Informtics Large Practical Coursework 2

Qiwen Deng s1810150

December 4, 2020

Abstract

The program is mainly about controlling the drone flying over the campus in order to collect the data from the sensors to generate an air quality map. The report includes 3 aspects to describe the program, software architecture, class documentation and the controlling algorithms.

1 Software Architecture

1.1 Class Description

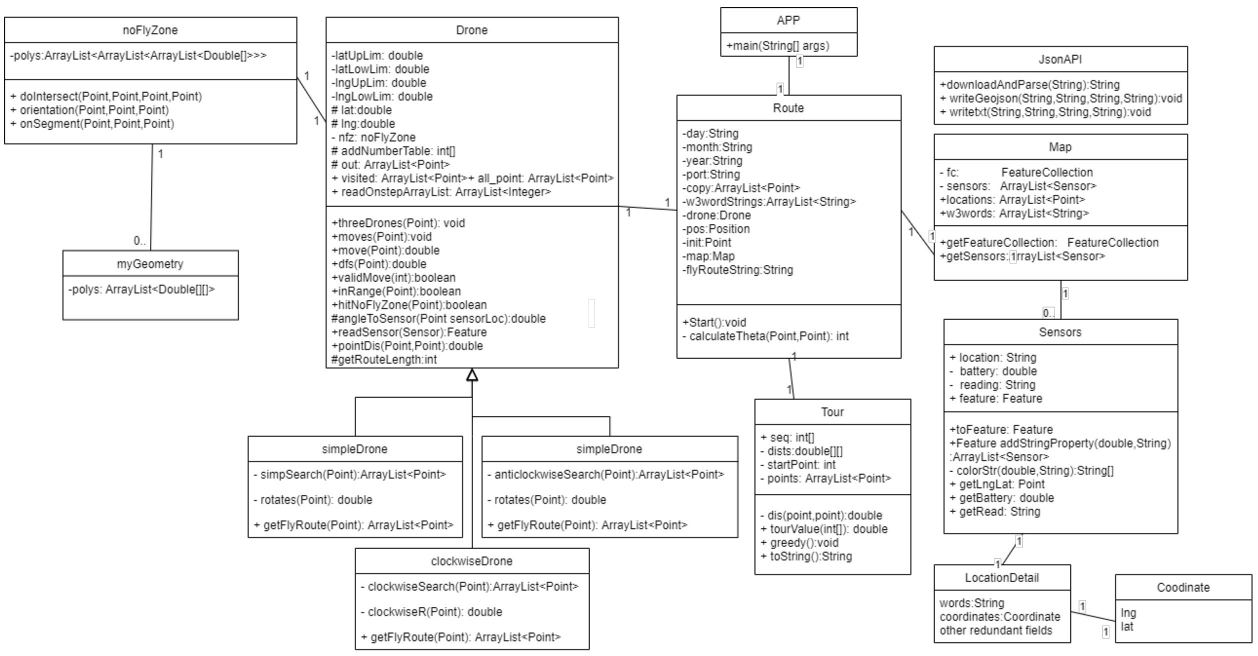

noFlyZone
-polys:ArrayList<ArrayList<ArrayList<Double[]>>>
+ doIntersect(Point,Point,Point,Point)
+ orientation(Point,Point,Point)
+ onSegment(Point,Point,Point)
myGeometry
-polys: ArrayList<Double[][]>
Drone
-latUpLim: double
-latLowLim: double
-lngUpLim: double
-lngLowLim: double
# lat:double
# lng:double
- nfz: noFlyZone
# addNumberTable: int[]
# out: ArrayList<Point>
+ visited: ArrayList<Point>+ all_point: ArrayList<Point>
+ readOnstepArrayList: ArrayList<Integer>
+threeDrones(Point): void
+moves(Point):void
+move(Point):double
+dfs(Point):double
+validMove(int):boolean
+inRange(Point):boolean
+hitNoFlyZone(Point):boolean
#angleToSensor(Point sensorLoc):double
+readSensor(Sensor):Feature
+pointDis(Point,Point):double
#getRouteLength:int
simpleDrone
- simpSearch(Point):ArrayList<Point>
- rotates(Point): double
+ getFlyRoute(Point): ArrayList<Point>
simpleDrone
- anticlockwiseSearch(Point):ArrayList<Point>
- rotates(Point): double
+ getFlyRoute(Point): ArrayList<Point>
clockwiseDrone
- clockwiseSearch(Point):ArrayList<Point>
- clockwiseR(Point): double
+ getFlyRoute(Point): ArrayList<Point>
APP
+main(String[] args)
Route
-day:String
-month:String
-year:String
-port:String
-copy:ArrayList<Point>
-w3wordStrings:ArrayList<String>
-drone:Drone
-pos:Position
-init:Point
-map:Map
-flyRouteString:String
+Start():void
- calculateTheta(Point,Point): int
Tour
+ seq: int[]
- dists:double[][]
- startPoint: int
- points: ArrayList<Point>
- dis(point,point):double
+ tourValue(int[]): double
+ greedy():void
+ toString():String
JsonAPI
+downloadAndParse(String):String
+ writeGeojson(String,String,String,String):void
+ writetxt(String,String,String,String):void
Map
- fc: FeatureCollection
- sensors: ArrayList<Sensor>
+locations: ArrayList<Point>
+w3words: ArrayList<String>
+getFeatureCollection: FeatureCollection
+getSensors:1rrayList<Sensor>
Sensors
+ location: String
- battery: double
- reading: String
+ feature: Feature
+toFeature: Feature
+Feature addStringProperty(double,String)
:ArrayList<Sensor>
- colorStr(double,String):String[]
+ getLngLat: Point
+ getBattery: double
+ getRead: String
LocationDetail
words:String
coordinates:Coordinate
other redundant fields
Coodinate
lng
lat
1
0..

