METRICS ON CROWD CONTROL
WITH OVERHEAD VIDEO AND VOCAL COMMANDS

A Thesis

Presented to

the Faculty of the Department of Electrical and Computer Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

By

Wei Yao

May 2016

METRICS ON CROWD CONTROL WITH OVERHEAD VIDEO AND VOCAL COMMANDS

Wei Yao

Approved:

Chair of the Committee
Aaron T. Becker, Assistant Professor
Electrical Engineering

Committee Members:

Zhu Han, Professor
Computer Engineering

Yan Yao, Assistant Professor
Electrical Engineering

Suresh K. Khator, Associate Dean
Cullen College of Engineering

Badri Roysam, Department Chair
Electrical and Computer Engineering

METRICS ON CROWD CONTROL

WITH OVERHEAD VIDEO AND VOCAL COMMANDS

An Abstract of a Thesis

Presented to

the Faculty of the Department of Electrical and Computer Engineering

University of Houston

In Partial Fulfillment

Of the Requirements for the Degree of

Master of Science

in Electrical and Computer Engineering

By

Wei Yao

May 2016

ABSTRACT

This thesis presents an agent-tracking framework for semi-structured, crowded video. This framework is used to investigate how large numbers of people respond to vocal commands with local feedback and an overhead camera video. We analyze a video showing an overhead view of more than 200 people, each holding an umbrella equipped with red, blue, and green LED lights. The crowd's motion under the vocal command formed a variety of patterns. We use K-means clustering to separate umbrella from each other. Kalman filtering is used to estimate how each umbrella moves and track their motion path. In particular, we present results on: (1.) Automatic segmentation and classification of each umbrella. (2) Swarm's response time to a simple command. (3) Time constant for a harder command. (4) Comparing accuracy. (5) "Shape-matching" ability (6) Documenting the position memory. (7) Distribution consensus simulation.

Keywords—K-means clustering, vision tracking, Kalman filter

CONTENTS

ABSTRACTS VIII

LIST OF FIGURES VIII

LIST OF TABLES XII

CHAPTER 1 INTRODUCTION.. 13

CHAPTER 2 EXPERIMENTS AND RESULTS ....**Error! Bookmark not defined.**

2.1 Verify each umbrella and collect data information 17

2.2 Tracking umbrellas' motion path 19

2.3 Data analysis 21

2.3.1 Crowd response to a command to change color............. 21

2.3.2 The time constant for a harder vocal command 23

2.3.3 Learning rate of the human swarm 24

2.3.4 Comparing accuracy between similar commands.......... 25

2.3.5 Shape-matching abilities 27

2.3.6 Forming a human swarm into a "snake"........................ 28

2.3.7 Accuracy of the "Bullseye" configuration.................... 30

2.3.8 The accuracy of the human swarm's memory 33

CHAPTER 3 SIMULATIONS.. 35

CHAPTER 4 CONCLUSIONS... 42

CHAPTER 5 APPENDIX.. 43

APPENDIX A: K-means algorithm support....................................43

APPENDIX B: Kalman filter algorithm support.............................44

APPENDIX C: Other equations applied...46

APPENDIX D: Codes.. 46

REFERENCES.. 68

LIST OF FIGURES

Figure 1.1 Umbrella equipped with A: blue, B: green, and C: red LED lights.......... 13

Figure 1.2 A sketch showing how the overview of scene looks like 14

Figure 1.3 This thesis analyzes an overhead video showing illuminated umbrellas. (1) Raw data, captured from overhead video. (2) Classified umbrellas in the processed image. (3) Umbrella color count as a function of time is one form of data that is generated 14

Figure 2.1 Centroids overhead on raw video data umbrellas' position data was first collected and stored, then it can be used in a Kalman Filter to track the motion paths. 18

Figure 2.2 How "assignment" and "update" steps help to find out each umbrella's center. 20

Figure 2.3 Applying Kalman filter to tracking umbrellas. Two umbrellas are circled in red 20

Figure 2.4 Change of the two umbrellas' shown in Fig.3 position and color as a function of time. 20

Figure 2.5 How crowd response to vocal command such as "I want everybody to turn them red." 22

Figure 2.6 Six aligned "color change" vocal commands reprsent human swarm's performance 23

Figure 2.7 For color-change vocal command such as "I want everybody to turn them red," "turn the red off turn the blue on" or "let's go to green" the swarm respond time tends to reduce, demonstrating that the swarm is learning . 24

Figure 2.8 Response time for command “when I say go I want you to turn them on and I want this whole group, this group that's gathered tonight to be one color but I'm not going to tell you what color that is” 25
Figure 2.9 Comparing the accuracy of three command “If you are red move”, “Let’s try that with green go!” and “When I say go I want red to freeze and the blues to move” In every 20 frames, if the umbrella’s new position is more than a quarter of its radius, this umbrella is moving............................... 30
Figure 2.10 Calculating the circularity of the human swarm when commanded “Red stop and bunch up, See how round you can be, keep circling around them greens” There are three circles with different colors, values closer to one means the swarm shape is closer to a true circle. 27
Figure 2.11 A definition of “snake”, a “snake” should consisted of at least 5 umbrellas sequencely, distance between successive umbrellas is less than four umbrella radius. 28
Figure 2.12 Time required to form snakes, y-axis shows how long the snake is, and the number of umbrellas are in (or not in) each snake............................ 34
Figure 2.13 Circularity of each circle in a bullseye given the command, “I would like to see three stripes, you know, like in the middle one color”.................. 30
Figure 2.14 Best fit circles for human swarm’s performance.................................. 31
Figure 2.15 Another equation applied to evaluate human swarm’s performance. 32
Figure 2.16 The distance between each two circles to evaluate the quality of bullseye as a function of time... 33
Figure 2.17 Accuracy of a swarm’s position memory. It compares the mean distance between everyone’s “original position” and the “returned position”.

Smaller distances indicate better memory. When they heard the vocal command “When I say go I want you to move”..................................... 33

Figure 3.1 200 nodes randomly initialized in a 2D space, each point selected a color from red, green, blue randomly. Each point turn color based on 7 nearest points... 36

Figure 3.2 All nodes choose one same color finally, takes 205 iterations.................. 36

Figure 3.3 200 nodes randomly initialized in a 2D space, each point selected a color from red, green, blue randomly. Each point turn color based on 9 nearest points... 37

Figure 3.4 All nodes choose one same color finally, takes 643 iterations 37

Figure 3.5 200 nodes randomly initialized in a 2D space, each point selected a color from red, green, blue randomly. Each point turn color based on 11 nearest points... 38

Figure 3.6 All nodes choose one same color finally, takes 95 iterations 38

Figure 3.7 Result of 7 nearest neighbors run simulation 100 times, plot ratio of major colors as a function of iteration times, also plot the “mean”, “mean + std”, “mean – std.” 39

Figure 3.8 Result of 9 nearest neighbors run simulation 100 times, plot ratio of major colors as a function of iteration times, also plot the “mean”, “mean + std”, “mean – std.” 40

Figure 3.9 Result of 11 nearest neighbors run simulation 100 times, plot ratio of major colors as a function of iteration times, also plot the “mean”, “mean + std”, “mean – std.” 40

Figure 3.10 With k = 7, 9, and 11, mean plots after 100 times simulations............... 41

Figure A.1 Constituent parts of Kalman Filter, how time update and measurement update work together.. 45

LIST OF TABLES

Table 1 Trial number and specific accuracy. 26

CHAPTER 1

Introduction

This thesis analyzes data from a crowd controlled by vocal commands and presents a method to track the motion paths of multiple agents in semi-structured crowded scenes.

