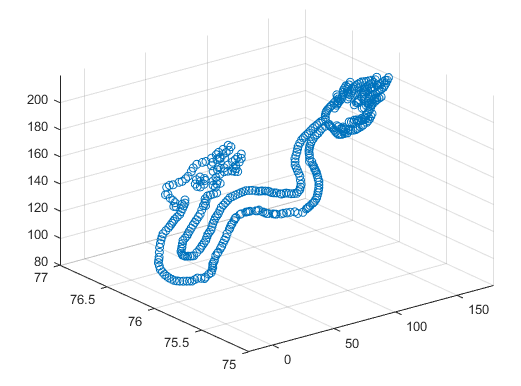
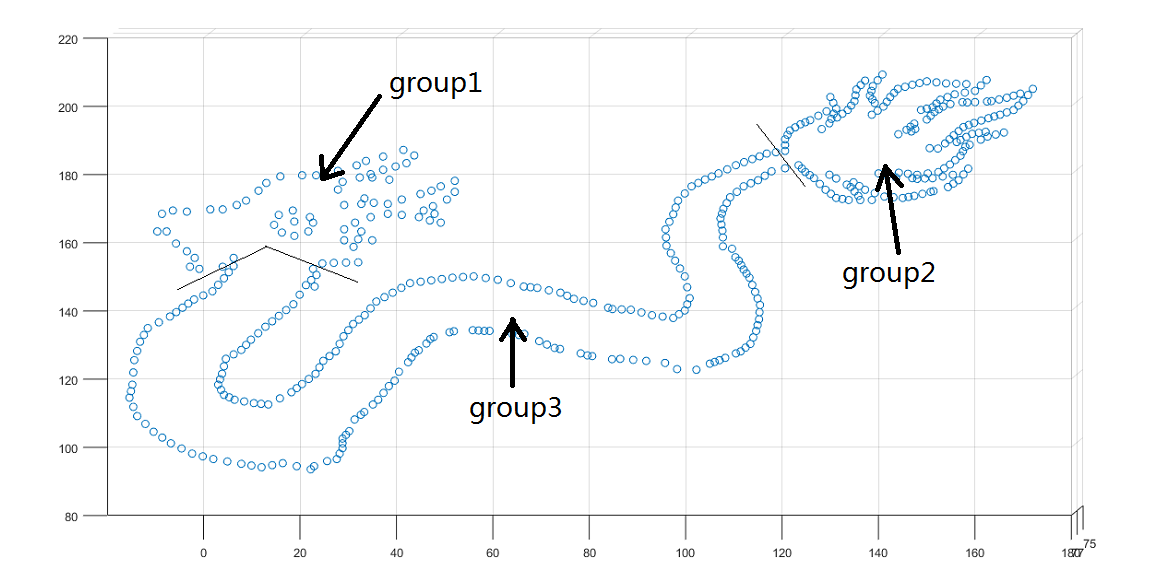
**The stage II flying process**

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 into three groups according to the shape of the dragon and the distribution of the drones. 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 detection function to find if there are drones that may crash into one another and adjust the matching method accordingly.



Our team first flip the Ferris wheel around central axis of y=76, z=150 to the previous position mentioned in the former process in order to lay the foundation for the following practice. The dragon mainly consists of three parts including the head part, the body part and the tail part, and our team divides the dragon into three groups in accordance with this pattern. We here define the head part of the dragon as group1, the tail part as group2 and the main body part as group3. 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

In this process, we do not match the drones based on the least Euclidean distance. Instead, we utilize binary integer programming in order to achieve our goal. The binary integer programming is a kind of special integer programming that decision variable is only 0 or 1. It can be used to find the global optimal solution, in other words, the global 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 use the method of binary integer programming in group1. We number the drones in the Ferris wheel from 1 to 477 and the drones in group1 from 1 to 78. Here we define ***dij*** as the distance between the drone numbered ***i*** in the Ferris wheel and the drone numbered ***j*** in group1. Then we define ***cij*** to judge the following situation, if the drone numbered i in the Ferris wheel match the drone numbered j in group1, then we define ***cij=1***, or we define ***cij=0***.

The total number of the drones in the group1 is 78. Therefore, according to the definition above, we can define the objective function in group 1 as



Every drone in the Ferris wheel can only match up to one drone in group 1, besides, every drone in group 1 must match one drone in the Ferris wheel. Therefore, the constraint conditions are as the following



Next, we utilize LINGO to run the binary integer programming, the details of the code are in the attachment.

