Intelligent low-altitude air traffic management system

Team number 30
Advisor: Prof. Peng Wei
Team members:
Humaid Alkaabi
Jun An Tan
Saad Alsudayri
Suhail Aldhaheri

Revised: 22 April 2018

Table of contents:

1. Introduction

1.1 Acknowledgement:

Our group is working with professor Peng Wei to create a system that manages air-traffic in low altitudes. The system will be a simulation interface, a software that simulates the delivery process using aircrafts, where customer request will be generated in a given manner, for a given location, and the user will choose between different types of simulation. The whole idea of creating such system is to create a platform for potential low altitude aircraft company users to be able to conclude which scenarios would be optimum in cost and benefit terms. Companies that would be interested in this industry could first make use of our product to simulate deliveries for different scenarios before going about investing and implementing actual aircrafts flights. Similar to a flight tracker for commercial planes, our software is designed to simulate deliveries to customer for different number of warehouses and the corresponding number of aircrafts used. To summarize, our group is working on creating a simulation software for aircraft deliveries that can be used to find the optimum location for the warehouse, corresponding number of aircraft and suitable delivery process, all in order to minimize the downsides of the low altitude aircraft delivery process.

1.2 Problem Statement:

In today's world of technology, wouldn't it be great if we had our package arrive just hours after placing an order? The idea of expanding all possibilities and putting customers first is every delivery company's dream. As such, using low altitude aircrafts has become an alternative to help speed up the delivery process. After all, the company's profits work hand in hand with the number of happy customers they have. However, this seemingly lucrative solution has a common problem i.e: Regulations from the FAA. Hence if we were to consider all of the rules listed by the FAA on low altitude aircrafts for the purpose of delivery or not, this project will not be able to fully provide solutions to all of the rules. However, for a start this project will be a starting point for future attempts on providing full solutions for abiding all of the list rules regarding low altitude aircrafts activities.

For a start, our project aims to create a simulation platform where users could reach a conclusion that indicates the best scenario in delivering. Our project aims to mainly provide the solution to common issues that might be faced by staffs responsible for the planning of flight routes of low altitude aircrafts.

In our system there will be two main parameters, warehouses and demands. Depending on the user's input settings, the low altitude aircrafts allocated to each warehouse will be solely based on the user settings. The added flexibility besides allowing users to "choose a preallocated

scenario", is that by manipulating the low altitude aircrafts availability at certain warehouses, users could observe what advantages or disadvantages that might bring about. Low altitude aircrafts will be responsible for the completion of demands appearing randomly on the map as time goes. The pathway of the low altitude aircrafts will always follows this rule, they will fly from their warehouse to the place of interest and eventually get back to their warehouse.

The simulation software provides the flexibility of choosing between two different algorithm approaches. One of which focuses on assigning demands based on "shortest distance" while the other works on allocating demands based on the rule of completing demands in the "fastest time".

1.3 Intended Users and Uses:

There will be two groups of users that we are targeting.

1) Delivery companies such as amazon air prime, Google/any other companies that are planning to venture into the low altitude aircrafts delivering market.
2) Transport companies such as uber elevate that plans to develop autonomous flying cars.

In any cases, our system would be able to simulate air traffic conditions at any time if in the event when the airspace is getting populated with different flying objects. The concept of delivering packages from point A to B is the same as delivering humans from point A to B. Flying objects must be able to reach their destination safely especially since they will be flying autonomously. Relevant flight information will be displayed to the users for analyzing purposes.

1.4 Assumptions and Limitations:

The assumptions we made will be:

1) Each warehouse has an unlimited number of goods for the purpose of delivery.
2) Each warehouse will be assigned a fixed number of low altitude aircrafts for the purpose of fulfilling customers demands.
3) Aircrafts will always depart from their warehouse to a demand location and get back to the same warehouse to restock for the next delivery.
4) Aircrafts will all be flying at the same altitude at a and fixed speed.
5) Other possible obstacles such as tall trees, building, power-plants… etc will not be present.
6) Preset trajectory paths would always be a "straight line" between two points.
7) No actual low altitude aircrafts will be in use as part of the simulation/testing process
8) Any natural conditions that could potentially affect the flight path of the low altitude aircrafts will not be considered to a great extent (tailwind/crosswind/blizzard/headwind)

that is why we had decided to include a combination of straight lines to simulate the slight disturbance of potential wind conditions

Limitations:

1) 3D map is not applicable.
2) FAA drone regulations to be followed in the future.
3) User can't change any information such as aircrafts number once the simulation starts.

1.5 Expected End Product and Other Deliverables:

Our final software now has functionalities that are meant to depict our client's definition of "ground delay" scenarios.

