For office use only
T1_____
T2_____
T3_____
T4 _____

For office use only
F1_____
F2_____
F3_____
F4 _____

2017

20th Annual High School Mathematical Contest in Modeling (HiMCM) Summary Sheet
(Please make this the first page of your electronic Solution Paper.)

Team Control Number: 7879
Problem Chosen: A

As technology for drones advances, large-scale drone light shows which are capable of forming animated complex displays emerged and have became a popular choice to provide a large crowd with a unique experience for technological innovation. As it is crucial for the mayor to understand specific details about the show before making the decision of whether to include the aerial light show in this year's festival, the purpose behind our mathematical and computational model is to simulate the whole process of the light show, with transition time minimised, safety guaranteed, displays aesthetically presented.

Our models consist of four steps: formation of displays, transition between displays, animation during each display and lastly take-off and landing. To construct the frames, we put the original images into Matlab and trace out the contour of every display. A point selection was performed to locate important points using an algorithms constructed by our own, generating a list of positions that need to be occupied by drones. Then, to minimise the transition time between every display for better light show experience, which is equivalent to minimising the maximum distance travelled by any drone, we employed bipartite matching and binary search. Animation for the three displays is in the form of rotation or shearing, both of which make use of transformation matrix. Lastly, to minimise takeoff and landing time as well as the area required for holding the drones, we decided to place the drone at a position on the ground directly below the position it will occupy in the sky for the first frame, while the landing process applies the same concept.

Based on our model, 480 drones will be employed to put up the performance so that the whole show will be big enough for thousands of people to view. A restricted ground area with a length of 430m and a width of 330m will be cordoned off and viewers are not allowed to enter this area for safety concerns. With the restricted area taken into account, the total air space required will be 17737500m3, while the required launch area will be 2400m2.Given each display lasting 60s, the duration of the whole aerial light show will be 329s.

To: Mayor of R City
From: Team 7879
Subject: Results of investigation by Team 7879
Date: 19 Nov 2017

We are deeply aware that the intention of this investigation is to evaluate the feasibility of adding a large-scale aerial light show to R City's annual festival. We understand that such an investigation will allow you to assess whether it is possible to use a reasonable number of drones to form the three displays you desire as well as to control the duration of the show and the cost within an acceptable range. By using a three-dimensional coordinate, our model will determine the exact position and flight path of each individual drone throughout the entire show, starting from the ground, the transition into the sky to display and animate a ferris wheel, a dragon, our city's mascot -- the double-headed eagle, and finally flying back to the ground. Employing modeling and programming, we have created a model that would determine the flight paths of drones in its transmission from one display to the next one and in the animation of each display.

Based on our model, in order to put up a perfect light show for the audience, the drones will have to fly up to a maximum height of 125 meters, which unfortunately exceeds the height limit for drones established by the Federal Aviation Administration (FAA). However, this requirement can be waived if the FAA deems that the performance can be operated safely. Given Intel's expertise in this field and its records of past successful performances by Intel® Shooting Star™, we believe that obtaining a waiver from FAA should not be too much of a hassle.

For this aerial light show, we will be using Intel® Shooting Star™ Drones which are widely used in past aerial light shows. Although we could find neither rental fees nor the price for this drone model, we discovered that the Intel website has a form to request information about light show performances. As such, we suggest that you obtain information from Intel regarding the fees for the aerial light show. Following that, you can weigh the cost (fees for the service, cost to coordinate with FAA, and publicity fees for promoting this aerial light show) against the benefits (increased reputation for the festival and R City, and more tourist revenue earned from this light show). Finally, you can make the decision on whether to add the aerial light show to the festival. From our perspective, we would highly recommend that you include this aerial light show as the reputation gained for R City and the festival is immeasurable, and the festival can possibly be established as a unique light festival, further attracting more tourists. Besides, the overwhelming response from the public after the show was put up in

Disneyland, Coachella and Singapore's National Day Parade serves as a strong testimony that you should support this aerial light show in our city.

Now, we will tell you the details about this show. 480 drones will be employed to put up the performance so that the whole show will be big enough for thousands of people to view. The displays will be within a cuboid of height 120m, length 120m and width 20m. Based on our extensive research, the safety distance will be 500 feet, i.e. around 155m, which is used in Intel's Walt Disney Light Show. This is to prevent the possible explosion of a damaged or punctured battery in the event of an accident. Hence, a restricted ground area with a length of $120m + 155m + 155m = 430m$ and a width of $20m + 155m + 155m = 330m$ will be reserved where people should not intrude. Thus, with the restricted area taken into account, the total air space required will be $(120m + 155m + 155m) \times (20m + 155m + 155m) \times (120m + 5m) = 17737500m^3$.

After considering the battery life of the drone (20 minutes) and the ample time that the audience will need for taking photos, we have decided that the duration of each display will be 60 seconds.

In our model, the transition time from one display to the next one will be based on the time needed for all drones to reach their designated area, which is the same as the travel time for the drone which travels the maximum distance during the transition. The transition time between displays 1 and 2 and between displays 2 and 3 will be 30s and 59s respectively. Provided that the take-off time and landing time will be 36s and 33s respectively, the duration of the aerial light show will be 329s. In the event that you want to add more displays to present a more fulfilling show for the audience, we highly recommend that the length of the show should be limited to 15 minutes, as our model is flexible enough to accommodate extra transition time for other displays.

We hope that our model will give you a better understanding about the details of the aerial light show and that you will be able to adopt our model accordingly to best suit the festival's needs.

Thank you for giving us the chance to contribute to this year's festival!

HiMCM Written Report

Team 7879

November 20, 2017

Chapter 1

Introduction

1.1 Background

Intel successfully defines a new form of entertainment – aerial light show. The night sky is the enormous stage where hundreds of drones perform perfectly choreographed moves, allowing hundreds and thousands of people to enjoy the breathtaking view at the same time. R City is a hypothetical city with an annual festival which aims to boost the tourist arrivals to the city. This year, the mayor of R City wants to incorporate the aerial light show to further enhance the reputation of this festival, and has thus tasked us to find ways to display a ferris wheel, a dragon, and the citys mascot - a two-headed eagle.

Figure 1.1: Three displays that we will build models upon

1.2 Restatement of Problem

The main challenge lies in determining the flight paths of drones during takeoff, transition from one display to another, animation, and landing. We need to decide the takeoff and landing area, determine the positions of drones in the sky to display the image and animation, and develop a path planning algorithm. We are faced with the problem of limited flight time of the drones. Therefore, we need to minimize the takeoff, transition,and landing time to leave more time for display and animation while ensuring safety and not demanding an excessively large safety clearance area.

Chapter 2

Definitions, Assumptions, Justifications, and Variables

2.1 Definitions

Here we will define the terms that will be used throughout the paper:

Show: This is the proposed outdoor aerial light show to be added to our citys annual festival.

Audience: The audience of the show.

Area of Drones: The smallest sized cuboid that can fit all the drones involved in the show (Figure 2.1)

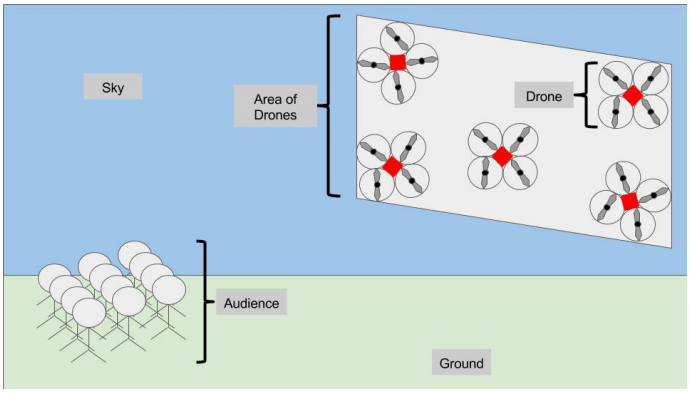


Figure 2.1: An area of drones in the sky with the audience on the ground

Frames: The set of positions of all drones at resting position, 1st display, 2nd display, 3rd display and return position.

2.2 Assumptions

The following assumptions are made:

Assumption 1: The model of the drones used in this show is the Intel® Shooting Star™drone, with the following specifications ("Intel® Shooting Star™Drones Featured in First-Ever Drone Integration during Pepsi Zero Sugar Super Bowl LI Halftime Show", 2017):

1 Each drone has a flight time of up to 20 minutes.
2 Each drone can travel at the speed of $3ms^{-1}$ at all times.
3 Each drone has a size of $384 \times 384 \times 93$ mm

Justification: Intel® Shooting Star™drones have already been used for aerial light shows in the past (as mentioned in the question) and are hence appropriate to be used as drones in the outdoor aerial light show for our citys annual festival.

