Autonomous Drone Program Report

Informatics Large Practical

Ethan Yang
s1862671

December 3, 2020

Contents

1 Software Architecture Description **3**
1.1 Introduction to the Autonomous Drone Program 3
1.2 Architectural Overview of the Program 3

2 Class Documentation **5**
2.1 Class LocationDetail 5
2.2 Class Coordinates 5
2.3 Class SensorDetail 5
2.4 Class FlightDetail 5
2.5 Class DataServer 6
2.6 Class GeoJsonServer 7
2.7 Class Drone 8
2.8 Class App 11

3 Drone Control Algorithm **12**
3.1 Introduction to the Algorithm 12
3.2 No-fly Zone Avoidance Algorithm 13
3.3 Swap Greedy Path Planning Algorithm 13
3.4 Path Re-Planning Algorithm 15
3.5 Path Minimization Algorithm 15

4 References **15**

5 Appendices **15**

1 Software Architecture Description

1.1 Introduction to the Autonomous Drone Program

This section will provide overview of the autonomous drone program. The autonomous drone program is developed to collect readings from air quality sensors distributed around the University of Edinburgh's Central Area. Drone flies around, avoids no fly zones and collects the sensor readings and produces scientific visualisations of the data in the form of GeoJSON file and produces a txt file which records flight information of drone.

1.2 Architectural Overview of the Program

This section will provide a comprehensive architectural overview by UML class diagram and briefly explain the purpose of each class designed and why these classes are the right ones for program. In addition, there are some explanations of advantages of this architectural design.