The analysis uses video data from *UP: The Umbrella Project* [1], a beautiful experiment conducted at night on a football field in which more than two hundred people were each given an instrumented umbrella equipped with an RGB LED as shown in figure 1.1.

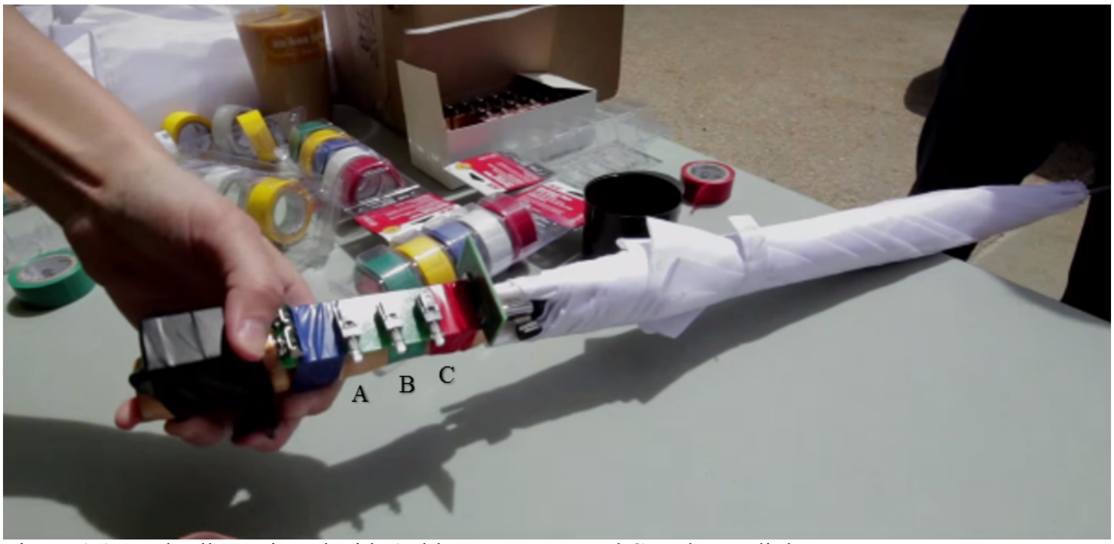


Figure 1.1: Umbrella equipped with A: blue, B: green, and C: red LED lights

Using vocal commands from a director on an elevated platform, and an overhead camera view projected on a large screen, the participants were divided into several groups according to their major, gender, or grade and then directed to form different shapes in various colors. The solution for vision tracking is coded in MATLAB, and is available at [2].

In figure 1.2 is a sketch showing the overview of scene.

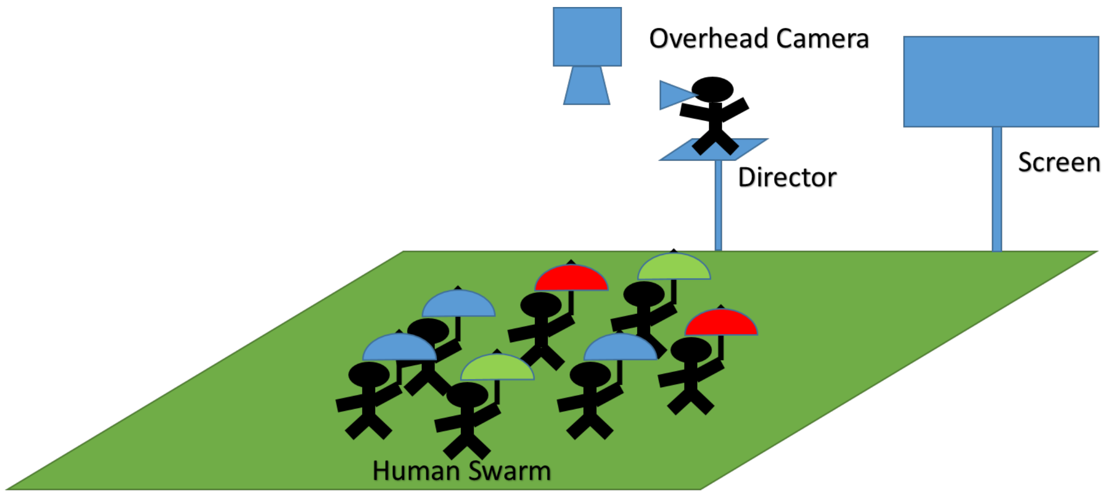


Figure 1.2:A sketch showing an overview of the scene.

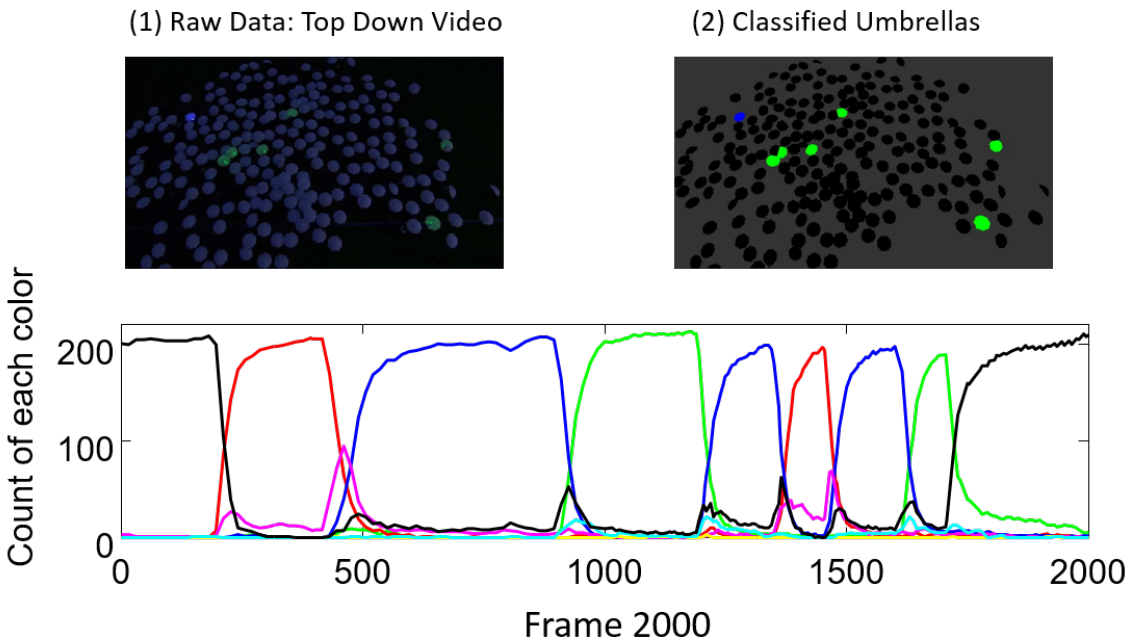


Figure 2.3: This thesis analyzes an overhead video showing illuminated umbrellas. (1) Raw data, captured from overhead video. (2) Classified umbrellas in the processed image. (3) Umbrella color count as a function of time is one form of data that is generated

