A singular drone can achieve a successful flight with little to no planning. A few more may require some extra coordination, but nothing that human hands and eyes cannot manage. A few hundred more, however? That scale simultaneously enters the realms of mathematics and performance: take Intel’s 500-drone light display at the 2016 Super Bowl halftime show, for example. Inspired by such a performance, our team has developed a mathematical model to plan a drone show of our own, giving us the capability to display nearly any image we choose using Intel’s Shooting Star™ drones. Tasked by the Mayor, we have utilized our model to plan a drone show of our own for our city’s annual festival, consisting of a sprawling dragon, a spinning Ferris wheel, and our own team number surrounded by a scintillating ring.

When creating our model, we began at the most important element of any performance -- what the audience sees, in this case the aerial image. In order to make the model applicable with any image input, we found Eigenvalues to systematically identify various aspects of nearly any inputted image and to catalog whether or not certain points should be included as drones. As a result, we minimized the number of drones necessary to map each image in the sky.

We then utilized a k-D tree nearest neighbor algorithm to determine the optimal set of starting points for the drones based on the final points of the image. This minimized the distance each drone has to travel from takeoff to the first image and from each image to the subsequent, thus minimizing the each drone’s travel time and allowing the actual images to be displayed for as long as possible. From there, a genetic algorithm optimized the distance from each starting point to each ending point, connecting every drone’s beginning and ending spot, accomplished by minimizing total drone distance traveled with each generation.

Ultimately, once the starting and ending points of each drone in each image were connected, we utilized 3-dimensional vectors to create a crash-prevention algorithm that takes into account velocity and wind to keep each drone on its shortest flight path with no collisions occurring. A similar set of parametrics, based on Newtonian mechanics, allowed us to animate the Ferris wheel and our team number.

We found that our model is quite applicable to almost any image, and thus could easily be utilized to host various drone shows. In the case of our show, our model utilized 560 drones, which may initially seem like a large number; our largest image, however, is 154 m by 91 m, no small space. The show runs for 13 minutes, with a three minute display of each formation, easily inside the Shooting Star’s 20-minute battery life. Of course, since safety is of great importance, we assessed the amount of space necessary for the show: a 12.5 m square launchpad for takeoff, and 17,175 meters squared of airspace, less than an average fireworks show.

The results for our city’s light show are quite encouraging indicators of our model’s applicability. Someday, when drone light shows become holiday tradition, it is our hope that our model will have helped light shows to achieve that status.
Lighting Up the Night

Team 8475

November 20, 2017
Dear Mayor,

As the chosen team to investigate the possibility of an outdoor aerial light show for next year’s festival, we have spared no effort in choreographing our own light show comprising of the images you suggested. The show begins with an oriental dragon stretching across the sky, followed by a Ferris wheel with cabins rotating above the skyline. The finale is our team number, 8475, inside of a spinning ring, before the fleet of glowing drones descend together toward the landing pad.

That is what your audience would see. Our team, on the other hand, has observed the logistics that it takes to make such a show possible. In order to go forward with a drone light show, we are planning on partnering with pioneers in the field, the Intel corporation, in order to utilize a fleet of their Shooting Star™ drones. The Shooting Star is the only drone model specifically created to put on light shows, perfect for our intended display for the festival.

Intel’s algorithms for mapping drone paths are proprietary, however, and all their own drone-piloting teams are occupied elsewhere at the time of the festival. This is where we come in. We will have access to Intel’s drones, but not their algorithms for designing shows and mapping drones, and thus we have developed our own algorithms for a drone light show.

Our model, now complete, outputs nearly all the needed information to put on the best possible drone show for the festival in our city. It allows us to use as few drones as possible for our display, in the hopes of minimizing the compensation cost to Intel for the use of their drones. Due to the ambitious size of the display, however - 91 meters by 154 meters - our city will claim the world record for Most Unmanned Aerial Vehicles (UAVs) airborne simultaneously, which is no small feat. While this may seem like an exorbitant cost for the festival, keep in mind that fireworks displays, the most obvious alternative, cost much more, as you are no doubt aware.

Our model also ensures that we need not sacrifice time for the drone show, despite the large number of drones. We have minimized the time necessary for the arrangement of the drones into each pattern, utilizing the speed of the Shooting Stars to their maximum capacity. Even under windy conditions, takeoff, transitions, and landing will only take a total of 13.1215 minutes of the Shooting Stars’ 20-minute battery life, of which 9 minutes are for onlookers to enjoy the breathtaking show.

The most important aspect of any show is of course the safety of the spectators, and as such, we have thoroughly considered whether or not the drone show will be safe for our city. We have determined that onlookers need to be kept at a distance of 200 meters, and a 12.5 meter by 12.5 meter box will have to be reserved as a launch area with a 91 meter by 154 meter chunk of airspace necessary to hold the show. However, the drone show developed using our model is significantly safer than its most obvious alternative, a fireworks show. Drones contain no explosive elements and thus cannot start fires, and the Shooting Star specifically has a propeller cage to make sure that even if one were to malfunction and crash, injury would be minimal.
The one drawback of a drone show against an alternative like a fireworks display, however, is its dependence on favorable weather. If the day of the festival turns out to be rainy, fireworks would prove a better option than drones - drones simply cannot fly under rain, while fireworks can still be launched. Of course, if lightning begins, then an outdoor show at all is out of the question, so neither would be doable. The show that we have developed for the festival is too large to perform in an indoor environment such as our football stadium, but using our model, a smaller, backup show utilizing a lesser number of drones could be performed if the weather is unfavorable.

Our final recommendation to you? Go ahead with the drone show. Our model will provide all the necessary information that our city requires in order to host the best drone show possible; all that is left to choreograph are the colors of the LEDs on the drones, a task much better left to our city’s talented community of artists than our team of math modelers. Logistically, hosting a drone show for our festival will create an influx of people from around the world, both as real and virtual visitors, who will tune in to watch such a momentous event - your own publicity as Mayor regarding the event will be wonderful once our model allows for a flawless showing. This is a chance for our city to not only save money on what could be a costly fireworks show or other traditional entertainment event, but to grasp a once-in-a-lifetime opportunity to set a world record while simultaneously lighting up the city night.

Please take your time and carefully consider our proposal; we will be eagerly awaiting your response.

Yours,
Team 8475
Contents

1 Introduction 4

2 Assumptions 5

3 Model 6
   3.1 Goals of the Model .............................................. 6
   3.2 Summary of Program ............................................. 6
   3.3 Influences .......................................................... 6
      3.3.1 Shi-Tomasi Minimum Eigenvalue Method .................. 6
      3.3.2 k-D Tree Nearest Neighbor Algorithm .................... 6
      3.3.3 Genetic Algorithms ........................................... 7
   3.4 Theory Section .................................................... 7
      3.4.1 Calculating the Positions of Each Drone ................ 7
      3.4.2 Determining the Number of Drones Needed ............... 8
   3.5 Model Building .................................................... 9
      3.5.1 Considerations ................................................. 9
      3.5.2 Size of the Light Show .................................... 9
      3.5.3 Initial Positions of Drones ................................ 11
      3.5.4 Choosing Which Drones to Use in the Show .............. 12
      3.5.5 Final Positions of Drones .................................. 14
      3.5.6 Calculating Drone Paths ................................... 16
      3.5.7 Analyzing External Conditions ............................. 17
   3.6 Animations ........................................................ 18

4 Model Results 18
   4.1 Our Light Show ................................................... 18
      4.1.1 Dragon Image ............................................... 18
   4.2 Ferris Wheel Image ............................................... 21
   4.3 Our Own Image: HiMCM Logo ................................... 23
   4.4 Timing of Performance ........................................... 25
   4.5 Animations ........................................................ 26

5 Strengths and Weaknesses 26
   5.1 Strengths ........................................................... 26
   5.2 Weaknesses ........................................................ 26
   5.3 Model Testing ....................................................... 26

6 Conclusion 27

7 References 28

8 Appendix 30
1 Introduction

Drone technology is beginning to permeate many aspects of life in the modern world, and for good reason. Delivery drones can efficiently deliver packages better than any existing methods. Medical drones can reach emergency patients faster than any ambulance. Security drones can surveil areas more quickly and more thoroughly than a team of human guards. The advent of commercial and industrial uses for drones, however, begets the vital question of how can drones be used outside the workplace. Intel’s answer? Entertainment.

Thus, the Intel Shooting Star™ drone was born, a quadcopter explicitly built for producing breathtaking aerial light shows similar in appearance to fireworks displays (Intel, n.d.). On October 7, 2016, 500 drones were used by Intel to set a Guinness World Record for the most Unmanned Aerial Vehicles (UAVs) airborne simultaneously. More impressively, however, is that only one pilot and one laptop were used to guide the show (Cheung, 2017). This “master computer” uses Intel’s proprietary drone-piloting algorithm to marshall each drone to its designated place in an aerial image, and once the show is complete, to bring each safely back to earth.

Our task is to develop a model to choreograph a drone light show with modeled flight paths for each drone that a similar “master computer” would control. We, however, must tackle issues Intel did not need to consider during their drone displays: those of economy and efficiency, by optimizing both the number and the placement of drones. We wish to perform a drone show optimized for both resource and time efficiency, and thus, the number of drones used in the show will be kept to a minimum to lower cost. The travel time for each drone (from liftoff to formation, and from one formation to each subsequent formation) during the show will be minimized as well – the Shooting Star’s battery lasts for approximately twenty minutes, and hence the drone show must run safely inside of that time frame. And of course, the less time spent positioning drones, the more viewing time will be available for the show!

During our light show, we will project the images of a dragon, a Ferris wheel, and our team number in the sky by creating corresponding sets of drone positions. However, the use of the model will not be limited to one show; in order to be of utility for future drone-show organizers, our model must be able to accept any image (given that there are enough drones available to properly display the image’s complexity) and convert it into a drone pattern. Our model must also take into account how wind affects drone flight performance. Based on our model’s optimized output, we can determine the number of drones our city must acquire to host our light show, and therefore the cost of staging the performance. We can also calculate the timing of the show, and any flight path adjustments necessitated by the wind. Using this information, we can then give the Mayor of our city an informed perspective of whether or not to pursue the option of a drone light show for our city’s annual festival this year.

However, our model will not be limited for use on merely one occasion, in one city, and for one holiday. Quite the contrary, our model will be implementable not only in our city, but cities around the globe to help establish drone shows as traditional holiday events, and to demonstrate that drones can be just as valuable for entertainment as they are for business.
2 Assumptions

**Assumption 1:** The functionality of each drone is exactly what is described by Intel.

*Justification:* We assigned the variables we used in this paper to values outlined by Intel’s own drones they use for their light shows. If some value is off or the functionality of a drone has an issue that was not previously accounted for, then the performance may not run too smoothly.

**Assumption 2:** Wind turbulence from the flight of one drone does not affect the flights of other drones.

*Justification:* According to Intel’s website, the maximum speed for each drone is 3 m/s. As the weight of each drone is 330 grams, the force needed to thrust the drone upwards would be negligible as 3 m/s is a relatively slow speed and the light weight of the machine would not require much power. Therefore, the wind turbulence from the drones is negligible.