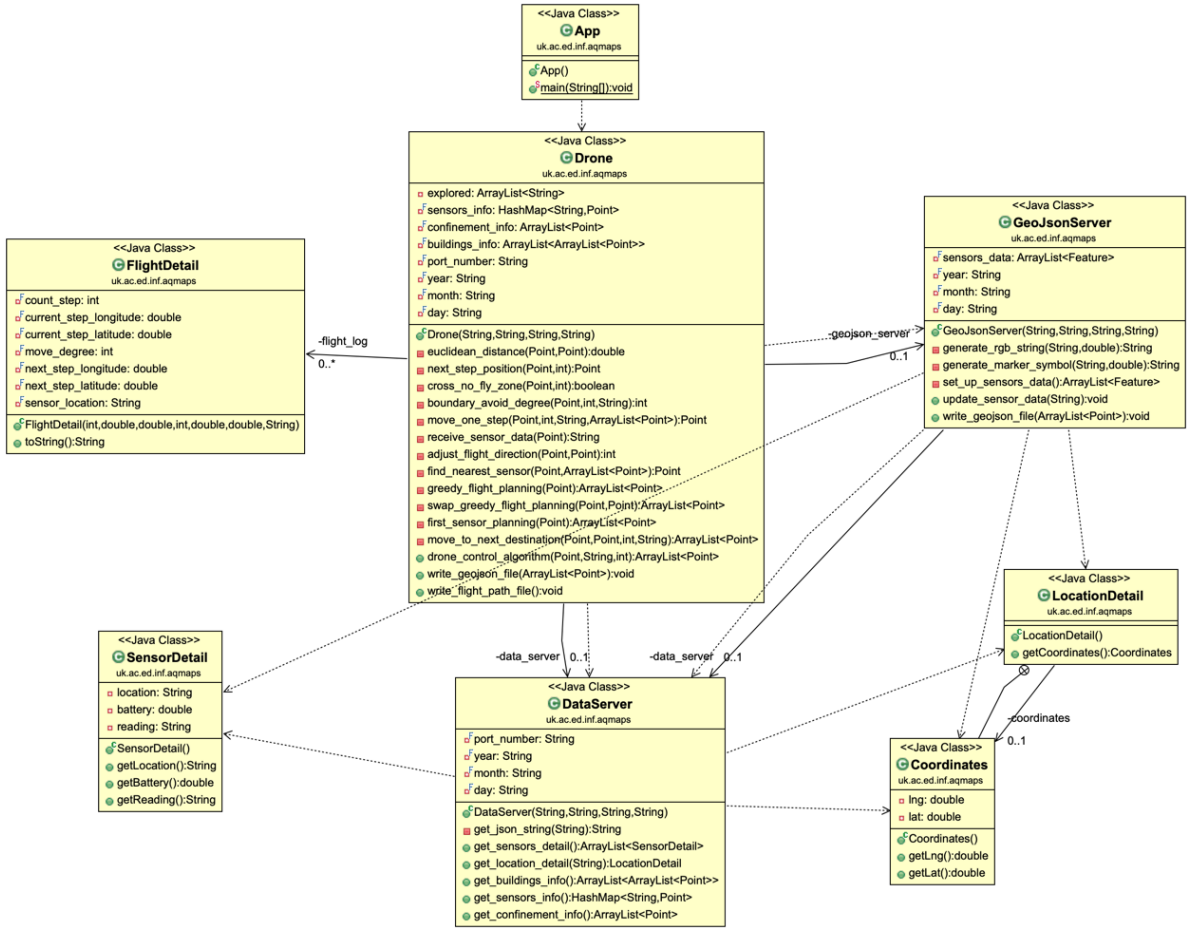

Figure 1: UML Class Diagram of Program's Architecture

Class App which is located at the top of the UML class diagram is the most significant class which contains the main method to run the program. This class only contains main method which integrates methods from Class Drone to write GeoJSON file and txt file.

Class Drone which is located in the middle is the biggest class and contains the most important algorithm to let drone fly around, avoid no fly zones and collect reading from sensors. This class focus on implementing drone control algorithm which is built by lots of components (small methods) and connect DataServer and GeoJsonServer which will be explained later to get data support and update sensor information to GeoJsonServer. It also write a flight path file after process of collection. This design makes Class Drone focus on single part of task without considering how to request data from web server and write GeoJSON file.

Class DataServer which is located at the bottom contains all methods that get data from web server. This class only focus on parsing JSON data from web server and processing data to data structures that Class Drone need. This class also help GeoJsonServer to update information.

Class GeoJsonServer which is located at the upper right focus on setting up initial state of GeoJSON map and updating information when drone receives sensor data. This class can write a GeoJSON file by using its field which stores sensors' information.

All these three classes are highly independent and focus on single part of tasks. Such design has great advantages in program maintenance. If DataServer fails due to change or error occurs in web server will not affect primary design of methods in other class. Changing the format of GeoJSON file will not affect any methods designed in other class. The error that drone fails to autonomously fly around only caused by some of components of Class Drone.

Class FlightDetail which is located at the upper left stores all the data needed to write flight path file. Each FlightDetail object stores one row of information in flight path file which contains step number, longitude and latitude of current and next position, direction of move in degree and location of sensor detected.

Class SensorDetail, LocationDetail and Coordinates are used to store the data parsed from JSON file in web server. They provide common objects for three classes introduced in the last few paragraphs to store some information about sensors' data and sensors' location.

Classes in last 2 paragraphs are just data classes. They are designed based on format of JSON file except FlightDetail class which is designed to separate writing function from drone control algorithm. This class benefits software testing because it does not need to write txt files every time when algorithm runs in the program.

There are some advantages of this architectural design. Small methods divide drone control algorithm into many single functions which are easy to maintain and find bugs when program has error. Besides the refactor of program, loosely coupled design in this architecture is really important for maintenance since the web server and output requirement change over time. There is no static field which is dangerous when share address in different objects.

2 Class Documentation

2.1 Class LocationDetail

Class Summary: This class is used to provide LocationDetail object which stores location data get from web server. This class is used when store web data into its object.

Field Detail:

private Coordinates coordinates - A field refer to a Coordinates object.

Constructor Detail:

public LocationDetail() - Construct a LocationDetail object by default.

2.2 Class Coordinates

Class Summary: This class is used to provide Coordinates object which contains longitude and latitude of what3words location. This is a sub class of Class LocationDetail.

Field Detail:

private double lng - A field refer to value of longitude.
private double lat - A field refer to value of latitude.

Constructor Detail:

public Coordinates() - Construct a Coordinates object by default.