In a semi-structured crowded scene, the motion of the crowd appears to be random, with different participants moving in different directions at different times [3]. This scenario has some structure because it is controlled by one person, the voice giving the vocal commands, but errors cannot be avoided completely. Moreover, tracking is challenging because the umbrellas switch colors rapidly and often overlap. Fig. 1.3 shows a representative screenshot, the results after we processed the raw video, and a plot showing umbrella color counts as a function of time.

Object tracking is a key problem in the field of computer vision, and is especially challenging when tracking multiple objects in unstructured, crowded scenes. Tracking moving objects in video has a variety of applications, including automated surveillance, military guidance, traffic management system, robot vision and artificial intelligence [4].

This original video is available at [5]. Tracking multiple objects is more difficult than tracking one object for several reasons. Data association, the matching between targets and observations, from frame to frame in a video sequence, is one difficulty [6]. Because objects are continually moving, they often overlap partially or completely. Sometimes the objects disappear and occasionally new objects enter the frame. To address these problems, this thesis uses Kalman filters to track multiple objects [7].

The first challenge is to segment individual umbrellas. The solution employed is to erode all components to shrink to points. These points will not overlap and denote the centroid of each object. In this thesis we apply data clustering to verify the centroids of each object. Data clustering is frequently used in many fields, including data mining, pattern recognition, decision support, machine learning and image segmentation [8]. In this thesis we adapt one of the most widely used formulations to solve this problem, the K-means clustering algorithm. Given a set of n data points in real d-dimensional space, R_d, and an integer k, the problem is to determine a set of k points in R_d, called centers, so as to minimize the mean squared distance from each data point to its nearest center [9].

The umbrellas in this project are not moving aimlessly. At one frame, all may be the same color, but later the all umbrellas may change to another color and later form a colorful image. In the video, umbrellas form a smiley face, and later change to a snake, and finally form a word. All these transformation occurred under the direction of a vocal command. This thesis presents an analysis on how the agents response to the vocal command.

This section, describes the approaches used: K-means clustering, Kalman filter algorithm, and techniques to, monitor the transformation of umbrellas' color and pattern.

CHAPTER 2

Experiments and Results

This section describes measurements obtained from *UP: The Umbrella Project.* In this thesis we will discuss lots of experiments during the time we doing research. Mainly related to verifying every umbrella in the crowd, analyzing the data collected based on human swarm, after that we also did simulations for a more complete thesis project, to further prove that the analysis we did is correct, and then results we get can be applied in the future.

2.1 Verify each umbrella and collect data information

The data is a video recorded by an overhead camera showing how umbrellas respond to a vocal command. The first step is to identify the umbrellas, and record their positions However, both the numbers and positions of umbrellas are not a constant, this number changes as umbrellas enter and leave the field of view or lose battery power.

The aim of the *K*-means algorithm is to divide *M* points in *N* dimensions into *K* clusters so that the within-cluster sum of squares is minimized [10]. K-means returns a locally optimal clustering solution, and is dependent on a trustworthy initial value. We use manual identification for the first frame, and use the centroids from the previous frame as initial seed values for each successive frame.

In this research project, we analyzed the raw video every 15th frame to get the data we want. Since all the umbrellas are changing their colors rapidly, moving quickly and they will overlap with each other frequently, so the first step I'm going to do is identify each umbrella from the raw video. And the basic thought here is erode all the components shrink into points, which means each point represents one umbrella respectively. As a solution, we use the K-means tool here. In the first frame, each umbrella's centroid was manually marked and those centroids were used as seeds for the next frame. In the next frame, K-means was used refine the seeds' position to ensure they are in the middle of each umbrella.

This analysis is implemented using MATLAB, code is accessible at [11]. The resulting data, saved as a video, is available at [12].