Assumption 2: The show will be part of an annual festival of a city in the United States of America, and will have to comply with rules related to drone operation stated by the Federal Aviation Administration (FAA) of the United States Department of Transportation (Federal Aviation Administration, 2017):

No.	Rule	Adherence
1	Drone must fly under the height limit of 121.92 m*	Not adhered to (refer to Assumption 4)
2	Drone must fly during the day and not at night*	Not adhered to (The show will be carried out at night.)
3	Drone must fly at or below $160 \cdot 934kmh^{-1}$	Adhered to
4	Drone must not fly over people	Adhered to

Justification: We chose United States of America as the country for our model as they have clear and quantitatively specified rules on drone operation that we can adhere to in our model. Note that items marked with * are negotiable.

Assumption 3: The show is carried out at night (Organizers need to obtain waiver for rule 2, i.e. Drone must fly during the day and not at night), with a clear, dark sky as the background, and there will be negligible light pollution in the vicinity of the show, such that the audience will be able to see a drone if and only if the light of that drone is turned on.

Justification: Given that the annual festival of a large city should be a major event that is carried out well, the show should be carried out under optimal viewing conditions for the audience. Intel has managed to obtain the

waiver for flying drones at night ("Intel Receives Drone Waiver from the FAA", 2016). Since this show is similar in nature to the one organised by Intel, the organizers should also be able to receive a waiver for rule 2 in Assumption 2.

Assumption 4: Minimum distance between drones is 1.5m, i.e. the distance between any two drones should not be within 1.5m at any point of time in air. It will not be a concern when the drones are on the ground.

Justification: A minimum distance between drones has to be set to prevent any collision between drones, which may result in damage to the drones with the drone or any broken parts of the drone becoming a falling-object hazard to the audience on the ground. In a previous aerial light show, five hundred Intel® Shooting Star™drones were spaced at least 1.5m apart from each other (aeronewstv.com, 2016) (Kaplan, 2016). We will adhere to this minimum distance as it has been tried and shown to be safe. It is reasonable to assume this distance need not be maintained by drones still on the ground as they are not interfering with moving drones.

Assumption 5: The drones will remain in the sky throughout the show. In other words, they will not return to the ground during the transition between consecutive displays. They will only do so after they finish the third and final display - the double-headed eagle.

Justification: For simplicity of the model and to minimise transition time between consecutive displays, the drones shall remain in the sky throughout the show. Instead, the light on each drone can be turned on or off (refer to Assumption 3) to give the audience the illusion that the number of drones in the sky is changing during the show. Furthermore, in Coachella Drone Light Show Delights Concert (KG, 2017), an aerial light show held in the past, the drones stayed in the air throughout the show, showing that it is indeed feasible to have such an arrangement.

Assumption 6: All drones will always fly in a straight line, except when two drones that are approaching each other are less than 2 metres apart, those two drones will move around each other in a circular path.

Justification: It is reported that this drone is pre-programmed to avoid collisions. (Glaser, 2016) For simplicity, we assume they can safely avoid any collision.

Assumption 7: All drones function normally and are equipped with motion sensors that can sense presence of birds in the sky and fly around any bird in its path via a circular path.

Justification: For simplicitys sake, we assume so.

Assumption 8: The safe distance between audience and drones will be 155m.

Justification: This safe distance is used in Walt Disney Light shows. ("4 Things You Need To Know Before Using Drones At Your Event", 2017)

2.3 Variables

We have identified the following variables:

I: A matrix representing the input image, whose entries are real numbers ranging from 0 to 255 representing the grayscale intensity of the pixel

N: number of drones used for the show

P_i: The list of N positions that drones should take to display an image in the i^{th} show

A_i: A matrix representing the transformation to be applied to the image in the i^{th} animation stage

v: The linear velocity of each individual drone

t_{Tot} : The total time of the show

$t_{Transition}$: transition time between the frames

t_{TL}: take off time and landing

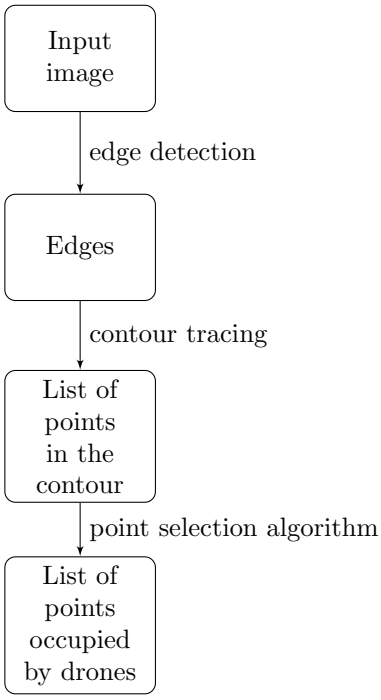
t_{show}: Duration of each animation sequence.

A: clearance area required

Chapter 3

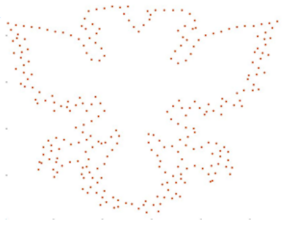
Model Description

3.1 Formation of the Frames



(a) Flow chart of the image processing algorithm

(b) Original image with cntour traced out in green



(c) Points selected in the contour by the algorithm

Figure 3.1: The image processing algorithm

Our frame of display consists of an array of points occupied by drones. To

generate the frame, we require an input of a binary image. Edge detection and contour tracing will be performed on the image to obtain a list of points on the contour (Fig 3.1b). The number of points in the contour is very large, and it is impractical to have drones occupy all the raw contour points. Therefore, we perform a point selection in the contour point list to narrow the number of points that need to be occupied by the drones to a few hundred (Fig 3.1c).

While edge detection and contour tracing is performed using library in Matlab, we developed our own point selection algorithm. For the algorithm, we select the first point on the contour, and continuous search for the next selected point until we finish scanning through the entire list of contour points. Suppose that the last selected point is represented by the red dot, and subsequent contour points for selection are represented by yellow dots. We calculate the distance between the last selected point to subsequent points for selection. An array `d[i]`, $i = 1, 2, \ldots$ representing the distance can be calculated as we move along the list. The next point `k` is selected when `d[k]` is greater than a threshold distance value and the selection process repeats, taking point `k` as the last selected point. This ensures relatively even spacing of the selected points along the contour.

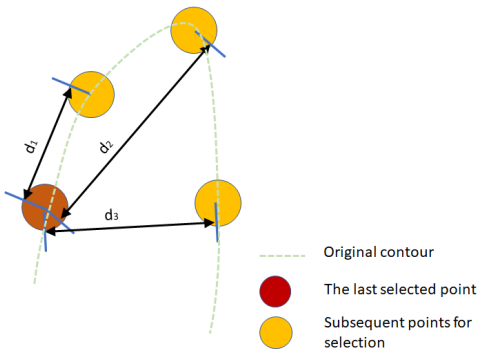
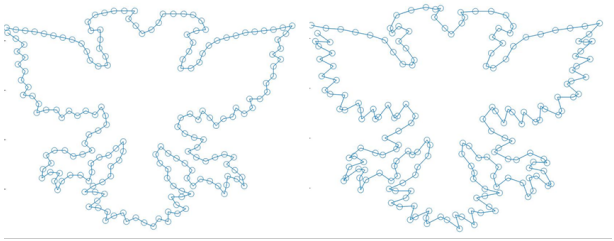


Figure 3.2: Illustration of the point selection process

However, this may not be sufficient. When the number of points is relatively low, features of the shape, especially those at sharp corners will be lost. To enhance such features, we modify our point selection to detect sharp corners for corner enhancement. Specifically, we compare `d[i]` with `d[i-1]`. If `d[i]` $<$ `d[i-1]` or as shown in Fig 3.2 where `d[3]` $<$ `d[2]`, there is a high possibility that a sharp corner is present. The algorithm will then select `d[i-1]` into the selected point list so that the feature of the sharp corner can be enhanced. The effect can be seen in Fig 3.3 where the point set with corner enhancement (Fig 3.3b) preserves much more details in the feature of the wing than the point set without corner enhancement (Fig 3.3a).

(a) Original image



(b) An image with the normal algorithm (left) and one with corner enhancement (right)

Figure 3.3: Comparison of different point selection algorithms

3.2 Transition between the displays

For transition between the different displays, we aim to minimise the maximum distance travelled by any individual drone. The transition time is limited by the time taken by the drone to cover the longest distance. Therefore, by minimizing the maximum distance travelled by any drone, the transition time can be minimized. Faster transition reduces waiting time and saves more time for more elaborated animation as the total amount of time drones can fly is limited. This can help to make the show more aesthetically pleasing.