2.3 Class SensorDetail

Class Summary: This class is used to provide SensorDetail object which contains location, reading and battery value of sensor. This class is used when store web data into its object.

Field Detail:

private String location - A field refer to sensor's what3words location.
private double battery - A field refer to value of sensor battery.
private String reading - A field refer to value of sensor reading.

Constructor Detail:

public SensorDetail() - Construct a SensorDetail object by default.

2.4 Class FlightDetail

Class Summary: This class is used to provide FlightDetail object which contains all the information needed for writing flight path file by Drone.

Field Detail:

private final int count_step - A field refer to step number of current step.
private final double current_step_longitude - A field refer to longitude of current step.
private final double current_step_latitude - A field refer to latitude of current step.
private final int move_degree - A field refer to direction of move in degree.
private final double next_step_longitude - A field refer to longitude of next step.
private final double next_step_latitude - A field refer to latitude of next step.
private final String sensor_location - A field refer to what3words location of sensor.

Constructor Detail:

public FlightDetail(int count_step, double current_step_longitude, double current_step_latitude, int move_degree, double next_step_longitude, double next_step_latitude, String sensor_location)
Construct a FlightDetail object given all the information needed for flight path file.

Method Detail:

public String toString()
This method returns String of a row of flight path file by using fields.
Returns: String of one row of flight path file

2.5 Class DataServer

Class Summary: This class provides methods to get data from web server and provides necessary information needed for Drone to use drone control algorithm and provides necessary information needed for GeoJsonServer to write GeoJSON file.

Field Detail:

private final String port_number - A field refer to port number of web server.
private final String year - A field refer to year of DataServer connection.
private final String month - A field refer to month of DataServer connection.
private final String day - A field refer to day of DataServer connection.

Constructor Detail:

public DataServer(String year, String month, String day, String port_number)
Construct a DataServer object given date (year, month, day) and port number.

Method Detail:

private String get_json_string(String urlString)
This method connects to web server and gets JSON String from web server based on specific URL String. URL String is the address of data on web server.
Parameters: urlString - URL String
Returns: JSON String

public ArrayList<SensorDetail> get_sensors_detail()

This method gets sensors' information from web server and stores in SensorDetail objects.
Returns: ArrayList of SensorDetail objects that represents sensors' information

public LocationDetail get_location_detail(String W3W_Location)
This method gets what3words info from web server and stores in LocationDetail object.
Parameters: W3W_Location - what3words location
Returns: LocationDetail object that represents what3words location

public ArrayList<ArrayList<Point>> get_buildings_info()
This method gets no fly zones info from web server and process data into a list of list of points of no fly zones which will be used by Drone.
Returns: ArrayList of ArrayList of Point objects contains points of no fly zones

public HashMap<String, Point> get_sensors_info()
This method generates a HashMap which contains 33 what3words locations of sensors as keys and corresponding points as values which will be used by Drone.
Returns: HashMap which contains what3words and location points

public ArrayList<Point> get_confinement_info()
This method generates a list of corners of confinement area which will be used by Drone.
Returns: ArrayList of Point objects which contains corners of confinement area

2.6 Class GeoJsonServer

Class Summary: This class is used to provide GeoJSON service for Drone to update information received from sensors and write the GeoJSON file.

Field Detail:

private final DataServer data_server - A field refer to DataServer object.
private final ArrayList<Feature> sensors_data - A field refer to sensors' data.
private final String year - A field refer to year of GeoJsonServer connection.
private final String month - A field refer to month of GeoJsonServer connection.
private final String day - A field refer to day of GeoJsonServer connection.

Constructor Detail:

public GeoJsonServer(String year, String month, String day, String port_number)
Construct a GeoJsonServer object given date (year, month, day) and port number.