**Assumption 3:** The 3D positioning of the drones in the sky can be interpreted as a 2D image from afar.

*Justification:* Images online and drawn on paper are in two dimensions. If we project those images in the same two dimensions in the sky, no significant distortion should be seen. A 3D image could look deformed based on the different locations from a viewers prospective. Moreover, many projectors, such as that in movie theaters, presentations, are in two dimensions as well. Therefore, 2D light show would be sufficient in providing entertainment for our audience.

**Assumption 4:** There are no obstacles in the paths of the drones and thus they may fly freely in the air.

*Justification:* The use of drones to create a light show relies on drones systematically moving from an initial to final position. If there were obstacles that hindered that movement, the drone would be unable to go to the desired destination, and the light show would not go on.

**Assumption 5:** There exists a large enough space in the city to accommodate a drone light show.

*Justification:* While the projection of certain images have a fixed height and width, different images will have different sizes and aspect ratios. Therefore, it is necessary to have a space that can accommodate a very large area. Moreover, the drones should not be constrained by space or else the restrictions movements could result in crashes.

**Assumption 6:** The drone light show will occur in visible conditions, and the drone lights will remain constantly on.

*Justification:* The only indication to viewers that the light show has started and/or is
going on is the lights on the drone. If some light suddenly fails or the audience cannot see
the lights and drones at all, that would hinder the performance of the entire light show.

3 Model

3.1 Goals of the Model

We want our model to address the following concerns in putting on a drone light show.

- Determine the number of drones required to achieve a clear, large image in the air,
while also minimizing the number of drones to keep costs down.

- Decide and simulate the flight paths of the drones to optimize the distance and the
time each drone has to travel.

- Establish a launch area and an air show space in our city to ensure that the safety
of the viewers is our first priority.

3.2 Summary of Program

We wrote a program in the programming language Mathematica that modeled drone
takeoff, transition, and landing in a drone light show scenario. The program was a
detailed, step-by-step simulation that took into account many factors associated with
hosting a drone light show. We first determine the number of drones to use based on the
preferred image for the light show. We proceed to determine the starting positions of
our drones, and then match each launch position to a respective final position for each
image. We then use 3D vectors to calculate the shortest possible drone flight path, and
then design a crash-prevention algorithm, followed by a parametric animation. The full
code of our simulation, written in Mathematica, can be found in the Appendix.

3.3 Influences

3.3.1 Shi-Tomasi Minimum Eigenvalue Method

The Shi-Tomasi Minimum Eigenvalue Method is used for edge detection and the finding of
corners in modern computer science. It is a useful tool for finding and locating coordinates
of corners and other artifacts in images. In our case, we use the Shi-Tomasi Minimum
Eigenvalue Method to determine the edges of the image we want to use, and the Shi-
Tomasi Minimum Eigenvalue Method gives us the coordinates of the points that follow
this edge.

3.3.2 $k$-D Tree Nearest Neighbor Algorithm

The $k$-D Tree Nearest Neighbor Algorithm is used in modern computer science to de-
termine numbers and points that are mathematically ”close” to a root node, a selected
point. In our case, we use the $k$-D Tree Nearest Neighbor Algorithm to determine the
nearest drone to a final drone position in an image, which is of utility when optimizing
takeoff, transition, and landing times.
3.3.3 Genetic Algorithms

Genetic algorithms are used in modern computer science for optimization. Modeled after the concept of natural selection, genetic algorithms use biological principles such as fitness, crossover, mutations, and reproduction to optimize solutions. In our case, we use genetic algorithms to optimize flight time for the drone light shows.

3.4 Theory Section

3.4.1 Calculating the Positions of Each Drone

We utilized a corner detection theory to identify the final positions of each drone to create a projected image in the sky. Notably, we adapted the Harris Corner algorithm in this work (Viragh et al., 2014). The fundamental theory behind this algorithm relies on characterizing points of interest within a certain image which can be robustly detected. In order to identify such points, we first processed images to black and white, using Otsu’s cluster variance maximization method, an algorithm often used in computer science to binarize images. With the black and white image, we analyzed a small window within the picture to possibly identify corners or edges that could be delineated by points, or eventually, drones.

The corners and edges within an image have distinct gradients, both horizontally and vertically, that can be captured to mathematically identify such features. We calculated the sum of the gradients within a small region of the picture to be:

$$E = \left( \frac{\sum I_x^2}{\sum I_y^2} \right)$$  \hspace{1cm} (1)

where $I_x$ and $I_y$ are the horizontal and vertical gradients of a given region. As $\sum I_x^2$ and $\sum I_y^2$ get increasingly larger, we can easily identify the corresponding region of a corner as that region will experience a notable change in appearance both in the x and y directions. Similarly, a high change in $\sum I_x^2$ may delineate a horizontal edge, while $\sum I_y^2$ may equivalently describe a vertical edge.

This algorithm is useful to accurately identify corners and edges with straight lines, as seen in Figure 1. However, when finding areas where the gradients may not be so distinct, such as when lines are diagonal and change in one direction more than they do in the other, Equation 1 may not be too accurate. This is seen in the center image of Figure 1 where the center corner would have a high $\sum I_x^2$ value but low $\sum I_y^2$ value. Based on Equation 1, the center corner would not be detected because of the low $\sum I_y^2$ value, or vertical gradient in that small window. Therefore, we utilized the Harris Corner algorithm to de-rotate the pre-specified window of the picture using eigenvalue decomposition. The Harris Detector for a certain shift, $[u, v]$ is given by the equation:

$$E(u, v) = \sum_{x,y} [I(x + u, y + v) - I(x, y)]^2$$  \hspace{1cm} (2)

where the shifted intensity is represented by $I(x + u, y + v)$. After computing a first order Taylor approximation of this equation, a matrix equation can be written as:

$$E(u, v) = \begin{bmatrix} u & v \end{bmatrix} \left( \sum \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$  \hspace{1cm} (3)
As eigenvalue decomposition maps the shifted image back to our original case as seen in the graphical representation in Figure 1, finding the eigenvalues, $\lambda_1$ and $\lambda_2$, of the matrix:

$$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

will then allow the identification of specific features of the given window in the image. If both $\lambda_1$ and $\lambda_2$ are small, no visible feature will be observed. On the contrary, a corner can be identified when both $\lambda_1$ and $\lambda_2$ are large, and a horizontal or vertical edge can be found when one $\lambda$ is large and the other, small.

![Figure 1: An ideal image used to identify corners and edges as denoted by Equation 1 (far right and left).](image)

If this image was rotated (center), the Harris Corner algorithm would be utilized to mathematically de-rotate the image back to the standard image based on a scaling factor using eigenvalue decomposition.

After determining all the corners and edges of our image, we fitted points along the picture to simulate drones projecting the image in the night sky.

### 3.4.2 Determining the Number of Drones Needed

The Shi–Tomasi corner detection algorithm was then adapted to help minimize the number of drones needed for our light show. Taking the two eigenvalues, $\lambda_1$ and $\lambda_2$ from matrix $M$, a measure of corner response, $R$, or the scoring function, can be calculated as:

$$R = \min(\lambda_1, \lambda_2)$$

Therefore, if $R$ is greater than a threshold value, $\lambda_{\text{min}}$, then it can be marked as a corner point, as denoted by the green area in Figure 2. If $R$ is greater than $\lambda_{\text{min}}$ for one $\lambda$ but not the other, as seen in the red area, then it will be classified as an edge. Otherwise, it will fail the test and not be concluded as a point in our projected image, seen in the grey area in Figure 2.
The Shi-Tomasi algorithm was used to minimize the number of drones for our light show by classifying points and categorizing regions of a specific image.

Because this algorithm relies on specifically classifying points and categorizing such regions of a picture, less points were needed to create the picture. Therefore, the number of drones used in our light show would be reduced as well.

3.5 Model Building

3.5.1 Considerations

A significant component in deciding how a light show is put on is determining how large the show will be. In order to represent

3.5.2 Size of the Light Show

An image illustrated by our light show should not be stretched or compressed. Therefore, when deciding how to a project a certain picture on to the night sky, we conserved the length and width of the image.

The maximum height of a drone can fly allowed by Federal Aviation Administration (FAA), is 400 feet, or approximately 121.92 meters. Accordingly given some image, with length \( \ell \) and width \( w \), we expressed the \( w \) as a function of \( \ell \) to constrain the height of our projected image and fall within FAA guidelines. Because the \( w \) can be represented by some constant \( \alpha \) times the length, we determined the width of a certain image to be:

\[
w = \frac{P_{\text{width}}}{P_{\text{length}}} \ell
\]

where \( P_{\text{width}} \) and \( P_{\text{length}} \) are the number of horizontal and vertical pixels, respectively, and \( \alpha = \frac{P_{\text{width}}}{P_{\text{length}}} \). Therefore, an image can be represented solely by a set length:
Figure 3: The drone light show conserves the aspect ratio and size of the original image while still following FAA guidelines.

Two problems of interest in determining the size of our show is then how large the projected image will be, and how far away viewers should stand. Therefore, we examined the average size of fireworks in order to determine $\ell$ and how far a viewer should ideally stand to watch the light show. The average height of a firework is given as 61 meters (US Firework Review, 2015). Then taking into consideration the minimum head tilt of a person to calculate a viewer’s line of vision, we can determine how far the viewer should stand. The normal neck range of motion backwards (thus looking up) is $45^\circ$ to $75^\circ$, and the height at which they should see our light show should be at the center of the height of all our drones. A graphical representation of viewers in respect to the light show can be seen below:

Figure 4: Graphical representation of the position of the light show in respect to a viewer’s perspective. Viewers $D$ distance away from the light show would see the total length of the light show to be $\ell$ (in meters).

Because the minimum neck range of motion is $45^\circ$, and the average height of a firework is 61 meters, we set our distance, $D$, which describes how far viewers would be from the light show, to be 61 meters through using trigonometry.

Calculating the height of our drone show, we set the boundaries of our show to be constrained by the average minimum height of fireworks, or 31 meters, and the maximum height a drone can fly allowed by the FAA, or 122 meters. Therefore, the height of our
light show is $122 - 31 = 91$ meters. We can then calculate the corresponding width of the projected image through Equation 5.

3.5.3 Initial Positions of Drones

Intel used 500 drones in their largest display, the 2016 Super Bowl halftime show (Eindhoven, 2017). In using our Shi-Tomasi Minimum Eigenvalue Method for various images, we expect that our drone show, no matter the complexity of the image, will use at maximum 625 drones. We inspect example images below to make sure that most, if not all of the images that the mayor chooses will need less than 625 drones.

**Table 1:** We inspected three common shapes with our Shi-Tomasi Minimum Eigenvalue Method to verify that images would necessitate less than 625 drones.