To reformulate the problem, suppose that drones currently occupy positions marked out by the blue dots in Fig 3.4(top right), and the next display requires drones to occupy positions marked out by red dots. We therefore need to decide which drone needs to move to which final position. Obviously, there are many possible ways, as illustrated in in Fig 3.4 (bottom left and bottom right), and the bottom right represents a better matching with the maximum distance travelled by any drones shorter. More formally, the problem can be stated as such:

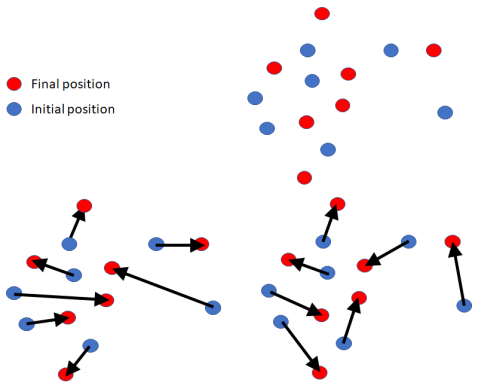


Figure 3.4: An illustration of the transition progress

Given two sets of points in the three dimensional space, P_1 and P_2, find a one-to-one relationship R from P_1 to P_2 such that the value $\max\{d(p_1, p_2) \mid (p1, p2) \in R\}$is minimised, where $d(p_i, p_j)$ is the Cartesian distance between p_i and p_j.

We can model this problem with a bipartite graph. The bipartite graph G can be generated with the nodes represented by the set $(P_1 \times \{0\}) \cup (P_2 \times \{1\})$, with the set of edges described as the set $\{((p_1, 0), (p_2, 1)) \mid p_1 \in P_1, p_2 \in P_2\}$, with the weight function defined as the cartesian distance between the two points. The problem then reduces to finding the perfect matching (a matching where every node is used, Fig 3.5b) such that the maximum weight of edge used is minimal.

The problem now is then to come up with an algorithm to compute the matching. The matching concerned here is one type of bipartite matching. A bipartite matching without any restriction on a graph can be easily generated using the Ford Fulkerson Algorithm.

To minimize the maximum distance travelled by any drone, we use a binary search method. We start with a lower bound of 0 and upper bound of the maximum distance between any two points. We take the middle point as a guessed value and delete from the complete bipartite graph those edges connecting the initial and final positions of drones whose distance is longer than the guessed value. This ensures that in the generated matching, the maximum distance is less than the guessed value. We then perform bipartite matching on the graph. If a perfect matching is possible, it means that a lower maximum distance than the guessed value is possible (Fig 3.5c). We thus update the upper bound to the guessed value. If a perfect matching cannot be found (Fig 3.5d), it means that the lowest possible maximum distance is higher than the guessed value. We thus update the lower bound to the guessed value. We repeat the cycle with updated lower and upper bounds until desired precision is achieved. More

formally, the algorithm is outlined in the following pseudocode.

Algorithm 1 Approximation to the MiniMax Maximal Cardinality Bipartite Matching

Require:
G is a bipartite graph with nodes N and edges E, and n is the number of nodes on each side

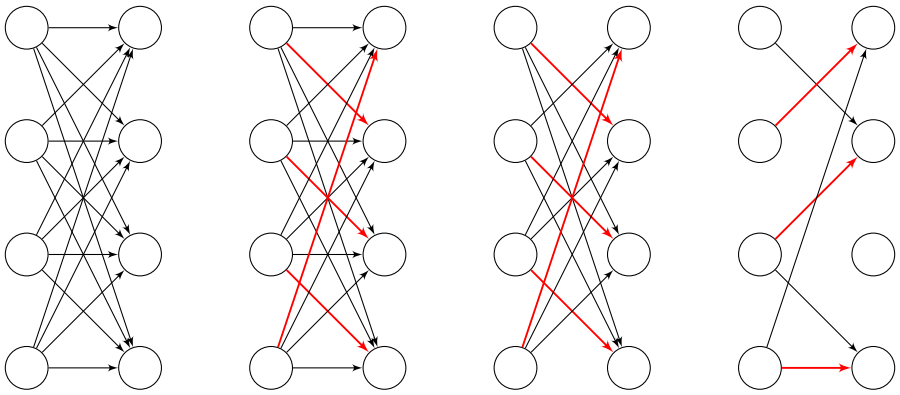
```
procedure MiniMaxMatching(G(N, E), n)
    mx ← {w(e) | e ∈ E}
    lowerbound ← 0                          ▷ lower bound to the binary search
    upperbound ← ⌊mx + 1⌋                   ▷ upper bound to the binary search
    while upperbound − lowerbound > 1 do    ▷ binary search the MiniMax
        G′ ← G
        guess ← ⌊(lowerbound + upperbound)/2⌋
        for e ∈ E do                        ▷ delete edges whose weight exceeds the guess
            if w(e) > guess then
                remove e from G′
            end if
        end for
        r ← maxmatching(G′)
        if r ≥ n then                       ▷ reduce search space to the lower half
            upperbound ← guess
        else                                ▷ reduce search space to the upper half
            lowerbound ← guess
        end if
    end while
    return Matchings(G)                     ▷ return the approximate optimal matching
end procedure
```

After the matching is generated, we can simulate the position of each drone at any given time t during the transition using the following function:

$$f : P \times \mathbb{R} \to \mathbb{R}^3$$

$$f(\boldsymbol{x}, t) = \begin{cases} \frac{matched(\boldsymbol{x}) - \boldsymbol{x}}{\|matched(\boldsymbol{x}) - \boldsymbol{x}\|} tv & \text{if } tv < \|matched(\boldsymbol{x}) - \boldsymbol{x}\| \\ matched(\boldsymbol{x}) & \text{otherwise} \end{cases}$$

, where $\boldsymbol{x}$ is a point from the original frame, $matched(\boldsymbol{x})$ is its matched position in the next frame, t is the time elapsed since the start of the transition.



(a) A complete bipartite graph

(b) A perfect matching on the bipartite graph

(c) Some edges are deleted, however, there is still a perfect matching

(d) Some edges are deleted. A perfect matching is no longer possible

Figure 3.5: Examples of matching. Edges in the matching are highlighted red

3.3 Animation of Each Display

3.3.1 General Description of the Method

We treat the animation of the image as essentially an image transformation. The image may be rotated, stretched, or transformed with a combination of both. Image transformation can be easily described using transformation matrix. Note that the transformation matrix also describes how the position of the drones transform as drone positions are originally points on the image. Therefore, to animate the image, we only need to move the drones to a new position obtained by applying transformation matrix to its original position. In cases where different portions of the image undergo different transformation, we only need to check which region the original position of the drone falls into and then apply the corresponding transformation matrix to obtain the new position. As such, we can represent an animation sequence with a collection of ordered pairs (A, P), where A is a transformation and P is the set of points it is to be applied to. If no animation is specified for a point, it will remain stationary in the air.

3.3.2 Animation by Rotation: Ferris Wheel

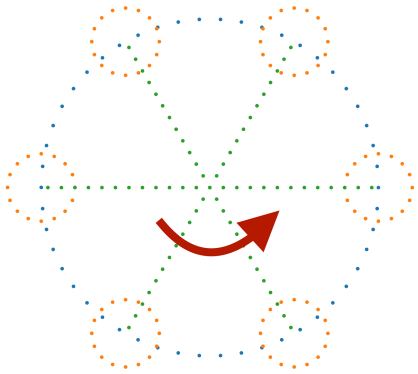


Figure 3.6: The Ferris wheel and its direction of rotation

We start by calculating the maximum number of rotations the Ferris wheel can have in the span of t_{show} seconds. For maximum aesthetic effect, the Ferris wheel should end up with a configuration similar to that in the start, that is, it can only rotate through $\frac{k\pi}{3}$ radian, $k \in \mathbb{Z}$. In our construction of the Ferris wheel, the outermost drone is at a distance of 60 meters from the center of the formation. As such, the amount of time required for it to rotate through $\frac{\pi}{3}$ radian is $\frac{60\pi}{3}\frac{1}{v} = \frac{20\pi}{v}$ seconds. The maximum number of 60 degree turns the Ferris wheel can perform is thus given by $\lfloor \frac{vt_{show}}{20\pi} \rfloor$. For ease of notation, we denote this quantity s.

The transformation in this case is simply an rotation about the z-axis by θ degrees. Hence, it can be written as the following matrix:

$$A_1 = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

with the parameter θ ranging from 0 to $s\frac{\pi}{3}$. To make the rotation uniform, θ should vary linearly with time. This gives $\theta = \frac{vt}{20\pi s} \times \frac{\pi s}{3} = \frac{vt}{60}$ when $t \leq \frac{20\pi s}{v}$ and $\theta = s\frac{\pi}{3}$ otherwise. Since the display remains stationary when $t > \frac{20\pi s}{v}$, we shall only describe the animation when $t \leq \frac{20\pi s}{v}$. If this is the case, by substituting relevant quantities into equation 3.1, the transformation matrix can be written as:

$$A_1 = \begin{pmatrix} \cos\left(\frac{vt}{60}\right) & \sin\left(\frac{vt}{60}\right) & 0 \\ -\sin\left(\frac{vt}{60}\right) & \cos\left(\frac{vt}{60}\right) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

Thus, the animation of the first display can be described by our criterion using the ordered pair (A_1, P_1).