Fig 1: UML Class diagram

First of all, initiate the program and acquire the needed data by using HTTP protocols from a server. For convince, I converts all the data into the instance of mine, such as detail.json will be converted into LocationDetail, no fly zone geojson will be converted into noFlyZone.

The UML class diagram illustrated the general class structure of the program. As can be seen, it consists of 15 classes. APP, Route, Map,Sensor, Tour, Drone,simpleDrone, clockwise-Drone, anticlockwiseDrone, noFlyZone, myGeometry, LocationDetail, Coordinates,JsonAPI and WirteFile. The UML graph is shown as a hierarchy of functionalities, these classes are identified by following the scheme of "Divide can Conquer", since we are breaking down the task into small pieces.

• App The App class contains the main method only. It is just calling out the Route instance. This keeps the class simple.

• Route. The Route class is responsible for creating the other 3 instances(Map, Drone, Tour), we using them to generate the algorithm, drone control methods and the map. The start method is where I imply the drone life cycle, the location information of next sensor will be sent to the drone instance, and receive the information it returns(how many steps? and what about the feature of this sensor). After we sent all sensor details to the drone, or we run out of the 150 steps, we write the fly path txt file and draw the air quality map.

• Map The Map class is basically just storing all of the sensors, it receives the string parameters from the route class, More importantly, it set up a HttpClient communication with the sever and create 33 sensor instances.

• Sensor Sensor class is used to efficiently parse and store the air-quality-map.json file into the instances in the program. In this class, I also have the method to transfer the sensor entity into a feature, and it will wait a read signal(the addStringProperty method) to be able to produce a feature with the air quality information.

• LocationDetail LocationDetail is just used to parse and store the coordinates of a specific sensor according to its what 3 words from a file called detail.json. It is necessary, since the sensor object we got before this step has no numerical coordinate information, we need this to tell the drone where the sensor is.

• Coordinate A class to represent the coordinate property of the LocationDetals we stored from the last step. Using this class, we are able to obtain the real longitude and latitude of the sensor from its corresponding detail json. file.

•Tour This is the class where I calculate the distances between all sensors and give a planned path for later uses.

•Drone Drone class includes the functional methods of the drone, such as how it moves and how it will read the sensors and what messages it carries.

•The 3 sub-classes of drone(simpleDrone, clockwiseDrone and anticlockwiseDrone) are the basically the different ways to overcome the non-fly-zone problem. I choose to create 3 sub-classes rather than just write 3 different method is because I would like to make it extendable. Considering the future development.

• NoFlyZone is the class where to download and parse the non-fly-zone into the object of part of the programe.