The problems that will portray "ground delay" conditions at the same time being able to simulate the conditions of low altitude aircrafts deliveries which we were expected to solve were:

1- Ensuring no path crossing between aircrafts depends on what scenarios were chosen.
2- Updating aircrafts latest position on the map by displaying it at the front end display
3- Erasing completed demands off the display and from the results section
4- Generation of demands on map display that has some correlation to the census of Ames population.
5- Creating scenarios such as satisfying demands in the fastest possible manner with respect to any number of drone Used per warehouses (grants flexibility for our client to do any test he wants)
6-Displaying simulated results on some form of User Interface (front end display) that makes sense for our client
7- parts 1-6 were the main conditions that were tasked but to better cater to our client's Intention, the code would be improved by handing the option of variable control to the user. Doing so would allow the option of having different scenarios
8- Having the option of being able to run this tool over other city/countries and not just over ames area.

Our client's main intention is to have a relatively realistic system that is capable of displaying relevant information for him to conclude which method would increase the efficiency of aircrafts delivery scenarios and what is the best option in balancing between effective use of resources vs shorter waiting times for deliveries to complete.

Our final product is made up of a combination of java(backend) and javascript(front end). We used Eclipse, a java IDE, to accomplish the tasks and also use javascript, HTML to create the display of our simulation process. Eventually all tests/product demo will be done with the use of this software.

2 Testing and Implementation

The Team have established a “standard testing procedure” for all kinds of code related work throughout the two semesters. In cases when our front end display isn’t ready we would always:

1) Test the written code individually as a function by itself

2) Test the same code again when the function is added to the main portion of the code

The Team’s method of attempting to try and solve a potential problem would comprise of :

- Personal attempt in solving a given task

- Researching on relevant methods that might help in the attempt to solve a given task

- Establishing a group discussion on tasks that cannot be solved alone

After our front end display is completed, it would be easier to spot abnormalities of the code. However, the troubleshooting process remains the same as above: where we would go to the function that we thought where the problem was and troubleshoot it.

Hence, using the above as a guide, we'll be using this approach to go about new and current problems in working towards our desired outcome. In cases where the completion of a function could not be achieved by a particular member, the team will then be working together to help resolve the issue.

Functional testing:

- The drone should start exactly at point A and end at point B. It should not exceed these two endpoints for a given trajectory and should always stay on the displayed trajectory path at all times.
    - If this works, we can be assured that our front end is reading the outputs correctly from the backend without the addition of any “unwanted data”.

- There should be distinct differences between symbols used to represent “demands”, “warehouses”and “drones”

- The front end background should be a precisely zoomed in version of google maps where it depicts the boundary of a city given by a user (Ames is used as our default testing City).

- There would be an allocated block somewhere in the front end display where it would display the “results” specified by our client. There will be current and ongoing flights for each warehouses and next available (earliest) departure time. When a specific flight is done, they will be removed from this “results” section.

- The rate at which demands appearing on the map can be varied accordingly to allow users to observe the delivering process. This “rate” would based on our client’s approval. Users would have the choice of choosing a census based distribution or a totally random distribution of demands.

- The rate of reading data from the back end to front end should be at a “decent” realistic rate of 1Hz to showcase the aerial view of the process of drone moving from one point to the other.

- The option of flexibility would be given to our users by allowing them to manipulate certain variables such as number of drones, number of warehouses , different scenarios like assigning demands based on closest distance or assigning demands based on earliest time.

Non functional testing:

Although it is not explicitly stated by our client that our project is basically made of blocks of codes, the code would be ensured to always be kept at minimum to allow the system to process at a “fast enough” speed for a smooth simulation. The baseline of our definition of lag would be benchmarked to under two seconds. This would probably not happen at all due to the fact that the code would not be accessing any hardware related library functions. In short, the display should not crash/lag during operation.

The display itself would allow users to zoom in and out whenever but, the initial preset zoomed is where users were able to observe drones doing their thing.

Our code would strictly be following the standard typical way of organizing and adding comments during any development process. In terms of user interface, we would make sure that

our system is user friendly and that the instructions that comes with it under appendix 1 would allow anyone to be able to set things up easily.

Implementation Details:

In order to approach this project and be able to deliver our final product on time, the team has decided to divide the project into two parts:

In the first part, the team will work mainly on all of the back end codes which simply fulfills the calculation, algorithm of all parts of our deliverables. Due to the fact that there will be no visual interactive output, the team will constantly be checking for accuracy of the written codes by validating results through a simple "printf" equivalent function in java. Parts that require validation of results through the use of front end would first be written and "marked" until the front end is completed. The team will also be making use of other preloaded programs such as excel and matlab for a temporary visual assurance of the printed results. This temporary visual reassurance acts like a redundancy check that we predict would be sufficient in the checking of most algorithms.