<table>
<thead>
<tr>
<th>Image</th>
<th>Analyzed Image</th>
<th>Drones Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Duck" /></td>
<td><img src="image2.png" alt="Duck Analysis" /></td>
<td>185</td>
</tr>
<tr>
<td><img src="image3.png" alt="Elephant" /></td>
<td><img src="image4.png" alt="Elephant Analysis" /></td>
<td>197</td>
</tr>
<tr>
<td><img src="image5.png" alt="Olympics" /></td>
<td><img src="image6.png" alt="Olympics Analysis" /></td>
<td>484</td>
</tr>
</tbody>
</table>

Another reason the number 625 was chosen is because of the simple implementation of a 25 by 25 array for the drone’s initial positions. This 25 by 25 array minimizes the setup required by a crew, as each drone lies inside a box in the grid, shown below in Figure 5.
Figure 5: The initial positions of all drones if every drone was utilized in our model.

This array also helps keep the show safe. The drones are concentrated in a 12.5m by 12.5m box where each drone each occupies a 0.5m by 0.5m square, a relatively small area that can easily be cordoned off from spectators to prevent interference with the drones (although a larger area will of course be necessary during the drone show to allow the drones to spread).

3.5.4 Choosing Which Drones to Use in the Show

Far fewer than 625 drones will be used in the majority of light shows, meaning only certain spots on the grid will be utilized. Therefore, we must determine which initial drone placements are to be used for any given image. To do this, we take each three-dimensional point generated by using our Shi-Tomasi Minimum Eigenvalue search on an image, and determine the drone on the grid closest to that point. We determine from this the drones that will be used in the light show, and those that will stay on the ground idle. To do this, we use a nearest neighbor algorithm (Nearest-Neighbor Search, 2011; Park, Kim, Kim, & Kim, 2016).

We use $k$-D trees to code our nearest neighbor algorithm. $k$-D trees are data structures that partition $k$-Dimensional spaces to organize points within an area (Sheehy, 2010). In $k$-D trees, the dimension of data is $k$, so in the 3-dimensional space of the drone show, $k=3$. Essentially, $k$-D trees are binary trees in which each node generates a hyperplane that splits the entire area under analysis into two parts (Bentley, 1975). Each of these parts is called a half-space. So, a $k$-D tree with six unique nodes would divide the overall space into 7 parts. For example, a two dimensional $k$-D tree made with the points (7,2), (5,4), (9,6), (2,3), (4,7), and (8,1) would look as below:
Figure 6: A sample $k$-D tree was generated with six unique nodes that yielded seven regions of the overall space. This $k$-D tree would divide its overall area into 7 parts, as it has 6 nodes. The parts are shown below:

Figure 7: The overall region was divided into 7 regions based on a $k$-D tree with 6 nodes.

For our algorithm, we aim to calculate the nearest unique drone start position for every unique drone end position. So, we begin by picking the highest point within our image, as this point has the largest Z-axis value and thus most likely requires the longest distance to cover. We then proceed as follows.

1. The highest point in the image is the current search point and thus is the root node, or top node, of our $k$-D tree. The points we utilize to determine proximity comprise the list of all unused drone start positions.

2. We then divide the overall 3D space of the drone start positions by the splitting hyperplane defined by the root node. This splitting hyperplane follows the x-axis.
This divides the points in the 3D space into two categories, creating our first binary branch in the $k$-D tree. This causes the root node to branch off to two leaf nodes. Each of these leaf nodes is chosen by selecting a point on either side of the splitting hyperplane.

3. We then create a hypersphere around each of the leaf nodes. This hypersphere has radius $r$ and will intersect the previous splitting hyperplane for some value of $r$. If this value of $r$ is smaller than the current best, then this value of $r$ is now the current best distance, and that node is the current closest point. If this is the case, we continue on that branch of the tree, and eliminate all other branches. If not, we repeat steps one and two for other branches of the $k$-D tree, first using a $y$-axis splitting hyperplane, then a $z$-axis splitting hyperplane, and then cycling back to a $x$-axis splitting hyperplane, and so on and so on, until the algorithm has finished processing for the root node and all the leaf nodes.

The algorithm then outputs the drone positions in the grid which are closest to the final positions in the aerial image.

In using this algorithm, we can optimize the drone start positions, and choose the drones we will use for the light show from the total 625 drones, or drone spots on the grid, available for use. For example, when we use the $k$-D tree nearest neighbors algorithm to match end points to start points for an example dragon drawing, we are able to optimize the drones used in our light show. Figure 8 shows an example. In order to create an aerial display based off of the sample image (different from the dragon in our actual drone show), we were able to use a certain optimized set of 278 drones instead of the full 25 by 25 drone array, as determined by $k$-D trees.

![Figure 8: Using the $k$-D tree nearest neighbors algorithm, the number of drones needed to model a sample dragon was minimized to 278 drones from the original 625.](image)

3.5.5 Final Positions of Drones

Once we have decided our starting drone positions, we must correspond each starting position with an ending position in the image. This is, of course, a necessary step to map each drone’s path through the light show.
Assume for the purposes of this explanation that we require \( n \) number of drones to create our image for the light show.

If we were to use a random assignment approach to this problem, we would have \( n! \) ways to assign a starting drone to an ending drone position. For large numbers (such as 625), calculating the most efficient solution of these \( n! \) is extremely computationally inefficient. Thus, a brute force method will not work here.

Instead, we use an adaptive stochastic optimization algorithm called a Genetic Algorithm. The genetic algorithm, an algorithm geared toward optimization problems, aims to mimic natural selection to find the best possible algorithm in any scenario to which it is applied (Rowland & Weisstein, n.d.; Suleman, 1997). The Genetic Algorithm mimics many aspects of evolution, specifically those of fitness, reproduction, crossover, and mutation.

We use this Genetic Algorithm in order to correspond each drone starting position with a drone ending position. Our Genetic Algorithm follows the following logic, as seen in Figure 9.

![Figure 9: The Genetic Algorithm follows cyclic logic with numerous iterations of crossover and mutation to yield optimized output via the principle of evolution.](image)

We begin by creating a random set of possible solutions. To accomplish this, we take a randomly generated arrangement of starting drone positions and match them each, again randomly, to an ending drone position. We do this for all starting and ending drone positions, creating one possible mapping solution. Our algorithm repeats the process 100 times, creating 100 random solutions. This is our initial solution population.

As in any population, some solutions will have higher fitness than others. In our model, the fitness function is defined as the variance of the distance each drone has to travel in a certain solution. The lower the variance of a solution, the higher its fitness. We established this definition for our fitness function to minimize travel distance, so that solutions with the least variance emerge in the end as the fittest. We calculate variance, \( \sigma^2 \), as:

\[
\sigma^2 = \frac{\sum_{i=0}^{n} (x_i - \bar{x})^2}{n - 1}
\]  

where \( x_i \) is an individual point, \( \bar{x} \) is the average of the data set, and \( n \) is the number of points. Logically, it is more ideal to have all drones travel a similar distance rather than having some drones move a short distance while others move a longer distance because a projected image can only be shown when each drone is in its final position. Having
drones take longer to reach their final destination will nullify the efforts of other drones which took less time to arrive at their given position. Therefore, by minimizing $\sigma^2$, we can systematically determine the flight paths for each drone. Each solution from our model is then assigned a calculated fitness score defined by its variance, where a lower score translates to higher fitness.

Subsequently, solutions then enter the “reproductive phase.” We utilize the elitism strategy to regulate reproduction in our genetic algorithm, where the extremely fit individuals are favored. Often, the elitism strategy is used when the problem has a single, linear solution, as we do in this case. So, we choose the 10 solutions with the lowest fitness score to reproduce.

Reproduction of solutions in our Genetic Algorithm is completed via crossover, or the switching of elements in two fit solutions, to ideally create a fitter and better solution as progeny. This is accomplished by randomly selecting two of our previously determined top 10 solutions at a time, and then randomly switching selections of initial and final drone positions, similar to what is displayed in Figure 10. The process repeats until 100 new solutions have been created, which comprise the next generation.

![Figure 10: Our Genetic Algorithm optimizes output via the mechanism of crossover, which yields a better "child," or solution with each generation.](image)

Just as in any true evolutionary system, our solutions also undergo a degree of mutation. During the mutation phase of our model, 0.2% of drone start positions are switched randomly in order to open our solution “gene pool” to fitter, more efficient possibilities. Most mutations, simply due to their random nature, will not be conducive to efficiency; however, akin to actual natural selection, mutations advantageous to fitness will be preserved.

Once we apply our entire Genetic Algorithm to the randomly generated solution set and allow it to run for 1000 generations, 10 optimized solution sets of drone paths result, with the distance between each drone’s initial and final point minimized. We can then input the most efficient of these into the next stage of the model.

### 3.5.6 Calculating Drone Paths

Using the nearest neighbor and genetic algorithm to systematically determine the final positions for each specific drone, we can calculate the flight paths of each drone in 3D space using the initial, $I$, and final, $F$, positions of the drone. Given these points $I$ and $F$,
On a coordinate plane, a direction vector $D$ is described as:

$$\vec{D} = \vec{F} = F(x_1, y_1, z_1) - I(x_0, y_0, z_0)$$

(7)

In order to describe this flight path as a function of time, we normalize the direction vector to find the unit vector, and multiply it by the maximum velocity of the drone, $\vec{v}_{max}$:

$$\vec{v}_f = \vec{v}_{max}\hat{D}$$

(8)

Given that the maximum velocity of each drone is 3 m/s according to Intel’s specifications, we can substitute that value for $\vec{v}_{max}$. Finding the position, $s$, of the drone with respect to time, $t$, can simply be modeled by the equation:

$$s(t) = (x_0, y_0, z_0) + t \cdot \vec{v}_f = (x_0, y_0, z_0) + t(\vec{v}_{xf}, \vec{v}_{yf}, \vec{v}_{zf})$$

(9)

As the flight path of each drone consists of a straight line connecting the initial and final positions of the drone, it relies on the assumption that the drone may fly freely in the air as there are no obstacles blocking its path. Therefore, we had to consider the possibilities of crashes during the performance.

With the location of each drone modeled at every time interval, we developed an algorithm to find if there would exist the possibility of a crash during the light show. This algorithm entailed defining a critical radius, $r_{critical}$, from a specific drone $A$, where any other drone inside this critical radius would be identified as a drone $B_i$ ($i = 1, 2, 3\ldots$ number of drones) that could potentially crash with drone $A$. This was done for every drone at a single time interval, and was calculated every $n$ seconds until each drone reached its final destination.

In order to determine the value of $r_{critical}$, we took the size of the drone, 0.3 m, and added 0.1 m of padding to ensure no crashes would occur, to achieve an $r_{critical}$ value of 0.5.

The value for $n$ was calculated by:

$$n = \frac{\text{Drone Size}}{v_{max}}$$

(10)

since a drone would not be able to completely pass another drone in any time less than $n$ seconds.