3.3.3 Animation by Shearing: Dragon

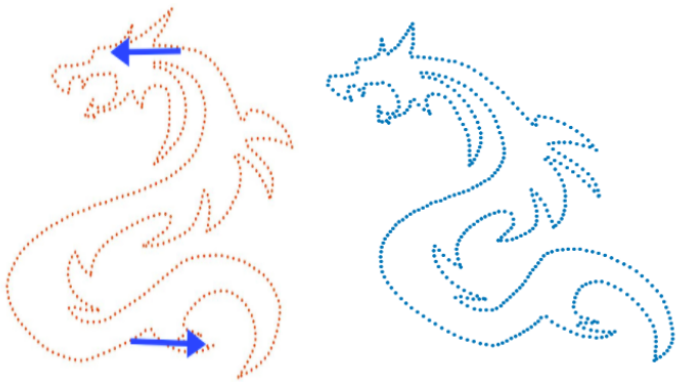


Figure 3.7: The dragon and its direction of sheering

As with the Ferris Wheel, we start by calculating the maximum shearing factor of the transformation. When a shearing of factor γ is applied to the image, the distance travelled by any point (x, y, z) can be given by $|\gamma y|$. This means that the distance travelled by any point is only dependent on the absolute value of its y-coordinate. In the dragon image, this value is exactly 50 meters. Therefore, given any t_{show}, the maximum value of γ is given by $\gamma_{max} = \frac{vt_{show}}{100}$.

The transformation in this case is just the well-known shear by a certain factor γ. It can thus be written as the following matrix:

$$A_2 = \begin{pmatrix} 1 & \gamma & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

with γ ranging from 0 to $\frac{vt_{show}}{100}$ depending on time. To make the animation uniform, γ should vary linearly with time. As such, γ can be written as $\gamma = \gamma_{max} - \left|\frac{2\gamma_{max}}{t_{show}}t - \gamma_{max}\right|$. Substituting this and relevant terms into equation 3.3, we have:

$$A_2 = \begin{pmatrix} 1 & \frac{vt_{show}}{100} - \left|\frac{vt_{show}^2}{50}t - \frac{vt_{show}}{100}\right| & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

Thus, the animation of the second display can be described by our criterion using the ordered pair (A_2, P_2).

3.3.4 Animation by Rotation - Double Headed Eagle

As with the previous 2 sections, we start by calculating the maximum angle θ_{max} that the wings can sweep out within the time t_{show}. In this time, the total distance travelled by any point will be $2r\theta_{max}$, where r is its distance from

the axis of rotation. The maximum distance travelled will thus be $2r_{max}\theta_{max}$, where r_{max} is the maximum distance from the axis of rotation across all points on the wing. Since θ_{max} is dependent on the maximum distance travelled, when v is constant, we have: $\theta_{max} = \frac{vt_{show}}{2r_{max}}$.

The transformation matrix in this case, however, is more complicated. Since the axis of rotation, represented by the 2 lines $l_1 : \boldsymbol{r} = (20, 0, 0)^T + \lambda(0, 1, 0)^T, \lambda \in \mathbb{R}$ and $l_2 : \boldsymbol{r} = (-20, 0, 0)^T + \mu(0, 1, 0)^T, \mu \in \mathbb{R}$ does not pass through the origin, rotation about them cannot be represented using a linear transformation in the 3-dimensional euclidean space. However, we can represent the points in P_3 using their homogeneous coordinates in the 3-dimensional projective space $\mathbb{P}_3(\mathbb{R})$. For the sake of convenience, we shall represent each point $(x, y, z) \in P_3$ with $x : y : z : 1 \in \mathbb{P}_3(\mathbb{R})$. The transformation sequence can then be carried out. We shall illustrate this using the left wing.

The first step is to translate the image such that the axis of rotation passes though the origin. This can be done easily by translating the image in the positive x direction by 20 units. This can be represented by the following transformation matrix:

$$A_{L_1} = \begin{pmatrix} 1 & 0 & 0 & 20 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.5}$$

The next step is to rotate the points anticlockwise by an angle of θ about l_1, which is now the y-axis. This is just the rotation matrix:

$$A_{L_2} = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

The last step is to re-position the image. This simply involves shifting the image in the negative x direction by 20 meters, which can be represented by the matrix:

$$A_{L_3} \begin{pmatrix} 1 & 0 & 0 & -20 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.7}$$

The full transformation sequence, A_L can thus be represented by the composition of the three matrices:

$$A_L = A_{L_3} A_{L_2} A_{L_1} = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 20\cos(\theta) - 20 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & -20\sin(\theta) \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.8}$$

Similarly, the full transformation sequence for the right wing can be represented by a similar matrix, except for this time, the rotation is done anticlock-

wise such that it remains symmetrical with the left wing:

$$A_R = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) & -20\cos(\theta) + 20 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & -20\sin(\theta) \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.9}$$

In both A_L and A_R, the value of θ can range from $-\frac{\theta_{max}}{2}$ to $\frac{\theta_{max}}{2}$. To ensure uniformity of animation, we also want θ to vary linealy with time, such that each point completes a complete oscilation and return to the original point. This implies that θ is a piecewise linear function with time, such that $\theta(t) = 0$ at $t = 0, t_{show}/2, t_{show}$, and $|\theta(t) = \theta_{max}/2|$ at $t = t_{show}/4, 3t_{show}/4$. A suitable function is $\theta(t) = \left|\left|\frac{2\theta_{max}}{t_{show}}t - \frac{1}{2}\theta_{max}\right| - \theta_{max}\right|$. Substituting this and relevant terms into equations 3.8 and 3.9, we are able to obtain the time dependent form of A_L and A_R, which are unfortunately too long to fit into the document. Thus, we can describe the animation on P_3 as the ordered pairs (A_L, P_L) and (A_L, P_L).

3.4 Takeoff and Landing

We aim to minimize takeoff and landing time and reduce the area required for holding the drones. The most efficient method is to place the drone at a position on the ground immediately below the position it will occupy in the sky for the first frame, as illustrated by Fig 3.8. The landing process is the simply the reverse of take-off process, so we will only discuss the takeoff process here.

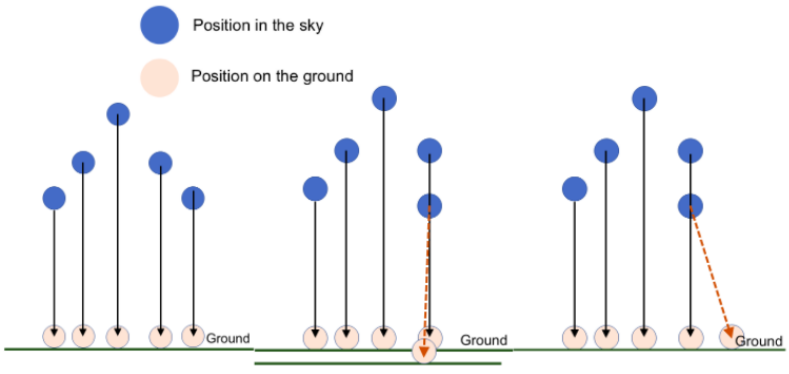


Figure 3.8: An illustration of the takeoff process

3.4.1 Efficiency

This is the most efficient method as the time taken for take-off cannot be shorter than the time taken by the drone that needs to rise to the highest position in the image. By employing this method, this minimum is indeed achieved.

The space required for this method is also minimal as the drones are closely packed about the line immediately below the image.

3.4.2 Prevention of clustering

Multiple positions might need be placed at the same position on the ground during take off. This can be resolved by placing the drones on the ground in multiple rows as illustrated in Fig 3.8 or in available space nearby. Additional length required is likely to be negligible, so this arrangement will not compromise time efficiency.

However, problem may still arise when many drones very close to each other takeoff together from the ground. This problem may be addressed by letting drones takeoff at different times. Drones that need to rise up higher will take off earlier and vice versa. This can effectively stagger the take-off. More mathematically, suppose the it takes time t_i for drone i to rise from ground to its image position in sky if it travels at max speed, and the drone that takes the longest time takes t_{max}. Then under the current scheme, drone i should take off at time $t_{max} - t_i$.

If this is done, it is easily shown that the distance between any two flying drones will not be shorter during the takeoff process than in the stationary image. As it is assumed that no safety distance is needed for drones on the ground, this method can address the problem of clustering almost completely.

