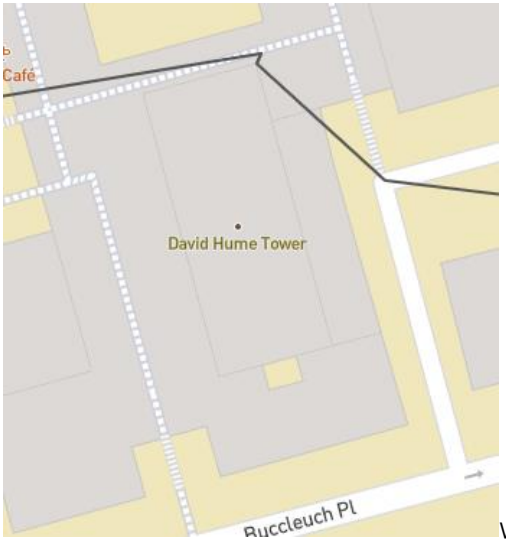
date: 12/01/2021 actual path

the predicted path:


The Studio
City Restaurant
Ten Hill Place
Checkpoint
The Chaplaincy
Namaste Kathmandu
Alison House
The Robertson Mausoleum
Hotel du Vin
KM Central
Koyama
The Mosque Kitchen
James Hutton
Ting Thai Caravan
Beirut
Bristo Sq
Marshall St
Uniview
A7
Paolozzi
City of Edinburgh Methodist Church
10 To 10 in Delhi
Lauriston Pl
Charlesfield
Maki & Ramen
Anatomical Museum
Inspace
Charles St Ln
Tesco Metro
Crichton St
The Nile Valley
Appleton Tower Cafe
Hugh Robson Building
Quarry Close
ground
Windmill St
56 North
Soul Sushi
University of Edinburgh - George Square Campus
Residence Inn
Sainsbury's Local
Buccleuch Parish Church Burial Ground
George Sq Ln
Söderberg
Kim's Mini Meals
DHT Café
Chapel of St Albert the Great
George Square Southside
The Chocolate Box
Business School
Old Police Call
Press Coffee
Buccleuch Pl
Buccleuch Pl Ln
N Meadow Walk
Square Ln
Edinburgh University Library
Sofi's Southside
Archers Walk
N Meadow Walk
Meadow Ln
George Sq Ln
N Meadow Walk
Ascham Court
Dagda
The Institute for Advanced Studies in the Humanities

date : 15/06/2021





we can see it is avoiding the no fly zone.

Actual path

The part of the code supposed help to avoid the no fly zone returns heap error and I failed to fix it. So it just follows the prediction path directly.

The algorithm is trying to find a route using all the sensor location before the drone moves, when it encounter noflyzone, it will fly near the no fly zone with a constant distance. The drone will use the prediction path to fly, however, the final output is not promising.

The way to produce **prediction** geojson map: uncomment the “output” at the bottom of the “App” class

```
System.out.println(fin.size());

//output(fin, marker(sensorsList,w3wList),input[0],input[1],input[2]);
////////////////
```

The way to produce **actual** txt and geojson file:

Uncomment the “output” at the bottom of the filetxt:

```
        }
    }
    System.out.println(result.size());
output(result,result2,day,month,year);
```

I am terribly sorry for not finishing the test section, again.