If a drone $B_i$ exists and a possible crash site is identified, a slowdown factor would be applied to the drone with the least amount of distance left to ensure that the crash would not occur. This would then minimize the amount of time needed for each drone to go from its initial to final location. The slowdown factor was calculated based on the fact that the crash area, or area within $r_{critical}$, would go away once one drone had left the area. As the maximum speed of each drone is 3 m/s, a drone $A$ should pass a possible crash zone in $n$ seconds. This is the same $n$ described in Equation 10. Because we can already determine the position of drones for the entire light show from Equation 9, we can then calculate all the possible times, $t$, for each drone where a potential crash zone exists, and inherently slow down the drone $3n$ seconds before each crash through using kinematic equations. The value for $3n$ was arbitrarily identified to ensure the drone does not abruptly stop and also help maintain its maximum speed to save time. After the potential crash zone had been voided, each drone would continue at $\vec{v}_{max}$.

### 3.5.7 Analyzing External Conditions

As our light show would be performed outside, our drones would be subjected to various weather conditions during our performance. In order to ensure the show can run...
smoothly, almost ideal conditions are needed (i.e. no rain, snow, etc.). Moreover, given
the maximum speed of each drone to be 3 m/s, certain criteria must be met in terms
of outside wind speed in order for our drones to run. The velocity vector describing
wind, $\vec{v}_w$, can be described by its components in the x, y, and z directions. Similarly, the
velocity of our drones can be divided into the same three components. Therefore, when
accounting for wind the final velocity of the drone, $\vec{v}_{fd}$, can be expressed as,

$$\vec{v}_{fd} = (\vec{v}_{xid} - \vec{v}_{xw}, \vec{v}_{yid} - \vec{v}_{yw}, \vec{v}_{zid} - \vec{v}_{zw})$$

where the initial velocity of the drone, $\vec{v}_{id} = (\vec{v}_{xid}, \vec{v}_{yid}, \vec{v}_{zid})$ and the velocity of the wind
is represented as $(\vec{v}_w = \vec{v}_{xw}, \vec{v}_{yw}, \vec{v}_{zw})$. Subtracting both these vectors will give the final
velocity of the drone.

If we were to project our image in the sky, we would need all component vectors to
be positive, as each drone would need to move into their respective position. Ideally,
we would have to project our image with drones moving with the direction of the wind.
However, if one component of the $\vec{v}_w$ is greater than the corresponding component of $\vec{v}_{id},$
and the drone is moving against the direction of the wind with respect to that specific
component, then our drone will not be able to arrive at its pre-determined location. Wind
may also significantly slow down the movement of the drone; however, the drone will still
be able to move if $\vec{v}_{fd}$ remains positive.

### 3.6 Animations

Once every drone has reached its final position, we decided to animate certain images,
which relied on setting new positions with respect to time, $t$. Because our projected
image would be in 2D space, or on the y-z plane, we can describe the movement of the
drones as points along a 2-variable parametric equation:

$$s(t) = \begin{cases} y(t) \\ z(t) \end{cases}$$

The final positions of each individual drone would then be updated to calculate the flight
path, as described in Equation 9, for the following image in our light show.

### 4 Model Results

#### 4.1 Our Light Show

##### 4.1.1 Dragon Image

Upon comparing various images of a dragon, we decided to use the following dragon
image for our light show due to its simplicity and high potential for visibility in the night
sky:
From this image, we can apply our Shi-Tomasi Minimum Eigenvalue Method to determine the number of drones needed as well as the drone positions to form the outline of the dragon. We scale the x, y, and z coordinates of our drone outline to fit our space restrictions, and also convert the coordinate system into the metric system rather than using the previously measured pixels.

We calculate we will need 554 drones to create our dragon light show. We then proceed to use our $k$-D tree algorithm to determine which starting drones to use. The $k$-D tree algorithm determines the nearest unused drone starting point to each end point, and gives us the following starting drone arrangement.
Figure 13: The optimized starting drone arrangement for displaying the image of the dragon in the light show.

From this drone starting position, we then have to determine which drones travel to which drone end position. To do this, we use our genetic algorithm. After 1000 generations, our genetic algorithm outputs an optimal start to end drone combination. We use this combination to create our final drone arrangement.

Figure 14: The final drone arrangement for our dragon in 3-dimensional space.

The above dragon, made out of lighted drones, is 124 meters across and 91 meters tall, and is suspended 200 meters in the air, as per our calculations for the best possible light show viewing experience. Each drone travels at its max speed of 3 meters per second until it reaches its destination, at which point the drones hover idle, waiting for the other drones to reach their respective locations. The drone with the longest distance to cover has 291.44 meters to cover, as per our optimized drone positions. Since each drone is traveling at the max speed of 3 meters per second, this drone will take $291.44/3 = 97.1467$ seconds to reach its final position in the dragon. Our dragon will then remain for viewing for 3 minutes, which is ample time for everyone to view the magnificence of our dragon.
4.2 Ferris Wheel Image

The second image in our light show is a moving Ferris Wheel. We use this image to design our Ferris Wheel:

![Ferris Wheel Image](image.png)

**Figure 15:** The image of the Ferris wheel that our drone light show will display.

From this image, we can apply our Shi – Tomasi Minimum Eigenvalue Method to determine the number of drones needed and the positioning of each drone. We then, like before, scale the x, y, z coordinate system to fit a 91-meter high image:

![Individual Point Breakdown](image.png)

**Figure 16:** The individual point breakdown for the image of the Ferris wheel as determined by the Shi-Tomasi Minimum Eigenvalue Method.

We calculate that we will need 305 drones to create our Ferris Wheel light show. We then proceed to use our $k$-D tree algorithm to determine which starting point drones to use. This $k$-D tree algorithm determines the nearest unused drone starting point to each
end point, and gives us the list of starting point drones to use. However, this time, many if not all of our starting drones will be coming from the previous image, the dragon. From the dragon, some drones will hover and switch off their lights, yet 305 of the drones will remain active and will move to form the Ferris wheel. Thus, we must apply our $k$-D tree nearest neighbors algorithm to the points from the dragon light show. In doing so, we isolate our 305 nearest starting drone positions.

Figure 17: The starting drone arrangement continued from the final drone arrangement for the dragon, which displays the image of the Ferris wheel in the light show.

From this drone starting position, we then have to determine which drones travel to which drone end position. To do this, we use our genetic algorithm. After 1000 generations of crossover, our genetic algorithm outputs an optimal start to end drone combination. We use this combination to create our final drone arrangement.

Figure 18: The final drone arrangement for the Ferris wheel in 3-dimensional space.

This Ferris wheel, consisting of lighted drones, is 79 meters across and 91 meters tall, and is suspended 200 meters in the air, the optimal conditions per our calculations for the best possible light show viewing experience. Each drone travels at its max speed of
3 meters per second until it reaches its destination, at which point the drones hover idle, waiting for the other drones to reach their respective locations.

The drone with the longest distance to cover has to travel 76.0754 meters from the dragon to the Ferris wheel. Since each drone is traveling at a max speed of 3 meters per second, this drone will take $76.0754/3 = 25.3585$ seconds to reach its final position in the Ferris wheel. Our Ferris wheel will then remain for viewing for 3 minutes, which is ample time for everyone to view the splendor of our Ferris wheel.

During these three minutes, the Ferris wheel carriages will rotate, according to our calculations we defined in Equation 12:

$$s(t) = \begin{cases} y(t) = r \sin(t) \\ z(t) = r \cos(t) \end{cases}$$  \hfill (13)

for each individual drone, which would constantly be a set distance $r$ away from the center of the ferris wheel.

### 4.3 Our Own Image: HiMCM Logo

The third and final image in our light show is our encircled team number, shown below:

![Figure 19](image.png)

**Figure 19:** Our team number that our drone light show will display.

For this image, we can apply our Shi – Tomasi Minimum Eigenvalue Method to determine the number of drones needed and the positioning of the drones. We then, akin to before, scale the x, y, z coordinate system to fit a 91-meter high image:
Figure 20: The individual point breakdown for the image of our team number as determined by the Shi-Tomasi Minimum Eigenvalue Method.

We calculate that we will need 560 drones to create the light show for our team number. We then proceed to use our $k$D tree algorithm to determine which starting drones to use. This $k$-D tree algorithm determines the nearest unused drone starting point to each end point, and gives us the list of starting drones to use. However, this time, many of our starting drones will be coming from the previous image, the Ferris wheel, and some will come from the dragon. So, we must apply our $k$-D tree nearest neighbors algorithm to the points from the Ferris wheel light show, and those unused remaining from the dragon light show. In doing so, we isolate our 560 nearest starting drone positions.

Figure 21: The starting drone arrangement continued from the final drone arrangement for the Ferris wheel, which displays the image of our team number in the light show.

These starting drone positions comprise of drones from the dragon that were unused and had their light turned off for the Ferris wheel, and also comprise of drones that were a part of the Ferris wheel. From this drone starting position, we then have to determine which drones travel to which drone end position. To do this, we use our genetic algorithm again. After 1000 generations of mutation, our genetic algorithm outputs an optimal start to end drone combination. We use this combination to create our final drone arrangement:
Figure 22: The final drone arrangement for our team number in 3-dimensional space.

Our team number, comprising of lighted drones, is 154 meters across and 91 meters tall, and is suspended 200 meters in the air, the optimal measurements per our calculations for the best possible light show viewing experience. Each drone travels at its max speed of 3 meters per second until it reaches its destination, at which point the drones hover idle, waiting for the other drones to reach their locations.

The drone with the longest distance to cover has 82.9145 meters to cover from the starting positions to the final team number position. Since each drone is traveling at the max speed of 3 meters per second, this drone will take $82.9145/3 = 27.6382$ seconds to reach its final position in the team number. Our team number will then remain for viewing for 3 minutes, which is ample time for everyone to learn about the team who set a new record for a drone light show.

Then, our drones must return to their starting positions on the ground in a smooth landing. We calculate that the drone with the longest distance to cover to return to its landing position has a distance of 291.44 meters to cover. Since each drone is traveling at the max speed of 3 meters per second, this drone will take $291.44/3 = 97.1467$ seconds to reach its landing location.

4.4 Timing of Performance

In total, our light show will follow the following time schedule:

<table>
<thead>
<tr>
<th>Movement</th>
<th>Time Needed (seconds)</th>
<th>Total Time Elapsed (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Dragon</td>
<td>97.1467</td>
<td>1.619112</td>
</tr>
<tr>
<td>In Dragon</td>
<td>180</td>
<td>4.619112</td>
</tr>
<tr>
<td>To Ferris wheel</td>
<td>25.3585</td>
<td>5.041753</td>
</tr>
<tr>
<td>In Ferris wheel</td>
<td>180</td>
<td>8.041753</td>
</tr>
<tr>
<td>To Team Number</td>
<td>27.6382</td>
<td>8.50239</td>
</tr>
<tr>
<td>In Team Number</td>
<td>180</td>
<td>11.50239</td>
</tr>
<tr>
<td>To Landing Locations</td>
<td>97.1467</td>
<td>13.1215</td>
</tr>
</tbody>
</table>

Our light show will run in a total of 13.1215 minutes, well within the 20 minute battery of the Intel drones. This leaves approximately 7 minutes of buffer time for any unexpected complications.
The launch area of our drones is the 12.5-meter by 12.5-meter launch box, and the maximum area needed in the air is for the team number, which is 91 meters tall and 154 meters wide.