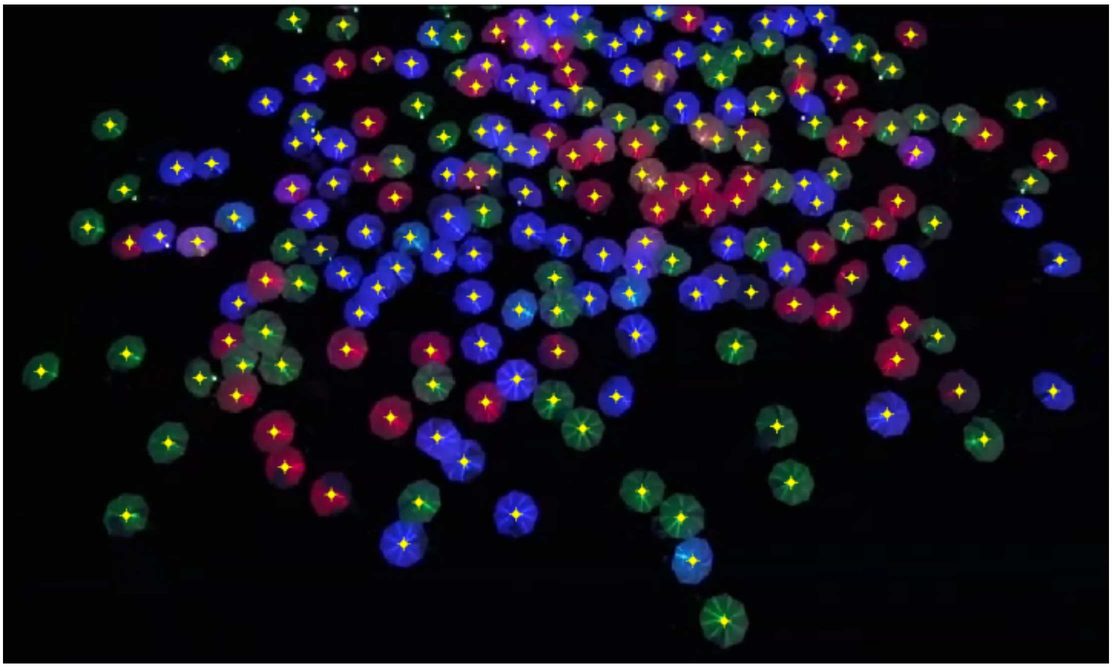
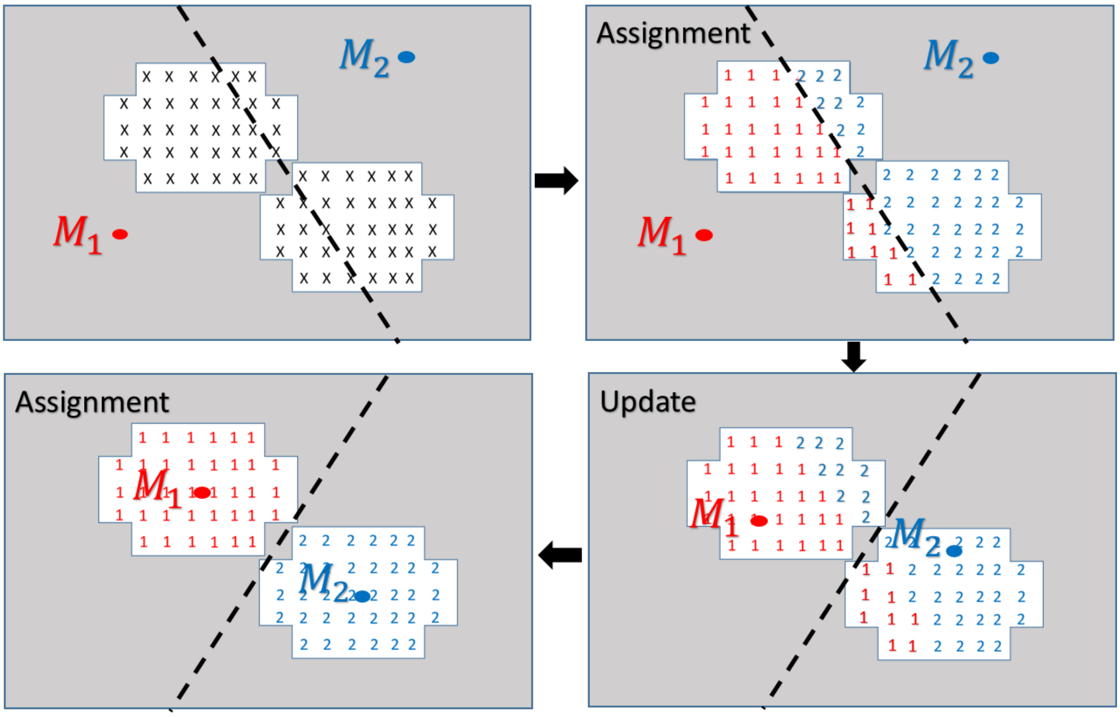


Figure 3.1: Centroids overhead on raw video data umbrellas' position data was first collected and stored, then it can be used in a Kalman Filter to track the motion paths.

As shown in Fig 2.1, each umbrella is marked by at its centroid. The centroid data and the average color of pixels in its neighborhood are stored to a file. The specific algorithm will be showed in appendix section later, to further explain how the K-means been applied here, I give the following figure.

Figure 2.2 shows how K-means algorithm worked in this thesis project in first step to identify each umbrella from the raw video. This algorithm can be divided into two groups, "Assignment" and "Update." As shown in Fig 3, for example, we want to find out where the centers of those two umbrellas are. So we assigned two points M_1 and M_2, which should be the centers of umbrellas, umbrellas here actually are clusters composed with large quantities of pixels. Here all pixels are divided into two groups based on their minimum distance to cluster's center, this is the "Assignment" step, then M_1 and M_2 will move to the center of pixels "1" and "2". After that, all pixels will find the nearest cluster's center to them, and they will be divided again, this is the



"Update" step.

Figure 2.2: How "assignment" and "update" steps help to find out each umbrella's center

So K-means will continue doing iteration between "Assignment" and "Update" till all clusters find their centers, at which point the means will not move any more.

2.2 Tracking umbrellas' motion path

To detect umbrella positions and show how they moved, umbrellas are represented as a set of circles with three parameters, the center of a circle, and the circle radius. The user interface uses two circles to evaluate how the umbrellas move that are green and red. The green circle is the observed position of umbrella and the red circle is the estimated position which is calculated after applying the Kalman Filter, giving the result shown in Fig 2.3, which shows that the two circles overlap well, which indicates the algorithm is working

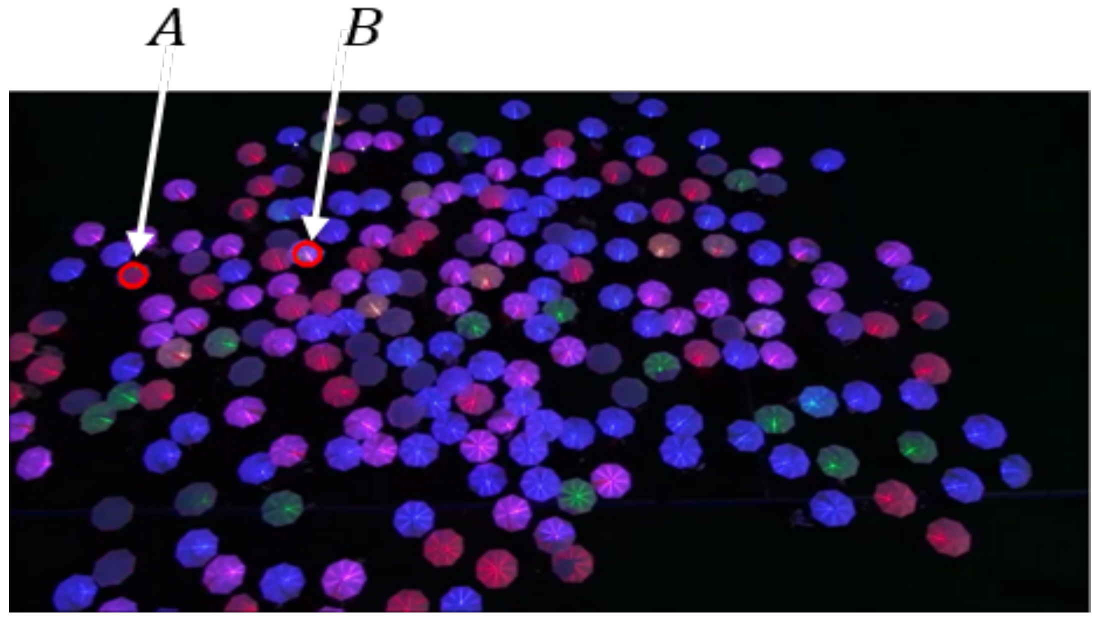


Figure 2.3: Applying Kalman filter to tracking umbrellas. Two umbrellas A and B are circled in red.

Fig 2.4 shows of *x* and *y* position of two umbrellas as a function of time with the lines drawn in the correct colors of the umbrellas, in the plot we show the changement of multiple umbrellas' positions.