1.2 Class Interaction

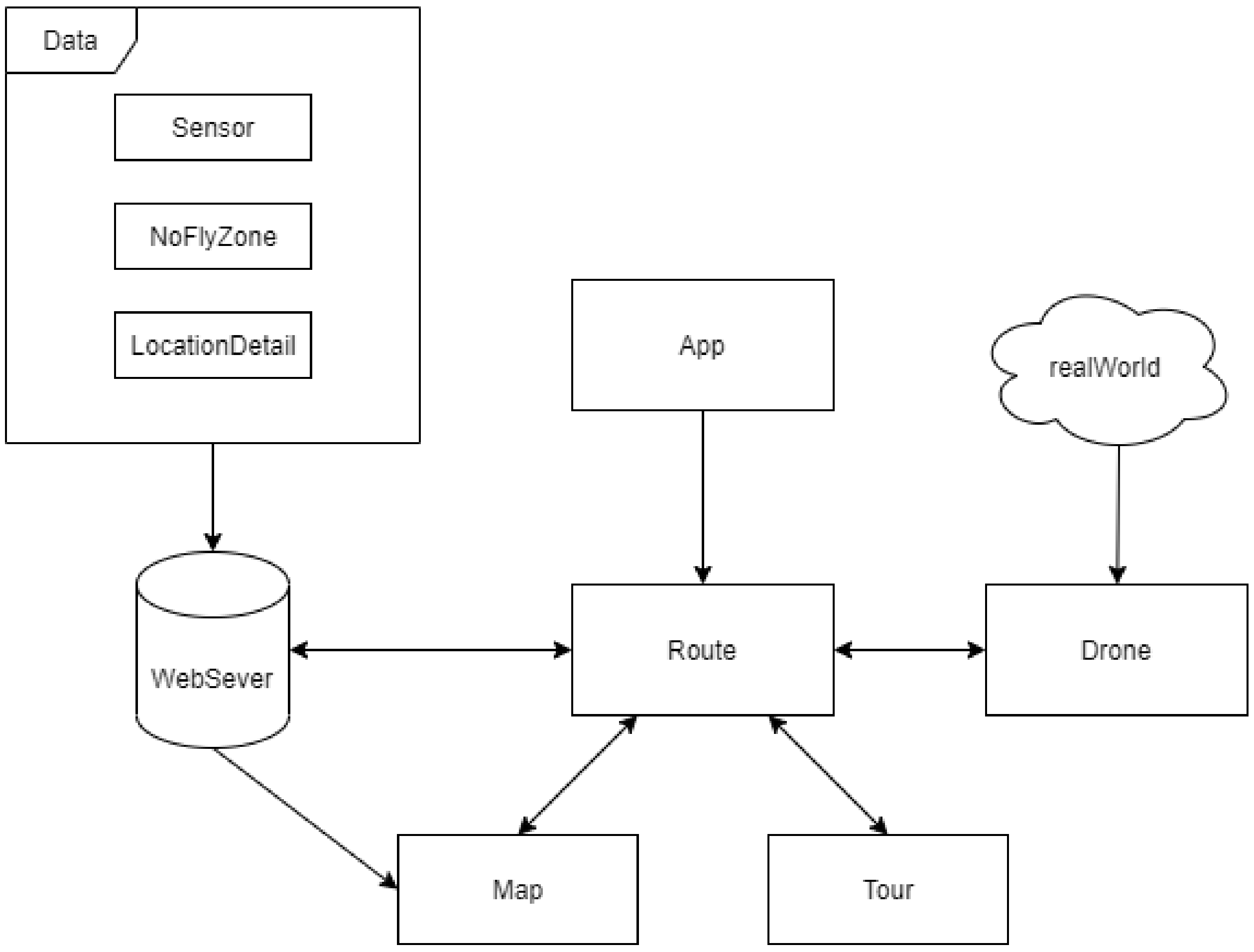


Fig 2: Class Interaction Diagram

The diagram above illustrates the class interaction. As can be seen, the Route class is the 'core class', since it communicates with basically all of the classes. The app class is the unique, which only sends message to route but never receive anything from it.

1.3 Workflow

1. App class read input and process it to the Route instance to initiate the program.
2. The Route class instance receives the message,it creates some objects of other classes and separate the data to the created objects.
   (a) The first object created will be the map object, it parse the air-quality-data.json, creates 33 sensor objects. After, the sensors are stored, we still need to make some

more steps in order to get enough details of the sensors. Since, when we created the sensor object, its coordinates do not come together with the information we get at this step. Additinonal step of connecting the sever, use the LocationDetails class instance to combine those information is needed.

   (b) The second object created will be the drone object, the messages of the initial position will be published to create the object. Meanwhile, the no-fly-zone,instance is created in the drone class, since the no-fly-zone is fixed object in the map, there is no message needed to be sent for this.

   (c) Te third object will be the Tour instance. Since we have got the sensor locations generated from the previous steps and the initial position of the drone. We are able to produce a route that visit all of the sensors with an acceptable costs.

3. Afterwards, the Route instance will try to send the message of next sensor to visit to the drone instance. The drone instance will determine how it is going to reach there within the drone class methods, and what to do if the drone is about to hit the no-fly-zone. The 3 subclasses of the drone class describes the technique of how the drone is going to make it.

4. Drone classes,the fundamental of 3 ways to avoid invalid flight step is similar which is at the moment of the drone is about to crash on the no-fly-zone when it is straightly moving towards to the next sensor that route class send to it. We will control it to change its direction, the simpledrone class is just trying to change for a smaller angle first, if it is not working, then it will change for a bigger angle, the clockwiseDrone is always changes the angle by -10 degree, where the anticlockwise is change the angle by +10 degree.