Since our light show is 200 meters away, the safety of the viewers is not a concern. The safety of the setup crew is taken into consideration, as the drones start and end in a compact 12.5 meter by 12.5 meter box, limiting the probability someone on the setup crew will be injured.

4.5 Animations
We have linked the animations for our simulations here:

- Dragon Animation: tinyurl.com/8475Dragon
- Ferris Wheel Animation: tinyurl.com/8475Ferris
- 8475 Logo Animation: tinyurl.com/8475Logo

5 Strengths and Weaknesses

5.1 Strengths

1. Our model can take into account nearly any 2D image found online or drawn by hand to systematically map and project that image using drones. The versatility and robustness of this method can create a diverse collection of images used to provide entertainment for audiences during a drone light show.

2. Our model makes very few assumptions and uses few independent variables. This allows any person, not only a mathematician, to use our algorithm to design a drone light show.

3. The model is extremely computationally efficient. We use optimization algorithms in our model, rather than brute force, causing the model to run multiple times faster than it would with a brute force method.

5.2 Weaknesses

1. Our model does not take into account color. Our model, if used to create a drone light show, would create a unicolor show. An artist would be required to assign each drone a color by hand, as our model does not factor in color.

2. Our model works best with simple images, while images with a lot of congestion do not work optimally. This is because we are attempting to minimize the number of drones, as to keep price down, and with a limited number of drones there is a limit to the detail that those drones can create. Given an infinite number of drones, our model would easily accommodate for those images with a lot of congestion.

5.3 Model Testing
We used three different stochastic minimization and/or optimization algorithms in our model. For all three algorithms, in order to make sure we were getting the best possible
solution, we ran the optimization algorithms multiple times, receiving the same results every time. This revealed the consistency of our algorithms. Even though the algorithms may include a random element, we receive similar results each time, demonstrating that our algorithms are optimizing to the best possible solution. We also compared solutions created by our algorithms to solutions created by brute force algorithms. Every time, our optimization algorithm created a better solution than brute force. In addition, these algorithms were more computationally efficient.

6 Conclusion

The results from our model indicate that a drone show is quite a safe and cost-effective way for a city to put on a show. Our results have proven that even on a large - in fact world-record-breaking - scale, it is quite energetically, computationally and chronologically efficient to command a swarm of drones from a single source. Even a massive amount of drones needs only a small space for takeoff and landing, in our case a 12.5 m by 12.5 m square, which could be utilized in instances beyond a light show. Human rescue missions, for example, could be much more efficiently completed by a swarm of drones than groups of humans, and could keep rescuers free from danger in risky situations. Swarms of drones could be used for infrastructure maintenance and management as well, inspecting the condition of bridges, skyscrapers, telecom towers, and pipelines safely and remotely. Optimized arrangement of drones would further aid in warfare and defense, as drones could be used for bomb detection, surveillance, and air strikes. There are many other examples of the utility of drones, yet it is futile to try and exhaust them all; the possibilities for the use of drone swarms are endless.

However, in context of our model’s purpose, the sheer possibility of light shows and other forms of drone entertainment are exciting to fathom. If our model today can efficiently map nearly any image that we wish with at most 625 drones, then there is no reason that someday, our model and models similar to it could help to command thousands of drones, enveloping cities in light shows visible for miles. Time will tell if, indeed, drones someday light up the night in cities around the world, becoming as well-established of a tradition as fireworks shows are today.
7 References


8 Appendix

Below is the code we used in our model, all written in Mathematica. We begin with the code used to design and create the light show, followed by the testing and initial implementations of the Shi-Tomasi Minimum Eigenvalue Method, the $k$-D Tree Nearest Neighbor Algorithm, and the genetic algorithm.
Implementation and Testing of the Shi - Tomasi Minimum Eigenvalue Method and k-D Tree Nearest Neighbor Algorithm
bimg = Binarize[img]

ImageKeypoints[bimg];
HighlightImage[bimg, %]
ic = ImageCorners[bimg];
HighlightImage[bimg, ic]
ic = ImageCorners[bimg, 2, 0, 2, Method -> "ShiTomasi"]; HighlightImage[bimg, ic]
ed = EdgeDetect[img]
ic = ImageCorners[ed, 2, 0, 3, Method -> "ShiTomasi"]; HighlightImage[ed, ic]

Length@ic
286
img = ; bimg = Binarize@img; ed = EdgeDetect[img];
ic = ImageCorners[ed, 2, 0, 1, Method -> "ShiTomasi"]; HighlightImage[ed, ic] Length@ic ListPlot[ic]
\textbf{Implementation of Initialization Algorithms.nb}

\texttt{img = \textbf{\textcircled{I}}; bimg = Binarize@img; ed = EdgeDetect[img];}

\texttt{ic = ImageCorners[ed, 2, 0, 5, Method \rightarrow \textquote{\textquote{ShiTomasi}}];}
\texttt{HighlightImage[ed, ic]}
\texttt{Length@ic}
\texttt{ListPlot[ic]}

\textbf{Printed by Wolfram Mathematica Student Edition}
img = ; ed = EdgeDetect[img];

icTabs = Table[ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"], {i, 1, 10}];
If[icTabs[[1]] >= 500,
  ic = ImageCorners[ed, 2, 0, 1, Method -> "ShiTomasi"];
  ic = ImageCorners[ed, 2, 0, Position[Nearest[icTabs, 500], icTabs][[1, 1]], Method -> "ShiTomasi"]];
HighlightImage[ed, ic]
Length@ic
ListPlot[ic]
edgePointer[img_] := Block[{icTabs, ed},
ed = EdgeDetect[img];
icTabs = Table[ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"], {i, 1, 10}];
ic = If[Length@icTabs[[1]] >= 500,
  ImageCorners[ed, 2, 0, 1, Method -> "ShiTomasi"],
  Position[Nearest[icTabs, 500], icTabs][[1, 1]], Method -> "ShiTomasi"];
{HighlightImage[ed, ic],
  Length@ic,
  ListPlot[ic]}]

Using Shi-Tomasi Minimum EigenValue Method to Detect corners of Image

EdgeDetect[ ]
Add Math stuff below

edgePointer
pointMinMaker[img_] := Block[{icTabs, ed},
ed = EdgeDetect[img];
icTabs = Table[ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"], {i, 1, 10}];
ic = If[Length@icTabs[[1]] >= 500,
    ImageCorners[ed, 2, 0, 1, Method -> "ShiTomasi"],
    ImageCorners[ed, 2, 0, Position[Nearest[icTabs, 500], icTabs][[1, 1]], Method -> "ShiTomasi"]];
{HighlightImage[ed, ic],
Length@ic,
ListPlot[ic]}]

Need to minimize number of points (i.e. drones) needed.

ed = EdgeDetect[

icTabs = Table[Length@ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"], {i, 1, 10}]
{HighlightImage[ed, ic],
Length@ic,
ListPlot[ic]}
{575, 362, 279, 202, 161, 137, 121, 101, 85, 76}

```
ListPlot: ic is not a list of numbers or pairs of numbers.
```

Space between center of one drone and center of other must be at least .532 m according to Intel
Average Arm Length is 0.6096m

\[
\text{\#Pixel Difference Width} = \frac{\text{\#Pixel in Width}}{\text{.6096 m}} \times .532 \ m \\
\text{\#Pixel Difference length} = \frac{\text{\#Pixel in Length}}{\text{.6096 m}} \times .532 \ m
\]

Imagelines

ImageDimensions[ed]
{500, 350}
575 / 25
23
\[ \text{ed} = \text{ImageCrop}\@\text{EdgeDetect}[\text{image}]; \]

\text{iDim} = \text{ImageDimensions}[\text{ed}];
\text{pxWid} = \text{iDim}[1];
\text{pxLen} = \text{iDim}[2];
\{\text{pxDifWid} = \frac{\text{pxWid}}{200}, 0.6096 \ast \text{Max}[\text{end}[[\text{All, 3}]]] \}
335.5

\text{end} = \text{Prepend}[\#, 0] - \{0, 250, 0\} \&/\@\text{ic};
\text{end}[[1]]
\{0, 265.5, 293.5\}

\text{ListPointPlot3D}[\text{end}]

\text{\textbf{start} = Flatten[Table[\{i, j, 0\}, \{i, 25\}, \{j, 23\}], 1];}
\texttt{In[88]} := \texttt{ListPointPlot3D[start, ViewPoint \rightarrow Above]}

\texttt{Out[88]}=

\texttt{vector} = \texttt{Nearest[end, \#] - \#}; &

\texttt{near} = \texttt{Nearest[end, \{0, 2, 1\}]}
\texttt{\{0, 56.5, 149.5\}}

\texttt{start[[1]]}
\texttt{\{1, 1, 0\}}

\texttt{near} = \texttt{Nearest[end, start[[1]]][[1]]}
\texttt{\{0, 56.5, 149.5\}}

\texttt{end} = \texttt{Drop[end, Position[end, near][[1]]]}

\texttt{vector} = \texttt{N[(near - start[[1]])/100]}
\texttt{\{-0.01, 0.555, 1.495\}}

\texttt{end} = \texttt{Prepend[\#, \{0, 250, 0\}]\&/\#ic;}
\texttt{vectors} = \texttt{Table[near = Nearest[end, start[[i]]][[1]];}
\texttt{end = Drop[end, Position[end, near][[1]]];}
\texttt{vector = N[(near - start[[i]])/100], \{i, 1, 575, 1\}};
vectors[[1]]

(-0.01, 0.555, 1.495)

Table[start[[1]] + vectors[[1]] * j, {i, 1, 1}, {j, 0, 100}]