Method Detail:

private String generate_rgb_string(String reading, double battery)
This method converts reading to RGB String given reading and battery.
Parameters:
reading - value of sensor reading
battery - value of sensor battery
Returns: RGB String based on reading

private String generate_marker_symbol(String reading, double battery)
This method converts reading to marker name given reading and battery.
Parameters:
reading - value of sensor reading
battery - value of sensor battery
Returns: String of marker name based on reading

private ArrayList<Feature> set_up_sensors_data()
This method sets up field sensors_data with default setting which contains 33 sensors' Features with no marker properties and with gray color RGB String properties.
Returns: ArrayList of Feature that represents sensors in default setting

public void update_sensor_data(String location)
This method updates the field sensors_data (add properties of Feature related to what3words location parameter based on sensor reading received by Drone).
Parameters: location - what3words location

public void write_geojson_file(ArrayList<Point> points)
This method writes GeoJSON file given output of drone control algorithm and sensors_data which has been updated by drone control algorithm.
Parameters: points - ArrayList of Point that output from drone control algorithm

2.7 Class Drone

Class Summary: This class is used to provide Drone object which contains drone control algorithm and method to write flight path file and GeoJSON file.

Field Detail:

private ArrayList<FlightDetail> flight_log
list of FlightDetail objects which record flight information of Drone.
private ArrayList<String> explored - list of what3words explored.
private GeoJsonServer geojson_server - GeoJsonServer owned by Drone.
private final DataServer data_server - DataServer owned by Drone.
private final HashMap<String, Point> sensors_info
33 what3words locations and corresponding points provided by DataServer.
private final ArrayList<Point> confinement_info - corners of confinement area.
private final ArrayList<ArrayList<Point>> buildings_info - points of no-fly zones.
private final String port_number - port number of web server connection.
private final String year - year set by Drone.
private final String month - month set by Drone.
private final String day - day set by Drone.

Constructor Detail:

public Drone(String year, String month, String day, String port_number)

Construct a Drone object given date (year, month, day) and port number.

Method Detail:

private double euclidean distance(Point point1, Point point2)
This method returns euclidean distance between two points.
Parameters:
point1 - Point object that represents a point
point2 - Point object that represents a point
Returns: double value that represents euclidean distance

private Point next step position(Point current step, int degree)
This method returns next point given current point and direction of move in degree.
Parameters:
current step - Point object that represents current point
degree - int value that represents direction of move in degree
Returns: Point object that represents next point

private boolean cross_no_fly_zone(Point current_step, int degree)
This method is used to determine whether current point move one step in direction of degree will intersect with boundaries of no fly zones or confinement area.
Parameters:
current_step - Point object that represents current point
degree - int value that represents direction of move in degree
Returns: True if move one step intersect with boundaries, False otherwise

private int boundary_avoid_degree(Point current_step, int degree, String mode)
This method returns direction of move in degree which avoids the boundaries of no fly zones or confinement area. Choosing the strategies to avoid boundaries based on mode.
Parameters:
current_step - Point object that represents current point
degree - int value that represents direction of move in degree
mode - String that can be 'clockwise', 'anticlockwise' or 'double'
Returns: int value that represents move degree that avoids boundaries

private Point move_one_step(Point current_step, int degree, String mode, ArrayList<Point> moved_points)
This method returns next point which avoids boundaries of no fly zones and confinement area and also make sure that next point is not in moved points.
Parameters:
current_step - Point object that represents current point
degree - int value that represents direction of move in degree
mode - String that can be 'clockwise', 'anticlockwise' or 'double'
moved_points - ArrayList of Point that represents moves in the past
Returns: Point object that avoids boundaries and does not occur in the past

private String receive_sensor_data(Point current_step)

This method finds sensors in area of Drone detection at current point and returns the what3words location of the closest sensor.
Parameters:
current_step - Point object represents current point
Returns: String that represents what3words location detected by Drone or "null"

private int adjust_flight_direction(Point current_step, Point destination)
This method returns adjusted flight direction in degree which is multiple of 10.
Parameters:
current_step - Point object that represents current point
destination - Point object that represents target sensor
Returns: int value that represents adjusted degree of move

private Point find_nearest_sensor(Point sensor, ArrayList<Point> search_domain)
This method finds the sensor that is nearest to current sensor in search domain.
Parameters:
sensor - Point object that represents point of current sensor
search_domain - ArrayList of Point that represents unexplored sensors
Returns: Point object that represents the nearest sensor