5. When we detect that the desired sensor is in range, we read it and convert it to a feature. In the Route class

6. Finally, when all of the sensor are read or the step count is exceed 150, we will output the current information to geojson and txt in the Route class.

2 Class Documentation

In order to make the program easier to maintenance, we design the program as robust and simple. The idea of separate the task into small pieces is held.

2.1 App,java

- Methods

public static void main(string[] args)
The app class is just contain a main method, it creates a Route object and parse the input.

2.2 Route.java

1. Private Fields(Strings)
   •day:String, month:string,year:string
   These private variables are used to generate other objects.
2. Private Fields(Object)
   • drone:Drone, pos:Tour, init:Point, map:Map
   These are the object of other classes, we will use them to out put our file results
3. Constructor
   Once the Route object is produced, it will produce another 3 objects.
4. Method
   •public method start()
   This method is where I imply the drone life cycle, the location information of next sensor will be sent to the drone instance, and receive the information it returns(how many steps? and what about the feature of this sensor). After we sent all sensor details to the drone, or we run out of the 150 steps, we write the fly path txt file and draw the air quality map.

2.3 Map.java

1. private fields
   •fc:FeatureCollection, sensors:ArrayList<Sensor>
   These fields are the set of 33 sensors, one is just the arrayList of the sensor, the other is the featureCollections of the sensor.
2. public fields
   •locations:List¡Point¿, w3words:List¡String¿
   These 2 fields are storing the sensor's coordinates and its coorsponding w3words.
3. Constructor
   In the constructor, the we parse and store 33 sensor info from air-quality-map info, but so far, we do not have numerical coordinates of the sensors.
4. Method
   •public getters()
   2 getter methods to return locations and w3words.

2.4 Sensor.java

1. Fields
   •location:String, battery:double,reading:String,feature:Feature

The field location, battery and reading are used to initialise the sensor object in the map class. The feature field is for transfer the sensor object into the geojson feature and return it.

2. Method
   •public Feature toFeature
   Transform into the geojson feature. However in this step, we only get a point without any string property or color with it.

   •public Fature addStringProperty
   The method to add the color and String properties to the feature generated from the method above.

   •public String[] colorStr()
   Return the necessary information for displaying sensors as a colored point of the geojson feature type. The transform depend on the battery and readings of the sensor.

   •public Point getCoordinate(String port)
   Set a http connection, parse the detail.json and get the coorespoding numerical coordinates of the W3words of the sensor.

2.5 LocationDetail.java

1. Fields
   •country:String, nearestPlace:String, words:String, language:String, map:String, squareOnMap:Square,coordinates:Coordinates
   This is the class where we parse the What3Word location into the program object.

2. Class
   The Coordinates coorespond to the what3word location. Only 2 fields for this class, lng:double, lat:double.

2.6 Tour.java

1. public fields
   •seq:int[]
   This is the final sequence of the sensors we are scheduled to visit.

2. priavte fields
   •dists:double[][],init:point, startPoint:int, points:ArrayList<Point >
   These fields are neccesary to calculate the cost(sum of estimate distances) of the current seq.

3. Constructor
   •Tour(ArrayList<Point> points,Point initPoint)
   Initialize the points, dists table, and the seq.

4. Method
   •private double dis(Point point, Point initPoint)
   A method to calculate the distance from all other point to a specific point. This is for producing the dist table and workout what should be the first point to visit.

   •public void greedy()
   Greedy algorithm to produce a sequence of visiting the sensors. The idea is always choosing the nearest point among the rest of the points as the next point.

2.7 Drone.java

1.

2. final private fields
   •latUpLim:double,latLowLim:double,lngUpLim:double,lngLowLim:double These 4 fields draws the valid square area that the drone is permitted to fly.

3. protected fields
   •lat:double,lng:double,port:String,addNumberTable:int[],out:ArrayList<Point>
   These are the field that the class share with the subclasses, since we need to alway keep the location of the drone, so we have lat(latitude), and lng(longitude), addNumberTable is a sequence of values that we considered to how to change our flight direction. The arraylist out, is the list of point we visited from the drone location that have just took the reading from the last sensor or its initial location(when the program started)

4. Public fields
   •all_point:ArrayList<Point>,readOnstepArrayList:ArrayList<Integer>
   The field is used to store the point we visited. ReadOnstepArrayList records at which step ,we read the sensor. These are public since we drawing the files in Routed class, return these 2 paprameter to output the file is needed.

5. Methods
   •public void threeDrones(Point nextSensor)
   In this method, I call there subclasses to fly if we are goint to meet a no-fly-zone on the way to next sensor, other wise the subclasses will not be created and called.

   •public void moves(Point nextSensor))
   A simple way to reach the next sensor, move until we are in range of 0.0002 degree to it. At the end of this method, I will add the point visited to the public field all_point, and record the current size of it store it in readOnstepArrayList.