Chapter 4

Analysis

4.1 Strengths and Weaknesses of our Model

Strengths:

- Our model prepares the whole model and algorithm flow, which can be easily adopted to other displays.
- The model can adopt other sizes of displays easily. The only consideration will be to maintain the minimum distance more than 1.5m.
- The model can be applied to various other problems as well. The transition algorithm, for instance, can be applied to services such as Uber, where the coordinator has to dispatch drivers to different passengers such that the waiting time of each passenger is minimised. In this case, one set of points can be the location of drivers, with the other being that of the passengers. The edge weight between a driver and passenger can be the estimated amount of time required to for the driver to reach the passenger.
- The transition time from one display to the next display is minimised in our model.

Weaknesses:

- Our model assume that the Intel Shooting StarTM Drones have pre-programmed to avoid collision. However, we cannot be 100
- Our model is for the contour of the input image only, and it needs to be modified if the display is a filled-up image.
- This model is computationally heavy.
- The image that display is based on needs to be binary and its boundary should be clear-cut.

4.2 Further Insight

4.2.1 Effect on take off time

Assuming that the eventual size of the displayed image is constant, reducing the number of drones will not reduce the take off time as it is limited by the height of the display only. As such, changing the number of drones will not offer an advantage in terms of take off time.

4.2.2 Image quality

It is obvious that reducing the number of drones will compromise the image quality, especially in cases where the image contains many sharp corners as those features may be lost even with corner enhancement as illustrated in Fig 4.1.

Figure 4.1: A comparison between image formed by different bumber of drones. It is clear that the one with 160 drones (middle) is of lower quality when compared to the one with 482 drones (left)

As such, a significant number of drones may be needed to achieve an aesthetically pleasing show. However, cost may be a concern when employing a very large number of drones.

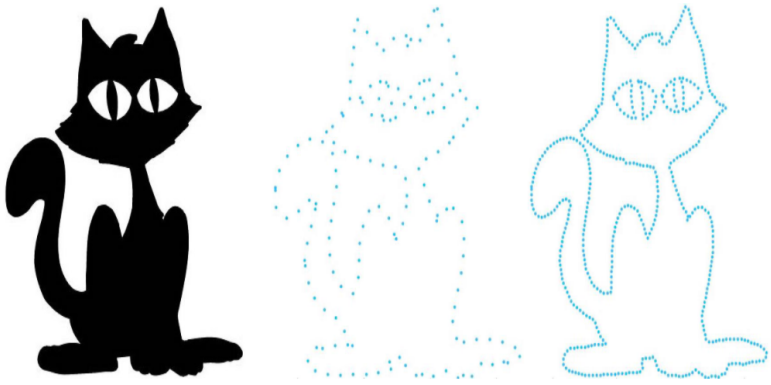


Figure 4.2: A comparison between display formed by different bumber of drones. The one with 160 drones (middle) is of comparable quality to the one with 482 drones (left)

Nonetheless, the problem may be mitigated in cases where the geometry is relatively smooth and has few sharp edges. In this case, the number of drones can be greatly reduced, with mere 160 drones being sufficient to form a reasonably good approximation compared to the display formed by 489 drones in Fig 4.2. Therefore, in cases where the number of drones available is limited, careful choice of image geometry is critical in ensuring a pleasing show.

4.2.3 Transition time

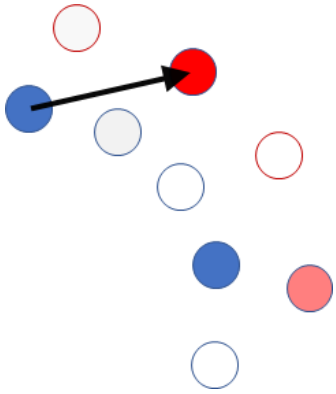


Figure 4.3: Maximum distance travelled by drones increase with a decrease in their numbers

Reducing the number of drones may have an adverse effect on transition time. Assuming that the final image size remains constant, when there are fewer

drones, there will be fewer drones close to the target position It might be necessary for drones further away to travel to the target position, increasing the transition time. This is illustrated in Fig 4.3 Thus reducing the number of drones will adversely affect the transition time.

4.2.4 A brief summary

As seen above, increasing the number of drones can help to improve image quality and reduce transition time. This can definitely improve the quality of the show. However, this does not mean that the number of drones used should be increased excessively, as when the number of drones is excessively high, the limit of control system powered by Intel, the area needed for taken off, and the cost of the display will be a concern. As such, it is important to strike a balance between the controlling the number of drones and improving the quality of the display.

Chapter 5

Results and Conclusion

We applied our frame generation method (with the code attached in B1) as described in Section 3.1 to the form the frames for display. For the first frame of the Ferris wheel, as it contains regular geometric shape only, we did not use the algorithm but directly assigned the points on the circle and lines for simplicity (The code is attached in A3). The resulting frames lit up by the drones are shown in Fig 5.1.The first display(Ferris wheel) and the third display(double-headed eagle) needs 480 drones, while the second display(dragon) employs 478 drones, with two drones resting at the side whose lights are switched off(Assumption 3,5). The approximate number of 480 drones is chosen to strike a compromise between limiting the number of drones and maintaining image quality (The point list generated is attached in A1 and A2 respectively). The actual size of the image are 120m *120m for the three display. We then scaled the coordinates of the points in the list to the actual coordinates according to image size. The image is 5m above the ground to avoid any construction on the ground. This height is not significantly exceeding the limit set by FAA (121.92m) (Assumption 2) and it is deemed to be acceptable as the drones have a professional driving and control system powered by Intel.

Figure 5.1: Frames lit up by drones

We then used the algorithm to compute the path for transition between frame and the time taken. It is found that the transition takes 30s and 50s from the first to second display and from the second to the third display respectively. For takeoff and landing, the time taken is limited by the height of the starting

and ending frame. With the starting frame and ending frame having maximum height of 108m and 100m the total time taken is (108+100)/3.0 = 69s. The space required by takeoff and landing can be easily constrained into a narrow strip immediately below the image as described in 3.4. To provide ample space, we assign a 20m * 120m area below the image. A further 155m boundary is set up as represented in red in Figure (Assumption 9) to prevent any falling broken parts of the drone becoming a falling-object hazard to the audience on the ground. With this we need a total safety clearance area (assumption x), we thus need a 430m * 330m clearance area below the image as illustrated in Fig 5.2.

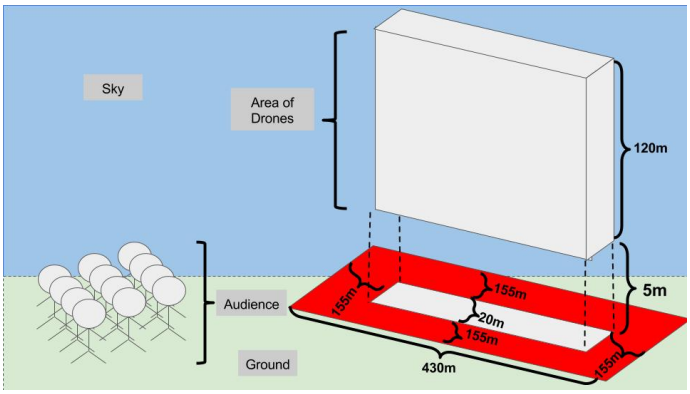


Figure 5.2: Clearance area for the drone display

For the animation, we applied the animation as described in 3.4, with the rotation ferris wheel, the shearing of dragon, and the flipping of wing of the double headed eagle. The time needed for the each animation is 60s (Assumption 8) . With this, the total time of the display from takeoff to landing is 69 + 30 + 50 + 60*3 = 329s, which is safely below the maximum flying time of the drone (1500 s). During the animation, the drones will be contained within the 120m* 120m* 20m cuboid shown in Fig 5.2. As such, the safety clearance distance worked out previously will still be sufficient.

5.1 Concluding Remarks

We successfully constructed a model to describe and optimize the a drone light show. It is shown that the show time is safely within the maximum flight time of the drones and the required safety clearance area is also within a reasonable size. It therefore seems feasible for the light show to be held. However, the model is still far from complete. Additional complication including the reliability of the collision avoidance system of the drones, errors in GPS positioning system in controlling the position of the drones, and even weather condition may impact the feasibility of the drone light show. A much more complete and complicated

model is needed before the feasibility of the light show can be conclusively determined.

Chapter 6

Citation and references:

Aeronewstv.com. (2016, December 11). World Record: Fleet of 500 drones. Retrieved November 19, 2017, from http://www.aeronewstv.com/en/industry/drones/3656-world-record-fleet-of-500-drones.html

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 26.2: The FordFulkerson method". Introduction to Algorithms (Second ed.). MIT Press and McGrawHill. pp. 651664. ISBN 0-262-03293-7.

