For office use only		Team Control Number	For office use only	
T1	________	7379	F1	________
T2	________		F2	________
T3	________	Problem Chosen	F3	________
T4	________	A	F4	________

2017
HiMCM
Summary Sheet

In response to the Mayor's asking of an outdoor aerial light show using drones, after a careful investigation, we would like to present our mathematical model, conclusions and recommendations for the aerial light show. In brief, we address the problem of optimizing the flight paths for each drone through mapping the locus function with three-dimensional system of coordinates, aiming at finding the solution that has the optimal total flight paths as possible and satisfies the safety requirements.

Based on the historical data from all the aerial light shows using drones that have already taken place so far, we set the lower bounds to avoid crashes and the upper bounds to guarantee a visual feast within the sight for the distance between two drones referring to the basic parameters of drones. Then we use the Roberts edge detection operator to form a simplified pattern of our tridimensional design of display----a Ferris wheel, a dragon and a map of China----and pick out points that satisfy the distance bounds. Next, we set up a left-handed tridimensional coordinate system and calculate the coordinates of all the drones.

In search of the optimal solution that accomplishes both of our goals, first, we calculate with Euclidean distance to find the minimum flight distance for each drone and use clustering analysis to group similar flight paths. Second, we utilize binary integer programming to find the global optimal solution. Third, we use the detect function with vectors to examine for colliding paths. Then, in order to adjust fight paths and avoid crashes, we generally apply two methods: one is to change the flight paths from a straight line to a curve using quadratic functions; the other is to revise the take-off time.

In additional, we add Bezier curves to the rotation of our design after forming the horizontal image to make the transition more vivid. Lastly, to ground this model in reality, we animate the whole display process through computer simulation and present the flight paths with mathematical functions.

The light show is arranged to last for 20 minutes while the actual flying time is about 2 minutes. The apron required for taking off has to cover approximately 69m×57m. All the drones are configured as a 24×20 rectangle with 3 blank points. As for the space needed in the midair, the flight height is within the range of 63m~224m. The best viewing position for audience is 123.48 meters from our light show. Since profitable shows are beneficial to the local economy, tickets can be required to watch the aerial light show so that the cost of the show can be covered. As for each flight path, we present the changes in coordinates and the functions in between. For instance, drone 1:

$$(0,0,0) \rightarrow (60,50,138) \rightarrow (5,57,113.775) \rightarrow (115.610,57.001,149.468) \rightarrow (57,0,0)$$

Our solution, which is easy to implement, includes a detailed aerial display program and the flight path for each drone. Our model showcases the improving process and the contrast among various methods on the road to finding the optimal solution. According to the result of the sensitivity analysis, we believe that our algorithm is broad and flexible enough to accommodate various conditions, safety concerns and different parameters of various kinds of drones. Since our model is based on the control and orientation of UAVs, our strategy may also contribute to other technologies related to drone including combat drone operation and orientation system.

Key words: Euclidean distance, Roberts edge detection operator, binary integer programming, Bezier curve, clustering analysis

Letter to the Mayor

Dear Mayor,

Our team has carefully planned the light show on the night of the annual festival and succeeded in creating three possible sky displays including the pattern of a Ferris wheel, a dragon and a map of China. The whole performance will include 477 drones, and all the people in this city can enjoy this well-organized fantastic visual feast.

Taking the maximum flight time limit of drones and the overall show plan demanded into consideration, our team arrange the drone light show to last for 10 minutes. To be more specific, the actual flying time in total is 2 minutes and the rest of the time would be used for performance. The apron required for taking off has to cover approximately 69m×57m. All the drones are configured as a 24×20 rectangle with 3 empty points. As for the space needed in the midair, the flight height is within the range of 63m~224m. According to our careful calculation, the audience's viewing position should be arranged approximately 123.48 meters from our light show in order to enjoy the visual feast from the best view. Since profitable shows are beneficial to the local economy, tickets can be required for audience to watch the aerial light show so that the cost of the show can be covered.

Based on the historical data from all the aerial light shows using drones that have already taken place so far, we set the lower bounds to avoid crashes and the upper bounds to guarantee a visual feast within the sight for the distance between two drones referring to the basic parameters of drones. Then we use the Roberts edge detection operator to form a simplified pattern of our tridimensional design of display----a Ferris wheel, a dragon and a map of China----and pick out points that satisfy the distance bounds. Next, we set up a left-handed tridimensional coordinate system and calculate the coordinates of all the drones.

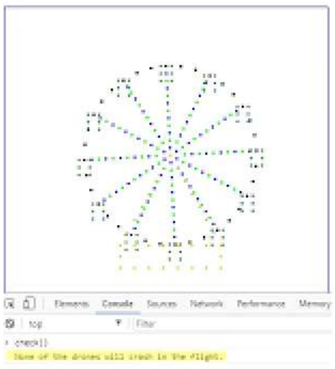
In search of the optimal solution that accomplishes both of our goals, first, we calculate with Euclidean distance to find the minimum flight distance for each drone and use clustering analysis to group similar flight coordinates. Second, we utilize binary integer programming to find the global optimal solution. Third, we use the detect function with vectors to examine colliding paths. Finally, in order to adjust fight paths and avoid crashes, we generally apply two methods: one method is to change the flight paths from a straight line to a curve using quadratic functions; the other method is to revise the take-off time.

The brightest shining point of our show is the third image designed. The broad territory of China fully unfolds before the audience which showcases the immensity of our motherland China, representing the uniqueness of our nationality. Another great brilliance that lies thoroughly in our model is how we improve our algorithm to achieve the goal of finding the optimal solution that has the shortest total flight distance possible with no crashes. Various methods are utilized to achieve this goal, resulting in remarkable progresses. With comparison and contrast between different methods and algorithms, we manage to accomplish our goal. 477 is undoubtedly an incredibly huge number for the drones required, and it's clearly a great challenge to ensure that any two flying tracks do not possess intersections. Yet we manage to conquer the challenge and successfully present a structural model in response to the task.

Our model effectively achieves the goal of finding the shortest total flight distance without any crashes to meet the safety requirements. It is definitely a feasible solution and could handle large quantities of data. Admittedly, our model may face few challenges when applying into the reality since we take account of several factors to simplify the model due to limited time. But we firmly believe that with more time to adjust and improve the flight paths, and more factors being taken into consideration, the model can be enhanced to a higher and more realistic level. In addition, our model generalizes the algorithm used in the control and orientation of UAVs, flexible and broad enough to accommodate various local conditions, safety concerns and other unexpected incidents. We proudly declare that the application of our model maintains a vast potential for future development including combat drone control and orientation system.

Attached below are our designed images for the aerial light show. We express our sincerest gratitude for your trust in us to organize this festive event, which we hope will develop into a worldwide carnival in which everyone enjoys and appreciates our aerial light show.

Best Wishes,
Team # 7379



```
> check()
None of the drones will crash in the flight.
```

Figure 1: The shape of the Ferris wheel

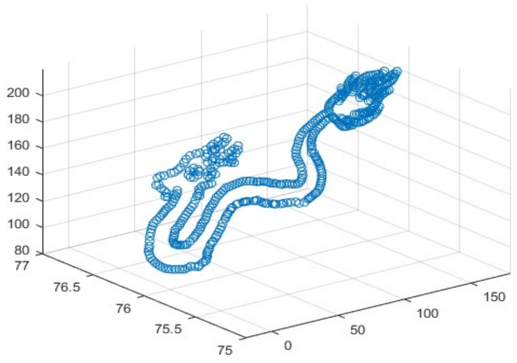


Figure 2: The shape of the dragon

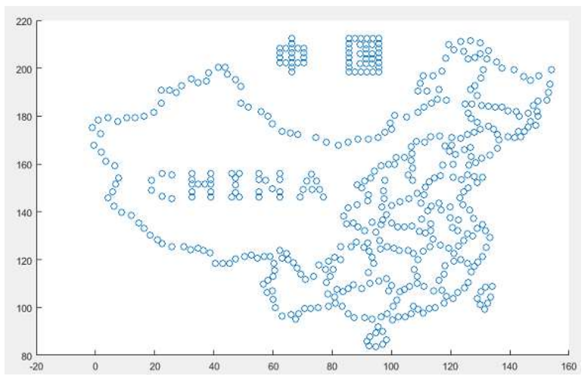


Figure 3: The shape of the map of China

Contents

1. Background

1.1 Current Research Status

Pilotless aircraft, often referred to as "unmanned aerial vehicle", with the abbreviation of “UAV”, is a pilotless aircraft operated by radio remote control equipment, its own program control device, or operated entirely or intermittently by the on-board computers. Since the birth of the aircraft in the early twentieth century, people have proposed the idea of unmanned aircrafts because of the safety problem of the aircrafts. In 1930s, the British Ferrell company remade a double-fixed wing aircraft into an unmanned drone, which was the first time UVA had entered the history of aviation. Since then, UVA has been used in a lot of domains including aerial photography, news report, wildlife protection and also performances despite the military use. In a recent UVA show performed by YiHang GHOSTDRONE 2.0, engineers designed a set of intelligent and efficient unmanned aerial vehicle remote control system, which realized the function of using only one computer as a ground control station to autonomously control, monitor the flight task of thousands of UAVs, and set the color change of aircrafts’ lights. They presented a large-scale visual feast in the form of fancy lighting show in only 15 minutes.

In our task, the main goal is to organize a beautifully performed light show by using approximately 480 drones and create 3 possible displays. The main challenge is how to minimize the total time the whole performance would cost due to the limited time a drone could fly constantly in the sky and how to reduce the total distance that drones would move from one displayed pattern to another.

1.2 Restatement of the Problem

We are asked to organize an outdoor aerial light show with the utilization of drones. The main challenge is to depict three possible images and the overall show process as well as determine the optimal flight paths of each drone device that would simulate our image on display. To be more specific, the required launch area, required number of drones, required air space and the transitioning flight paths between two images for our three images----the Ferris wheel, the dragon and the map of China. The great barrier lies upon us is how to avoid crashes of the flight paths during the display and the transitions between two images. The effect of changes in the total distance, average distance, and time required based on the consideration of safety concerns, limited space and limited flying speed will also have to be investigated to explore the advantages we may achieve in future events.

2 Assumptions

2.1 Assumptions

We make the following assumptions about the initial conditions and basic parameters of the drones in this paper. The functions and performance parameters of each drone may have a slight difference, but in order to simplify the model, we assume all of them to be the same. Through the reference of the Shooting Star™ drones used in the Intel® light show [1], some key parameters of the drones are assumed and set as following in table 1:

Table 1: Basic Parameters of the drones

Size	384mm×384mm×93 mm
Propeller diameter	6 inches (15 cm)
Maximum take-off weight	280 grams
Maximum time of flight	25 minutes
Maximum distance of flight	1.5 km
Maximum airspeed of flight	10 m/s
Maximum airspeed of light show mode	3 m/s

Each drone can automatically measure its horizontal position and vertical height with the utilization of GPS and barometer. Each drone can determine the target location of the next moment according to the preset track. With the flight control system for navigation, the drones can complete the overall effect of structural formation. The flying path of each drone may vary from the preset track, but we approximate each flight path to be a linear function to simplify our model.

Considering the convenience of formation, the take-off site and landing position of each drone is fixed to form a rectangle. To minimize the required launch space, we shape all the drones to remain a regular triangle distance with each other instead of simply fitting them into the points in a rectangle. To prevent mutual interference and ensure safety, the distance between each other during the whole display is further than the minimum safety distance by our definition.

Meteorological conditions can be complicated and unpredictable in reality; therefore, we ignore the factors related to meteorological conditions such as the wind speed and non-ideal weather. Other unexpected incidents like breakdowns of drones and defaults are assumed to be impossible as well. Additional assumptions are made to simplify analysis for individual sections. These assumptions will be discussed at the appropriate locations.

2.2 Definitions

We define the flight from the take-off apron to the Ferris wheel as the **stage Ⅰ flying process**. Similarly, the flight from the Ferris wheel to the dragon, the flight from the dragon to the map of China, and the flight from the map of China to the landing apron are defined as **stage Ⅱ flying process**, **stage Ⅲ flying process**, and **stage Ⅳ flying process**.

To better build our model, we defined the minimum safety distance between the center of any two drones is A meters, and the maximum observing distance between the center of any two drones is B meters. As a result, the distance between any two drones would be

$$A \le \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \le B$$

In our case, each drone possesses a size of 384mm×384mm×93mm, with the propeller of approximate 15cm-diameter. Therefore, we plug A=1m to be the minimum safety distance between any two drones by our assumption when taking both the lower and upper bounds into consideration. According to the upper and lower bounds of the distance between any two drones, and the preset size of the Ferris wheel design, we determine that 477 drones are required to accomplish the display. In other words, the preset size of the Ferris wheel is 108 m-diameters. We then use the size of the Ferris wheel to calculate the drones needed to form the shape. The result is about 480. And in later practices, we discover that there are 3 drones of no use in shaping the pattern. Therefore, in the name of economy, we eliminate those 3 drones and determine the final number of drones in total is 477.