   • public double move(Point sensorLoc)
   Function to move, this will be called in the method moves above. And return a angle.

   •protected boolean validMove(int theta)
   To determine whether the direction we trying to move to is valid. It will consider if the current go will hit the no-fly-zone, or it will fly out of the campus area or it is flyting to the point we have already visited for this go.

•private boolean hitNoFlyZone(Point q2)
q2 will be the point of the next point the drone wish to move to. The basic logic is if the line of current location to q2 has intercept with one of edge of any of the noflyzone polygons.

•public boolean inRange(Point sensorLoc)
Determind wither a point (the sensor) is in range of 0.0002

•protected double angleToSensor(Point sensorLoc)
Calculate the angle from the drone position to the sensor.

•public Feature readSensor(Sensor s)
Take the reading of the sensor, return the geojson feature of it.

•public double pointDis(Point p1, Point p2)
ultility function to calculate the distance between 2 points

•protected int getRouteLength()
Return the number of steps in one go to the next sensor, this method is mainly used in the subclasses. We need this in order to choose to which protocol to avoid crashes to the no-fly-zone is the most efficient.

6. the subclasses
•simpleDrone.java, clockwiseDrone.java,anticlockwiseDrone.java
The structure of 3 subclasses is similar, I divide them from their parent class is because for convenience of future extension and maintenance. The 3 subclasses are consists of its constructor and 3 methods. In this constructor, it just call a super(lng,lat,port), the methods are search, rotates and getFlyRoute. The search method is just similar to the method moves(the method in the parent class) and rotation is the spacial protocol for each of the class to chose how are they going to change its direction. The getFlyRoute is just return the route obtained in the method search.
Into more details, of how are the subclass drones changes its direction of moving, for simpleDrone, it first try to rotates for a small angle(10,-10) if it is not working then the angle it changes become larger and eventually if rotates 180 will still not be a valid move, then it will just return null arrayList, for clockwise, it -10 degree if the current direction is not valid. for aniticlockWise is +10 degree. Calling it clockwise and anticlockwise is because following this protocal, the drone will eventually, overcome the no-fly-zone just very near to the no-fly-zone in 2 direction, one is clockwise another is anticlockwise

2.8 NoFlyZone.java

1. public field
•ArrayList<ArrayList<ArrayList<Double[]>>> polys
Since, each point is 2 doubles, it is a Double[], the Arraylist¡Double[]¿ represented an edge of the polygon of no-fly-zone, and the ArrayList¡Arraylist¡Double[]¿¿ represent a single polygon. Since the no-fly-zone feature contains 4 geojson polygons, so we have this structure to parse and store the no-fly-zone.

2. constructor
   •public NoFlyZone(String port)
   The constructor takes a geojson feature as input, and then using geojson type casting break down them to my structure above.

3. Methods
   •public boolean doIntersect(Point p1, Point q1, Point p2, Point q2)
   The method determinds whether the line p1q1 and p2q2 intersects. it will check the orientation of the point of a line to points of another line and check if the point are onSegment.

   •private int orientation(Point p, Point q, Point r)
   In a simple word, the method will just check if the 3 point are colinear or check the direction if we go from p to q to r(clockwise or anticlockwise)

   •private static boolean onSegment(Point p, Point q, Point r)
   This method checks if the 3 points is colinear, then we check the unchecked point(did not check from the last orientaion method) whether it is on segment to the other line

2.9 MyGeometry

public field public ArrayList<Double[][]> coordinates; This is nessccary to parse the geojson no-fly-zone into object easily.

3 Drone Control

This section will discuss details of algorithm in contolling the drone and planning the sequence of sensors to visit.

3.1 Drone controlling

After discussed the details of how the drone will do for avoiding the no-fly-zone. We enlarge the scale. This section will discuss the life cycle of the drone and how it is recording the flight path.

- Flying angles towards the sensor

  First of all, since assessing the sensor do not the drone to reach the exact coordinate of the sensor, but anywhere inside the campus and outside the no-fly-zone will be fine. However, we do not really need to care about if moving at the direction of multiple of 10 degrees will not reach the area that can read data from sensor if we are too close to the sensor, but actually this problem do not exist for general situations(exclude the ones where no-fly-zone is considered in place). The reason is we can consider the 2 situation for the step before the drone can read the sensor.