{{{1., 1., ... center of drones

Needs to be at least .532 between center of drones
pointMinMakerScaled[
    img_] := Block[
    {ed, imgDim, pxWid, pxLen, len, wid, pxSize, diff},
    ed = ImageCrop[EdgeDetect[img]];
    imgDim = ImageDimensions[ed];
    pxWid = imgDim[[1]]; pxLen = imgDim[[2]];
    len = 61;
    wid = N[pxWid * len];
    pxSize = N[len / pxLen];
    diff = Ceiling[.532 / pxSize];
    icTabs = Table[Length[ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"]], {i, diff, 10, 1}];
    ic = If[Length[icTabs[[1]]] <= 500, ImageCorners[ed, 2, 0, diff, Method -> "ShiTomasi"],
        ImageCorners[ed, 2, 0, Position[Nearest[icTabs, 500], icTabs][[1, 1]],
        Method -> "ShiTomasi"];
    {diff, HighlightImage[ed, ic],
        Length@ic, ListPlot[ic]}]

pointMinMakerScaled2[
    img_] :=
    Block[{ed, imgDim, pxWid, pxLen, len, wid, pxSize, diff, maxW, maxL, ic},
    ed = ImageCrop[EdgeDetect[img]];
    imgDim = ImageDimensions[ed];
    pxWid = imgDim[[1]]; pxLen = imgDim[[2]];
    len = 61;
    wid = N[pxWid * len];
    pxSize = N[len / pxLen];
    diff = Ceiling[.532 / pxSize];
    icTabs = Table[Length[ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"]], {i, diff, 10, 1}];
    ic = If[Length[icTabs[[1]]] <= 500, ImageCorners[ed, 2, 0, diff, Method -> "ShiTomasi"],
        ImageCorners[ed, 2, 0, Position[Nearest[icTabs, 500], icTabs][[1, 1]],
        Method -> "ShiTomasi"]];
    maxW = Max@ic[[All, 1]]; maxL = Max@ic[[All, 2]];
    scaledIc = {wid / maxW * #[[1]], len / maxL * #[[2]]} & /@ ic;
    {diff, HighlightImage[ed, ic],
        Length@ic, ListPlot[ic, PlotMarkers -> {Automatic, Medium}]}]
In[2]:= pointMinMakerScaled2[
pointMinMakerScaled2[
pointMinMakerScaled2[
}
```mathematica
pointMinMakerScaled2[

\{5, 100, 484, 200\}]
```

```
In[3]:= end = Prepend[#, 0] - \{0, Median[scaledIc[[All, 1]]], -200\} & /@ scaledIc;
```

```
In[4]:= start = Flatten[Table[\{i, j, 0\}, \{i, 25\}, \{j, 25\}], 1];
```

Go through each in Start instead of End

```
distances = Table[near = Nearest[start, end[[i]]][[1]]; start = Drop[start, Position[start, near][[1]]]; Norm[end[[1]] - near], \{i, 1, Length@end, 1\}];
```

```
Max[distances]
```

```
67.4005
```

```
Min[distances]
```

```
1.01344
```

### Animation Only

```
In[11]:= vectors = Table[near = Nearest[start, end[[i]]][[1]]; start = Drop[start, Position[start, near][[1]]]; vector = N[(near - end[[i]])/100], \{i, 1, Length@end, 1\}];
```

```
```
```
```
```
vectorsCALC = Table[near = Nearest[start, end[[i]]][[1]]; start = Drop[start, Position[start, near][[1]]]; vector = N[(near - end[[i]])], {i, 1, Length@end, 1}];

In[152]:= Manipulate[ListPointPlot3D[
    Table[end[[i]] + vectors[[i]] * j, {i, 1, Length@end}, {j, 0, 101}][[All, z]],
    PlotRange -> {{-100, 300}, {-100, 300}, {-2, 300}},
    {z, 1, 102, 1}, ContinuousAction -> False]
```

Speed of Vectors
Sqrt[#[[1]]^2 + #[[2]]^2 + #[[3]]^2] & /@ vectors

{2.00096, 2.61002, 2.43101, 2.53069, 2.60097, 2.00095, 2.38964, 2.32756, 2.17998, 2.34482, 2.20564, 2.56217, 2.51818, 2.5520, 2.56757, 2.57589, 2.4145, 2.57571, 2.52183, 2.52969, 2.48402, 2.33354, 2.30032, 2.48398, 2.4656, 2.55472, 2.3627, 2.32883, 2.54838, 2.09759, 2.42272, 2.43648, 2.56538, 2.5433, 2.57145, 2.56625, 2.5916, 2.1802, 2.55249, 2.41212, 2.53157, 2.44744, 2.39223, 2.37931, 2.02867, 2.47304, 2.40332, 2.26499, 2.1342, 2.3884, 2.50305, 2.28719, 2.2795, 2.54614, 2.31104, 2.56488, 2.50208, 2.42558, 2.5048, 2.52405, 2.48365, 2.51808, 2.43848, 2.51747, 2.49094, 2.19475, 2.20151, 2.03922, 2.11546, 2.3286, 2.42197, 2.33657, 2.48354, 2.41253, 2.54838, 2.57566, 2.57322, 2.58005, 2.43855, 2.43674, 2.41807, 2.46201, 2.38857, 2.36484, 2.35621, 2.30194, 2.55072, 2.59655, 2.47697, 2.35842, 2.348, 2.55626, 2.47507, 2.38551, 2.23832, 2.21327, 2.03798, 2.22812, 2.12763, 2.48851, 2.53734, 2.22595, 2.54879, 2.49286, 2.39531, 2.38339, 2.31355, 2.28219, 2.04008, 2.38276, 2.56692, 2.43705, 2.35611, 2.12282, 2.50247, 2.46909, 2.44792, 2.37562, 2.37252, 2.13221, 2.1205, 2.05492, 2.11167, 2.45057, 2.57968, 2.32079, 2.40843, 2.43134, 2.44219, 2.32439, 2.4126, 2.43177, 2.5748, 2.45325, 2.57518, 2.41369, 2.56435, 2.43424, 2.56345, 2.43174, 2.46565, 2.44074, 2.37466, 2.24682, 2.57761, 2.2459, 2.17775, 2.59161, 2.28586, 2.05882, 2.01827, 2.07376, 2.19481, 2.01149, 2.55118, 2.47749, 2.0758, 2.03005, 2.47807, 2.46099, 2.43626, 2.47245, 2.17098, 2.16547, 2.56092, 2.37836, 2.13824, 2.57315, 2.82217, 2.39435, 2.07171, 2.06221, 2.23766, 2.58541, 2.1145, 2.60379, 2.48788, 2.38198, 2.14235, 2.03009, 2.48468, 2.21326, 2.2082, 2.15764, 2.14491, 2.57052, 2.5973, 2.22077, 2.58723, 2.25889, 2.18585, 2.20601, 2.32866, 2.36256, 2.54572, 2.5157, 2.33815, 2.60031, 2.55822, 2.40196, 2.35808, 2.44436, 2.14542, 2.1278, 2.28835, 2.43194, 2.57444, 2.5755, 2.49056, 2.30033, 2.28007, 2.26721, 2.00922, 2.54181, 2.31156, 2.61082, 2.02493, 2.2907, 2.58822, 2.37508, 2.27874, 2.26289, 2.26679, 2.13565, 2.60281, 2.50081, 2.25242, 2.02804, 2.39591, 2.28967, 2.09783, 2.00637, 2.68315, 2.59541, 2.5937, 2.58675, 2.58322, 2.51464, 2.38245, 2.34543, 2.28965, 2.30239, 2.27447, 2.13009, 2.09598, 2.02407, 2.61279, 2.59376, 2.44509, 2.38602, 2.38577, 2.2486, 2.13129, 2.36006, 2.1035, 2.60815

fitnessFunction[Table[Norm[vectorsCALC[[j]]], {j, 1, Length@vectorsCALC, 1}]]

301.867

SetDirectory@NotebookDirectory[]

C:\Users\student\Documents\Shyam\HIMCM 2017

Export["Start.xls", startG]

Start.xls

Export["End.xls", end]
Drawing and Creating the Light Show

Dragon

```
pointMinMakerScaled2[img_] :=
    Block[{ed, imgDim, pxWid, pxLen, len, wid, pxSize, diff, maxW, maxL, ic},
        ed = ImageCrop[EdgeDetect[img];
        imgDim = ImageDimensions[ed];
        pxWid = imgDim[[1]]; pxLen = imgDim[[2]]; len = 91.44;
        wid = N[pxWid*len];
        pxSize = N[len/pxLen];
        diff = Ceiling[.532/pxSize];
        icTabs = Table[Length[ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"], {i, diff, 10, 1}];
        ic = If[Length[icTabs[[1]]] <= 500, ImageCorners[ed, 2, 0, diff, Method -> "ShiTomasi"],
            ImageCorners[ed, 2, 0, Position[Nearest[icTabs, 500], icTabs][[1, 1]],
            Method -> "ShiTomasi"]];
        maxW = Max[ic[[All, 1]]]; maxL = Max[ic[[All, 2]]];
        scaledIc = {wid, maxW}[[1]]/pxWid, {len, maxL}[[2]]/pxLen} & /@ ic;
    {diff, HighlightImage[ed, ic],
        length@ic,
        ListPlot[ic, PlotMarkers -> (Automatic, Medium)])
```
In[241]:= pointMinMakerScaled2
Out[241]= 

Out[242]=

In[243]:= ListPlot[scaledIc, PlotMarkers -> {Automatic, Medium}]
Out[243]=

Printed by Wolfram Mathematica Student Edition
In[101]:= \text{Mean}[\text{scaledIc}[[\text{All}, 1]]]
Out[101]= 39.7873

In[243]:= end = \text{Prepend}[#, 0] - \{-12.5, \text{Median}[\text{scaledIc}[[\text{All}, 1]]], -200\} \& @ \text{scaledIc};

In[245]:= \text{vectors} = \text{Table}\{\text{near} = \text{Nearest}[\text{start}, \text{end}[[1]], (**, \text{Method} \rightarrow \text{"KDTree"}**)][[1]]; \text{start} = \text{Drop}[\text{start}, \text{Position}[\text{start}, \text{near}][[1]]]; \text{vector} = \text{N}\left[(\text{near} - \text{end}[[1]])/100\right], \{i, 1, \text{Length}@\text{end}, 1\}\};

In[168]:= \text{Length}@\text{start}
Out[168]= 218

In[246]:= \text{startG} = \text{Flatten}\{\text{Table}\{\{i, j, 0\}, \{i, 25\}, \{j, 25\}\}, 1\};

In[247]:= \text{Do}[\text{startG} = \text{Drop}[\text{startG}, \text{Position}[\text{startG}, \text{start}[[1]]][[1]]], \{i, 1, \text{Length}@\text{start}\}]

In[177]:= \text{ListPointPlot3D}[\text{startG}, \text{ViewPoint} \rightarrow \text{Above}]

\begin{figure}
\centering
\includegraphics[width=\textwidth]{plot.png}
\caption{3D plot of startG.}
\end{figure}

In[289]:= \text{Length}@\text{startG}
Out[289]= 305

In[218]:= \text{randSamp} = \text{Table}[\text{RandomSample}[\text{startG}], \{i, 100\}];
AbsoluteTiming@Do[gen1 = Table[fitnessFunction[Table[Norm[randSamp[[numb, j]] - end[[j]]], {j, 1, Length[end]}]], {numb, 1, Length[randSamp, 1]}];
goods = Table[Position[gen1, RankedMin[gen1, i]][[1, 1]], {i, 1, 10, 1}];
gennex = randSamp[[goods]]; orandsamp = randSamp;
randsamp = RandomSample[Join[gennex, Table[RandomSample[startG], {i, 90}]]], {i, 1, 100}]
{105.428, Null}

vectors2 = Table[Normalize[(orandsamp[[goods[[1]], i]] - end[[i]])]*3, {i, 1, Length[orandsamp[[goods[[1]]]], 1]}];
distances = Table[Abs[Abs[orandsamp[[goods[[1]], i]] - end[[i]]], {i, 1, Length[orandsamp[[goods[[1]]]], 1]}];