In the second part, after the completion of the backend coding, the team will then move onto the creation of the front end display (GUI). This front end would then be responsible for the display of our intended results and relevant informations to users. Although this part involves only a "one step process" in processing the backend and transferring it over to the front end, we would foresee that there would be various obstacles associated to it. This would mainly be attributed to the team composition and unfamiliarity with the process of creating a GUI. Ultimately, we would try to complete the first phase as fast and as much as possible so we could have more time to work on our display portion of our approach.

In the last part, should any conditions need to be modified, the completed front end display would come in handy to help with any potential debugging/verification situations.

2.1 INTERFACE SPECIFICATIONS

The client has no preference on the types of programming languages for the development of the project. Since the fact that this project is software based, and that it requires us to create the software from start, we were given the freedom of choice in the selection of programming

language. Therefore, the team has agreed and decided that ultimately this project will be delivered through the use of Java for the back-end calculations and JavaScript, HTML and CSS for the front-end display.

There will only be one interfacing specificification: to make sure that the output GUI ,JavaScript, is able to interface with the back-end code, Java.

2.2 Hardware and software

A java based software,eclipse was used to write our simulation project. In the development phase especially when the focus is on the backend portion, the testing and validation of the work will be done using excel, matlab and google maps as a temporary form of checking our work without the need for the official GUI to be up and running. Being a fact that it's a simulation based project, there will not be any hardware interfacing process or any hardware assembly process. The model of the drone specification will be referenced to be that of DJI Phantom 3 with the assumption that it will have unlimited flight time. The final GUI will eventually made using javascript that displays the entire simulation process.

2.3 Standards

The only IEEE standards that would be adhering to would be "standard for consumer drones: privacy and security". This IEEE standard is by far the most applicable with regards to the nature of our project. In our earlier sections, we mentioned that this project has to follow the FAA regulations on unmanned drones that it has to be within line of sight between the pilot and the drone. Since, this is just a simulation and its for the sake of educational/ research purposes in determining the criteria for resources allocation vs efficiency, the other consideration that would be adhered to would be the height at which the drone flies would be at an altitude that is way below commercial flying altitude.

2.4 Process details

In order to achieve the deliverables, we decided to create different functions each fulfilling specific tasks. By making use the functionally of ArrayLists prominent in java to act as databases, where it would be keeping informations related to customer demands, initiating drone clearance between the pathway from the warehouse to a demand, storing of warehouse information, data for ongoing flight and the time information at which when a drone would be arriving and departing for its next demand.

Acting as one of our main function to store information for our front end display, a trajectory function was created. It will be responsible for the constant update of current location while

displaying the drone's location on the map. In order to make sure that it is precise, this function will be using specifically the data from the ongoing flight arraylist.

The combination and utilization of such specific task functions that we had, made it possible for us to create the connection between such functions. Some tasks of back end functions that will be responsible through modifications in achieving our deliverables at different stages of our project are shown below.

The functions are:

1) Distance function:

Calculates distance between two points of the map using latitude and longitude parameters. These information can be obtained and verified from google maps with some logical effort.

2) Calculating current drone position function:

Calculate current drone position using basic math techniques of sine,cosine, tangent properties, projection of axis. For e.g, in a x-y plot, when x and y are both positive, angle is positive. Likewise, when x-y is negative, angle is negative. So by incorporating all four possibilities of " different angles" we can get our current drone position.

3) Random demand generator:

Using a random algorithm to generate different locations for the representation of "random customer demands". This function exists to get things started, while eventually it would be modified to display "random demands" according to the user's input of census data for an area.

4) Creating arbitrary map function(when our front end is not ready):

This serves as an arbitrary map to get the latitude and longitude information by creating a boundary with 10000 points vs 10000 points. It acts as our temporary test to allocate demands on a non existent front end map for the purpose of checking deliverables.

5) Collision checking function:

This will be one of the more complex function that uses previous functions to ensure that whenever there is an intersecting path/paths, permission will not be granted for both flight paths. In the event where one path is granted permission(ongoing), the other intersecting path/paths will not take off, it will be assigned to fulfil other non-intersecting paths demands. We assumed that when two trajectory paths cross each other, it equates to collision. The other possible way that we could go about is that, we could track the location of each drones and their flight paths and make a decision accordingly with the help of the time function java provides.

6) Few arraylist for storing different types of data (acts as a database of the system)

There will be four arraylist used as a database of our system. In the future should there require a modification of our current system, arraylist can easily be added for that purpose. The four lists include: warehouse list, flight-request list, demands-list and ongoing flight list. The primary data found in these lists would include the latitude, longitude information and possibly time information at which the demand has been made.
The amount of data parameters will be constantly evaluated during the process of developing the code.

7) Transferring demands data to flight request list:

This function will literally do the task as indicated by its name

8) Selecting nearest warehouse function:

Initially the purpose of this function is to allocate demands to the nearest warehouse associated to it. However, as part of our modification to simulate the realisticness of drones having a short flight time, this algorithm is modified to include a "time" factor where basically we would had stored time information(somewhere) of when the demands had been requested and make sure that the upcoming demand always have the precedence over other pre allocated demands for various warehouses.