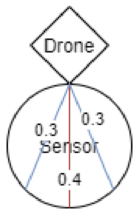


Fig2 the extreme close situation

The left figure shows that the drone is very close to the sensor's detecting area, since the detect distance is 0.0002 so the diameter for the area(circle) to be able to read the sensor is 0.0004(denote as just 4 in the graph). We can calculate the angle between the red lines and blue lines as the area they formed covers the fly route of next step which can make the drone read the sensor. acos(3/4) = 41 degree (ignore the decimals), thus we have 4 directions of freedom in this case.
Another Situation will be the distance between sensor and drone is just 0.0005, where on step direct towards the sensor will be able to read the sensor. Since we are only able to move at the direction of multiple of 10 degrees. This may not able to make the drone reach the area. In this situation, the solution is simple, we go for 2 steps for this one. Therefore we grantee that moving in at the direction of multiple of 10 degrees is not a problem. Thus at the moment when we trying to moving towards to the next sensor, we just calculate the direction and round it to the multiple of 10.

- Return to the initial position Return to the initial position is done by the drone take the initial position as another sensor. It will return to the intial position in range of 0.0002 degree.

3.2 Greedy

A greedy algorithm is a simple, intuitive algorithm that is used in optimization problems. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem. The procedure contains the follwing steps.

1. Find the point that is nearest to the starting initial position of the drone. Put it into a new integer list as a first item.
2. loop through the list of rest of the points. using the dists table(obtained when the Tour instance is created), we are looking for the nearest next point j for our current point i, and push j into the new integer list we have just created from the last step.
3. After we have produced a new sequence, deep copy it to the global seq variable.

Algorithm 1 Greedy

```
procedure GREEDY

    for $i = 0, i < 33, i++$ do
        $x = dis(point[i], init)$
        if $x < minimum$ then
            $minimum = x$
            $Start = i$

    $newSeq.add(start)$
    for $i = 0, i < 33, i++$ do
        $k = -1$
        $min = Double.max_value$
        for $j = 0, j < 33, j+$ do
            if *newSeq.contains(j)* then
                if *dists[newSeq.get(i) ¡ min]* then
                    $min = dists[newSeq.get(i)]$
                    $k = min$
        $newSeq.add(k)$
    for $i = 0, i < 32, i+$ do
        $seq[newSeq.get(i)] = i$
```

3.3 Avoiding Mechanism

In order to avoid the no-fly-zone on the campus, we need a efficient method to react with this. The core thought of avoiding mechanism is simple. The avoiding mechanism consists of 2 parts, detect if there is going to have a crash on the no-fly-zone and change the direction of current planned movement.The details of how to check the intercept will be discussed below.

- Detect Crashes
  The key idea to detect the crashes on the no-fly-zone is to check if the line segment of the movement will have intersections with the edges of the four polygons of the non-fly-zone. Given two line segments (p1, q1) and (p2, q2), find if the given line segments intersect with each other. Before we discuss solution, let us define notion of orientation. Orientation of an ordered triplet of points in the plane can be
  - counterclockwise
  - clockwise

- colinear

The following diagram shows different possible orientations of 3 points (a, b, c)

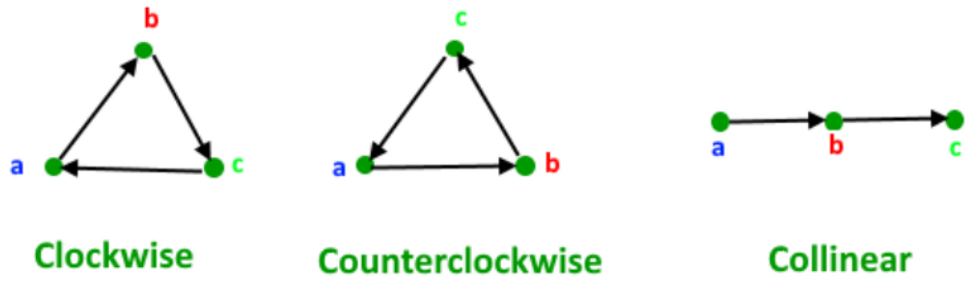
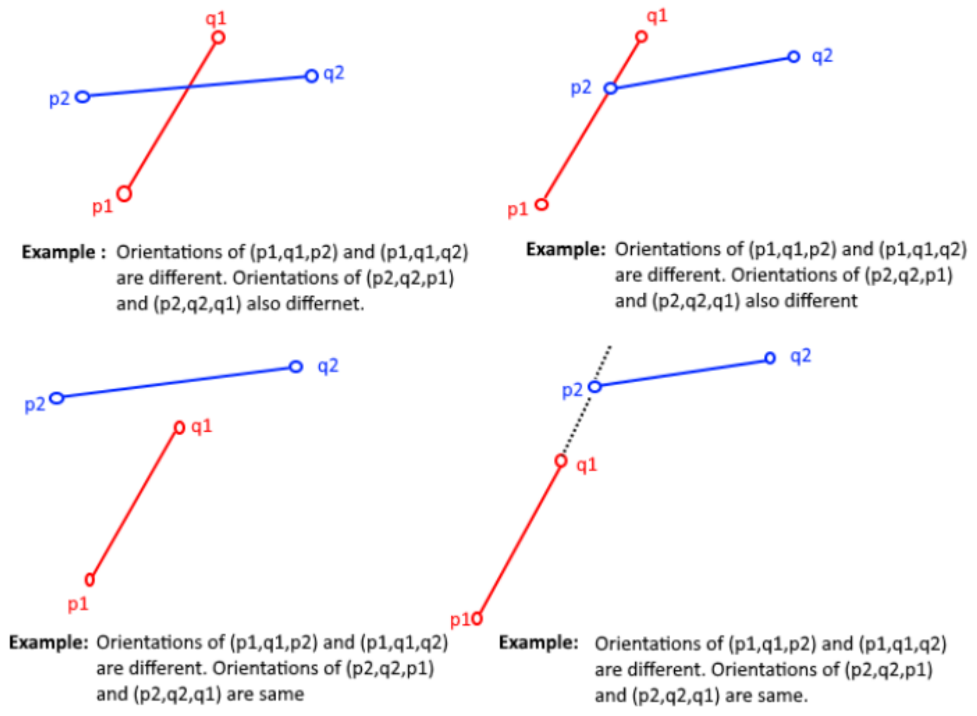