In order to simplify our calculation and unify the coordinates in the three displays, we mark all the drones as drone 1-477. To clarify, the number of each drone does not change from one image to another. In this way, we can easily present the transition track of each drone's coordinates during the overall 5 stages flying process.

The notation in the locus functions that appear in the rest of our paper is defined in the following table 2:

Table2: the definition of notations in the locus function

Notation	Definition
x_0	The matrix containing all the drones' tridimensional coordinates
s_i	The sleeping time before take-off time of the i^{th} drone
d_i	The actual flying time of the i^{th} drone
l_i	The hovering time after arriving at the target position of the i^{th} drone
T	The time to the start time of the current process

3 Models

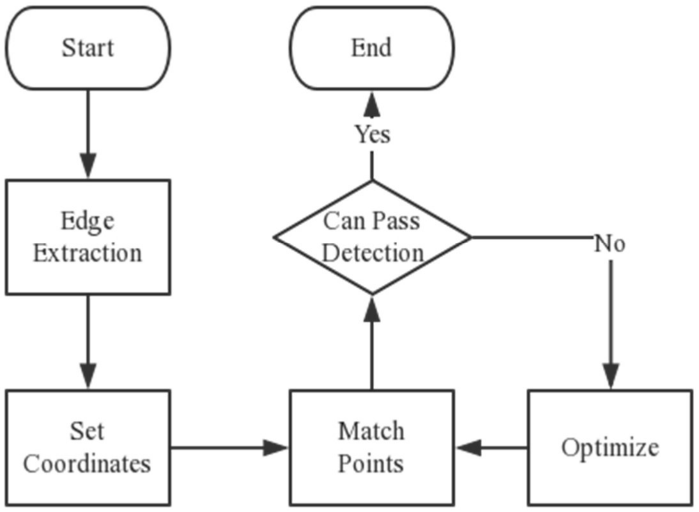
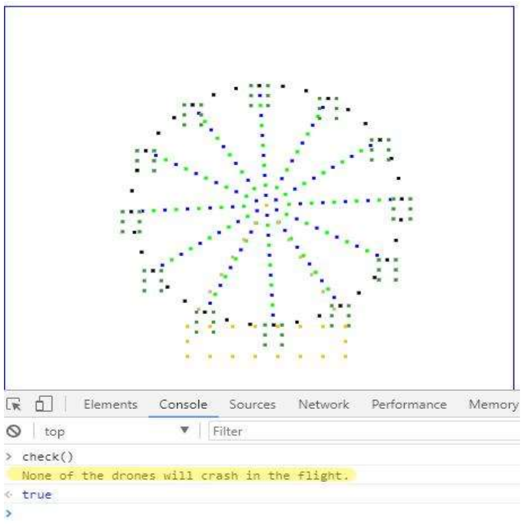


Figure 4: The flow chart of the whole modeling process

3.1 Modeling

In response to the Mayor's demand of investigating the idea of organizing an aerial light show, we start off our model by designing the three possible images that we would like to present. The first two images----the Ferris wheel and the dragon----are mandatory. With the help of reference images, we are able to design the stick figures of these two displays. With respect to the third one, we designed a map of China with a Chinese flag occupying the large territory in the west, representing the theme of "what ethnic is what worldwide".

After establishing the images on display, we set up a left-handed three-dimensional coordinate system with the origin at the lower left corner of the parking apron. To be more specific, x-axis stands for the north direction; y-axis stands for the east direction; z-axis stands for the height. Then we pick out points from the images with the criterion that the coordinate difference between any two adjacent points remains no less than the minimum safety distance. Under this prerequisite, our minimum number of drones required would be 477, as each point selected out from the images represents a drone in reality. To validate the feasibility of the images in our model, we figure out a detection function to examine whether the distance between every two drones is further than the minimum safety distance. The main method being used in the function, or the "program", is to detect whether there is a drone in a circumference within the radius of the minimum safety distance or not. More specifically speaking, for each drone in the Ferris wheel, if there is a drone detected within the range of the minimum safety distance, it means that the two drones would ultimately crash with each other. The test result for the Ferris wheel from our computer simulation is shown below:



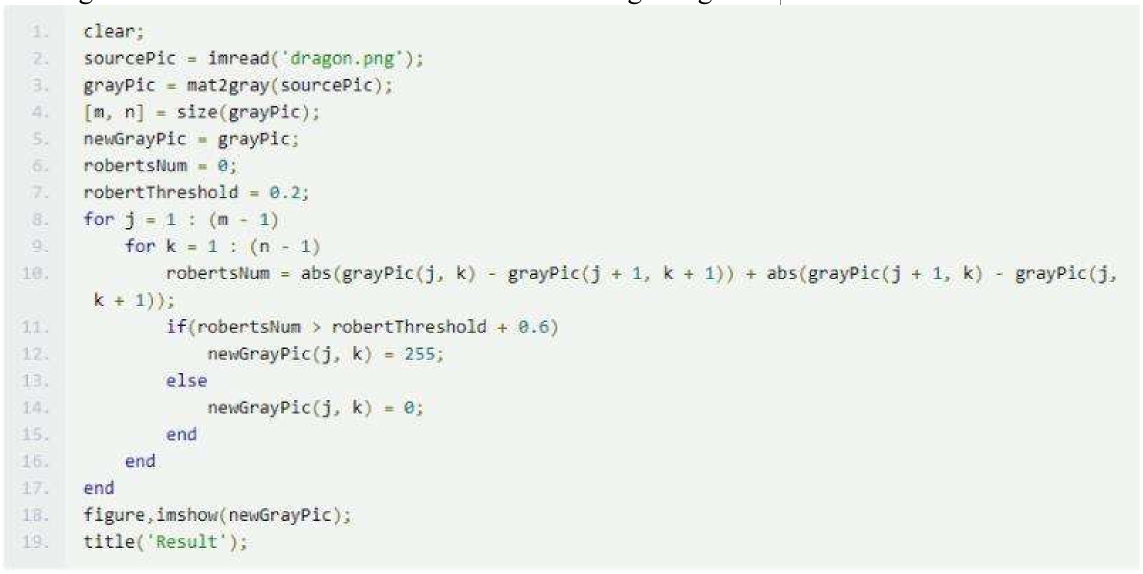
```
> check()
  None of the drones will crash in the flight.
< true
>
```

Figure 5: The result of the testing program of the Ferris wheel

From the figures, we can naturally reach the conclusion of any two adjacent drones won't crash with each other. With respect to the other two images----the dragon and the map of China, the crashing test result is shown in the appendix. All the test reports clear the possibility of crashes between drones.

3.1.1 Edge Extraction

As for the design process of the dragon pattern, we first download a picture of dragon and erase the dragon's claws to simplify our model and reduce the total number of drones needed to gain better control. Then we use the Roberts edge detection operator [2] to create the simplified pattern of the dragon. The Roberts edge detection operator computes the principle of gradient based on the difference between any pair of perpendicular directions, using the difference between adjacent pixels in the diagonal direction. The code is shown as following in figure 6:



```
clear;
sourcePic = imread('dragon.png');
grayPic = mat2gray(sourcePic);
[m, n] = size(grayPic);
newGrayPic = grayPic;
robertsNum = 0;
robertThreshold = 0.2;
for j = 1 : (m - 1)
    for k = 1 : (n - 1)
        robertsNum = abs(grayPic(j, k) - grayPic(j + 1, k + 1)) + abs(grayPic(j + 1, k) - grayPic(j, k + 1));
        if(robertsNum > robertThreshold + 0.6)
            newGrayPic(j, k) = 255;
        else
            newGrayPic(j, k) = 0;
        end
    end
end
figure,imshow(newGrayPic);
title('Result');
```

Figure 6: The MATLAB code of the Roberts edge detection operator

The simplified version of the dragon pattern is shown below in figure 7:

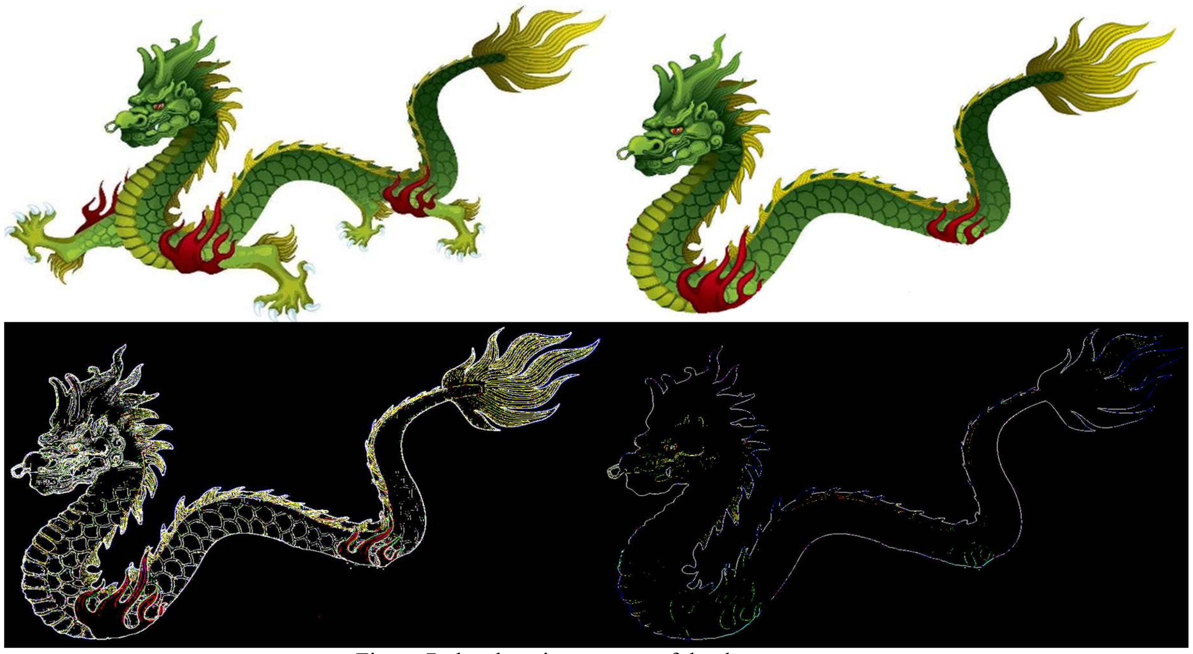


Figure 7: the changing process of the dragon pattern

Then we apply the Sobel operator to automatically generate the scatter diagram as below. By using this program, we do not need to pick out the point and take down the coordinates by ourselves; instead, it can all be done automatically by the computer. The result is shown in figure 8 and the code is shown in figure 9:

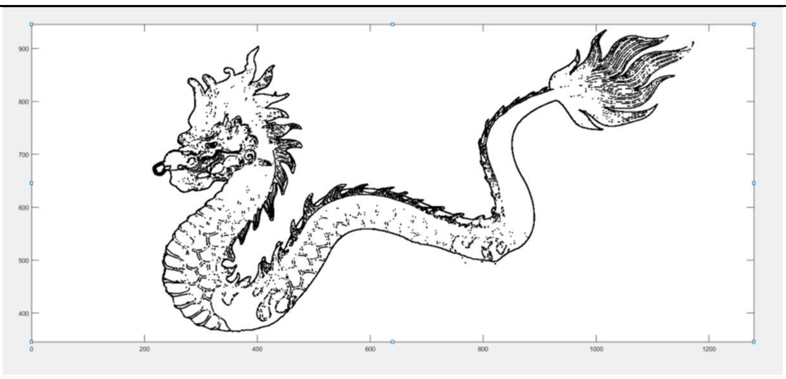
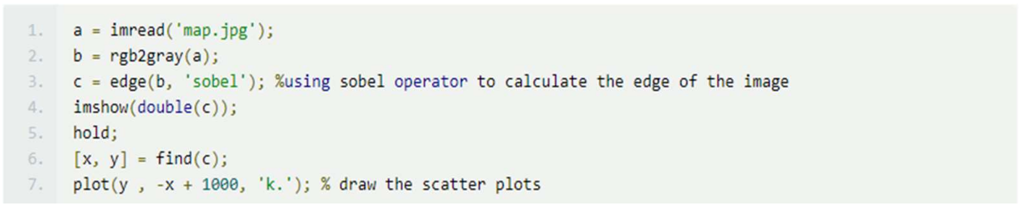