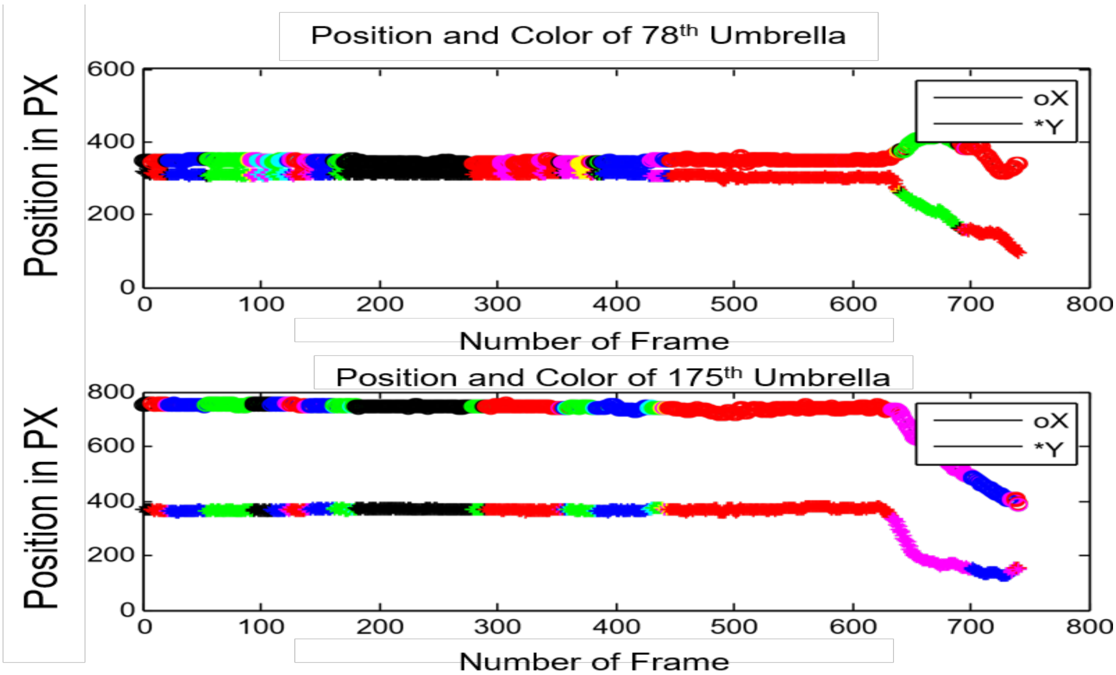


Figure 2.4: Change of the two umbrellas' shown in Fig.2.3 position and color as a function of time

We tracked two umbrellas 78th and 175th from the beginning of the raw video till 800 frames. During the tracking process, we applied Kalman Filter, which will introduce more details in later sections. We got the x and y velocity as a function of time with the lines drawn in the correct colors of the umbrellas too. However, the plot we get showing the changement of umbrellas' velocity is not easy to read, so we chose not show it in this section, and will show it in other forms later. From the plot of x and y position, we used the umbrella's position data, every time the umbrella moves from this frame to another, there will be a displacement distance on both x and y direction, so the x and y position are updated.

Similarly, the x and y velocities are obtained by approximating motion as a straight-line movement between successive frames and dividing the distance travelled by time.

2.2 Data analysis

Finding the motion path of each umbrella is just the first step toward understanding the swarm response. We want to know more details about the crowd, whether they performed well under the vocal command, and whether the swarm learns to better follow the directions of the vocal commands. This part is one of the most interesting experiments in the research.

2.2.1 Crowd response to a command to change color

In the video, there are several vocal commands which required people to change their umbrellas' color. Commands included "I want everybody to turn them red", and "Now turn the red off. Turn the blue on!"

This test analyzes how people responded to eight color change commands. Those vocal commands mentioned above can be seen as a serious of basic vocal commands, in the raw video there are more harder commands, we will show how human swarm did following the vocal command step by step.

Results for this set of vocal commands are summarized in Fig 2.5.

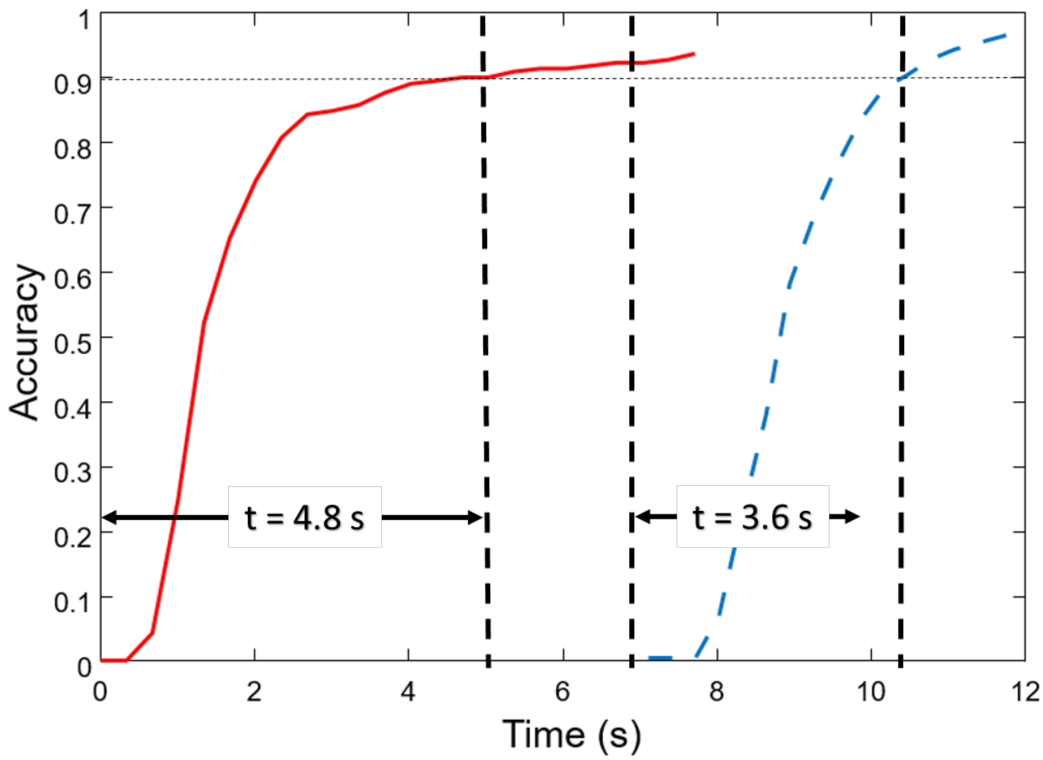
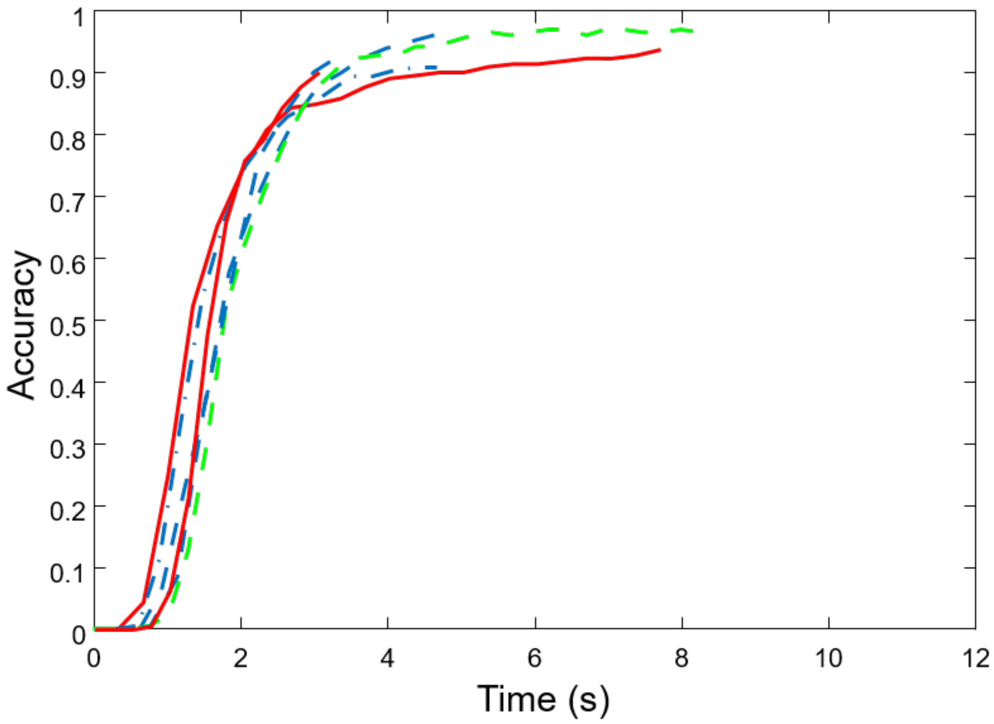