Max[distances]
291.44

N@291.44/3
97.1467

Min[distances]
0.0032
In[191]:= Norm/@vectors
Out[191]= {2.24153, 2.11371, 2.00396, 2.43876, 2.61751, 2.42746, 2.47201, 2.27145, 2.3436, 2.19498, 2.28922, 2.37758, 2.39835, 2.10513, 2.37683, 2.25168, 2.23832, 2.15917, 2.3851, 2.33932, 2.1173, 2.4286, 2.51741, 2.09138, 2.04679, 2.56829, 2.3252, 2.10513, 2.27159, 2.19498, 2.28922, 2.37758, 2.39835, 2.10513, 2.37683, 2.25168, 2.23832, 2.15917, 2.3851, 2.33932, 2.1173, 2.4286, 2.51741, 2.09138, 2.04679, 2.56829, 2.3252, 2.10513, 2.27159, 2.19498, 2.28922, 2.37758, 2.39835, 2.10513, 2.37683}
In[250]:=
ListPointPlot3D[end, ViewPoint \[Rule] \{1.2, -1, .5\}]

Out[250]=

In[103]:=
Manipulate[
ListPointPlot3D[Table[
  If[pt = end[[i]] + vectors[[i]] \[Times] t;
   pt[[1]] \[LessEqual] 0, orandsamp[[goods[[1]], i]], pt]
 , {i, 1, Length@end}, (t, 0, 102)][[All, z]],
PlotRange \[Rule] \{\{-100, 100\}, \{-100, 100\}, \{-2, 500\}\}],
{z, 1, 102, 1}, ContinuousAction \[Rule] False
]

In[251]:=
Length@end

Out[251]= 554

In[252]:=
endDragon = end;

In[253]:=
Length@endDragon

Out[253]= 554
Ferris Wheel

```mathematica
pointMinMakerScaled2[img_] :=
  Block[{ed, imgDim, pxWid, pxLen, len, wid, pxSize, diff, maxW, maxL, ic},
    ed = ImageCrop[EdgeDetect[img]];
    imgDim = ImageDimensions[ed];
    pxWid = imgDim[[1]];
    pxLen = imgDim[[2]];
    len = 91.44;
    wid = N[pxWid * len];
    pxLen
    pxSize = N[len / pxLen];
    diff = Ceiling[.532 pxSize];
    icTabs = Table[Length@ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"], {i, diff, 10, 1}];
    ic = If[Length@icTabs[[1]] <= 500, ImageCorners[ed, 2, 0, diff, Method -> "ShiTomasi"],
      ImageCorners[ed, 2, 0, Position[Nearest[icTabs, 500], icTabs][[1, 1]],
        Method -> "ShiTomasi"],
        Max@ic[[All, 1]], Max@ic[[All, 2]]];
    maxW = Max@ic[[1]]];
    maxL = Max@ic[[2]]];
    scaledIc = {wid, len} &@ic;
    fc = FindClusters[scaledIc]; ListPlot[fc]
```
In[255]:= ListPlot[scaledIc, PlotMarkers -> {Automatic, Small}]

Out[255]= 

\[8\]

In[259]:= end = Prepend[#, 0] - {12.5, Median[scaledIc[[All, 1]]], -200} & /@ scaledIc;

In[258]:= start = endDragon;

In[257]:= startG = endDragon;

In[260]:= vectors = Table[near = Nearest[start, end[[i]] (*, Method -> "KDTree" *)][[1]];  
   start = Drop[start, Position[start, near][[1]]];  
   vector = N[(near - end[[i]])/100], {i, 1, Length@end, 1}];

In[261]:= Length@start
Out[261]= 249

In[262]:= startG = endDragon;

In[263]:= Do[startG = Drop[startG, Position[startG, start[[i]]][[1]]], {i, 1, Length@start}]

In[264]:= Length@startG
Out[264]= 305
In[265]:= ListPointPlot3D[startG]

Out[265]=

In[266]:= ListPointPlot3D[end]

Out[266]=

endFerris = Join[end]

In[278]:= Length[end]

Out[278]= 385

In[285]:= Length[start]

Out[285]= 249

In[287]:= finalFerris = Join[end, start];
In[286]:= ListPointPlot3D[start]

In[272]:= randSamp = Table[RandomSample[startG], {i, 100}];

In[274]:= AbsoluteTiming[Do[
   gen1 = Table[fitnessFunction[Table[Norm[randSamp[[numb, j]] - end[[j]]], {j, 1, Length@end}]],
   {numb, 1, Length@randSamp, 1}];
   goods = Table[Position[gen1, RankedMin[gen1, i]][[1, 1]], {i, 1, 10, 1}];
   gennex = randSamp[[goods]]; orandsamp = randSamp;
   randSamp = RandomSample[Join[gennex, Table[RandomSample[startG], {i, 90}]]];,
   {i, 1, 100}];

Out[274]= {184.418, Null}

vectors = Table[Normalize[orandsamp[[goods[[1]], i]] - end[[i]]]] * 3,
   {i, 1, Length@orandsamp[[goods[[1]], 1]]}, 1];

In[275]:= vectors3 = Table[Abs[Abs@orandsamp[[goods[[1]], i]] - Abs@end[[i]]],
   {i, 1, Length@orandsamp[[goods[[1]], 1]]}, 1];

In[282]:= Max@vectors3
Out[282]= 76.0754

In[283]:= N@%/3
Out[283]= 25.3585

In[277]:= Min@vectors3
Out[277]= 0.
In[103]:= Manipulate[ListPointPlot3D[Table[
  If[pt = end[[i]] + vectors[[i]] * t;
  pt[[1]] ≤ 0, orandsamp[goods[[i]], i]], pt]
  , {i, 1, Length@end}, {t, 0, 102}][[All, z]],
PlotRange -> {{-100, 100}, {-100, 100}, {-2, 500}},
{z, 1, 102, 1}, ContinuousAction -> False]

Logo

In[268]:= pointMinMakerScaled2[img_] :=
  Block[{ed, imgDim, pxWid, pxLen, len, pxSize, diff, maxW, maxL, ic},
  ed = ImageCrop@EdgeDetect[img];
  imgDim = ImageDimensions[ed];
  pxWid = imgDim[[1]]; pxLen = imgDim[[2]];
  len = 91.44;
  wid = N[pxWid + len]; pxlen
  pxSize = N[len/pxLen];
  diff = Ceiling[.532/pxSize];
  icTabs = Table[Length@ImageCorners[ed, 2, 0, i, Method -> "ShiTomasi"], {i, diff, 10, 1}];
  ic = If[Length@icTabs[[1]] <= 500, ImageCorners[ed, 2, 0, diff, Method -> "ShiTomasi"],
  ImageCorners[ed, 2, 0, Position[Nearest[icTabs, 500, icTabs][[1, 1]],
  Method -> "ShiTomasi"]],
  maxW = Max@ic[[All, 1]]; maxL = Max@ic[[All, 2]];
  scaledIc = {{N[pxWid + #[[1]]], N[len + #[[2]]]} &@ic;
  {diff, HighlightImage[ed, ic],
  Length@ic, ListPlot[ic, PlotMarkers -> {Automatic, Medium}]}]

Out[269]= pointMinMakerScaled2[8475]
```
In[271]:=
ListPlot[scaledIc, PlotMarkers -> {Automatic, Medium}]

Out[271]=

In[298]:=
end = Prepend[#1, 0] - {\(-12.5, \text{Median}[\text{scaledIc}[[\text{All, 1}]]], -200\)} \& scaledIc[[;; -7]]; 

In[299]:=
start = finalFerris;

In[300]:=
vectors = Table[near = Nearest[start, end[[i]][1, Method -> "KDTree"]][[1]]; 
   start = Drop[start, Position[start, near][[1]]]; 
   vector = N[(near - end[[i]])/100, {i, 1, Length@end, 1}];

In[301]:=
startG = finalFerris;

In[306]:=
ListPointPlot3D[startG, ViewPoint -> \{1.2, -1, .5\}]
```
In[315]:= ListPointPlot3D[end, ViewPoint -> {1.2, -1, .5}]

Out[315]=

In[307]:= Do[startG = Drop[startG, Position[startG, start][[1]]][[1]], {i, 1, Length@start}]

In[308]:= randSamp = Table[RandomSample[startG], {i, 100}];

In[310]:= AbsoluteTiming@Do[gen1 = Table[fitnessFunction[Table[Norm[randSamp[[nump, j]] - end[[j]]], {j, 1, Length@end}]], {nump, 1, Length@randSamp, 1}];
goods = Table[Position[gen1, RankedMin[gen1, i]][[1, 1]], {i, 1, 10, 1}];
gennex = randSamp[[goods]]; orandsamp = randSamp;
randSamp = RandomSample[Join[gennex, Table[RandomSample[startG], {i, 90}]]], {i, 1, 100}];

Out[310]= {618.219, Null}

In[97]:= vectors = Table[Normalize[orandsamp[[goods[[1]], i]] - end[[1]]] * 3, {i, 1, Length@orandsamp[[goods[[1]], 1]]}, 1];

In[311]:= vectors4 = Table[Abs[Abs@orandsamp[[goods[[1]], i]] - Abs@end[[1]]], {i, 1, Length@orandsamp[[goods[[1]], 1]]}, 1];

In[312]:= Max@vectors4

Out[312]= 82.9145

In[99]:= N@% / 3

Out[99]= 27.6382

In[103]:= Manipulate[ListPointPlot3D[Table[
   If[pt = end[[i]] + vectors[[1]] * t;
   pt1 = 0, orandsamp[[goods[[1]], i]], pt]
   , {i, 1, Length@end}, {t, 0, 102}][[All, z]],
   PlotRange -> {{-100, 100}, {-100, 100}, {-2, 500}}],
   {z, 1, 102, 1}, ContinuousAction -> False]
In[317]:= Length@start
Out[317]= 625

In[318]:= vectors5 = Table[Abs[Abs@end[[i]] - Abs@start[[i]]], {i, 1, Length@end, 1}];

In[319]:= Max@vectors5
Out[319]= 291.44
Implementation and Testing of the Genetic Algorithm

```
In[7]:= startG = Flatten[Table[{i, j, 0}, {i, 25}, {j, 25}], 1];
   start[[1]]
   {5, 18, 0}
   Position[startG, start[[1]]]
   {{118}}

In[12]:= Do[startG = Drop[startG, Position[startG, start[[i]][[1]]][[1]]], {i, 1, Length@start}]

In[13]:= Length@startG

Out[13]= 278

In[8]:= Clear[fitnessFunction];
   fitnessFunction[x_] := Variance[x];

In[14]:= randSamp = Table[RandomSample[startG, {i, 100}]];

In[9]:= gen1 = Table[fitnessFunction[Table[Norm[randSamp[[numb, j]] - end[[j]]], {j, 1, Length@end}]],
   {numb, 1, Length@randSamp, 1}];
   goods = Table[Position[gen1, RankedMin[gen1, i]][[1, 1]], {i, 1, 10, 1}]
   {20, 66, 64, 43, 70, 52, 72, 24, 9, 58}
   gennex[[1]] = randSamp[[20]]
   True
   fitnessFunction[Table[Norm[randSamp[[20, j]] - end[[j]]], {j, 1, Length@end}]]
   301.424
   gennex = randSamp[[goods]];
   randSamp = Join[gennex, Table[RandomSample[startG], {i, 90}]];

In[15]:= randSamp = Table[RandomSample[startG], {i, 100}];
```