9) Plotting of current drone position function through the use of stored database:

This function would be the main function used to output our code to a front end GUI.

10) time taken function:

This function acts as an upgrade for the realisticness of our product. This function will calculate the time taken from point A to point B and work out the arrival and departure times for the drones. This would be mainly used to log in, display departure timings on the front end and determining which demands to be completed next.

11) Shortest time function

This function calculates the time it will take a given warehouse's aircraft to make a delivery and it can be setup in one of two ways, either find the shortest time without considering trajectory crossing or calculate the closest trajectory that has no crossing with the previous scheduled trajectories.

There were other functions that were not stated which plays a part in allowing us to achieve our final deliverables, but as listed were some of the basic and crucial functions that allowed our system to worked the way it should.

2.5 Design analysis

The software is made up of multiple functions including the ArrayLists which acts as our databases. Functions discussed earlier involved in doing specific tasks. Example of such tasks is checking for the possibility of trajectory crossing with the previous scheduled trajectories,

comparing of time taken between different flights, assigning of demands according to the wishes of the users and more. In one of the scenarios when flight paths are crossed, we will have either one of them allowed for flight, temporary prevent flight authorization for both of them or compares the timing at that particular intersection to determine if a trajectory crossing will occur. The software will then assign the drone to a "next in flight" for it to complete its next delivery.

All in all, the basis of our design works by first getting the locations of demands, then transfer them into a database(arraylist in our case) , then assign them to their warehouse, following which it checks for potential collision by looking at their pathways. Finally, when all is done internally, the last function will be the real time plotting which showcases the movement of the drones on the map.

The strengths of our proposed design includes:

1) Algorithm that fulfills demands the most efficient way.
2) Allowing users to brainstorm for more possible scenarios.
3) Allowing users to save money on real time aircrafts testing scenarios for data interpolation .
4) User friendly usage of software.
5) Depicting various scenarios.

The weakness of our proposed design includes:

1) Not much options as to what the users can do when running the simulation (minimal user to software interaction).
2) Assumptions of ideal conditions
3) Drone's flight paths are only in a "straight line"
4) Software is in 2D and not 3D/ VR

Observations of the proposed solution:

1) A good starting prototype that showcases some scenarios that allows users to play with
2) Massive amount of assumptions on creating an ideal environment
3) Includes flexibility in changing variables which allows realism

2.6 Results

There had been some minor failures encountered during our coding process but fortunately they were solved within a couple of days. These minor failures that cause things to work a little weird were due to the lack of understanding and the lack of experience in Java. Such setbacks were expected mainly due to the composition of our team and that half of us had to learn on the spot a new programming language. Judging from how we have our plans laid out, our progress has been great when attempting to solve the back-end problems, we managed to achieve the minor milestones by completing functions that are the solution to our designated deliverables.

There had been a period where after our successful pairing of back end to front end, abnormalities of our results were shown on the display. A persistent constant error that makes our drone exceeds the line of trajectory between two points occurs after the completion of the first allocated demand.This is shown below:

Fig.1 Results 1

It eventually took us some time to finally pinpoint the error which leads to the team's consensus to rewrite the functions that were used as a database for our simulation which turns out to be the problem.

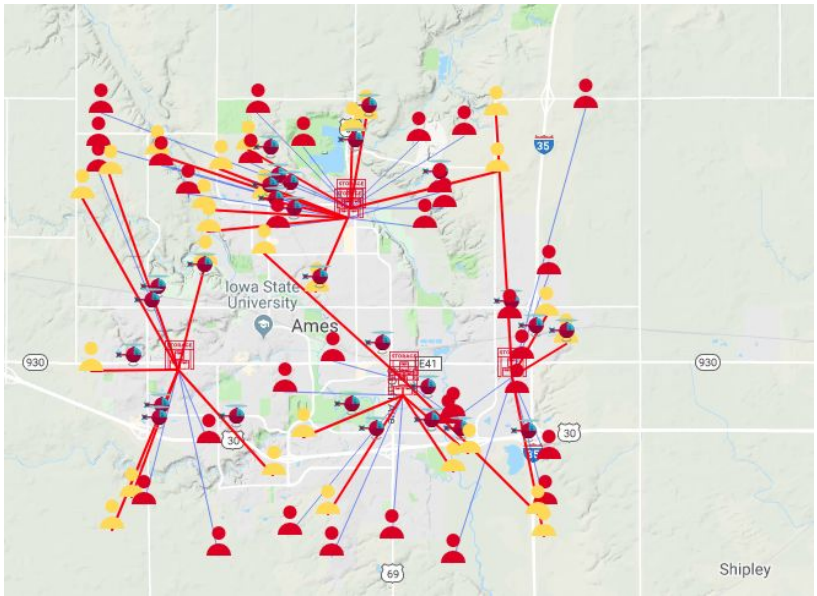

Fig.2 Results 2

After which, since we couldn't directly import the data from the census website, we began to find ways to replicate the actual census data of a particular area. We created "zones" to represent the majority of where residents were and allowed the generation of demands to be within such "zones".