Figure 2.5: How crowd response to vocal commands such as "*I want everybody to turn them red.*"

In the raw video, there are a series of vocal command can be seen as "color-change" commands, in fig 2.5, we comparing the first vocal command as "I want everybody to turn them red" and the command "Now turn them blue." As shown in the figure, for example, after the vocal command *"I want everybody to turn them red,"* people began to switch their umbrellas' color at t=1s, and at t= 5.8s, 90% of the umbrellas' color changed to red. The accuracy here is defined as $Accuracy = \frac{Number\ of\ Major\ Color}{Number\ of\ All\ colors}$. We define the response time as the time of the first response to when more than 90% of the umbrellas' color change. In this case, the time constant is 4.8 seconds. During successive color change commands, it takes less time for 90% of the swarm to turn their umbrellas to one color. For next command as we can see in the figure, everybody was required to turn blue, people begin to change color at t=6.4s and till the moment 90% umbrellas change to blue, it takes 3.6s only, obeviously human swarm is improving their performance in this kind of experiments.

2.2.2 Learning rate of the human swarm

In the video, totally six vocal color-change commands were recorded. All data is aligned so the command begins at t=0s. This means the swarm's performance is



increasing. The result we get is shown in Fig 8

Figure 2.6: Six aligned "color change" vocal commands reprsent human swarm's performance

To further prove our conclusion, we display human swarm's learning rate, with a best fit. The human swarm was asked to change colors six times. Fig 2.7 displays the time required for 90% of the swarm to achieve the desired value.

Fig 2.7 shows the time for accuracy to reach *90%*. One experiment inserted a new vocal command before *90%* of the human swarm achieved the desired color, so this analysis defined a successful convergence as when the ratio of major color reach *80%*.

The overall trend for human swarm is to take less time, showing that the human swarm is learning.

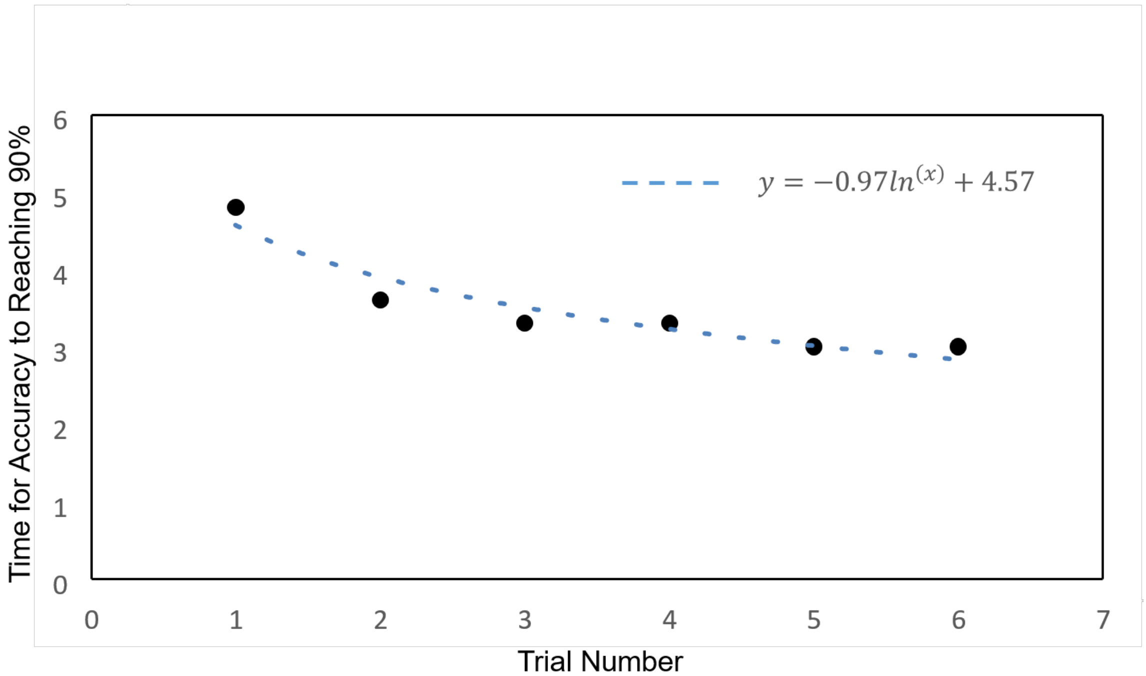


Figure 2.7: For color-change vocal command such as "*I want everybody to turn them red*," "*turn the red off turn the blue on*" or "*let's go to green*" the swarm respond time tends to reduce, demonstrating that the swarm is learning

2.2.3 The time constant for a harder vocal command

For simple color-change vocal command, people were able to achieve the goal in a short time. This section analyzes the time response to more complex commands, For example, at time 01:23, the vocal command was "*When I say go I want you to turn them on and I want this whole group, this group that's gathered tonight to be one color but I'm not going to tell you what color that is.*" So actually we will see how long exactly it will take human swarm to accomplish the vocal command they heard.