Figure 8: the scatter diagram of the map of China


```
a = imread('map.jpg');
b = rgb2gray(a);
c = edge(b, 'sobel'); %using sobel operator to calculate the edge of the image
imshow(double(c));
hold;
[x, y] = find(c);
plot(y , -x + 1000, 'k.'); % draw the scatter plots
```
Figure 9: the program used to generate the scatter diagram

After the generation of the scatter diagram, we pick out the points with an interval of 1.5m on the edge of the diagram with help of MATLAB. After the adjustment by our team, the ultimate design of the dragon pattern and the map of China as well as the pattern of Ferris wheel is shown below in figure 10, figure 11 and figure 12:

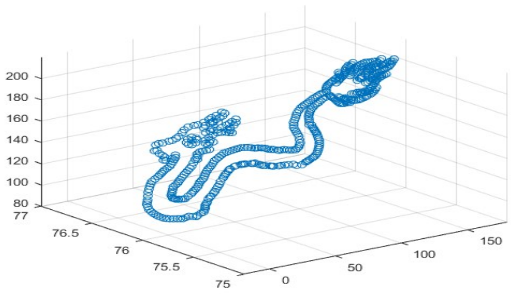


Figure 10: The shape of the dragon

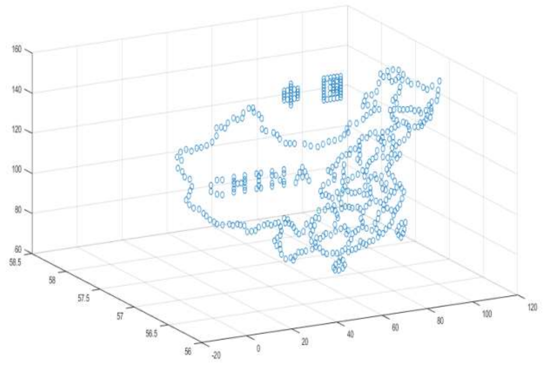


Figure 11: The shape of the Ferris wheel

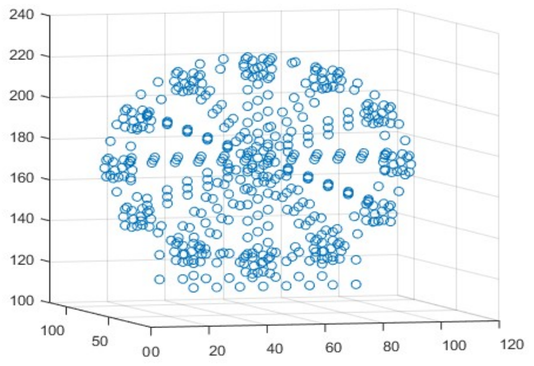


Figure 12: The shape of the Ferris wheel

Lastly, we calculated the three-dimensional coordinate of each drone and use the coordinates to form matrixes for later steps in modeling. All the coordinates in the three images are attached in the appendix.

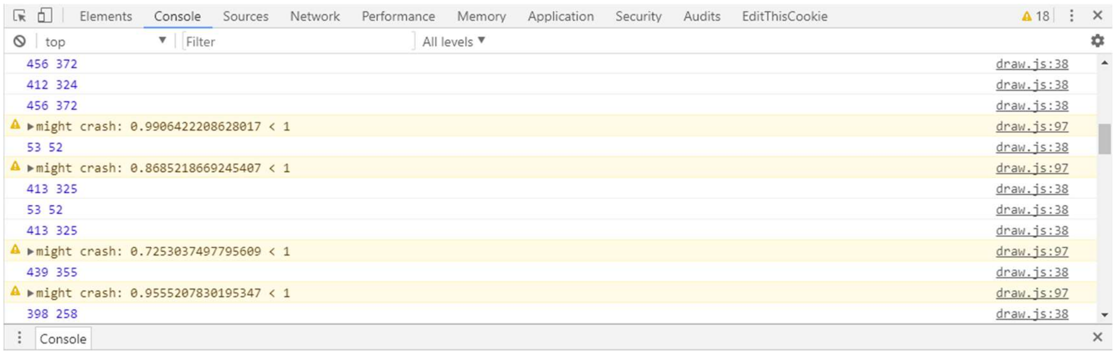
3.1.2 Detection Function

This part will mainly discuss the detection function and its optimization. The detection function is of vital importance in the modeling when taking safety concerns into consideration. It will make the bugfixes and adjustment of crashed drones much more quickly and thus improve our efficiency. Therefore, the following text will cover the algorithm of the detection function and how it can be optimized.

According to the minimum safety distance mentioned in the Definition part, the distance between any two drones has to be further than this minimum value to meet the safety demands. First, our team utilizes the Euclidean distance as part of the detection function. We calculate the Euclidean distance between every two drones at the same time by the following formula

$$\mathrm{d(t)} = \sqrt{(\mathrm{x}_1 - \mathrm{x}_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

In the formula above, x, y, and z represent the x-axis, y-axis, and z-axis coordinates of the drones respectively. If the distance calculated by the predetermined formula is lower than the minimum safety distance, then the program will display an error message. The program can calculate the total number of the error messages and report them to our team members. An example is presented in figure 13:



```
456 372                                          draw.js:38
412 324                                          draw.js:38
456 372                                          draw.js:38
▶might crash: 0.990642208628017 < 1              draw.js:97
53 52                                            draw.js:38
▶might crash: 0.8685218669245407 < 1             draw.js:97
413 325                                          draw.js:38
53 52                                            draw.js:38
413 325                                          draw.js:38
▶might crash: 0.7253037497795609 < 1             draw.js:97
439 355                                          draw.js:38
▶might crash: 0.9555207830195347 < 1             draw.js:97
398 258                                          draw.js:38
```

Figure 13: an example of using the detection function

Then we improve the detection function using vectors. It can be applied to detecting the minimum distance between two straight lines with the advantage that it calculates the distance without the restrictions of time since time is no longer considered as an independent variable of the detection function.

Suppose the starting and ending points of two drones are $\mathrm{A}(\mathrm{x}_1, \mathrm{y}_1, \mathrm{z}_1)$, $\mathrm{B}(\mathrm{x}_2, \mathrm{y}_2, \mathrm{z}_2)$, $\mathrm{C}(\mathrm{s}_1, \mathrm{t}_1, \mathrm{w}_1)$ and D $(\mathrm{s}_2, \mathrm{t}_2, \mathrm{w}_2)$. Then we can define the vector **n** as

$$\boldsymbol{n} = \boldsymbol{AB}$$

The vector **n** is perpendicular to both line AB and CD in the space. Then we calculate the minimal distance between two lines AB and CD by the formula

$$d = \frac{\boldsymbol{n}}{|\boldsymbol{n}|} \cdot \boldsymbol{AC}$$

Vector **AC** can be replaced by any vectors whose starting point is on line AB and ending point is on line CD. The notation *d* in the formula above represents the projection of vector **AC** on the vector **n**'s direction, which equals to the minimal distance between the two lines. Then the program will follow the process in figure 14:

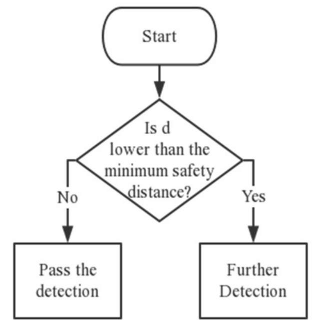


Figure 14: the process flowchart

A sketch of the vector method used in optimizing detection function is presented in figure 15:

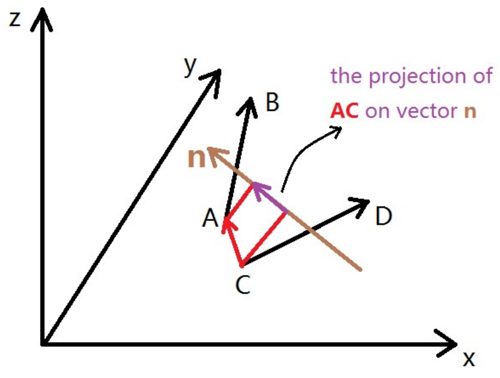


Figure 15: the sketch of the vector method

In conclusion, the second algorithm can be used as a filter program to reduce the workload of the program by straining off the matches that certainly won't collide with one another. As for the comparison and contrast between the two detection methods, the common method is easy to operate while the vector method has a smaller calculation amount. Considering that the computer can handle large quantity of calculation, we tend to adopt the common method to detect the distance between any two drones.

3.2 The Stage Ⅰ Flying Process

The main challenge in this part of our model is how to distribute the flight path for each drone and make sure that there’s no intersection in any two flight paths. Since our design of the Ferris wheel is a tridimensional spatial structure which has 5 layers including the front, the back, the frame and the carriages. The following figure 16 shows the three views of the Ferris wheel:

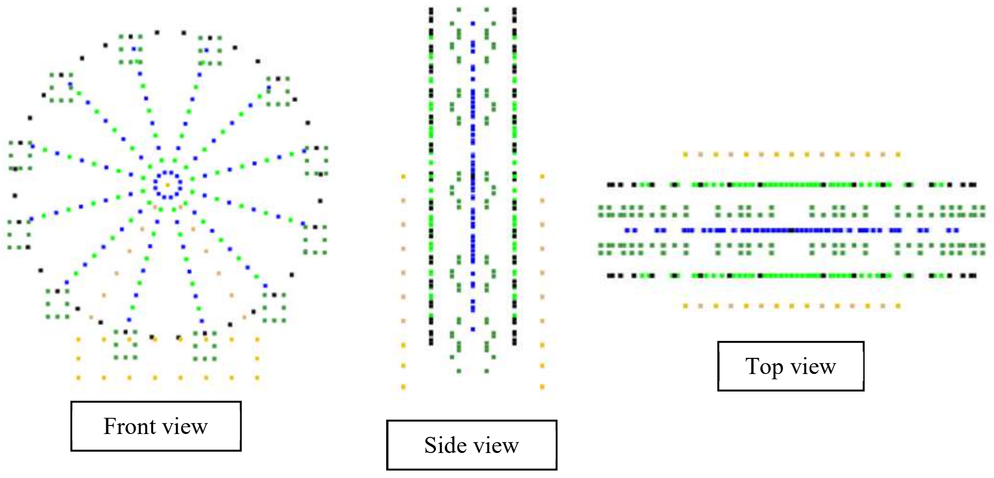


Figure 16: the three views of the Ferris wheel

The first step in our plan is controlling the drones to take off from the apron and rise up in the midair to form a horizontal Ferris wheel, and then flipping it to the upstanding position. In order to accomplish this goal, we divide all the drones in the horizontal Ferris wheel into several flying groups using the clustering analysis [3]. The cluster criterion is the z-axis parameters. In other words, by the standard of height, we group the drones into 5 levels to gain a better control. The clustering result is shown in figure 17:

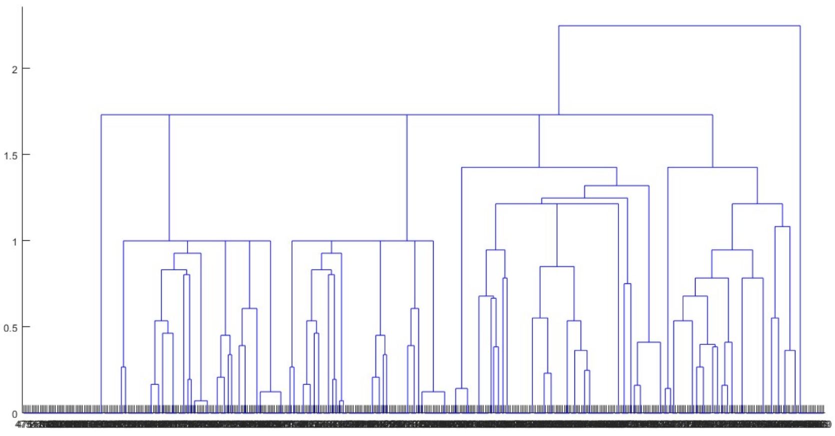


Figure 17: the clustering analysis result of the Ferris wheel

Our next step is numbering the clusters from top to bottom as cluster 1-5 and determining the priority of being distributed to a coordinate on the take-off apron for each drone. According to the clustering analysis result, the priority order would be cluster 5, cluster 3, cluster 2, cluster 1, and cluster 4; also, drones that have higher priority will take off earlier than those that have lower priority.

First, we match every drone in cluster 5 with a drone on the parking apron that has the least Euclidean distance, which is

$$\mathrm{pdist} = \sqrt{(\mathrm{x}_1 - \mathrm{x}_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Then we match every drone in cluster 3 with a drone from the set of drones remaining on the parking apron that has the least Euclidean distance. Similarly, we repeat the matching procedure for another three times with the order of cluster 2, cluster 1, and cluster 4, and successfully form an overall match from drones forming the Ferris wheel to drones on the apron.