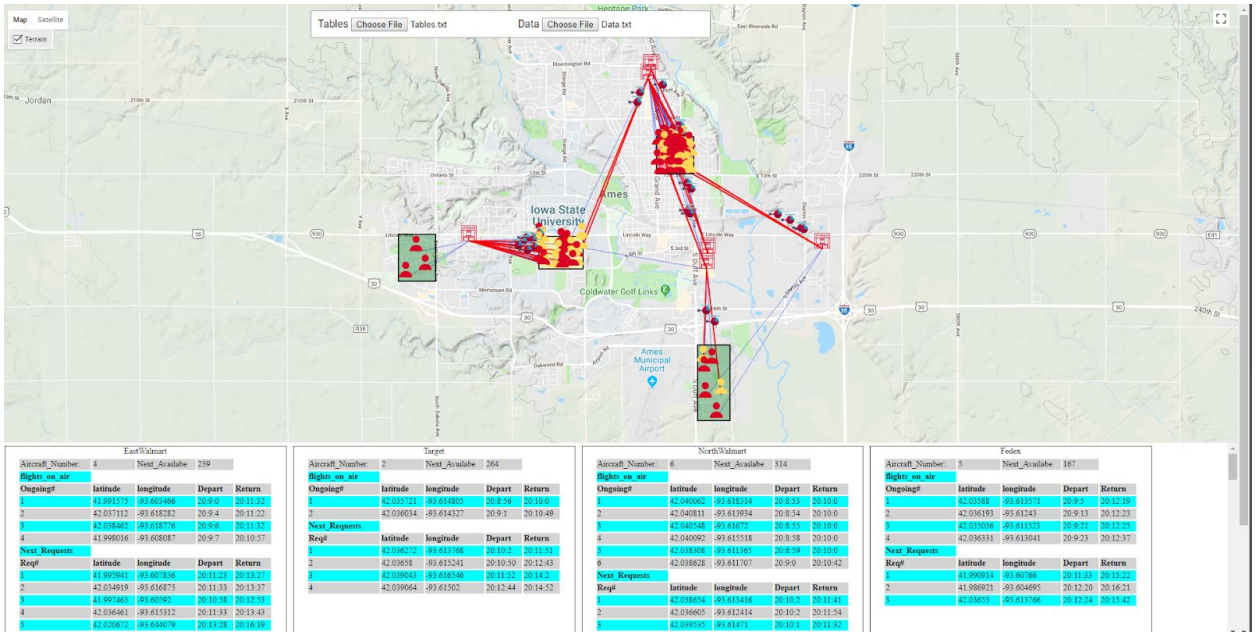

Fig.3 Results 3

3 Conclusion

In conclusion, this project was essentially meant to be an educational tool with the main intention of creating a platform where you could analyze the flow of drones and come up with a appropriate plan that works best. This project was tasked in a way that was classified as a "ground delay" scenario according to our client. This is because for the scenarios that we had implemented some of which works only when there is no intersection between any flight paths and thus drones would sometime have a "ground time" associated to it. This is especially evident in cases where you have little demands to fulfill. Ultimately users were given the option of changing certain variables such as warehouse location, quantity of drones for each warehouse, to enter the census of a particular area of interest, demands allocation according to shortest distance from warehouse / based on highest efficiency and a combination of shortest distance allocation along with highest efficiency option.

This project can definitely be improved on by considering other types of relevant scenarios such as different flightpath allocation. For realism purposes, this product would be a good starting prototype for any group that wishes to improve on.

3. Appendix I - “Operation Manual”

Step 1: Preparing input of the software:

The user has to prepare two text files with the information of the warehouses and generation of customer demands.

- First, warehouses.txt

The first text file should be named “warehouses.txt” and it will be used to create the list of warehouses in the simulation. The user should input a name, location then the number of the aircraft of this warehouse, below is a clarification for the format.

“Name1” “latitude” “longitude” “number of aircrafts”

Format Details:

- The name should have no spaces, but underscore can be used.
- The latitude and longitude are double numbers.
- Number of aircrafts should be an integer.
- Each warehouse information should be separated by different line.