This experiment is a classic distributed consensus problem. In this experiment, all people in the crowd must adjust their own color with their neighbors, but since the vocal command, is not specific on which color they need to turn, the process takes about 10 seconds. For this analysis, color umbrellas' amount is changing every frame, then we can find out which color the human swarm going to change. At the same time we can get the ratio of major color. An exponential function is fit to the data, giving 1-e$^{-0.32\,t}$.

Fig 2.8 shows the response and the time constant.

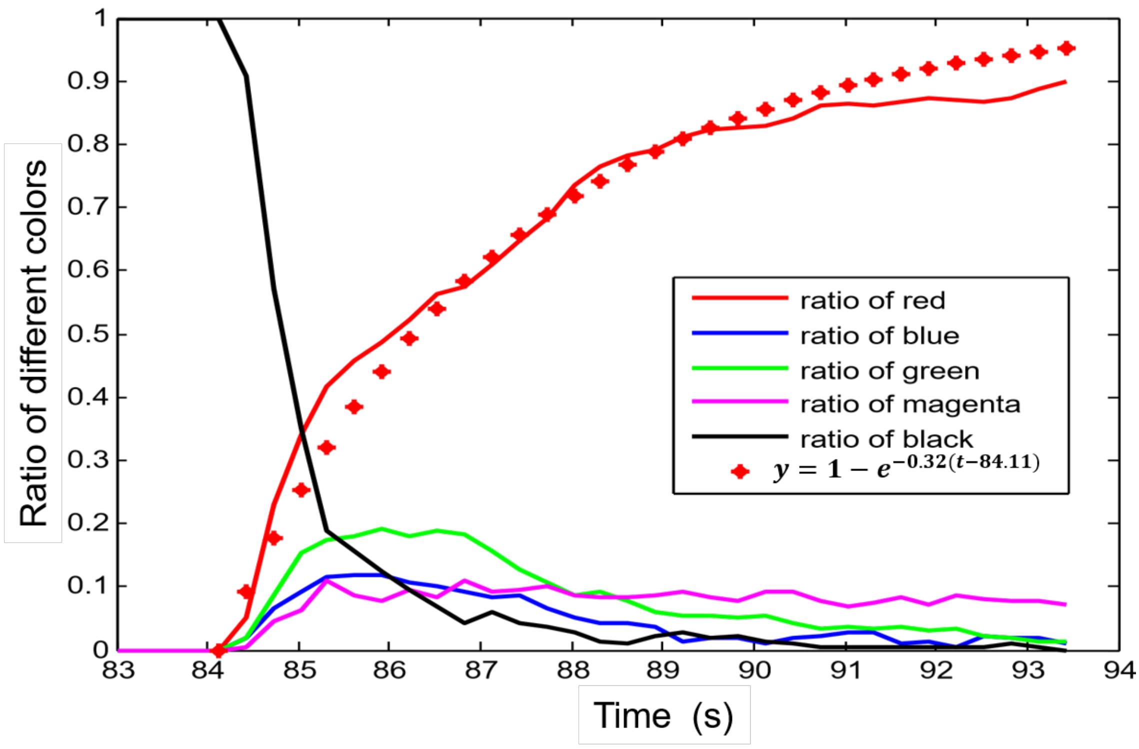


Figure 2.8: Response time for command "when I say go I want you to turn them on and I want this whole group, this group that's gathered tonight to be one color but I'm not going to tell you what color that is"

People start to switch colors around *t=82s,* and by *t=94s*, all people turn to same color. The red dotted line shows an exponential fit with a time constant of *3.125s*

2.2.4 Comparing accuracy between similar commands

There are several very similar vocal commands. We want to know in those kinds of commands, which one they performed better. For example, for the command "if you're red Move!" how accurate were they? Compared to "When I say go I want the red to freeze and the blue's to move … Go!" And "Let's try that with the green Go!" An analysis is shown in Fig 2.9.

Fig 2.9 shows that the swarm accuracy increased in response to "*When I say go I want red to freeze and the blues' to move.*" An exponential function fit shows when the swarm achieved 0.9 accuracy, which means more than 90% of the umbrellas are following the vocal command. Each respond has an exponential function fit, for "red move" is $0.92(1 - e^{-1.61t})$ for "red freeze blue move" is $0.95(1 - e^{-1.06t})$ which is same with "green go."

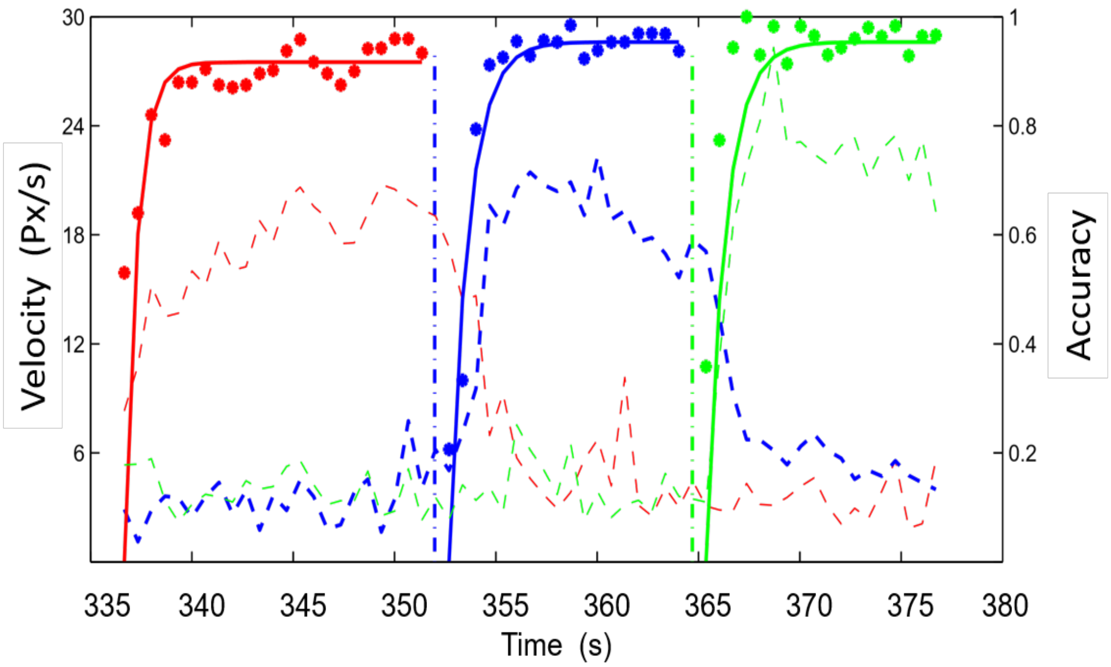


Figure 2.9: Comparing the accuracy of three command "*If you are red move*", "*Let's try that with green go!*" and "*When I say go I want red to freeze and the blues to move*" In every 20 frames, if the umbrella's new position is more than a quarter of its radius, this umbrella is moving.