After affirming the initial coordinates and the final coordinates, we shift our focus to the flight paths that connect the initials with the finals. What we need to deal with now is to avoid the crashes between any two drones during the whole stage Ⅰ flying process. In the purpose of eliminating the possibility of crashes between any two drones, we adopt the method of forming the Ferris wheel on a horizontal plane with the parameter on the z-axis to be 150 first and then flipping the Ferris wheel around central axis of y=76, z=150 to shape the final image of the Ferris wheel that we desire to display.

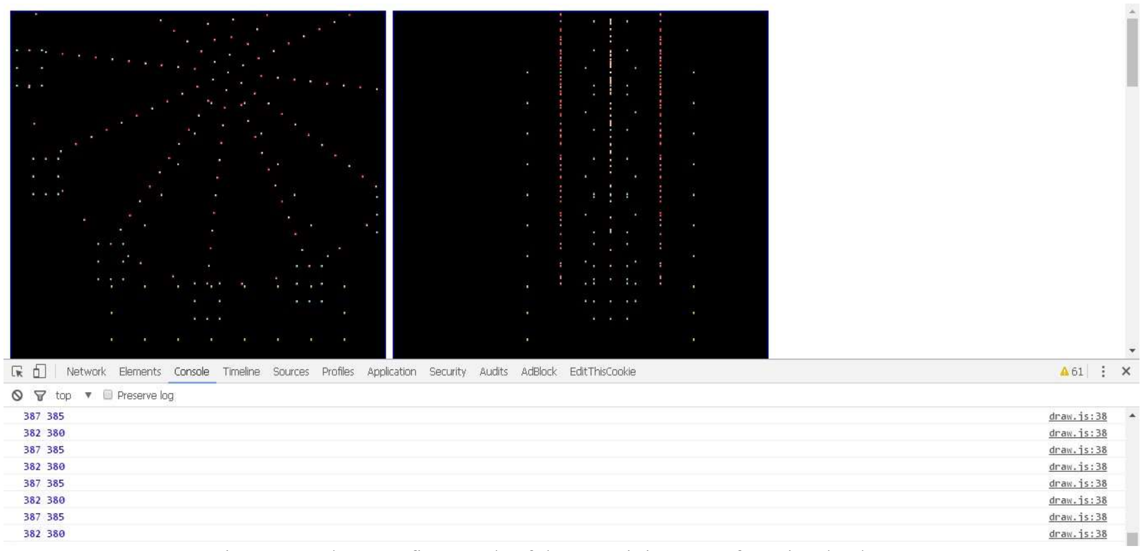
As for the first part of the stage I flying process in which the drones fly to the horizontal plane, we connect the initial coordinate on the apron with the final coordinate in the horizontal Ferris wheel using a straight path for each drone. Hence, the general flight path for the drones can be described in a linear function, which is

$$x_i(t) = \begin{cases} x_{0i} & (0 \le t < s_i) \\ \dfrac{x_{1i} - x_{0i}}{d_i + s_i} \cdot (t - s_i) + x_{0i} & (s_i \le t < d_i + s_i) \\ \dfrac{x_{2i} - x_{1i}}{d_i + s_i + l_i} \cdot (t - d_i - s_i) + x_{1i} & (d_i + s_i < t \le d_i + s_i + l_i) \end{cases}$$

$$y_i(t) = \begin{cases} y_{0i} & (0 \le t < s_i) \\ \dfrac{y_{1i} - y_{0i}}{d_i + s_i} \cdot (t - s_i) + y_{0i} & (s_i \le t < d_i + s_i) \\ \dfrac{y_{2i} - y_{1i}}{d_i + s_i + l_i} \cdot (t - d_i - s_i) + y_{1i} & (d_i + s_i < t \le d_i + s_i + l_i) \end{cases}$$

$$z_i(t) = \begin{cases} z_{0i} & (0 \le t < s_i) \\ \dfrac{z_{1i} - z_{0i}}{d_i + s_i} \cdot (t - s_i) + z_{0i} & (s_i \le t < d_i + s_i) \\ \dfrac{z_{2i} - z_{1i}}{d_i + s_i + l_i} \cdot (t - d_i - s_i) + z_{1i} & (d_i + s_i < t \le d_i + s_i + l_i) \end{cases}$$

The next step after setting a general function for the flight paths of the drones is to validate the feasibility through computer animation. We begin our calculation by running a test program similar to the program mentioned earlier that is used to examine the distance between any two drones in the static initial state of each image to examine the flight locus of each drone under the condition that launch time interval between any two drones is $\frac{1}{90}$ second. The result of the first examining test is shown in the following figure 18:



```
387 385                                                      draw.js:38
382 380                                                      draw.js:38
387 385                                                      draw.js:38
382 380                                                      draw.js:38
387 385                                                      draw.js:38
382 380                                                      draw.js:38
387 385                                                      draw.js:38
382 380                                                      draw.js:38
```

Figure 18: The very first result of the examining test of Ferris wheel

From the diagram we can summarize that there are 61 drones in total that may crash with one another during the whole flying process. More specifically, the crashed pairs of drones include drone (387, 385), drone (382, 380), etc. However, the crash of planes cannot be tolerated due to safety concerns. To optimize this result, we continue to adjust the launch time and time interval between each other for the crashed drones and manage to reduce the number of crashed drones to 27. With the utilization of manually postponing the take-off time of the crashed drones, we manage to completely eradicate any remaining possibility of crashing and the computer simulation validates the feasibility of our model. The specific postponing time lag is listed in the following table 3 and the testing report on the crashed drones is shown in figure 19 below:

Table 3: the adjustment of the take-off time for crashed drones in process 1

Group	Delayed drone number	Revised time lag
1	386, 139, 152, 234, 233, 348, 366, 381, 383, 400, 185, 214, 209, 210, 360, 416, 424, 436	1s later
2	152	1.5s later
3	344, 228, 322, 324, 283	1s earlier
4	230, 208, 209	1.5s later

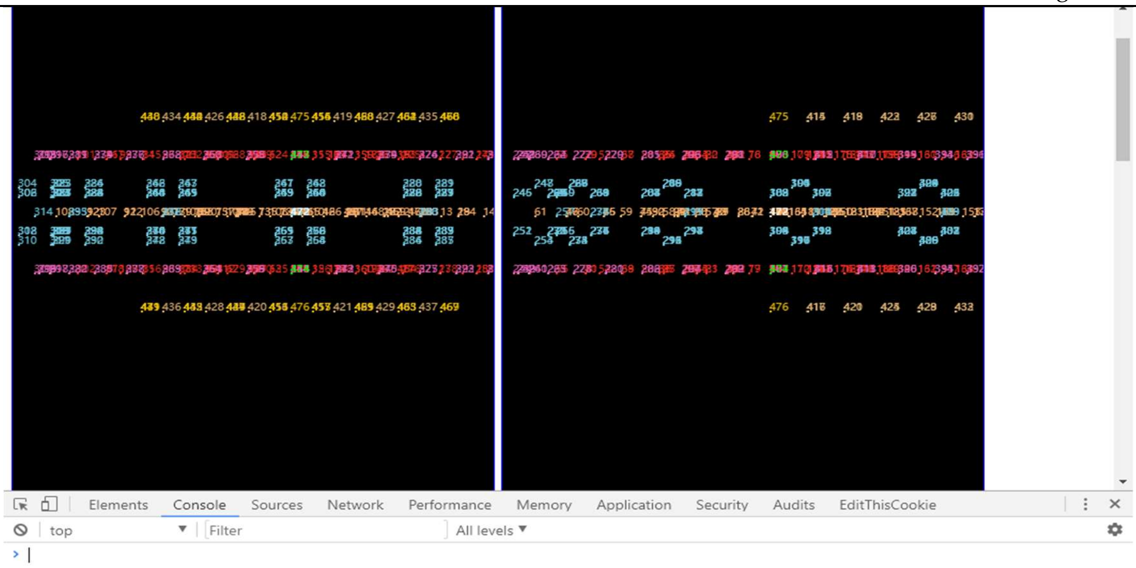


Figure 19: The testing report of the crashed drones on Ferris wheel

From the testing report of the crashed drones on Ferris wheel, it can be concluded that none of the drones would crash during the flying process till shaping the horizontal Ferris wheel. Then we flip the horizontal Ferris wheel into the upstanding position. The operative procedure is to make all the drones rotate at the same angular velocity to the upstanding shape. More detailed explanation is discussed later in the section titled “**The Rotation of the Designs**”.

At this moment, our model has accomplished the flight up until forming the Ferris wheel pattern. The stage I flying process is well accomplished.

3.3 The Stage II Flying Process

The main challenge of this process is how to minimize the total distance of drones flying in the air and make sure the flying pattern is optimal. In order to achieve this goal, we first divide the dragon pattern into three groups according to the shape and the outer contour of the dragon for later distributing drones on the dragon pattern to drones on the Ferris wheel. In the next step, we utilize binary integer programming to find solution that guarantees the total distance to be the shortest in each group. Then we use the testing program to check if there are drones that may crash into one another and adjust the matching method accordingly.

The dragon consists of three parts including the head, the body and the tail part, and our team divides the dragon into three groups in accordance with the shape. We here define the head part of the dragon as group 1, the tail part as group 2 and the main body part as group 3. The division of the dragon pattern is presented in figure 20:

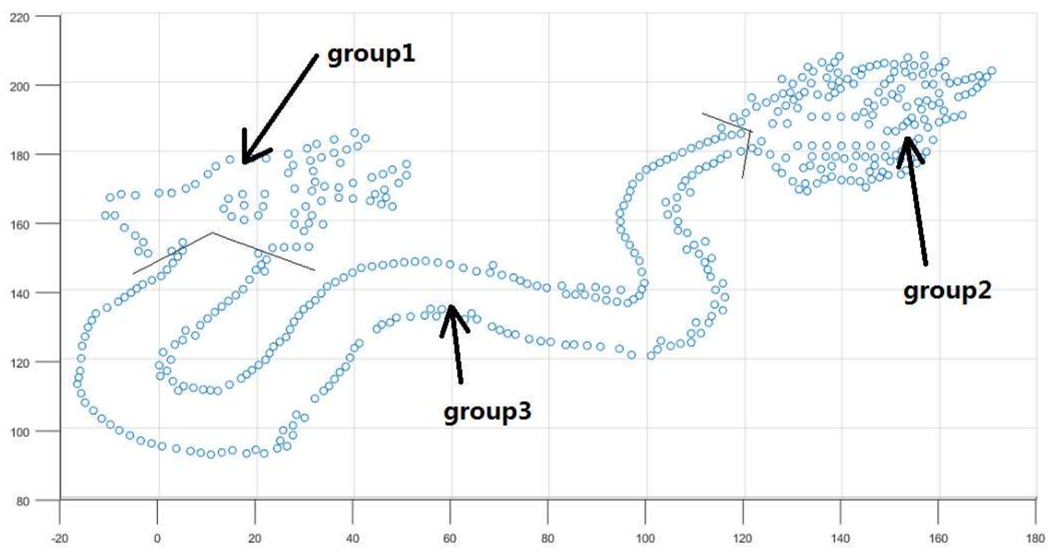


Figure 20: The division of the dragon

Then we calculate the distance between every drone in the Ferris wheel (the one after rotation) and the dragon by using Euclidean distance as mentioned in the former process, which is

$$\mathrm{pdist} = \sqrt{(\mathrm{x}_1 - \mathrm{x}_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

In this process, we do not match the drones based on the minimum Euclidean distance any more. Instead, we utilize binary integer programming to achieve our goal. The binary integer programming is a kind of special integer programming in which the conclusive variables include only 0 and 1. It can be used to find the overall optimal solution. In other words, the overall optimal solution in the process being discussed now is the shortest distance drones fly in total. We will give the specific explanation in the text below.

Our team first uses the method of binary integer programming in group 1. We number the drones in the Ferris wheel from 1 to 477 and the drones in group 1 from 1 to 78. Here we define d_{ij} as the distance between the i^{th} drone in the Ferris wheel and j^{th} drone in group 1. Then we define c_{ij} to judge the following situation: if the i^{th} drone in the Ferris wheel matches the j^{th} drone in group 1, then we define $c_{ij} = 1$. Otherwise, we define $c_{ij} = 0$.