After matching the drones in group1, the total number of drones remain is 399. Then we apply the method of binary integer programming in group2. Again we number the drones remains in the Ferris wheel from 1 to 399 and the drones in group2 from 1 to 167. Similarly, we define ***dij*** as the distance between the drone numbered ***i*** remains in the Ferris wheel and the drone numbered ***j*** in group2. Then we define ***cij*** to judge the following situation, if the drone numbered i remains in the Ferris wheel match the drone numbered j in group2, then we define ***cij=1***, or we define ***cij=0***.

The total number of the drones in the group2 is 167. Therefore, according to the definition above, we can define the objective function in group 2 as



Every drone remain 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



Then we again use LINGO to run the binary integer programming, the details of the code are in the attachment.

Then, there are only 232 drones in group3 remain. We respectively number the drones in the Ferris wheel and group3 from 1 to 232. As mentioned above, we similarly defined ***dij*** and ***cij***.

The total number of the drones in the group3 is 232. Therefore, according to the definition above, we can define the objective function in group 3 as

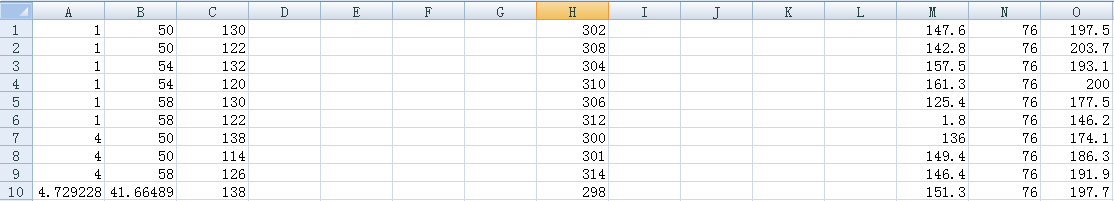


Every drone in group 2 must match one drone remain in the Ferris wheel. Therefore, the constraint conditions is as the following



Then we again use LINGO to run the binary integer programming, the details of the code are in the attachment.

Now every drone in the second process is matched, part of the matching results are as the following,



Column A, B, C respectively represents the x-coordinate, y-coordinate and z-coordinate of drones in the Ferris wheel. Column represents the drone's original number. Column M, N, O respectively represents the x-coordinate, y-coordinate and z-coordinate of drones in the dragon.

The complete results of the matching in the second process are in the attachment.

**The stage III flying process**

The binary integer programming has a defect in this process, that is 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 use the quadratic function to adjust the drones' flying path. In other words, instead of match the drones' coordinate in a line, we utilize the shape of quadratic function as the drones' flying pattern. By modeling in this way, we can use the space more efficiently and avoid the collision at the same time.

We first suppose every drone's flying path function is

The shape and opening size of a quadratic function is determined by the coefficient of , so obviously *a* is of great importance in the function. In order to avoid the crash, we need to make the coefficient *a*in each drone's function as different as possible. Besides, the larger |*a*| is, the longer the path will be, so 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* and the number in the *velocity* in the following pattern. If the remaining of the original number after dividing 14 is *k*(0), and the remaining of the original number after dividing 2 is *t* (0), then we define *a* as the product of the number in the determinant *slope* the 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

-0.3\*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.

However, this algorithm can't ensure that all the drones won't crash, so the adjustment of the path and the time are also needed. The diagram below shows the crash of drones numbered 470 and 389 and the crash of drones numbered 77 and 8. The total number of crushing up to now is 13.



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 postponing time lag is listed in the following table

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

|  |  |  |
| --- | --- | --- |
| Group | Delayed 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 |

Second, if time is not proper to adjust, then we can also choose to adjust the pattern of path by changing the value of coefficient *a* in the flying path function. If the two drones have been detected to have the chance to crash, then we will change the coefficient *a* in these two drone's path functions into opposite numbers in order to avoid the possible collision. The specific changes that are made to the coefficient *a* are presented in the following table.

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

|  |  |  |
| --- | --- | --- |
| Group | Changed 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 |  |
| 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 |  |
| 3 | 339, 423 | 0(changed to a line) |

The complete results of the matching in the third process by using quadratic functions are in the attachment.

**The 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

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

The adjustment of the time is shown in the table below

|  |  |  |
| --- | --- | --- |
| 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 |