Federal Aviation Administration. (2017, February 10). Beyond the Basics. Retrieved November 19, 2017, from https://www.faa.gov/uas/beyond_the_basics/#waiver

Federal Aviation Administration. (2017, July 31). Getting Started. Retrieved November 19, 2017, from https://www.faa.gov/uas/getting_started/

Glaser, A. (2016, November 4). Intel invented a way for a single operator to fly hundreds of drones at once. Retrieved November 20, 2017, from https://www.recode.net/2016/11/4/13517550/intel-single-operator-fly-hundreds-drones-shoo

Intel, Newsroom. (2017, February 5). Intel Shooting StarTM Drones Featured in First-Ever Drone Integration during Pepsi Zero Sugar Super Bowl LI Halftime Show [Press release]. Retrieved November 19, 2017, from https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/02/super-bowl-halftime-drone-show-fact-sheet.pdf

Intel, Newsroom. (2016, September 1). Intel Receives Drone Waiver from the FAA [Press release]. Retrieved November 19, 2017, from https://newsroom.intel.com/chip-shots/intel-receives-drone-waiver-from-faa/

Kaplan, K. (2016, November 4). 500 Drones Light Night Sky to Set Record. Retrieved November 19, 2017, from https://iq.intel.com/500-drones-light-show-sets-record/

Knight , R. (2017, June 29). Swarming the Skies. Retrieved November 19, 2017, from http://insideunmannedsystems.com/swarming-the-skies/

KG. (2017, April 17). Retrieved November 19, 2017, from https://www.youtube.com/watch?v=Ra_cdHjFH8Q&t=36s

4 things you need to know before using drones at your event. (2017, February 10). Retrieved November 19, 2017, from

http://fmav.ca/blog/4-things-you-need-to-know-before-using-drones-at-your-event/

Chapter 7

Appendix

7.1 Appendix A

7.1.1 A1 Point lists for dragon

```
{72,883},{75,865},{79,848},{87,832},{94,816},{103,800},{112,785},\\
{124,771},{136,758},{150,746},{164,735},{178,724},{192,713},{207,703},\\
{223,695},{239,686},{255,677},{271,669},{288,663},{305,657},{322,650},\\
{338,642},{355,637},{372,633},{389,626},{406,620},{423,615},{440,610},\\
{457,605},{474,599},{491,592},{508,585},{524,578},{540,570},{556,563},\\
{571,553},{585,543},{599,532},{612,520},{625,508},{636,494},{645,479},\\
{653,463},{658,446},{664,429},{667,411},{671,393},{673,375},{674,357},\\
{672,339},{668,321},{662,304},{655,287},{646,272},{634,258},{620,248},\\
{605,238},{589,230},{572,224},{555,219},{537,216},{519,217},{519,229},\\
{537,231},{554,237},{568,247},{581,260},{593,274},{601,290},{608,307},\\
{612,324},{614,342},{615,360},{613,378},{610,396},{606,413},{599,429},\\
{591,445},{580,459},{568,472},{556,486},{544,499},{530,511},{519,518},\\
{528,502},{535,486},{543,470},{551,454},{558,437},{562,420},{564,402},\\
{565,384},{565,366},{565,348},{565,330},{562,312},{555,295},{539,287},\\
{521,285},{504,292},{491,305},{485,322},{482,340},{481,349},{468,336},\\
{459,320},{454,303},{438,311},{429,326},{416,338},{401,348},{384,355},\\
{367,360},{349,363},{334,373},{337,379},{319,376},{310,373},{317,357},\\
{325,341},{333,327},{331,345},{348,350},{366,349},{382,340},{395,327},\\
{402,310},{406,292},{404,274},{401,256},{384,249},{366,251},{348,252},\\
{330,254},{316,266},{308,282},{302,290},{301,272},{291,258},{273,262},\\
{263,277},{261,295},{258,306},{241,299},{227,287},{218,272},{210,256},\\
{220,242},{232,228},{250,226},{268,226},{286,223},{304,220},{321,214},\\
{331,200},{335,183},{350,173},{368,172},{386,169},{403,164},{418,155},\\
{430,141},{440,126},{449,111},{452,107},{448,125},{442,142},{449,154},\\
{464,144},{480,135},{494,125},{508,113},{522,102},{536,90},{549,78},\\
{562,66},{572,57},{565,74},{558,90},{550,106},{541,122},{531,136},\\
{520,150},{517,165},{535,165},{553,166},{571,168},{589,171},{607,173},\\
{625,177},{643,180},{660,187},{675,197},{689,208},{702,221},{712,235},\\
{722,250},{732,265},{741,281},{748,298},{755,315},{762,332},{768,349},\\
```

```
{774,366},{781,383},{792,386},{799,369},{813,358},{831,361},{849,365},\\
{865,374},{880,383},{894,395},{908,406},{918,420},{927,436},{933,453},\\
{937,468},{921,459},{904,453},{887,448},{869,444},{851,443},{833,443},\\
{815,445},{798,449},{803,466},{816,478},{829,490},{841,503},{853,516},\\
{864,530},{873,546},{878,563},{878,581},{876,599},{872,616},{868,618},\\
{864,601},{857,585},{848,569},{839,553},{829,539},{817,525},{804,512},\\
{790,502},{775,492},{764,498},{774,513},{779,530},{781,548},{780,566},\\
{778,584},{773,601},{766,618},{757,633},{750,650},{740,665},{729,679},\\
{717,693},{703,704},{688,714},{678,719},{682,701},{684,683},{686,665},\\
{687,647},{684,629},{681,611},{670,598},{665,615},{662,633},{658,650},\\
{652,667},{645,683},{637,699},{626,713},{612,724},{597,733},{581,741},\\
{564,748},{547,755},{530,760},{512,762},{494,763},{486,762},{501,753},\\
{516,743},{531,733},{545,723},{558,710},{567,694},{569,676},{553,668},\\
{535,668},{518,673},{501,678},{484,685},{468,693},{452,701},{436,710},\\
{420,719},{405,729},{391,739},{377,751},{364,763},{352,776},{342,791},\\
{335,807},{330,824},{327,842},{326,850},{312,840},{300,826},{291,811},\\
{286,794},{279,782},{269,796},{262,813},{258,831},{257,849},{260,867},\\
{265,884},{273,900},{285,914},{298,926},{313,936},{329,945},{346,952},\\
{349,955},{331,955},{313,954},{295,951},{278,946},{260,949},{270,964},\\
{283,977},{299,986},{317,990},{335,990},{353,989},{370,984},{386,977},\\
{400,966},{415,957},{430,947},{445,937},{460,927},{475,917},{489,907},\\
{503,895},{517,883},{531,871},{545,859},{559,848},{573,836},{587,825},\\
{601,813},{617,805},{634,799},{652,795},{670,793},{688,792},{706,791},\\
{724,792},{742,796},{759,800},{776,805},{793,811},{808,820},{822,830},\\
{837,840},{851,850},{863,864},{874,878},{882,894},{889,910},{896,927},\\
{900,945},{900,963},{898,981},{895,999},{891,1017},{886,1034},{881,1051},\\

{874,1068},{866,1084},{855,1098},{843,1111},{829,1123},{814,1133},{799,1142},\\

{783,1150},{774,1154},{784,1140},{793,1124},{802,1109},{810,1093},{817,1077},\\

{823,1060},{826,1042},{826,1024},{824,1006},{820,988},{815,971},{806,955},{797,939},\\

{785,925},{771,914},{756,904},{739,898},{721,897},{703,900},{686,905},{670,913},\\

{656,924},{643,936},{631,950},{620,964},{611,980},{608,998},{618,1013},\\
{635,1019},{652,1026},{660,1030},{642,1031},{624,1031},{606,1031},{593,1041},\\

{609,1048},{626,1055},{644,1059},{662,1061},{680,1062},{698,1060},{683,1069},\\

{666,1075},{648,1079},{630,1081},{612,1078},{594,1075},{577,1068},{561,1061},\\

{546,1051},{532,1040},{519,1028},{510,1012},{492,1015},{478,1027},{463,1036},\\

{447,1045},{431,1053},{415,1061},{399,1069},{383,1076},{366,1083},{349,1090},\\

{332,1097},{315,1102},{297,1105},{279,1103},{262,1099},{245,1093},{228,1087},\\

{211,1080},{195,1071},{181,1061},{167,1049},{154,1036},{141,1024},{129,1010},\\
```

```
{117,997},{106,983},{97,968},{89,952},{82,935},{77,918},{74,900},{823,0},{756,0}
```