The total number of the drones in the group 1 is 78. Therefore, according to the definition above, we can define the objective function Z as the total flight distance of all the drones in group 1 as

$$\min Z = \sum_{j=1}^{78} \sum_{i=1}^{477} d_{ij} c_{ij}$$

On one hand, each drone in the Ferris wheel can only match up to one drone in group 1; on the other hand, each drone in group 1 must match one drone in the Ferris wheel. Therefore, the constraint conditions are as the following:

$$\sum_{j=1}^{78} c_{ij} \le 1, \;\; i = 1, 2, \cdots, 477$$

$$\sum_{i=1}^{477} c_{ij} = 1, \;\; j = 1,2, \cdots ,78$$

Next, we utilize LINGO to run the binary integer programming; the code we used can be font in the **Appendix** part.

After matching the drones in group 1, there are 399 drones left. Then we apply the method of binary integer programming in group 2. Similarly, we number the drones remain in the Ferris wheel from 1 to 399 and the drones in group 2 from 1 to 167. Accordingly, we define d_{ij} as the distance between i^{th} drone remaining in the Ferris wheel and j^{th} drone in group 2. Then we define c_{ij} to judge the following situation. To be more specific, if the i^{th} drone remaining in the Ferris wheel matches the j^{th} drone in group 2, then we define $c_{ij} = 1$. Otherwise, we define $c_{ij} = 0$.

The total number of the drones in the group 2 is 167. Therefore, according to the definition above, we can define the objective function Z for group 2 as

$$\min Z = \sum_{j=1}^{167} \sum_{i=1}^{399} d_{ij} c_{ij}$$

Every drone remaining in the Ferris wheel can only match up to one drone in group 2, besides, every drone in group 2 must match one drone in the Ferris wheel. Therefore, the constraint conditions are as the following:

$$\sum_{j=1}^{167} c_{ij} \le 1,\ \ i = 1,2,\cdots,399$$
$$\sum_{j=1}^{399} c_{ij} \le 1,\ \ j = 1,2,\cdots,167$$

Then we again use LINGO to run the binary integer programming, the code can be found in the **Appendix** part.

Then, there are only 232 drones left in group 3. We respectively number the drones in the Ferris wheel and group 3 from 1 to 232. As mentioned above, we similarly define d_{ij} and c_{ij}. The total number of the drones in the group3 is 232. Therefore, according to the definition above, we can define the objective function Z for drones in group 3 as

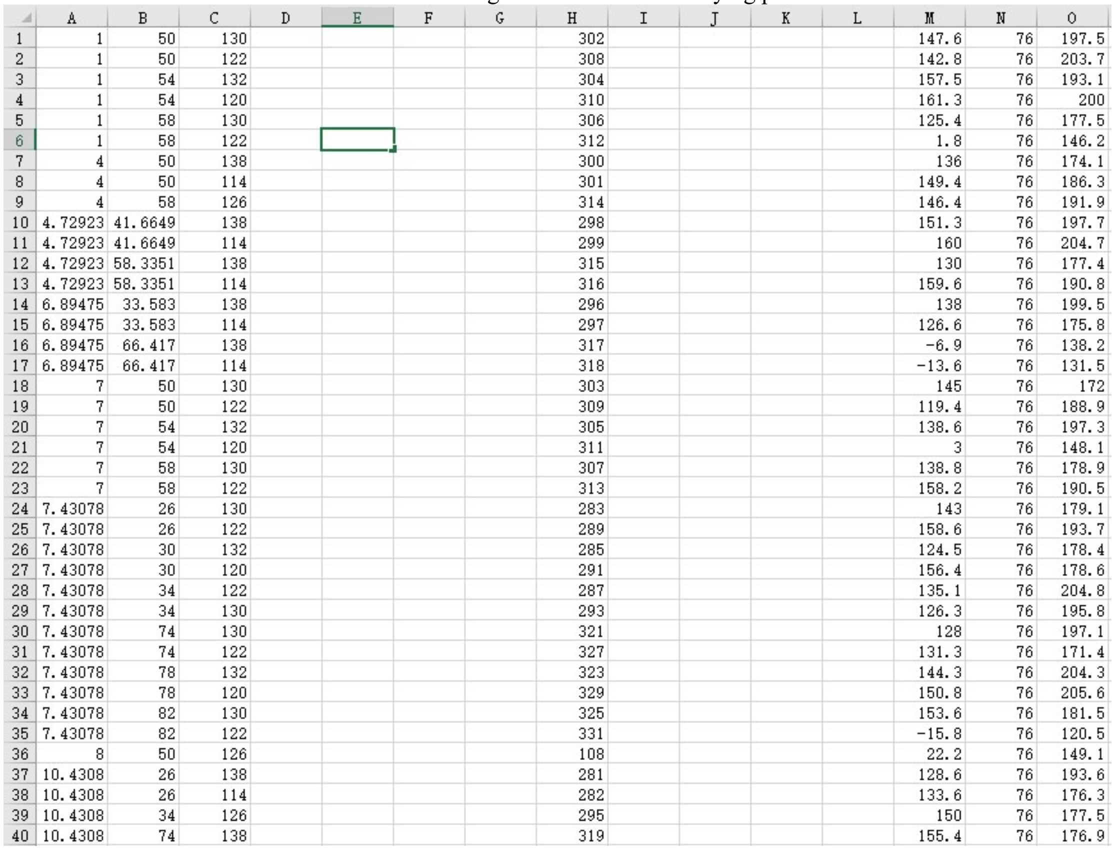
$$\min Z = \sum_{j=1}^{232}\sum_{i=1}^{232} d_{ij}c_{ij}$$

Each drone in group 3 must match one drone remaining in the Ferris wheel. Therefore, the constraint conditions are as the following:

$$\sum_{j=1}^{232} c_{ij} = 1, \qquad i = 1,2,\cdots,232 \qquad j = 1,2,\cdots,232$$

Again, we use LINGO to run the binary integer programming; the code can be found in the **Appendix**. Till now, every drone in stage Ⅱ process is matched. Part of the matching results is shown in the following table 4:

Table 4: The matching result of the second flying process



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	1	50	130					302					147.6	76	197.5
2	1	50	122					308					142.8	76	203.7
3	1	54	132					304					157.5	76	193.1
4	1	54	120					310					161.3	76	200
5	1	58	130					306					125.4	76	177.5
6	1	58	122					312					1.8	76	146.2
7	4	50	138					300					136	76	174.1
8	4	50	114					301					149.4	76	186.3
9	4	58	126					314					146.4	76	191.9
10	4.72923	41.6649	138					298					151.3	76	197.7
11	4.72923	41.6649	114					299					160	76	204.7
12	4.72923	58.3351	138					315					130	76	177.4
13	4.72923	58.3351	114					316					159.6	76	190.8
14	6.89475	33.583	138					296					138	76	199.5
15	6.89475	33.583	114					297					126.6	76	175.8
16	6.89475	66.417	138					317					-6.9	76	138.2
17	6.89475	66.417	114					318					-13.6	76	131.5
18	7	50	130					303					145	76	172
19	7	50	122					309					119.4	76	188.9
20	7	54	132					305					138.6	76	197.3
21	7	54	120					311					3	76	148.1
22	7	58	130					307					138.8	76	178.9
23	7	58	122					313					158.2	76	190.5
24	7.43078	26	130					283					143	76	179.1
25	7.43078	26	122					289					158.6	76	193.7
26	7.43078	30	132					285					124.5	76	178.4
27	7.43078	30	120					291					156.4	76	178.6
28	7.43078	34	122					287					135.1	76	204.8
29	7.43078	34	130					293					126.3	76	195.8
30	7.43078	74	130					321					128	76	197.1
31	7.43078	74	122					327					131.3	76	171.4
32	7.43078	78	132					323					144.3	76	204.3
33	7.43078	78	120					329					150.8	76	205.6
34	7.43078	82	130					325					153.6	76	181.5
35	7.43078	82	122					331					-15.8	76	120.5
36	8	50	126					108					22.2	76	149.1
37	10.4308	26	138					281					128.6	76	193.6
38	10.4308	26	114					282					133.6	76	176.3
39	10.4308	34	126					295					150	76	177.5
40	10.4308	74	138					319					155.4	76	176.9

Column A, B, C represents the x-coordinate, y-coordinate and z-coordinate of drones in the Ferris wheel respectively. Column H represents the drone's original number. Column M, N, O represents the x-coordinate, y-coordinate and z-coordinate of drones in the dragon respectively. The complete results of the matching in the second process are in the attachment.

The general flight path in Stage II flying process can be described in the function below:

$$\begin{cases} x_i(t) = \begin{cases} x_{0i}\ (t < s_i) \\ \dfrac{x_{1i} - x_{0i}}{d_i} \times (t - s_i) + x_{0i}\ (s_i \le t < l_i) \end{cases} \\ y_i(t) = \begin{cases} y_{0i}(t < s_i) \\ \dfrac{y_{1i} - y_{0i}}{d_i} \times (t - s_i) + y_{0i} + g_i(t - s_i)(s_i \le t < l_i) \end{cases} \\ z_i(t) = \begin{cases} z_{0i}\ (t < s_i) \\ \dfrac{z_{1i} - z_{0i}}{d_i} \times (t - s_i) + z_{0i}\ (s_i \le t < l_i) \end{cases} \end{cases}$$

Where

$$g_i(t) = a_i\left[\left(t - \frac{d_i}{2}\right)^2 - \frac{{d_i}^2}{4}\right](0 \le t < d_i)$$

Where

$$a_i = v_i \times A_i$$

3.4 Stage Ⅲ flying process

The main challenge of this process consists of two goals. One is to minimize the total transition time from the dragon pattern to the map of China by means of optimization of the total distance. The other is to ensure that none of the drones would crash with each other in purpose of satisfying the safety concerns. To accomplish both of the goals, we apply the multi-target programming to our model. In other words, we find the optimal solutions that meet either goal separately and then weighting the two solutions to find the overall best-fit model.

With the same way as the stage II, we calculate the distance between every drone in the two patterns using Euclidean distance to utilize binary integer programming. However, in lieu of dividing the entire patterns into several clusters, we regard the overall drone fleet as a whole to simplify the model. Possessing 477 drones in total, the formulas are as following likewise:

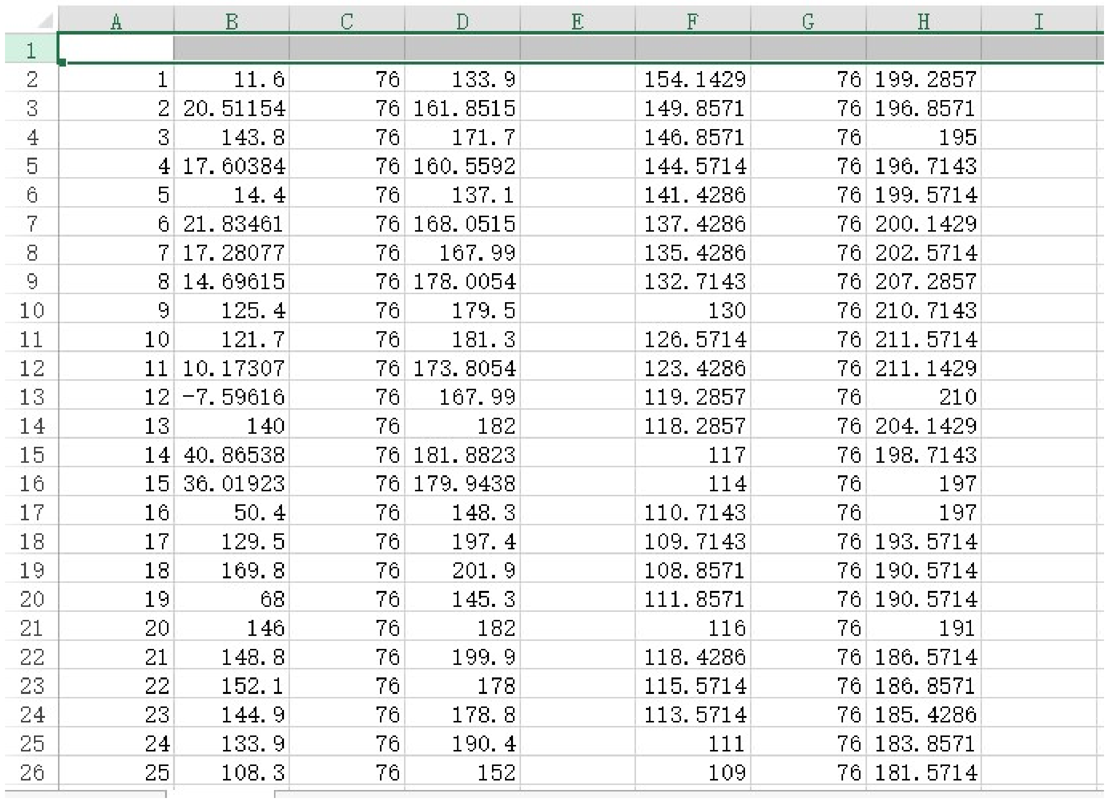
$$\mathrm{pdist} = \sqrt{(\mathrm{x}_1 - \mathrm{x}_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

$$\min Z = \sum_{j=1}^{477}\sum_{i=1}^{477} d_{ij}c_{ij}$$

$$s.t.\begin{cases} \displaystyle\sum_{j=1}^{477} c_{ij} \le 1,\ \ i = 1,2,\cdots,477 \\ \displaystyle\sum_{i=1}^{477} c_{ij} = 1,\ \ j = 1,2,\cdots,477 \end{cases}$$