An example of a valid “warehouses.txt” file is below

EastWalmart 42.016115 -93.607313 4 Target 42.019224 -93.606873 2 NorthWalmart 42.053202 -93.622611 6 Fedex 42.020060 -93.576719 5 UPS 42.055123 -93.621929 4 West_Hyvee 42.021593 -93.670005 7

- Second, simulation_requests.txt

The second text file should be named “simulation_requests.txt”, and it will be used to determine where the customer requests will pop up. The requests will be random, and will be generated at a rate that can been chosen to have value between 1 and 60 requests in the minute. so , the text file will have the location and rate, and the format for this text file should be as follow:

“North Latitude” “South Latitude” “East Longitude” “west Longitude” “Requests Rate”

Format Details:

- The first four variables should be double numbers.
- The request rate has to be an integer.

An example of a valid “simulation_requests.txt” is below:

```
42.041992 42.034723 -93.610774 -93.620559 60
42.022524 42.016149 -93.639826 -93.651575 30
42.001219 41.986418 -93.601118 -93.609615 9
42.022898 42.013714 -93.678838 -93.688623 4
```

Step 2: Run Java

After getting the two text files ready, it is time run the source code. Everything for the simulation has been written in one Java code file to make it easier to compile and run so the user can use any compiler to run the code. We recommend to use Eclipse to run the code. Regardless, make sure you place the two text files one the right directly depending on what compiler you are using when you run the java code, there will be two output text files from the code, “Data.txt” and “Tables.txt”, and these will be used in the next step.

Step 3: connect the data to webpage

Finally, the simulation will be shown on google maps, and the code for that has been written using google JavaScript API. So, open “Simulation.html” on the simulation folder and you will see on the top center two button for the two output from running the Java code,“Data.txt” and “Tables.txt”.

Picture below shows the two button before inputting.

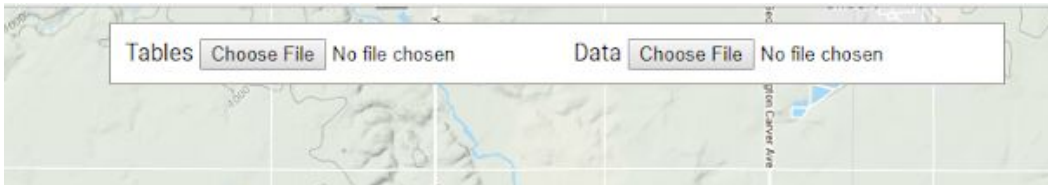

Fig.4 Where to select files

After finishing steps 2-3:

When you are done with steps 2-3, the html page and the output of the Java code should be connected, and display will like similar to the following

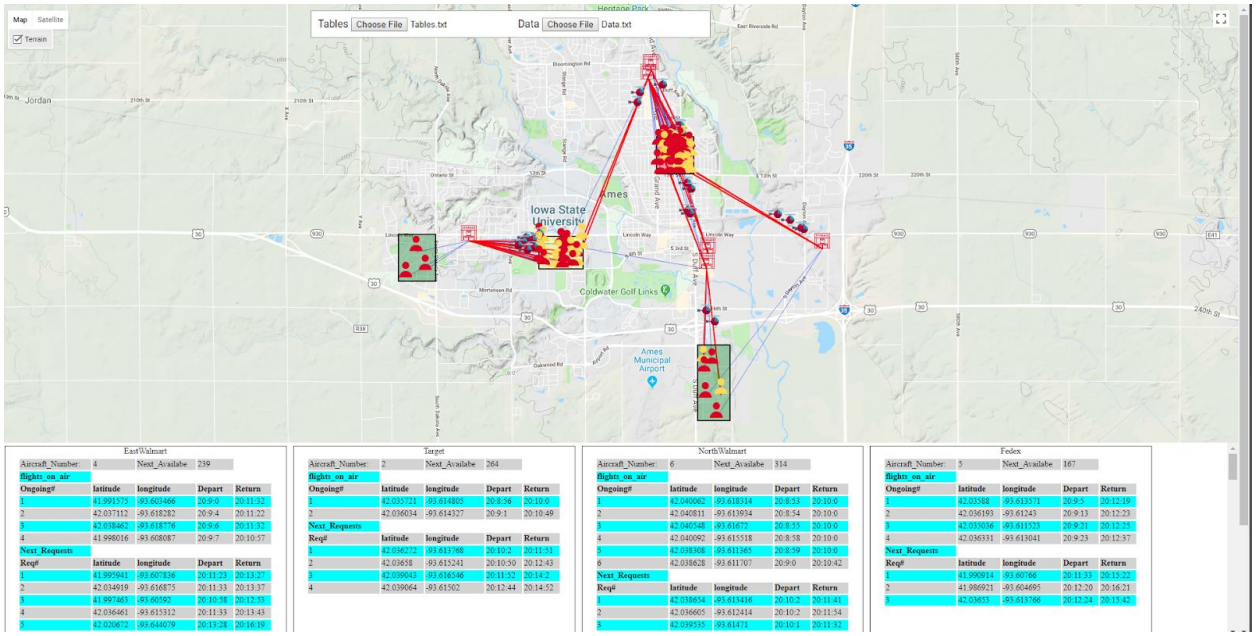

Fig.5 Census type of output example

Symbols Guide:

- Helicopters represts aircrafts.
- Red lines represent the trajectory for aircrafts on air.
- Blue lines represents scheduled trajectories.
- Yellow symbol represent customers with aircrafts on the way, and will will stay yellow until the aircraft return to the warehouse.
- Red symbol represents customers who has not been served yet.
- Squares represent where the requests will be generated, and each color indicate different value for the request rate:
    - Green squares represent rate less than 20 per minute.
    - Yellow squares represent st rate between 20 and 39 per minute.
    - Red squares represent rate between 40 and 60 per minute.