7.1.2 A2 Point lists for double headed eagle

```
{154,247},{158,247},{162,249},{167,250},{173,250},{178,251},{184,251},{189,252},\\
{194,253},{199,254},{203,256},{208,257},{213,258},{218,259},{222,261},{226,263},\\
{229,266},{231,270},{232,275},{234,279},{236,283},{239,286},{244,287},{250,287},\\
{252,283},{250,279},{249,274},{246,271},{243,268},{239,266},{238,263},{241,260},\\
{245,258},{247,254},{245,250},{241,248},{236,249},{233,252},{230,254},{230,250},\\
{229,245},{232,242},{234,238},{236,234},{241,233},{246,232},{251,231},{256,230},\\
{261,229},{266,230},{271,231},{275,229},{276,232},{273,235},{271,239},{274,242},\\
{277,245},{279,249},{282,252},{285,255},{288,254},{291,251},{294,248},{297,245},\\
{300,242},{298,238},{296,234},{301,233},{307,233},{313,233},{319,233},{325,233},\\
{331,233},{336,234},{339,237},{342,240},{344,244},{346,248},{345,253},{342,252},\\
{338,250},{332,250},{328,252},{326,256},{328,260},{332,262},{336,264},{333,267},\\
{330,270},{327,273},{324,276},{322,280},{320,284},{321,289},{326,290},{330,288},\\
{335,287},{338,284},{340,280},{341,275},{343,271},{346,268},{349,265},{352,262},\\
{356,260},{361,259},{365,257},{370,256},{375,255},{379,253},{384,252},{389,251},\\
{395,251},{400,250},{405,249},{411,249},{416,248},{421,247},{425,245},{429,247},\\
{426,250},{423,253},{419,255},{416,258},{412,260},{408,262},{412,264},{417,263},\\
{421,265},{418,268},{415,271},{411,273},{407,275},{408,278},{413,279},{419,279},\\
{416,282},{412,284},{409,287},{405,289},{403,292},{407,294},{411,296},{415,298},\\
{412,301},{407,302},{402,303},{397,304},{399,308},{402,311},{406,313},{409,316},\\
{405,318},{401,318},{396,319},{390,319},{388,322},{389,327},{391,331},{392,336},\\
{388,336},{383,335},{379,333},{375,331},{372,329},{371,334},{371,340},{370,345},\\
{366,343},{363,340},{360,337},{357,334},{356,339},{356,345},{353,343},{350,340},\\
{347,337},{345,333},{341,331},{340,336},{338,340},{337,345},{334,343},{332,339},\\
```

```
{330,335},{328,331},{325,332},{323,336},{320,339},{317,342},{317,346},{320,349},\\
{323,352},{327,354},{330,357},{334,359},{339,360},{345,360},{343,364},{340,367},\\
{336,369},{332,371},{328,373},{331,376},{335,378},{339,380},{343,382},{349,382},\\
{352,379},{356,377},{360,375},{366,375},{370,377},{374,379},{377,382},{377,386},\\
{373,384},{367,384},{363,386},{361,390},{367,390},{372,391},{376,393},{379,396},\\
{381,400},{382,405},{380,408},{378,404},{375,401},{371,399},{367,397},{362,398},\\
{362,402},{365,405},{365,411},{363,415},{361,419},{360,414},{359,409},{356,406},\\
{353,403},{348,402},{344,400},{339,401},{335,403},{333,407},{329,407},{328,402},\\
{328,398},{330,394},{332,390},{329,387},{326,384},{323,381},{320,378},{316,380},\\
{312,378},{309,375},{307,371},{304,368},{300,366},{297,368},{297,374},{298,379},\\
{300,383},{303,386},{306,389},{309,392},{308,397},{309,402},{311,406},{315,408},\\
{319,410},{323,412},{328,413},{332,415},{330,418},{324,418},{318,418},{312,418},\\
{312,422},{315,425},{319,427},{323,429},{327,431},{325,434},{320,435},{314,435},\\
{310,433},{306,435},{308,439},{309,444},{308,449},{305,446},{302,443},{299,440},\\
{295,438},{291,438},{293,442},{294,447},{294,451},{291,448},{287,446},{284,443},\\
{280,441},{277,440},{272,441},{268,443},{265,446},{263,443},{264,438},{266,434},\\
{262,434},{258,436},{254,438},{251,441},{248,439},{250,435},{252,431},{250,428},\\
{245,429},{239,429},{234,428},{230,426},{234,424},{239,423},{243,421},{246,418},\\
{249,415},{245,413},{239,413},{233,413},{228,412},{231,409},{235,407},{240,406},\\
{244,404},{248,402},{251,399},{254,396},{255,391},{254,386},{258,384},{262,382},\\
{265,379},{266,374},{267,369},{266,364},{263,363},{260,366},{257,369},{255,373},\\
{252,376},{249,374},{246,375},{243,378},{239,380},{236,383},{233,386},{235,390},\\
{236,395},{235,400},{232,403},{229,400},{226,397},{222,395},{217,396},{213,398},\\
{209,400},{206,403},{204,407},{203,412},{200,409},{200,403},{201,398},{204,395},\\
{203,392},{199,392},{194,393},{190,395},{188,399},{187,404},{185,400},{186,395},\\
```

```
{188,391},{191,388},{194,385},{199,384},{205,384},{204,381},{201,378},{195,378},\\
{191,380},{191,376},{195,374},{199,372},{203,370},{209,370},{212,373},{215,376},\\
{220,377},{225,376},{229,374},{233,372},{237,370},{235,367},{231,365},{228,362},\\
{224,360},{226,357},{231,356},{235,354},{239,352},{243,350},{246,347},{250,345},\\
{252,341},{252,337},{249,334},{246,331},{244,327},{241,329},{240,334},{237,337},\\
{235,341},{232,338},{231,333},{229,329},{226,330},{223,333},{221,337},{218,340},\\
{215,339},{215,335},{213,332},{210,335},{207,338},{203,340},{201,337},{202,332},\\
{201,327},{197,327},{194,330},{189,331},{185,333},{182,331},{184,327},{185,322},\\
{186,317},{181,316},{175,316},{170,315},{168,312},{172,310},{175,307},{178,304},\\
{175,301},{170,300},{165,299},{162,296},{166,294},{170,292},{174,290},{172,287},\\
{168,285},{164,283},{160,281},{164,279},{169,278},{174,277},{171,274},{167,272},\\
{163,270},{160,267},{158,263},{163,264},{169,264},{170,261},{166,259},{162,257},\\
{159,254},{155,252}
```

7.1.3 A3 Code for Generation of Ferris Wheel

```
from math import sin, cos, sqrt
from numpy import pi

R = 50
r = 10

points = []
param1 = []

for i in range(179):
    param1.append(i * 2 * np.pi / 150)

x = R * np.cos(param1)
y = R * np.sin(param1)

x1 = np.array([])
y1 = np.array([])

for i in range(6):
    for j in range(25):
```

```
        x1 = np.append(x1, R * cos(i * pi / 3) + r * cos(j * 2 * pi / 25))
        y1 = np.append(y1, R * sin(i * pi / 3) + r * sin(j * 2 * pi / 25))

l1x = [i * 2 for i in range(-24, 25)]
l1y = [0 for i in range(49)]
l2x = [i for i in range(-49, 50) if abs(i) <= 25]
l2y = [sqrt(3) * x for x in l2x]
l3y = [-sqrt(3) * x for x in l2x]

xlst = np.concatenate((l1x, l2x, l2x))
ylst = np.concatenate((l1y, l2y, l3y))
```

7.2 Appendix B

7.2.1 B1 Point Selection Algorithm

```
I = imread('Double-Headed Eagle final.jpg');
BW = imbinarize(rgb2gray(I));
B = bwboundaries(BW);
boundary = B{2};
scatter(0,0);

%plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 1) % y coordinate , x coordinate

EC_x = zeros(1,50000);
EC_y = zeros(1,50000);
ECS_x = zeros(1,20000);
ECS_y = zeros(1,20000);
n_points = 0;
n_selected = 0;
for i = 2:2 % skip the first one representing the boundary of the image
    %copy the segment out for easy reference
    boundary = B{i};
    n_points = length(boundary);
    for j = 1:length(boundary)
        EC_x(j) = boundary(j,2);
        EC_y(j) = boundary(j,1);
    end
    n_selected = 1;
    ECS_x(1) = EC_x(1);
    ECS_y(1) = EC_y(1);
    previous_distance = 0;
    j = 1;
    while (j<=n_points)
```

```
        current_distance = D(ECS_x(n_selected),ECS_y(n_selected),EC_x(j),EC_y(j));
        if (current_distance > 15) || (current_distance<previous_distance)  %originally 4
            n_selected = n_selected +1;
            ECS_x(n_selected) = EC_x(j-1);
            ECS_y(n_selected) = EC_y(j-1);
            current_distance = 0;
            j = j + 5;
        end
        previous_distance = current_distance;
        j = j+1;
    end
    hold on
    scatter(ECS_x(1:n_selected)-150, 460-ECS_y(1:n_selected),5,'filled');
    hold off
    %plot(ECS_x(1:n_selected)-150, 460-ECS_y(1:n_selected),'o-');
    n_selected
    fileID = fopen('DHE191.txt','w');
    for j = 1:n_selected
        nbytes = fprintf(fileID,'{%d,%d},',ECS_x(j),ECS_y(j));
    end
    %plot(, 'r', 'LineWidth', 1) % y coordinate , x coordinate
end

for i = 1:length(boundary)
  %  plot(boundary(i,2), boundary(i,1), 'r', 'LineWidth', 1)
end
```