Printed by Wolfram Mathematica Student Edition
In[16]:=
AbsoluteTiming@
Do[
gen1 = 
Table[
fitnessFunction[
Table[
Norm[randSamp[[numb, j]] - end[[j]]],
{j, 1, Length@end}]],
{numb, 1, Length@randSamp, 1}];

goods = Table[Position[gen1, RankedMin[gen1, i]][[1, 1]], {i, 1, 10, 1}];

gennex = randSamp[[goods]]; orandsamp = randSamp;
randSamp = RandomSample[Join[gennex, Table[RandomSample[startG], {i, 90}]]],
{i, 1, 1000}]
Out[16]= {478.428, Null}

Create Best Graph

In[30]:=
best = {};

In[37]:=
AbsoluteTiming@
Do[
gen1 = 
Table[
fitnessFunction[
Table[
Norm[randSamp[[numb, j]] - end[[j]]],
{j, 1, Length@end}]],
{numb, 1, Length@randSamp, 1}];

goods = Table[Position[gen1, RankedMin[gen1, i]][[1, 1]], {i, 1, 10, 1}];

gennex = randSamp[[goods]]; orandsamp = randSamp; AppendTo[best, fitnessFunction[
Table[Norm[orandsamp[[goods[[1]], j]] - end[[j]]], {j, 1, Length@end}]]];
randSamp = RandomSample[Join[gennex, Table[RandomSample[startG], {i, 90}]]], {i, 1, 1000}]
Out[37]= {533.554, Null}

In[17]:=
goods
Out[17]= {36, 76, 9, 45, 44, 94, 17, 91, 53, 83}

In[18]:=
fitnessFunction[Table[Norm[orandsamp[[36, j]] - end[[j]]], {j, 1, Length@end}]]
Out[18]= 298.769
In[19]:= orandsamp[[36]]
Out[19]= {{6, 4, 0}, {1, 4, 0}, {13, 4, 0}, {4, 3, 0}, {3, 9, 0}, {8, 25, 0}, {18, 3, 0}, {1, 22, 0}, 
{3, 17, 0}, {15, 5, 0}, {5, 7, 0}, {1, 23, 0}, {18, 5, 0}, {17, 2, 0}, {10, 10, 0}, 
{1, 25, 0}, {4, 18, 0}, {17, 5, 0}, {13, 9, 0}, {1, 9, 0}, {11, 9, 0}, {4, 20, 0}, {6, 13, 0}, 
{4, 17, 0}, {8, 7, 0}, {2, 20, 0}, {14, 4, 0}, {1, 19, 0}, {12, 1, 0}, {6, 14, 0}, 
{11, 1, 0}, {5, 11, 0}, {10, 8, 0}, {5, 1, 0}, {4, 16, 0}, {17, 1, 0}, {2, 15, 0}, 
{12, 2, 0}, {5, 10, 0}, {17, 6, 0}, {8, 3, 0}, {12, 5, 0}, {1, 6, 0}, {11, 10, 0}, 
{8, 24, 0}, {1, 12, 0}, {6, 23, 0}, {16, 6, 0}, {23, 1, 0}, {14, 2, 0}, {3, 16, 0}, 
{1, 1, 0}, {4, 5, 0}, {18, 2, 0}, {9, 11, 0}, {4, 10, 0}, {13, 2, 0}, {8, 10, 0}, {6, 9, 0}, 
{4, 23, 0}, {8, 13, 0}, {2, 16, 0}, {2, 12, 0}, {10, 25, 0}, {11, 4, 0}, {2, 18, 0}, 
{2, 11, 0}, {10, 4, 0}, {10, 12, 0}, {5, 21, 0}, {12, 9, 0}, {7, 24, 0}, {18, 1, 0}, 
{3, 23, 0}, {5, 4, 0}, {1, 5, 0}, {20, 2, 0}, {6, 2, 0}, {11, 6, 0}, {7, 6, 0}, {1, 18, 0}, 
{5, 6, 0}, {13, 1, 0}, {3, 15, 0}, {10, 3, 0}, {20, 3, 0}, {19, 3, 0}, {3, 20, 0}, 
{8, 11, 0}, {1, 13, 0}, {2, 10, 0}, {16, 4, 0}, {6, 6, 0}, {6, 7, 0}, {5, 15, 0}, {14, 1, 0}, 
{4, 9, 0}, {16, 1, 0}, {16, 3, 0}, {6, 5, 0}, {2, 17, 0}, {7, 10, 0}, {2, 8, 0}, {1, 11, 0}, 
{5, 13, 0}, {19, 1, 0}, {4, 11, 0}, {13, 8, 0}, {3, 5, 0}, {3, 2, 0}, {6, 11, 0}, {9, 12, 0}, 
{1, 3, 0}, {5, 16, 0}, {4, 4, 0}, {1, 10, 0}, {12, 6, 0}, {14, 7, 0}, {5, 25, 0}, 
{6, 22, 0}, {8, 1, 0}, {3, 8, 0}, {18, 4, 0}, {3, 18, 0}, {3, 12, 0}, {3, 25, 0}, 
{6, 24, 0}, {3, 4, 0}, {1, 8, 0}, {10, 5, 0}, {3, 21, 0}, {11, 2, 0}, {4, 2, 0}, {7, 15, 0}, 
{6, 25, 0}, {14, 8, 0}, {1, 7, 0}, {9, 4, 0}, {20, 4, 0}, {10, 6, 0}, {2, 2, 0}, {2, 3, 0}, 
{9, 9, 0}, {8, 6, 0}, {13, 6, 0}, {13, 3, 0}, {22, 2, 0}, {13, 5, 0}, {4, 19, 0}, 
{2, 13, 0}, {7, 12, 0}, {8, 14, 0}, {2, 23, 0}, {15, 2, 0}, {4, 15, 0}, {2, 9, 0}, 
{2, 4, 0}, {21, 2, 0}, {4, 8, 0}, {3, 7, 0}, {8, 12, 0}, {4, 12, 0}, {6, 10, 0}, {7, 2, 0}, 
{3, 19, 0}, {9, 13, 0}, {15, 6, 0}, {5, 24, 0}, {9, 10, 0}, {1, 17, 0}, {8, 2, 0}, 
{7, 11, 0}, {2, 19, 0}, {2, 24, 0}, {14, 6, 0}, {8, 9, 0}, {4, 25, 0}, {3, 13, 0}, 
{7, 23, 0}, {11, 3, 0}, {12, 4, 0}, {11, 5, 0}, {12, 8, 0}, {8, 8, 0}, {4, 21, 0}, {6, 1, 0}, 
{7, 4, 0}, {11, 7, 0}, {3, 22, 0}, {15, 7, 0}, {7, 14, 0}, {7, 3, 0}, {12, 10, 0}, 
{2, 22, 0}, {4, 22, 0}, {9, 7, 0}, {4, 24, 0}, {3, 1, 0}, {2, 21, 0}, {3, 3, 0}, {7, 13, 0}, 
{3, 24, 0}, {1, 2, 0}, {22, 1, 0}, {5, 12, 0}, {2, 7, 0}, {2, 6, 0}, {9, 3, 0}, {14, 3, 0}, 
{10, 11, 0}, {9, 1, 0}, {6, 15, 0}, {2, 25, 0}, {1, 21, 0}, {7, 9, 0}, {5, 23, 0}, 
{12, 7, 0}, {5, 14, 0}, {5, 17, 0}, {8, 5, 0}, {4, 13, 0}, {6, 8, 0}, {17, 3, 0}, {16, 2, 0}, 
{4, 6, 0}, {9, 2, 0}, {15, 1, 0}, {7, 7, 0}, {10, 2, 0}, {5, 8, 0}, {2, 5, 0}, {14, 5, 0}, 
{20, 1, 0}, {6, 12, 0}, {1, 16, 0}, {10, 9, 0}, {10, 7, 0}, {4, 1, 0}, {21, 1, 0}, 
{2, 14, 0}, {3, 10, 0}, {6, 3, 0}, {9, 25, 0}, {9, 8, 0}, {4, 14, 0}, {16, 5, 0}, 
{1, 24, 0}, {5, 3, 0}, {11, 8, 0}, {15, 3, 0}, {19, 4, 0}, {7, 8, 0}, {4, 7, 0}, {16, 7, 0}, 
{5, 2, 0}, {8, 4, 0}, {5, 9, 0}, {17, 4, 0}, {1, 14, 0}, {1, 20, 0}, {13, 7, 0}, {12, 3, 0}, 
{9, 5, 0}, {3, 14, 0}, {15, 4, 0}, {5, 5, 0}, {1, 15, 0}, {7, 25, 0}, {19, 2, 0}, {2, 1, 0}, 
{7, 5, 0}, {9, 6, 0}, {5, 22, 0}, {7, 1, 0}, {21, 3, 0}, {3, 6, 0}, {3, 11, 0}, {10, 1, 0}]

Animation Only

def = Prepend[#, 0] & /@ scaledIc;

vectors = Table[
    Normalize[(orandsamp[[34, i]] - end)[[1]]] + 3, {i, 1, Length@def, 1}];
Manipulate[ListPointPlot3D[
  Table[end[[i]] + vectors[[i]] * t, {i, 1, Length@end}],
  PlotRange -> {(-100, 100), (-100, 100), (-2, 300)},
  {z, 1, 102}, ContinuousAction -> False
]
Part: Part 373 of

\[
\begin{align*}
(0.135, 0.000255145, -2.61), & (0.135, -0.00974486, -2.0004), (0.135, 0.177803, -2.0004), (0.145, 0.392463, -2.155), (0.135, 0.195012, -2.61), (0.135, 0.00224546, -2.61), (0.155, 0.177803, -2.61), \langle 37 \rangle, (0.175, 0.189793, -2.16061), (0.155, 0.181759, -2.16061), (0.145, 0.005522, -2.16061), (0.145, 0.00748809, -2.16061), (0.135, 0.0494542, -2.16061), (0.135, 0.0314203, -2.16061), \langle 322 \rangle \end{align*}
\]

\( \) does not exist.

General: Further output of Part::partw will be suppressed during this calculation.

SetDirectory@NotebookDirectory[]

C:\Users\student\Documents\Shyam\HIMCM 2017

Export["OStart.xls", orandsamp[[34]]]

OStart.xls

\[87\] := ListPointPlot3D[orandsamp[[36]], ViewPoint \rightarrow Above]