Fig3 relationships of 3 points

We have a definite theorem here. Two segments (p1,q1) and (p2,q2) intersect if and only if one of the following two conditions is verified

1. General Case:
    (a) (p1, q1, p2) and (p1, q1, q2) have different orientations
    (b) (p2, q2, p1) and (p2, q2, q1) have different orientations.[1]



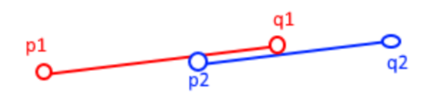
Example : Orientations of (p1,q1,p2) and (p1,q1,q2) are different. Orientations of (p2,q2,p1) and (p2,q2,q1) also differnet.

Example: Orientations of (p1,q1,p2) and (p1,q1,q2) are different. Orientations of (p2,q2,p1) and (p2,q2,q1) also different

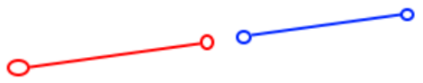
Example: Orientations of (p1,q1,p2) and (p1,q1,q2) are different. Orientations of (p2,q2,p1) and (p2,q2,q1) are same

Example: Orientations of (p1,q1,p2) and (p1,q1,q2) are different. Orientations of (p2,q2,p1) and (p2,q2,q1) are same.

2. Special Case:
   it occurs when, there is colinear situation among 3 points of the 4. Example:(p1, q1, p2), (p1, q1, q2), (p2, q2, p1), and (p2, q2, q1) are all collinear and
   (a) the x-projections of (p1, q1) and (p2, q2) intersect
   (b) the y-projections of (p1, q1) and (p2, q2) intersect orientations



Example: All points are collinear. The x-projections pf (p1,q1) and (p2,q2) intersect. The y-projection of (p1,q1) and (p2,q2) also intersect



Example: All points are collinear .The x-projections of (p1,q1) and (p2,q2) not intersect. The y-projection of (p1,q1) and (p2,q2) do not intersect

[1] Using this method, we can compute the line intercept very efficiently.

- Change direction
  The procedure contains the following steps.

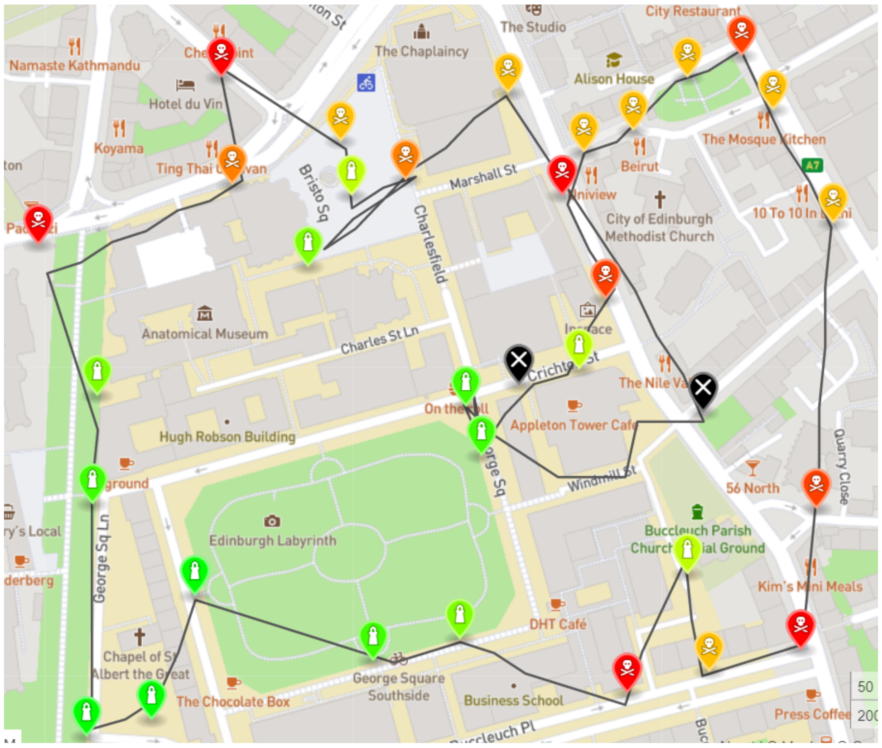
  1. the direction of move not valid?
     (a) for simple drone, try +
     $$(-1)^k * ((k+1)/2) * 10$$
     degree to the direction towards to the next sensor,
     (b) for clockwise drone, try -10*k degree to the direction towards to the next sensor.
     (c) for anticlockwise drone try +10*k degree to the direction towards to the next sensor