We also apply LINGO to run the binary integer programming, the details of the code are in the attachment, and part of the matching results is as the following in table 5:

Table 5: The matching result of stage Ⅲ flying process



	A	B	C	D	E	F	G	H	I
1									
2	1	11.6	76	133.9		154.1429	76	199.2857	
3	2	20.51154	76	161.8515		149.8571	76	196.8571	
4	3	143.8	76	171.7		146.8571	76	195	
5	4	17.60384	76	160.5592		144.5714	76	196.7143	
6	5	14.4	76	137.1		141.4286	76	199.5714	
7	6	21.83461	76	168.0515		137.4286	76	200.1429	
8	7	17.28077	76	167.99		135.4286	76	202.5714	
9	8	14.69615	76	178.0054		132.7143	76	207.2857	
10	9	125.4	76	179.5		130	76	210.7143	
11	10	121.7	76	181.3		126.5714	76	211.5714	
12	11	10.17307	76	173.8054		123.4286	76	211.1429	
13	12	-7.59616	76	167.99		119.2857	76	210	
14	13	140	76	182		118.2857	76	204.1429	
15	14	40.86538	76	181.8823		117	76	198.7143	
16	15	36.01923	76	179.9438		114	76	197	
17	16	50.4	76	148.3		110.7143	76	197	
18	17	129.5	76	197.4		109.7143	76	193.5714	
19	18	169.8	76	201.9		108.8571	76	190.5714	
20	19	68	76	145.3		111.8571	76	190.5714	
21	20	146	76	182		116	76	191	
22	21	148.8	76	199.9		118.4286	76	186.5714	
23	22	152.1	76	178		115.5714	76	186.8571	
24	23	144.9	76	178.8		113.5714	76	185.4286	
25	24	133.9	76	190.4		111	76	183.8571	
26	25	108.3	76	152		109	76	181.5714	

Column A, B, C represents the x-coordinate, y-coordinate and z-coordinate of drones in the Ferris wheel respectively. Column H represents the drone's original number. Column M, N, O represents the x-coordinate, y-coordinate and z-coordinate of drones in the dragon respectively. The complete results of the matching in the second process are in the attachment.

Though the binary integer programming we use seems quite effective in search of the solution, there exists a flaw. All the drones have to fly in the same surface; therefore, it's highly possible that the drones may crash into one another. In order to tackle this serious problem, our team then uses the quadratic function to adjust the drones' flying path. In other words, instead of applying the flight path to a simple straight path, we utilize the shape of quadratic function as a drone’s flying pattern. By modeling in this way, we can use the space more efficiently and avoid the collision at the same time.

First, our team utilizes trigonometric function to match the drones on the dragon with drones on the map. We set a reference point O (57, 57, 112.5), the center of the map pattern, to generate the relative vectors of each coordinate. As is known to us all, within the range from 0 degree to 180 degrees, the closer the cosine value of the vector angle is to 1, the nearer the two vectors are. Since the range of the cosine function defined in 0 degree to 180 degrees is from -1 to 1, we can reach the conclusion that the greater the cosine value is, the near they are. Therefore, we calculate the vectors from the reference point to the coordinate of each drone in the two patterns. The transformation technique is shown as below:

$$(x_1, y_1, z_1) \rightarrow [(x_1 - 57), (y_1 - 57), (z_1 - 112.5)]$$
$$(x_2, y_2, z_2) \rightarrow [(x_2 - 57), (y_2 - 57), (z_2 - 112.5)]$$

Then we calculate the cosine value of each vector in the dragon and each vector in the map. Suppose there are two vectors **OA** and **OB**, and their included angle is θ, then we can calculate the value of cosine between these two vectors by the following formula:

$$\cos\theta = \frac{\mathbf{OA} \cdot \mathbf{OB}}{|\mathrm{OA}| \times |\mathrm{OB}|}$$

In our case, the formula accordingly changes to the following:

$$cos([(x_1-57),(y_1-57),(z_1-112.5)],[(x_2-57),(y_2-57),(z_2-112.5)])$$

$$=\frac{[(x_1-57),(y_1-57),(z_1-112.5)]\cdot[(x_2-57),(y_2-57),(z_2-112.5)]}{|[(x_1-57),(y_1-57),(z_1-112.5)]|\times|[(x_2-57),(y_2-57),(z_2-112.5)]|}$$

$$=\frac{(x_1-57)(x_2-57)+(y_1-57)(y_2-57)+(z_1-112.5)(z_2-112.5)}{\sqrt{(x_1-57)^2+(y_1-57)^2+(z_1-112.5)^2}\times\sqrt{(x_2-57)^2+(y_2-57)^2+(z_2-112.5)^2}}$$

After repeating the steps above, we are finally able to get a better matching pattern. However, there are still a lot of collisions after the pattern above is realized. Therefore, we need to further optimize our program. In the next step, we apply the quadratic function to optimize the path of every drones. Assuming the flight path function of each drone is

$$path(i)=ax^2+bx+c \ \ (1\le i\le 477, i\in N^*)$$

The shape and opening size of a quadratic function is determined by the coefficient of x^2, so obviously a is the conclusive element in the function. In order to avoid the crash, we need to make the coefficient *a* in each drone's function as various as possible. Besides, the larger $|a|$ is, the longer the path will be. Thus, the numerical value of a need to be close to zero as possible in order to minimize the total distance that drones will fly in the space.

In order to achieve this, we use a special distribution pattern to determine the value of *a* as the following. We first define the determinant *slope* as

$$slope=[-1.6,-1.5,-1.3,-1,-0.8,-0.6,-0.3,0.3,0.6,0.8,1,1.3,1.5,1.6]$$

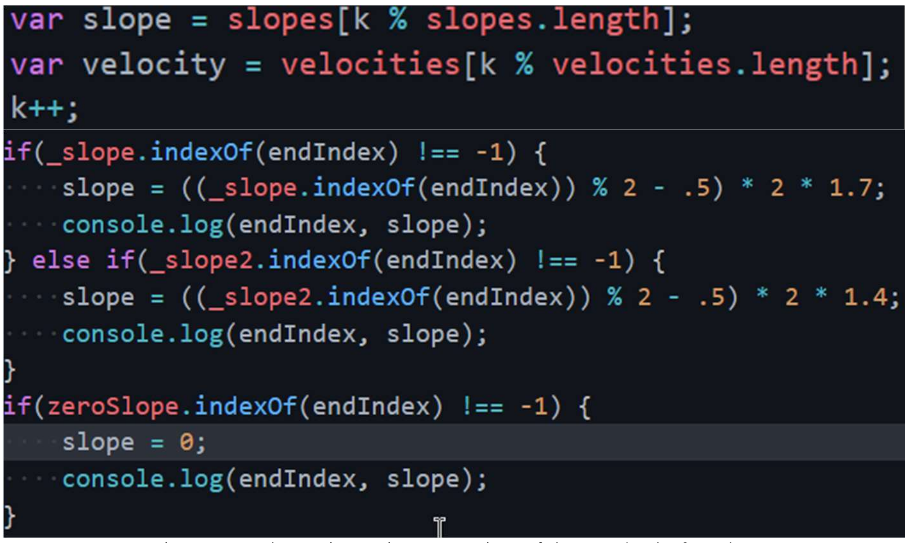
Then we define the determinant *velocity* as

$$velocity = \ 0.7, 1]$$

In the next step, we calculate the value of *a* by multiplying the number in the *slope* with the number in the *velocity* in the following pattern. If the remaining of the original number divided by 14 is *k* ($0\le k\le 13, k\in N^*$), and the remaining of the original divided by 2 is *t* ($0\le t\le 1, t\in N^*$), then we define *a* as the product of the $(k+1)^{\text{th}}$ number in the determinant *slope* and the $(t+1)^{\text{th}}$ number in the determinant *velocity.* For instance, if the original number of the drone is 20, then the value of the coefficient *a* in this drone's path function is

$$a= \ -0.3\times 0.7 \ = -0.21$$

In that case the values of the coefficient a of drones that are numbered close to each other will be different, and therefore we can avoid the potential crash of drones.



```
var slope = slopes[k % slopes.length];
var velocity = velocities[k % velocities.length];
k++;
```

```
if(_slope.indexOf(endIndex) !== -1) {
    slope = ((_slope.indexOf(endIndex)) % 2 - .5) * 2 * 1.7;
    console.log(endIndex, slope);
} else if(_slope2.indexOf(endIndex) !== -1) {
    slope = ((_slope2.indexOf(endIndex)) % 2 - .5) * 2 * 1.4;
    console.log(endIndex, slope);
}
if(zeroSlope.indexOf(endIndex) !== -1) {
    slope = 0;
    console.log(endIndex, slope);
}
```

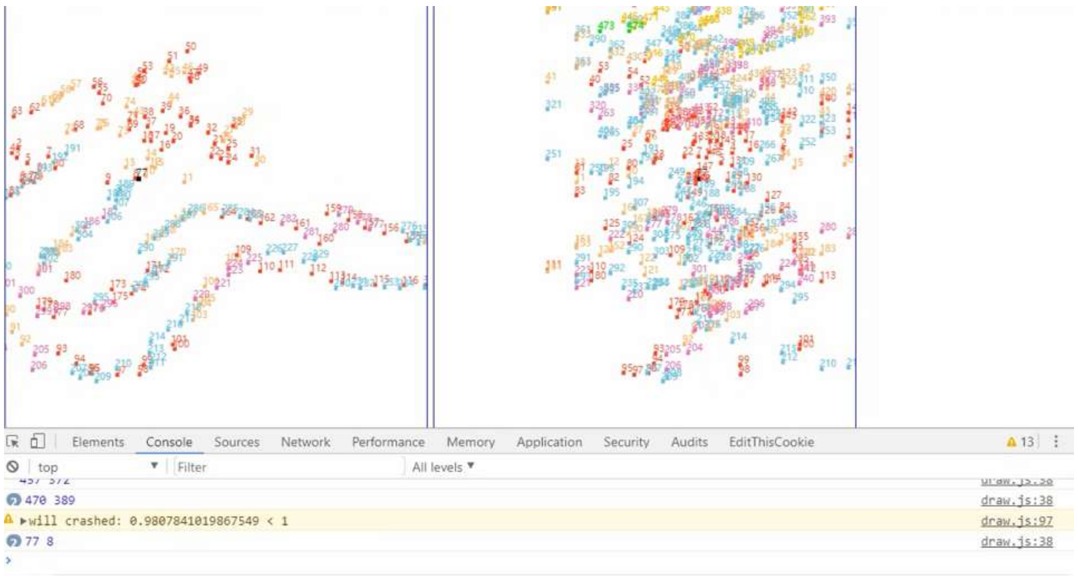
Figure 21: The assistant interpretation of the quadratic function

Besides, we also add the *sleeptime* for every drone in the similar way, the determinant *sleeptime* is as following:

$$sleeptime = [0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8]$$

Similarly, we add the *sleeptime* according to the remaining after the original drone's number is divided by ten. The unit of the *sleeptime* is second. The *sleeptime* is also applied to avoid the potential crash of the drones by staggering the take-off time.

However, this algorithm can't guarantee that none of the drones would crash into each other, so the adjustment of the path and the time are also needed. The figure xx below shows the crash of drones numbered 470 and 389 and the crash of drones numbered 77 and 8. The total number of crashes up to now is 13.



```
457 372                                                   draw.js:38
470 389                                                   draw.js:38
▸will crashed: 0.9807841019867549 < 1                     draw.js:97
77 8                                                      draw.js:38
>
```

Figure 22: the testing report of crashed drones

Therefore, our team's main task now is how to adjust the drones' path and time properly. First, we adjust the drone's starting time by postponing the time that the drones take off or by making the drone take off earlier. The specific revised time lag is listed in the following table

Table 6: the adjustment of the take-off time for crashed drones in process III

Group	Drone number	Revised time lag
1	160,443,101,467,458,253,288	1s later
2	441, 472,129, 431	1.5s later
3	362,440,466,357,457,53,410, 324, 337, 355, 413, 369	1s earlier
4	30, 31, 413, 454, 453, 230, 470, 469, 256, 450, 356, 117, 358, 432, 423	1.5s later

In addition, if time is not proper to adjust, we can also choose to adjust the pattern of path by changing the value of coefficient *a* in the flying path function. The specific changes that are made to the coefficient *a* are presented in the following table.