private ArrayList<Point> greedy_flight_planning(Point first_sensor)
This method makes a flight plan of sensors based on greedy method.
Parameters:
first_sensor - Point object that represents position of first sensor
Returns: ArrayList of Point that represents list of sensors to be visited by order

private ArrayList<Point> swap_greedy_flight_planning(Point start_position, Point first_sensor)
This method makes a flight plan of sensors based on greedy method with swap heuristic.
Parameters:
start_position - Point object that represents start position of Drone
first_sensor - Point object that represents position of first sensor
Returns: ArrayList of Point that represents list of sensors to be visited by order

private ArrayList<Point> first_sensor_planning(Point start_position)
This method returns an ArrayList of sensors from closest to furthest given drone position.
Parameters:
start_position - Point object that represents start position of Drone
Returns: ArrayList of Point that represents list of sensors from closest to furthest

private ArrayList<Point> move_to_next_destination(Point current_step, Point destination, int count_steps, String mode)
This method controls the Drone move from current position to target sensor.
Parameters:
current_step - Point object that represents current position
destination - Point object that represents next sensor to detect

count_steps - int value that represents current step number
mode - String that can be 'clockwise', 'anticlockwise' or 'double'
Returns: ArrayList of Point that contains all points in process of move

public ArrayList<Point> drone_control_algorithm(Point start_position, String mode, int tolerant)
This method controls Drone to collect data from sensors following plan made by planning algorithm.
Parameters:
start_position - Point object represents start position of Drone
mode - String that can be 'clockwise', 'anticlockwise' or 'double'
tolerant - int value that represents max times of replanning.
Returns: ArrayList of Point that contains all points of move during process

public void write_geojson_file(ArrayList<Point> path)
This method writes GeoJSON file by GeoJsonServer's method given flight path of Drone.
Parameters:
path - ArrayList of Point that contains points of move in whole flight process

public void write_flight_path_file()
This method writes flight path file by using field flight_log.

2.8 Class App

Class Summary: This class is used to provide the main method which integrates methods from Drone to write both GeoJSON file and flight path file based on command line inputs.

Method Detail:

public static void main(String[] args)
This methods writes GeoJSON file and txt file by choosing one output of drone control algorithm that has minimum number of steps among three different mode. All parameters come from command line.
Command Line Input:
day month year latitude longitude seed port_number
where latitude and longitude are drone start position.

3 Drone Control Algorithm

3.1 Introduction to the Algorithm

This section will introduce the whole drone flight process in detail. And in next few sections, there are comprehensive explanations of essential algorithms used during process. The general idea of drone control algorithm is really simple. Drone firstly makes a plan to decide what order of sensors that it will go to collect data and then follows the plan and flies directly towards those sensors one by one until all the sensors' data have been collected and finally return to the start position. During the process, it will avoid all the no fly zones between one sensor and another sensor, so it will change the direction of move if next move of drone will hit the no fly zones. This general process has been shown in the figure below which is produced by geojson.io:

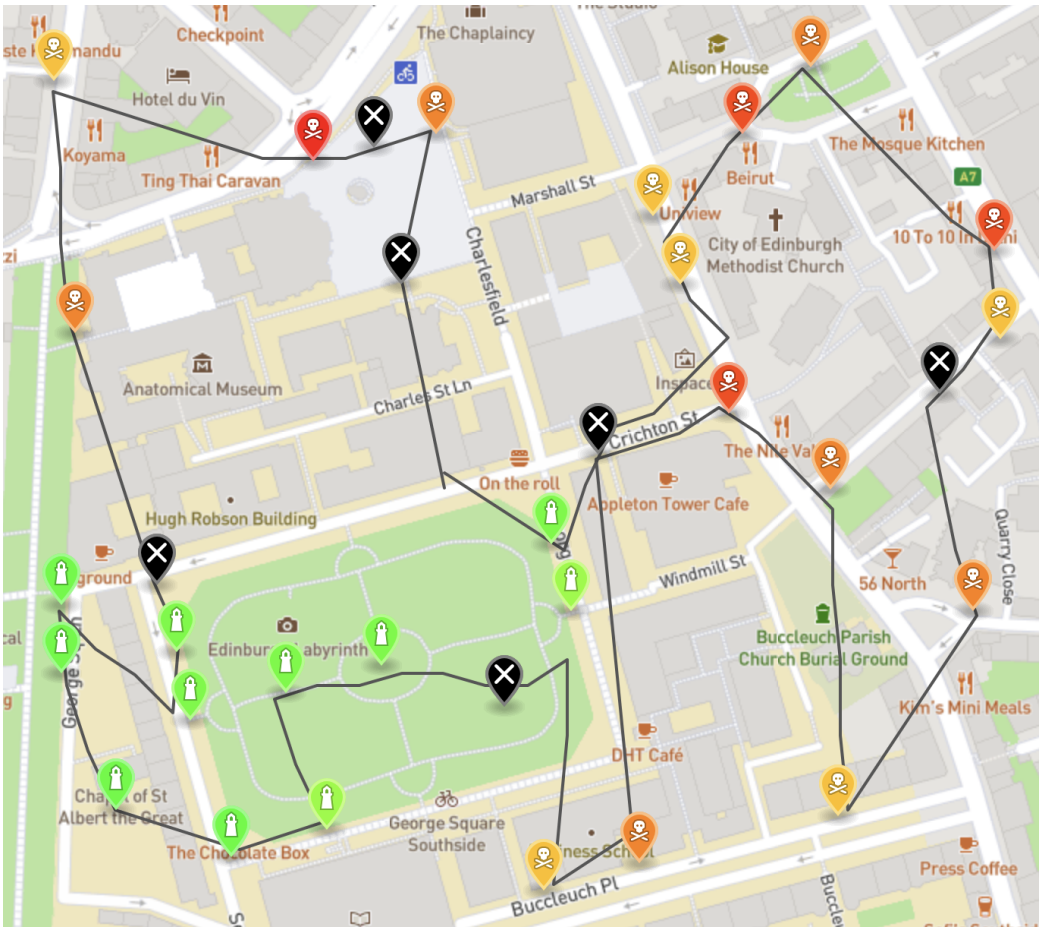

Figure 2: A sample output map for the date 08/08/2020.

From figure above, we can see that the drone starts at central area of the university and go to each sensors step by step, while avoiding no fly zones. It is clear to see that drone cross narrow street between Informatics Forum and Appleton Tower which are two no fly

zones commonly between one sensor and another sensor. Drone collects sensor's data within 0.0002 degree of sensor so some parts of path are not close enough to sensors. Overall, the drone use a reasonably good strategy to plan the order of sensors so that the flight path is really efficient.

3.2 No-fly Zone Avoidance Algorithm

This section will explain no-fly zone avoidance algorithm which play a significant role in the drone control algorithm. The figure in last section has shown how this algorithm avoids no fly zones and in this section, it will be explained in detail. This algorithm has been implemented in **cross_no_fly_zone**, **boundary_avoid_degree** and **move_one_step** methods in **Class Drone**. The general idea of algorithm is that if the flight path between current position and next position of drone in one move intersects with any boundaries of no fly zones, the drone can be considered as 'hit the no fly zones'.

When drone hits the no fly zones, the drone has three different strategies to solve the problem. The choice of strategies is based on 'mode' parameter. There are three values for 'mode' parameter - 'clockwise', 'anticlockwise' and 'double'. 'clockwise' is a strategy that turns the degree of move which towards target sensor in clockwise direction until next position will not hit the no fly zones. In similar way, 'anticlockwise' is a strategy in the same way but turns degree in anticlockwise direction. 'double' is more smart than those two and will determine which direction will turn less degree to avoid hitting no fly zones. The drone uses those three strategies and a consistent rule that in every steps, drone will towards the target sensor that it want to collect data. This makes flight path of drone similar to fly along the edge of no fly zones which is shown in figure 2. This algorithm still make sure that if drone does not hit no fly zones, it will move towards target sensor directly.