Comment: the symbols can be modified in the display folder by replacing the images with the desired symbol.

Table Guide:

The lower part of the page will display information regards all the warehouses that been inputted by the user, and a max number of 4 tables can be displayed in each row. Each table will contain the following

- Name of the warehouse on the top.
- Number of the aircrafts available in each warehouse.
- Time in seconds for the next available aircraft.

Appendix II: “Old Version”

1- First Prototype:

Our first prototype of the software was just an .exe file that do the calculations and print the information based on a simple guide, and it was as follow:

- Can’t handle more than one aircrafts per warehouse.
- User had to input the warehouses info directly to the source code
- Request generation was fixed to one boundary at demands can be just generated manually.

The pictures below show the first prototype.

```
               Welcome to                    Low-Altitude Air Traffic Managment System
To clear and print all the data type 123
To make a demand Enter 1
To make a flight request Enter 2
To Assign all the current Demands to the closest warehouses Enter 44
To check and give all the current requests permission to fly Enter 55
To print the list of warehouses Enter 6
To print Demand list Enter 7
To print the Flight Request list Enter 8
To print the ongoing Flight list Enter 9




To create a matlab code to show you the simulations for the current flights on the air please Enter 999
to generate a random 10 demands Enter 9910
to generate a 100 random demand Enter 99100


 to print this guide type 321
```

```
current time is: Tue Dec 05 09:10:10 CST 2017
*******************************Demand List*******************************
List Is Empty
*******************************Flight Request List*******************************
Flight Request Number: 4, Departure point: (42.055123,-93.621929), Destination point: (42.05882677384485,-93.5921884017202),
One way Traveling distance= 2489.680444681937 Request has been made at: Tue Dec 05 09:09:14 CST 2017 Required Time: 155
////////////////////
*******************************OnGoingFlightList*******************************
Ongoing Flight Number: 1, Departure point: (42.016115,-93.607313), Destination point: (42.00040576337642,-93.59776275475157),
One way Traveling distance= 1916.7422094810563, Total Traveling distance= 3833.4844189621126
Required Time in Seconds for oneway is: 119
Departure Time is: Tue Dec 05 09:10:00 CST 2017
Expected time of arriving to Customer is: Tue Dec 05 09:11:59 CST 2017
Expected time of return is: Tue Dec 05 09:13:58 CST 2017
Current location is: (42.014935, -93.606595)
\\\\\\\\\\\\\\\\\\\\\\\\\
Ongoing Flight Number: 2, Departure point: (42.055123,-93.621929), Destination point: (42.05871559317499,-93.64510106437399),
One way Traveling distance= 1954.348356478474, Total Traveling distance= 3908.696712956948
Required Time in Seconds for oneway is: 122
Departure Time is: Tue Dec 05 09:10:00 CST 2017
Expected time of arriving to Customer is: Tue Dec 05 09:12:02 CST 2017
Expected time of return is: Tue Dec 05 09:14:04 CST 2017
Current location is: (42.055388, -93.623636)
\\\\\\\\\\\\\\\\\\\\\\\\\
Ongoing Flight Number: 3, Departure point: (42.021593,-93.670005), Destination point: (42.00105269735498,-93.67446170437535),
One way Traveling distance= 2313.4673520082565, Total Traveling distance= 4626.934704016513
Required Time in Seconds for oneway is: 144
Departure Time is: Tue Dec 05 09:10:00 CST 2017
Expected time of arriving to Customer is: Tue Dec 05 09:12:24 CST 2017
Expected time of return is: Tue Dec 05 09:14:48 CST 2017
Current location is: (42.020314, -93.670282)
\\\\\\\\\\\\\\\\\\\\\\\\\
Flight number: 1 is still in the air, and its location is (42.014935, -93.606595)
Flight number: 2 is still in the air, and its location is (42.055388, -93.623636)
Flight number: 3 is still in the air, and its location is (42.020314, -93.670282)
```

Fig.6 Prototype 1 output

When we created our first prototype, we barely had enough knowledge in Java and we did not have any experience in any of JavaScript, HTML and CSS.