Table7: the adjustment of the value of coefficient *a* for crashed drones in process III

Group	Drone number	the value of ***a*** after revision
1	(192, 230) (457, 53) (410, 356) (413, 440) (470, 357) (472, 447) (238, 458) (456, 70) (160, 252) (466, 200) (195, 469) (6, 228) (174, 443) (283, 151) (217, 26) (1, 432) (45, 339) (360, 359)	± 1.7

2	(61, 55) (337, 311) (276, 318) (252, 245) (415, 410) (463, 195) (454, 11) (453, 362) (348, 425) (424, 75) (74, 122) (124, 354) (353, 469) 467	± 1.5
3	339, 423	0

From the table, we can conclude that the coefficient a in the two drones that are likely to collide with each other is changed into opposite values so that they would fly towards completely opposite directions. Especially, drone number 339 and 423 are revised to fly in straight path to avoid collision. The complete results of the matching in the third process by using quadratic functions are in the attachment.

To put in a nut shell, both of the methods have strength and weakness. The binary integer programming focuses on the optimization of total distance of drones; However, according to the computer simulation ran by the detect program, an astonishing number of drones would crash with each other. As a result, there is unfortunately no guarantee of the safety consideration. In sharp contrast, the quadratic function method prioritizes the safety concerns and ensures that the solution does not result in crashes no matter how far the total distance is. When combining the two methods together with the utilization of weighting and multi-target optimization. Under the prerequisite that safety always comes first, we determine to value more of the quadratic function. The only flaw lies upon this method is probably the waste of show time. Yet time is never a crucial variable as time is of great abundance for the aerial light show when compared to the maximum flight time of drones.

As is mentioned at the beginning of this section, in order to reconcile the dilemma between the shortest path and safety concerns, we apply a multi-target programming to devise the correspondence of each drone from the dragon to the map of China. We introduce the cosine value of the angle of two vectors to measure and quantify the flight paths. Then we calculate the Euclidean distance between each two coordinates as a preparation to the multi-target programming.

We number the drones in the Ferris wheel from 1 to 477. Here we define d_{ij} as the distance between the i^{th} drone in the dragon and i^{th} drone in the map, as well as e_{ij} as the cosine vector angle between the i^{th} drone in the dragon and j^{th} drone in the map. Then we define c_{ij} to judge the following situation: if the i^{th} drone in the dragon matches the j^{th} drone in the map, then we define $c_{ij} = 1$. Otherwise, we define $c_{ij} = 0$.

Considering the first goal, we define function Z, the minimum total distance, as one of our target function. We define function W, the maximum total cosine vector angle, as another target function.

$$min\, Z = \sum_{j=1}^{477}\sum_{i=1}^{477} d_{ij}c_{ij}$$
$$max\, W = \sum_{j=1}^{477}\sum_{i=1}^{477} e_{ij}c_{ij}$$

In a similar way, the functions are subject to

$$\sum_{j=1}^{477} c_{ij} \le 1,\ \ i = 1,2,\cdots,477$$
$$\sum_{i=1}^{477} c_{ij} = 1,\ \ j = 1,2,\cdots,477$$

Combining the two functions into one function, we define λ1 as the weight of the first function and similarly λ2 as the weight of the second function. Hence, we have

$$min\, Y = \lambda_1 Z - \lambda_2 W$$

$$s.t.\ \lambda_1 + \lambda_2 = 1, 0 \le \lambda_1, \lambda_2 \le 1$$

Since the two conditions are equally important to the final consequence, we attach the two functions to equal weight, which means $\lambda_1 = \lambda_2 = 0.5$.

We also apply LINGO to run the multi-purpose binary integer programming; the code can be found in the **Appendix** part. Unfortunately, due to the limited computing resources, we fail to collect the final data of the program. The general locus function of the drones in stage III flying process is shown below:

$$\begin{cases} x_i(t) = \begin{cases} x_{0i}\ (t < s_i) \\ \dfrac{x_{1i} - x_{0i}}{d_i} \times (t - s_i) + x_{0i}\ (s_i \le t < l_i) \end{cases} \\ y_i(t) = \begin{cases} y_{0i}(t < s_i) \\ \dfrac{y_{1i} - y_{0i}}{d_i} \times (t - s_i) + y_{0i} + g_i(t - s_i)(s_i \le t < l_i) \end{cases} \\ z_i(t) = \begin{cases} z_{0i}\ (t < s_i) \\ \dfrac{z_{1i} - z_{0i}}{d_i} \times (t - s_i) + z_{0i}\ (s_i \le t < l_i) \end{cases} \end{cases}$$

Where

$$g_i(t) = a_i\left[\left(t - \frac{d_i}{2}\right)^2 - \frac{{d_i}^2}{4}\right](0 \le t < d_i)$$

Where

$$a_i = v_i \times A_i$$

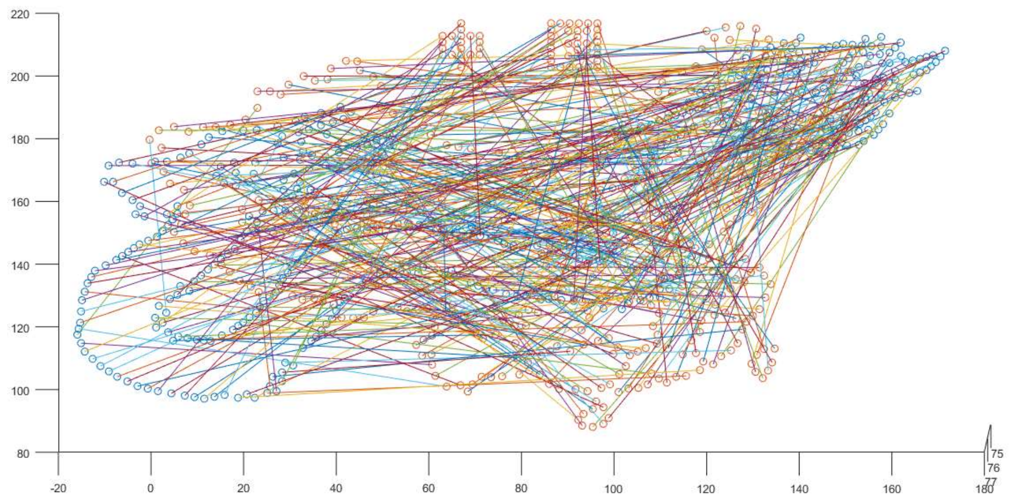


Figure 23: the links from dragon to the map

3.5 Stage IV Flying Process

The forth part of the flying process is the landing process. Similar to the process one, the stage IV also calculates the Euclidean distance between the drones on the map of China and the coordinates of the points on the apron. The only thing that is different from the fist process is that we do not use the clustering analysis.

The general flight path for the drones can be described in a linear function, which is

$$\begin{cases} x_i(t) = \begin{cases} x_{0i}\ (t < s_i) \\ \dfrac{x_{1i} - x_{0i}}{d_i} \times (t - s_i) + x_{0i}\ (s_i \le t < l_i) \end{cases} \\ y_i(t) = \begin{cases} y_{0i}(t < s_i) \\ \dfrac{y_{1i} - y_{0i}}{d_i} \times (t - s_i) + y_{0i}(s_i \le t < l_i) \end{cases} \\ z_i(t) = \begin{cases} z_{0i}\ (t < s_i) \\ \dfrac{z_{1i} - z_{0i}}{d_i} \times (t - s_i) + z_{0i}\ (s_i \le t < l_i) \end{cases} \end{cases}$$

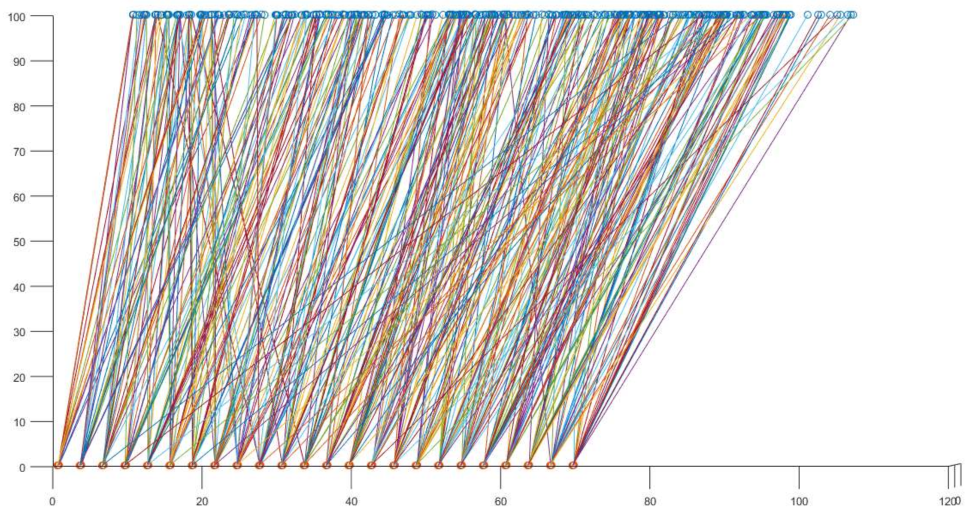


Figure 24: The links of the map of China and the apron

The adjustment of the time is shown in the table below

Table 8: the adjustment of the time in Stage IV Flying Process

Group	Delayed drone number	Revised time lag
1	297, 225, 321, 176, 259, 305, 178	1s later
2	306, 298, 393, 466, 309, 130, 370	1.5s later
3	310, 348, 296, 307, 129, 151, 457, 308, 205, 313, 474, 182, 142, 47, 233	1s earlier
4	145, 53	1.5s later

4. Optimization

4.1 Audience's Best Viewing Angle Optimization

This part of the essay is going to discuss the best viewing distance for audience to enjoy the visual feast.
Assuming the best horizontal distance is *X*. If the highest value of z-coordinate is *A*, the lowest value of z-coordinate is B, then we can calculate the best viewing angle θ by using the following formula

$$\theta = arctan\frac{A}{X} - arctan\frac{B}{X} \qquad X \in (0, +\infty)$$

Then we calculate the derivative of θ to find out its minimum value. We define θ' as the derivative of θ and it can be calculated by the following formula [4]

$$\theta' = -\frac{A}{X^2 + A^2} + \frac{B}{X^2 + B^2} = \frac{-(A - B)(X^2 - AB)}{(X^2 + A^2)(X^2 + B^2)}$$

When $\theta = 0$, θ has its minimum value, and we can calculate the value of *X*,

$$X = \sqrt{AB}$$

According to the z-coordinate of the first design, the Ferris wheel, the highest point of z-axis is approximately 224 meters and the lowest point of z-axis is approximately 116 meters. Therefore, the best viewing distance is

$$X_1 = \sqrt{AB} = \sqrt{224 \times 116} = 161.20m$$

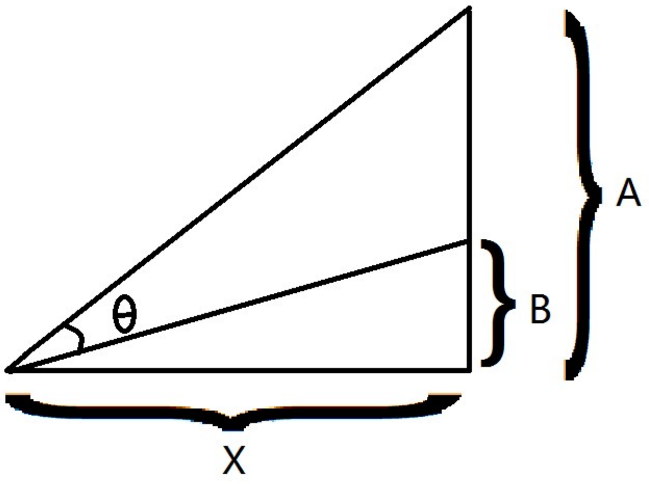


Figure 25: the sketch of the best viewing distance

According to the z-coordinate of the second design, the dragon, the highest point of z-axis is approximately 156 and the lowest point of z-axis is approximately 70. Therefore, the best viewing distance is

$$X_2 = \sqrt{AB} = \sqrt{156 \times 70} = 104.50m$$

According to the z-coordinate of the third design, the map of China, the highest point of z-axis is approximately 159 and the lowest point of z-axis is approximately 63. Therefore, the best viewing distance is

$$X_3 = \sqrt{AB} = \sqrt{159 \times 63} = 104.74m$$

Since audiences' viewing position is fixed, the three best distances listed above should be averaged in order to find the best horizontal distance for audiences to view the three patterns. The average best distance is

$$\mathrm{X} = \frac{X_1 + X_2 + X_3}{3} = 123.48m$$

4.2 The Rotation of the Designs

In this part of the essay we will discuss the functions of the rotation of the Ferris wheel, the dragon and the map of China being mentioned in the former parts.