If you think about algorithm rigorously, you will find bugs in these strategies. For examples, if the edge of no fly zones is too long, the 'double' will fail by moving left and right repeatedly and finally get stuck. There is a solution given in this algorithm. If next position of drone is where has been reached in the past, the drone will change the degree to opposite direction of that degree and find an appropriate degree which will not hit no fly zones. This helps 'double' get out of local trap. Overall, this algorithm is really robust to avoid no fly zones although sometimes drone will move strangely when it meets groove.

3.3 Swap Greedy Path Planning Algorithm

This section will explain swap greedy path planning algorithm which helps drone to make a wise choice to plan the order of sensors. This algorithm is implemented in two methods **find_nearest_sensor** and **swap_greedy_flight_planning** in **Class Drone**. In figure 3, we can see that most of time the drone will choose the closest sensor to visit and this process will repeat over and over again until all the sensors has been visited. This is the idea of greedy part of algorithm. However, greedy algorithm has a limitation that it is easy to get local

optimal results rather than global optimal result. The pure greedy algorithm sometimes can not get efficient plan since plan is based on straight line approximation without considering the real situation such as avoiding no fly zones.

Figure 3: A sample output map for the date 01/01/2020.

There is an improvement for greedy algorithm called swap heuristic. The general idea of swap heuristic is that swapping two sensors in a list when total distance of sensors decreases after swapping. This process will repeat until all possible swaps in this list has been tried. This is swap part of algorithm. There is also a heuristic part. If swap occurs in the whole tries, the whole tries will start again until there is no any swap occurs in the whole tries. Swap greedy path planning algorithm combines both greedy algorithm and swap heuristic. The algorithm will firstly use greedy to get a intuitively good plan and then use swap heuristic to help greedy get out of local optimal. Swap greedy algorithm always give a circular path which hardly ever has intersecting lines in flight path. This has been shown in upper right of figure 3 where drone decides to go to a furthest sensor below rather than nearest red sensor and then visits a list of sensors which are close to each other from below to above. Swap Heuristic unties the intersecting lines made by greedy and shorten the path.

3.4 Path Re-Planning Algorithm

This section will explain path re-planning algorithm which is hardly ever used in drone control algorithm but plays a significant role when it is necessary. This algorithm is implemented in **first_sensor_planning** and **drone_control_algorithm** in **Class Drone**. If you think about the algorithms shown in last few sections, there is a risk that drone possibly flight around in a groove and wast a lot of steps to get out of groove and finally cannot finish collection within 150 steps. And in extreme situation, it is possible that drone still get stuck in local trap. Although those situation is rare, algorithm failure still need to be considered.

One way to solve this problem is re-planning the flight path which changes the first sensor planned by swap greedy path planning algorithm. Greedy algorithm is sensitive to choice of first sensor due to its local optimal drawback. This drawback can be used as a way to re-plan the flight path by trying different sensors as first sensor. The order to try different sensors should be intuitive. So algorithm sorts all sensors by distance between sensors and drone start position from closest to furthest and use this order to try re-planning. This algorithm has a parameter 'tolerant' which can be set to any integer value between 1 and 33. This parameter can decides how many algorithm failure can be tolerated. Over this value, the algorithm will stop regardless it success or not. This parameter design is due to Java memory limit and thread limit.

3.5 Path Minimization Algorithm

This section will explain last algorithm in drone control algorithm which is implemented in **main method** of **Class App**. This algorithm uses three different modes introduced in no fly zones avoidance algorithm and get three different results. After that choosing one which has minimum steps among three modes and writing flight path file and GeoJSON file based on it. This algorithm helps drone to choose a more efficient flight path and largely decrease the risk of algorithm failure.

4 References

Third party library Mapbox Java SDK is used in program - click here
Jibble Web Server - Web Server used in program - click here
Swap and Greedy Heuristic - Travelling salesman problem in Wikipedia - click here
Greedy Algorithm - Introduction in Wikipedia - click here
Two Opt Swap Algorithm - Introduction in Wikipedia - click here
All other algorithms are self designed without references

5 Appendices