2- Second Prototype and Third Prototype:

By the beginning of the second semester, we started to study JavaScript, HTML and CSS in order to create a display for our code. However, the first attempts were as follow:

- Required using of AJAX and jQuery.
- Required using server to update and data.
- Required a lot of inputs.
- No way to use symbols.
- No table to represents data.

The following picture shows how the display was in the beginning of the semester.

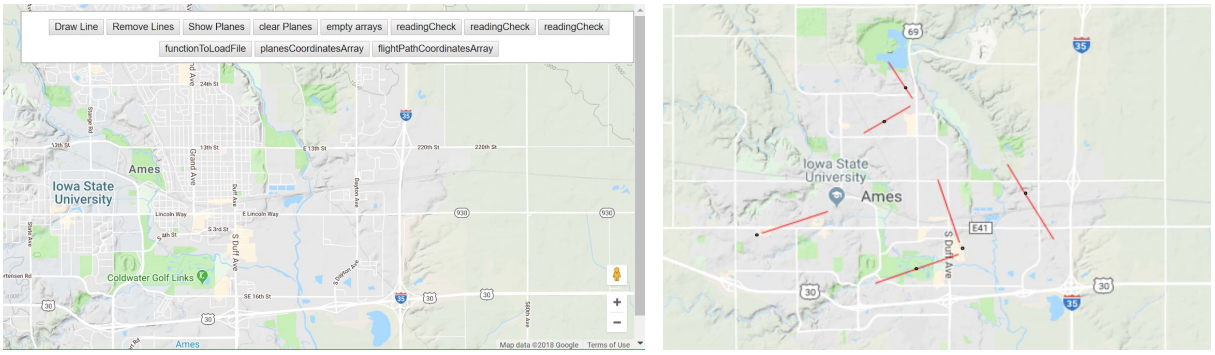

Fig.7 Prototype 2 & 3

3- Final version

We needed more data to be displayed on the front-end and more convenient way to connect the data, so we had to improve this prototype to have the following

- Symbols for each piece of information the user might be interested in.
- Data transfer that does not require AJAX and jQuery.
- Tables that show all the data user need to keep track of.

So, we ended with the version shown in the picture below:

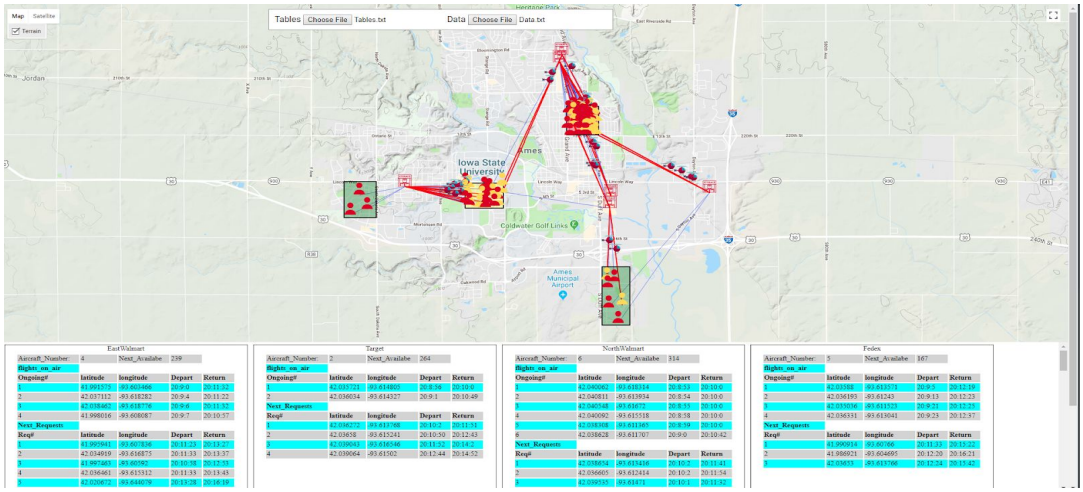

Fig.8 Census based results

Appendix III:

Challenges:

By the end of this project we had a lot of experience and challenges. We have wasted a lot of time in developing the wrong codes, or learning about useless tools that did not come handy later on the project. First of all, this project was a big challenge for our group because of two big reasons. First, the group consist of just Electrical Engineers. Second, we did not had any previous knowledge in the programing languages used in this project, except that two of us took ComS 207, which is an introduction class to Java programing. So, more than 80% of the project was about learning and writing codes in new languages.

Mistakes:

In the beginning of the project, we did not had any knowledge in developing softwares, so we assumed that we will have to create everything from the scratch, we did not know by the existence of the APIs, so we wasted a lot of time in learning how we can create a software from the scratch, however, in the beginning of the second semester we have learned about Google Maps JavaScript API.

Import Note:

In writing the code, we did our best in order to write the code in a way that enable future improvement. So the code is a bunch of different functions that can be improved in the future to add new details. For example, the user can change the simulation type by changing the function used for assigning the warehouse, based on time or distance.