If the coordinates of a point *A (x, y, z)* in the space is known, then the coordinates of point *A'(x, y, z)* after rotating around the axis *y*=0, *z*=0 for an angle θ can be calculated by the following formula [4]

$$(x', y', z') = (x, y, z) \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

In this way we can successfully calculate the coordinates of the terminal point. According to the formula, we can also calculate the path functions, which the designs rotate around the axis y=76, *z*=150, as the following

$$\begin{cases} x_i(t) = 76 \\ y_i(t) = 76 + \cos(\theta_0 + \theta) \\ z_i(t) = 150 + \sin(\theta_0 + \theta) \qquad (d_i + s_i + l_i \le t < d_i + s_i + l_i + t_0) \end{cases}$$

In the functions above, the θ_0 represents the angle between the vertical axis of the starting point to the axis of rotation and the axis *x*=76, *z*=150. θ represents the angle that the point rotates around the axis. The sketch of the rotation of the designs is presented in figure 26:

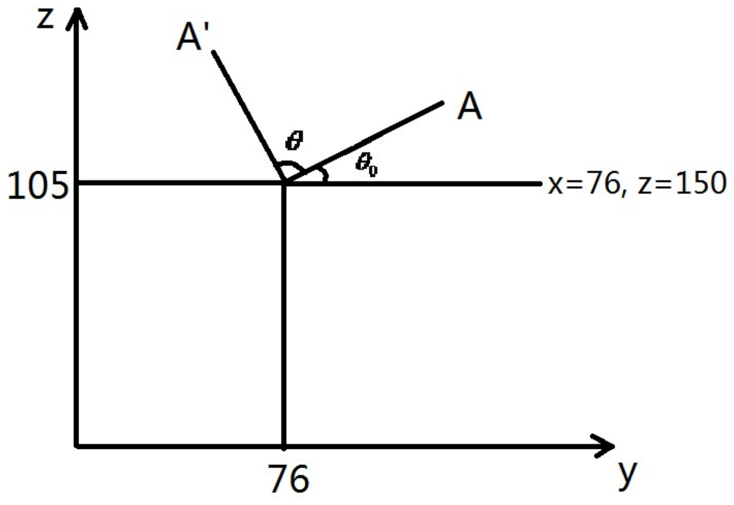


Figure 26: the sketch of rotation of the designs

The function between angle θ and time t is as following

$$\theta_i(t) = \frac{\pi}{8} \times t \times B(\frac{t}{d_i})$$

Where

$$B(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3 \quad t \in [0,1]$$

Where

$$P_0(0,0), P_1(0.42,0), P_2(0.58,1), P_3(1,1)$$

We utilized the Bezier Curve [5] since it could make the animation more vivid, and the speed of the drones is more close to real situation.

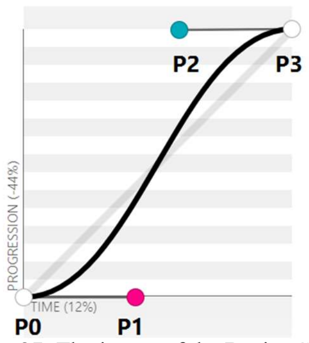


Figure 27: The image of the Bezier Curve

To make our design of Ferris wheel stand out, we make our Ferris wheel rotate around the axis y=76, z=150. This effect can be made by adding the following functions since it can make Ferris wheel more vivid and prevent audience's visual fatigue:

$$Pi(t) = (x_i, y_i, z_i) \begin{bmatrix} cos\theta_{i(t)} & 0 & sin\theta_{i(t)} \\ 0 & 1 & 0 \\ -sin\theta_{i(t)} & 0 & cos\theta_{i(t)} \end{bmatrix}$$

Where

$$\theta_{i(t)} = \theta_{0i} + \frac{\pi}{180} \times t \quad (l \le i \le 438\,)$$

$\theta_{i(t)}$ represents the included angle between the coordinates of drone number i and the axis *y*=76, *z*=150. The values of $\theta_{i(t)}$ are listed in the appendix. Especially, the drones numbered 439 to 477 are on the brackets of the Ferris wheel or are the fixed point, therefore they are not included in formulas.

4.3 The Honeycomb Optimization

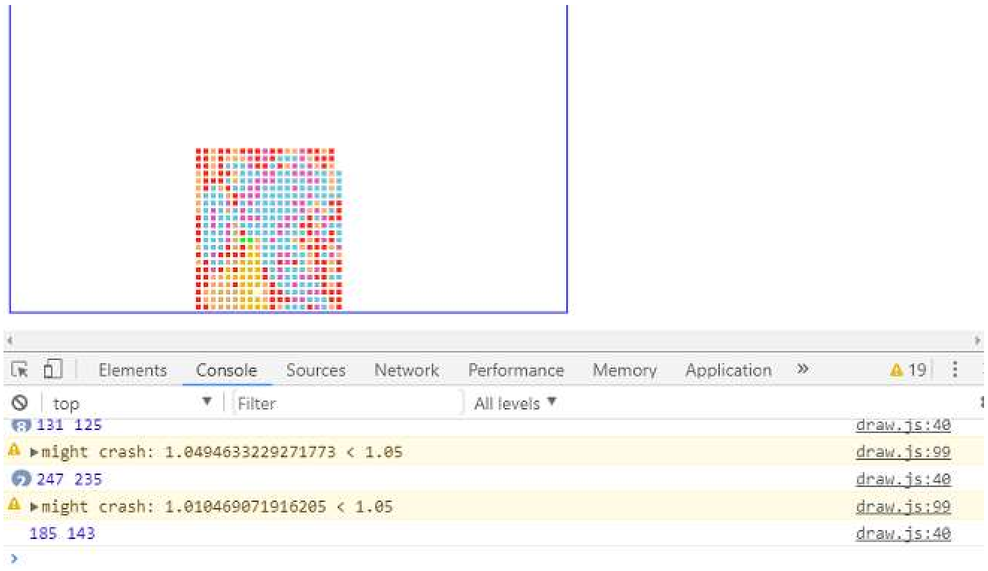
In order to minimize the floor area on the apron, we determine to use the honeycomb shape instead of the common rectangle shape. The honeycomb shape refers to the shape formed with regular hexagons. According to our calculation, the area required when using the common rectangle shape is 3933 square meters. In sharp contrast, the area required when using the honeycomb shape is 3367 square meters. Therefore, we can conclude that the honeycomb shape is a potential optimization of the floor area required. However, when we run the detection function to check if there is crash or not, we find 700 crashes in total. Due to the limited time, we decide to adopt the common rectangle shape so that none of the drones would crash.

5. Conclusion

5.1 Sensitivity Analysis

Sensitivity analysis is a method of studying and analyzing the sensitivity of the model to changes in system parameters or surrounding conditions. In our team's optimization methods, it can detect the stability of our model, especially when the given data is not accurate.

In this part we will mainly discuss the sensitivity of the minimum safety distance. In the **Definition** part, we set the minimum safety distance A to be 1 according to the size of the drones, and we successfully create the designs and flight paths that meet the requirements. We also calculate the number of crashes that will happen if the minimal safety distance A changes, the result is presented in figure 28:



```
131 125                                              draw.js:40
▶might crash: 1.0494633229271773 < 1.05              draw.js:99
247 235                                              draw.js:40
▶might crash: 1.010469071916205 < 1.05               draw.js:99
185 143                                              draw.js:40
```

Figure 28: The sensitivity analysis of the minimum safety distance

If we give *A* an increase of five percent, by changing the value of A on the program, we discover that the number of crashes increase from 0 to 19 during the whole display. The total crush number after the change is small enough for us to make further adjustment. Therefore, it is acceptable in the modeling. This sensitive analysis also indicates that our model has universality and can be applied to more situations. For instance, if the size of the plane is larger, then the minimal safety distance will increase correspondingly, but our model can also be applied since the drones that need further adjustments are under control. Therefore, our model is relatively stable.

5.2 Strength and Weakness

Throughout the entire modeling and optimization procedure, we utilize numerous reasonable and prevailing methods to address the problems, taking into account various factors, including the accessibility, economy, feasibility, security, sustainability, etc., which allows the strength far outweighs the drawback.

First, we set up a three-dimensional coordinate system as well as leading in the matrix to express and explain the issue conveniently and clearly. We introduce Roberts edge detection operator to gain the coordinate of each irregular dot efficiently and improve the exchange quality, which can also be modified in a degree. We take the advantage of clustering to simplify the devising process. We insert considerable number of charts and diagrams which is intuitive and apt to understand and comprehend. We use such a large sum of methods, only to hunt for the most economic way.

Our model effectively achieves all of the goals we set initially. It is fast and can handle large quantities of data, but also have the flexibility we desired. Though we do not test all possibilities, we show that our model optimizes state districts for any of a number of variables. If we had better equipment which can handle larger data, we could have produce the design and show of higher quality with virtually no extra difficulty. As well, our method is robust.

Our computer model agrees with both despite working on different principles, implying it behaves as we want. From Euclidean distance to included angle cosine, from greedy algorism to binary programming, and finally multipurpose programming, every algorithms we bring about are apt to cope with the problem that finding a proper match between drone groups independently. Pursuing an ideal brilliant solution to the issue, we carry out diverse techniques, lest we miss the solution. Moreover, we apply Bézier curve in which fits the speed of drones in lieu of constant speed which is idealistic, going our great length to mimic the real condition.

Among all the advantages, security is the most vital one. We utilize the quadratic function and fit in the flight path of drones in case the drones should crash. In the detection program, we apply the vector to ensure there is no mistaken crash or wrong trail.

As a model from the real world, we take several practical situations into consideration. Promising a magnificent feast for the audience, we take the derivative to calculate the best viewing angle while requiring as many as 477 planes being involved into the performance. It must be admitted that there is unavoidable deviation, we do the sensitivity analyzes to estimate the possible and underlying problems that may exist. Running various programs in distinct platforms, consisting of JavaScript, MATLAB and LINGO, we are able to utilize the utmost platforms in different condition based on need, which was replicable had we met similar problems.

Indeed, there is a fundamental tradeoff here between realism and elegance, and our model arguably veers toward over realism. Possessing such advantages, the flaws do exist, which majorly focus on that our model is not adequate in credibility. For instance, due to the limit of time and equipment, we fail to get the final result of several programs, while we do not examine the effect that the wind speed may have.

As what I have mentioned, our models are viable by various means, as our model is in enormous edibility, robust framework.

5.3 Conclusion

Our team comes up with a strategy on what would be the most efficient way to create a aerial light show outdoor in response to the Mayor's requirement. We utilize the algorithms including Euclidean distance, binary integer programming, quadratic function and Bezier curve to model and optimize the flight path. We divided the whole modeling process into four parts and utilize different algorithms in each part and also provide more than one plan for each in order to optimize the whole model.

In the first taking-off process, we calculate the Euclidean distance and clustering analysis in order find out the minimal value of the total flight paths. Then we change our algorithm into integer programming to find the global optimal solution. We also utilize a lot of functions to improve our model in different aspects. For instance, the Bezier curve is applied to make the animation of flying more vivid, and quadratic function is used to avoid the potential crash for the safety concerns. All these theories and programs lend strength to our model and make it more feasible and suitable.

Our team also does the sensitive analysis which calculates the potential crashes after changing the minimal safety distance, and the result shows a relatively strong stability of our model, which is definitely one of our model's advantages. Besides, our model can also be applied to more complicated situations since all the programs and algorithms are generally applicable and also all of them are very commonly use in nowadays mathematic modeling so they are not difficult to realize.

While our approaches and models were effective and produced results, there also remain several types of model weaknesses including the difficulty of multi-target programming. However, our team believes based on the model we have built, the audiences will definitely enjoy this visual feast.

6. Reference

[1] http://static.nfapp.southcn.com/content/201702/12/c279785.html High technologies in the drone show in Guangzhou Haixinsha
[2] *Sell Linear Operator Theory in Engineering and Science* by Arch W. Naylor, George R. World Book Inc, November 2015
[3] *Spatial Cluster Analysis and Application* by Deng Min, Science Press, October 2011
[4] *Mathematical Analysis* by Zhuo Liqi, Higher Education Press, January 2007
[5] *Mathematical Physics Equations* by Yan Zhenjun, Press of University of Science and Technology of China, January 2015