The reason why using extension of sub-classes instead of just write methods is because I would like to offer a more conveniences way to do the extensions If there are some other ways to do the real work on avoid the no-fly-zone.

For a consequence of my algorithm on this, the simple drone did well on some situation, such as 1 or 2 step is sufficient to overcome the no-fly-zone, but if more step is needed, it is likely to stuck in a loop. The other 2 drone is created to solve the problem, since we have a priority of a single direction to change our direction of flying, this will grantee that we can overcome the no-fly-zone, but just in some situation it is not as efficient as the first one.

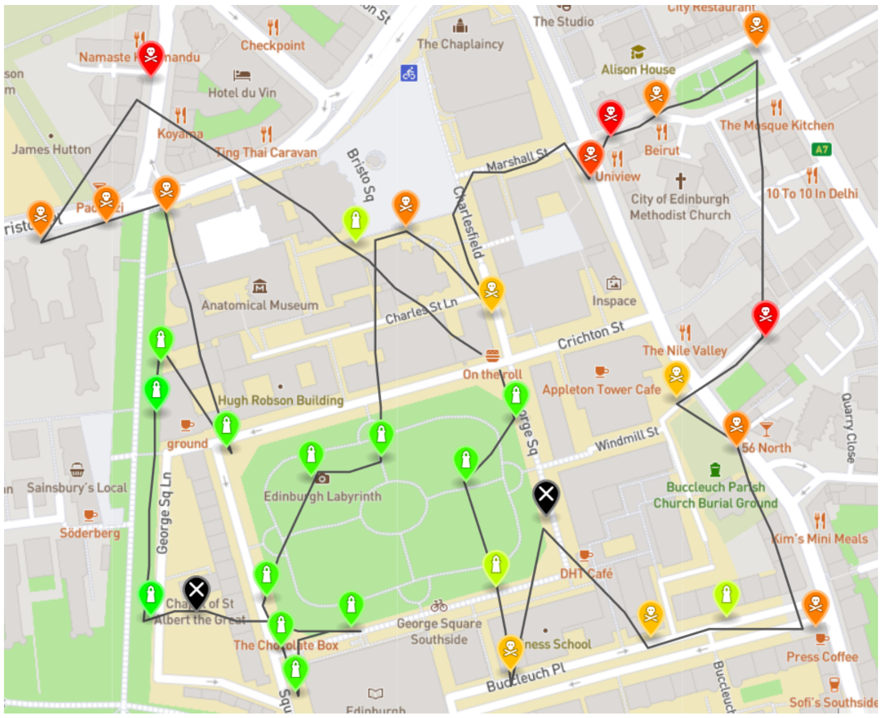
3.4 Computational results and features

To show the result outcome, I will also paste few examples of air-quality-map computed from data in the sever.



Readings -01-01-2020.geojson

As can be seen, if we just look at the local area performance, the greedy algorithm did a great job on planning the route. However, there are some route was not walked efficiently, since greedy considers only the closest between 2 sensor not the cost of the whole.
Then randomly look at another graph.


Readings -10-10-2020.geojson

The problem of this graph is similar.
All in all, this could be improved by change the planning route algorithm, such as a* or other algorithms that solve the TSP problem efficiently. But from these 2 graphs, we can see that there are no crossing with the no-fly-zone or any entangled path caused by algorithms as expected. Therefore we can guarantee that the avoid mechanism is doing very well.

4 Summery

In conclusion, the report describes the functionality of the drone control algorithm, by demonstrating its software architecture, detailed class documentation and the algorithms. There also some examples shown in this report, to show what is doing well and what is doing not very well.

References

[1] https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/