7.2.2 B2 MiniMax Bipartite Matching

```
#include <bits/stdc++.h>
using namespace std;
#define ABS(X) (((X)>0)?(X):((-X)))
#define SQ(X) ((X)*(X))

typedef long double ll;
typedef pair<int, ll> pif;
typedef pair<ll, ll> pll;

ll max_dist_constraint;
long long st, ed;

struct AugPath {
    int A, B;   //size of left, right groups
    vector<vector<pif> > G; //size A
```

```
    vector<bool> visited;   //size A
    vector<int> P;          //size B

    AugPath(int _A, int _B): A(_A), B(_B), G(_A), P(_B, -1) {}

    void AddEdge(int a, int b, ll w) {    //a from left, b from right
        G[a].push_back({b, w});
    }
    bool Aug(int x) {
        if (visited[x]) return 0;
        visited[x] = 1;
        /* Greedy heuristic */
        for (auto it:G[x]) {
            if(it.second > max_dist_constraint) continue;
            if (P[it.first] == -1) {
                P[it.first] = x;
                return 1;
            }
        }
        for (auto it:G[x]) {
            if(it.second > max_dist_constraint) continue;
            if (Aug(P[it.first])) {
                P[it.first] = x;
                return 1;
            }
        }
        return 0;
    }
    int MCBM() {
        int matchings = 0;
        for (int i = 0; i < A; ++i) {
            visited.resize(A, 0);
            matchings += Aug(i);
            visited.clear();
        }
        return matchings;
    }
    vector<pair<int, int> > GetMatchings() {
        vector<pair<int, int> > matchings;
        for (int i = 0; i < B; ++i) {
            if (P[i] != -1) matchings.emplace_back(P[i], i);
        }
        return matchings;
    }
};
```

```
// input
// line 1:      an integer N, describing the number of points in each list
// line 2-2N+1: 2 long doubles on each line, describing the coordinates of each point

// output
// N lines, each line containing a matching

ll cart_dist(tuple<ll, ll, ll> s1, tuple<ll, ll, ll> s2) {
    ll x1, x2, y1, y2, z1, z2;
    tie(x1, y1, z1) = s1;
    tie(x2, y2, z2) = s2;
    // cout << x1 << ' ' << x2 << '\n';
    return sqrt(SQ(x1 - x2) + SQ(y1 - y2) + SQ(z1 - z2));
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int n;
    tuple<ll, ll, ll> l1[505], l2[505];

    cin >> n;
    auto AP = AugPath(n, n);

    for(int i=0; i<n; i++) {
        ll a, b, c;
        cin >> a >> b >> c;
        l1[i] = {a, b, c};
    }

    for(int i=0; i<n; i++) {
        ll a, b, c;
        cin >> a >> b >> c;
        l2[i] = {a, b, c};
    }

    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++) {
            ll cost = cart_dist(l1[i], l2[j]);
            // cout << cost << '\n';
            AP.AddEdge(i, j, cost);
            ed = max((long double)ed, cost);
        }
    }
```

```
// max_dist_constraint = 200;
// cout << AP.MCBM() << '\n';

ed ++;

// cout << ed << '\n';

while(ed - st > 1) {
    long long guess = (st + ed) / 2;
    max_dist_constraint = guess;
    fill(AP.visited.begin(), AP.visited.end(), 0);
    fill(AP.P.begin(), AP.P.end(), -1);
    int ans = AP.MCBM();
    // cout << st << ' ' << ed << ' ' << ans << '\n';
    if(ans >= n) ed = guess;
    else st = guess;
}

ll stf = (long double) st;
ll edf = (long double) ed;

for(ll i=stf; i<=edf; i+=0.1) {
    // searching for the first dp
    max_dist_constraint = stf;
    fill(AP.visited.begin(), AP.visited.end(), 0);
    fill(AP.P.begin(), AP.P.end(), -1);

    int ans = AP.MCBM();

    if(ans == n) {
        stf = i-0.1;
        edf = i;
        break;
    }
}

for(ll i=stf; i<=edf; i+=0.01) {
    // searching for the second dp
    // cout << i << '\n';
    max_dist_constraint = stf;
    fill(AP.visited.begin(), AP.visited.end(), 0);
    fill(AP.P.begin(), AP.P.end(), -1);

    int ans = AP.MCBM();

    if(ans == n) {
```

```
            stf = i-0.01;
            edf = i;
            break;
        }
    }

    cout << edf << '\n';
    max_dist_constraint = edf;
    fill(AP.visited.begin(), AP.visited.end(), 0);
    fill(AP.P.begin(), AP.P.end(), -1);
    AP.MCBM();
    auto match = AP.GetMatchings();

    for(auto p : match) {
        cout << p.first << ' ' << p.second << '\n';
    }
}
```

7.3 Appendix C Code for Animation

```
def linear_animate(init, dest, total_frame, frame_step) -> np.array:
    # returns an iterator that generates the coordinates of all points at this step
    process=subprocess.Popen(['./min_cost_max_flow'],
                         stdin=subprocess.PIPE,
                         stdout=subprocess.PIPE,
                         stderr=subprocess.PIPE)
    stdoutdata,stderrdata=process.communicate(input=get_output(init, dest).encode())
    print(int(stdoutdata.decode('utf-8').split('\n')[0]))
    match = stdoutdata.decode('utf-8').split('\n')[1:]
    match_clean = np.array([[int(i.split(' ')[0]),int(i.split(' ')[1])] for i in match if i

    delta = np.array([np.array(dest[i[1]]) - np.array(init[i[0]]) for i in match_clean])

    normed = [norm(i) for i in delta]
    delta = [i / norm(i) for i in delta]

    print(max(normed))

    yield init # avoid stupid stuff
    yield init
    yield init

    counter = 0
```

```
    for i in np.linspace(0, total_frame, total_frame / frame_step):
        # print(counter)
        counter += 1
        newpoints = np.array([np.array([0,0,0]) for i in match_clean])
        for j, k in zip(match_clean, range(0, len(match_clean))):
            if(i * 3 >= normed[k]):
                newpoints[k] = init[j[0]] + delta[k] * normed[k]
            else:
                newpoints[k] = init[j[0]] + delta[k] * i * 3
        yield np.array(newpoints)

# current animation sequence:
# take-off -> ferris wheel - 40 sec
# ferris wheel -> dragon - 30 sec
# dragon -> double headed eagle - 30sec

fig = plt.figure()
ax = p3.Axes3D(fig)

iter1 = linear_animate(fat_mat, ferris_points, 40, 0.1)
iter2 = linear_animate(ferris_points, dragon_points_3d, 30, 0.1)
iter3 = linear_animate(dragon_points_3d, dhe_3d, 50, 0.1)
iter4 = linear_animate(dhe_3d, fat_mat, 63, 0.1)

def update(frame):
    if(0 <= frame < 40):
        pt_next = next(iter1)
        print(frame)
        line.set_data(pt_next[:, 0], pt_next[:, 2])
        line.set_3d_properties(pt_next[:, 1])
        return [line]
    elif(40 <= frame < 70):
        pt_next = next(iter2)
        line.set_data(pt_next[:, 0], pt_next[:, 2])
        line.set_3d_properties(pt_next[:, 1])
        return [line]
    elif(70 <= frame < 100):
        pt_next = np.array(next(iter3))
        line.set_data(pt_next[:, 0], pt_next[:, 2])
        line.set_3d_properties(pt_next[:, 1])
        return [line]
    else:
        pt_next = np.array(next(iter4))
        line.set_data(pt_next[:, 0], pt_next[:, 2])
        line.set_3d_properties(pt_next[:, 1])
```

```
        return [line]

ax.axis('equal')
ax.set_xlim3d([-60, 60])
ax.set_xlabel('X')

ax.set_ylim3d([-60, 60])
ax.set_ylabel('Y')

ax.set_zlim3d([-60, 60])
ax.set_zlabel('Z')

line, = ax.plot(fat_mat[:, 0], fat_mat[:, 2], fat_mat[:, 1], 'ob', markersize=1)
ani = FuncAnimation(fig, update, np.linspace(0, 183, 1830), interval=10, blit=True).save('3d
```