Through this figure, we can see the difference between the red's velocity and blue's velocity. Before the vocal command "*Go*" red umbrellas are moving, blue umbrellas are not moving, after that the command, reds freeze and blues move instead.

Table 1: Trial number and specific accuracy

Accuracy Comparing	
Trial	Accuracy
Red Move	0.92
Blue Move	0.95
Green Move	0.95

In table 1, we show the accuracy for each trial, it is apparently that human swarm is improving their performance during this period, which means they spend less time to change their colors, move faster.

2.2.5 Shape-matching abilities

Later in the video, the human swarm was given harder commands including to form circles. The accuracy of circle formation of the human swarm is shown in Fig 2.10. The equation $C^2/4\pi A$ is used to evaluate the circularity, where C is the circumference and A is the area of the convex hull. Values close to 1 indicate a more accurate circle. The smaller the value, the more round the circle is, otherwise if the value is larger than 1, we can make a conclusion that the circle is not real round. Three circles were formed by the three different colors, and each color became increasingly more round.

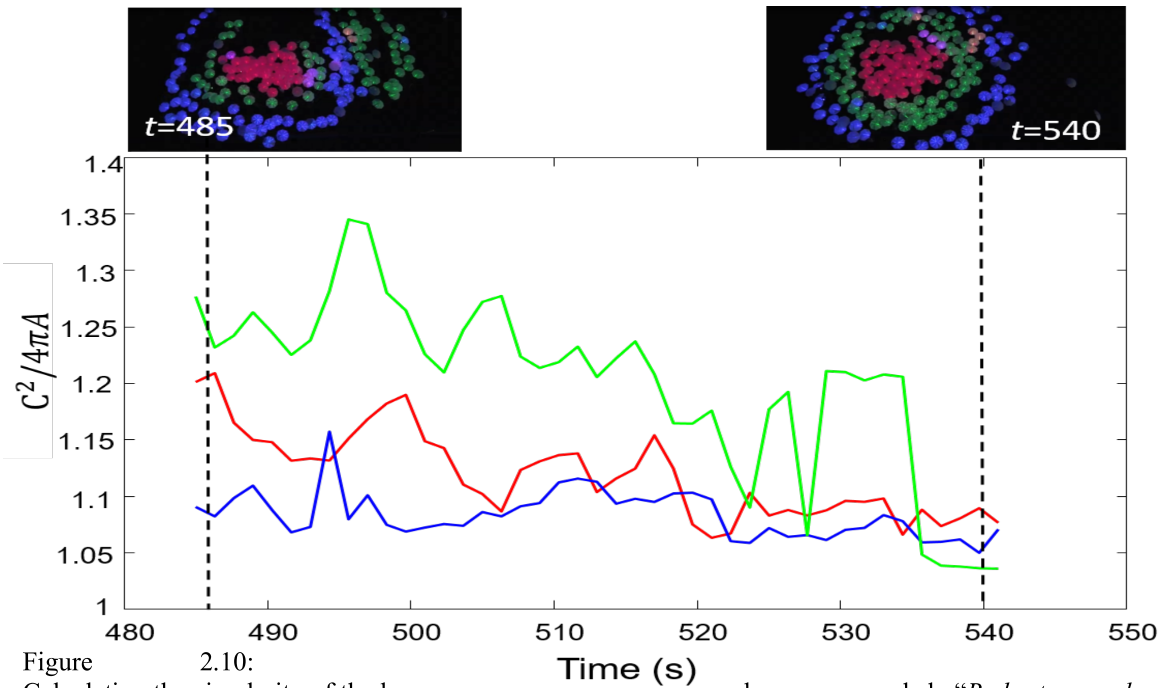


Figure 2.10: Calculating the circularity of the human swarm when commanded "*Red stop and bunch up, See how round you can be, keep circling around them greens*" There are three circles with different colors, values closer to one means the swarm shape is closer to a true circle

Fig 2.10 illustrates that at *t*=485 second, the human swarm was given the vocal command *"Red stop and bunch up. See how round you can be, keep circling around* *them greens."* and then they began to move. The human swarm followed this command till t=540 seconds, when a new command was announced. By calculating the circularity, human swarm is not able to perform a perfect circle obviously, and the green umbrellas is more unstable comparing to other two colors. But during this period, human swarm did improve their performance gradually, and till t=540 the circlularity is close to 1, which means circles became increasingly round.

We can say that comparing with "color change" vocal commands, human swarm does not perform so accurate when matching the specific shape, however, human swarm can still improve, human can learn at the same time.

2.2.6 Forming a human swarm into a "snake"

In this small section we will show the results how human swarm perform "snakes". The human swarm was told to form a "snake", which means they were divided into three groups based on their color, and asked to connect with their neighbors to move like a snake. To evaluate whether the "snake" is good or not, the number of umbrellas in the snakes as a function of time, and how many umbrellas were not in a snake is shown in Fig 2.11.

But how can we define it is a snake or not? We need a rule to evulate whether human swarm had form a real snake or not. For this experiment, a snake is defined as: there should be at least five same-colored umbrellas, they are connected with each other one by one, where each successive umbrellas is within four umbrella radius of a neighbor.

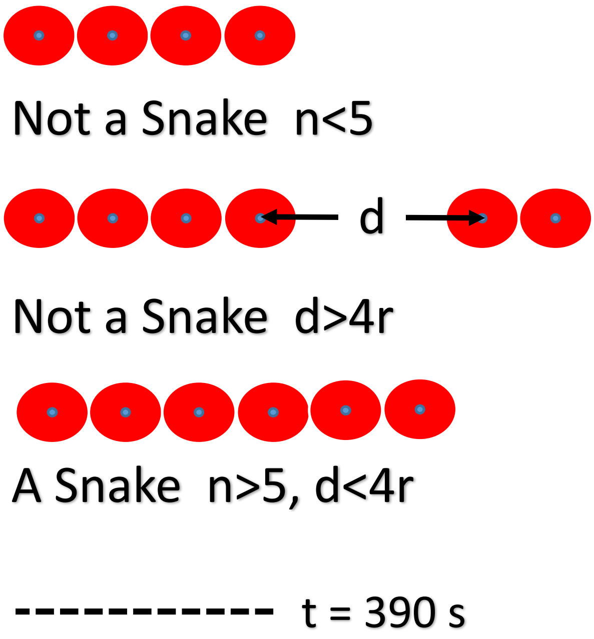


Figure 2.11: A definition of “snake”, a “snake” should consisted of at least 5 umbrellas sequencely, distance between successive umbrellas is less than four umbrella radius.

The results of how human swarm performed are shown in Fig 2.12.

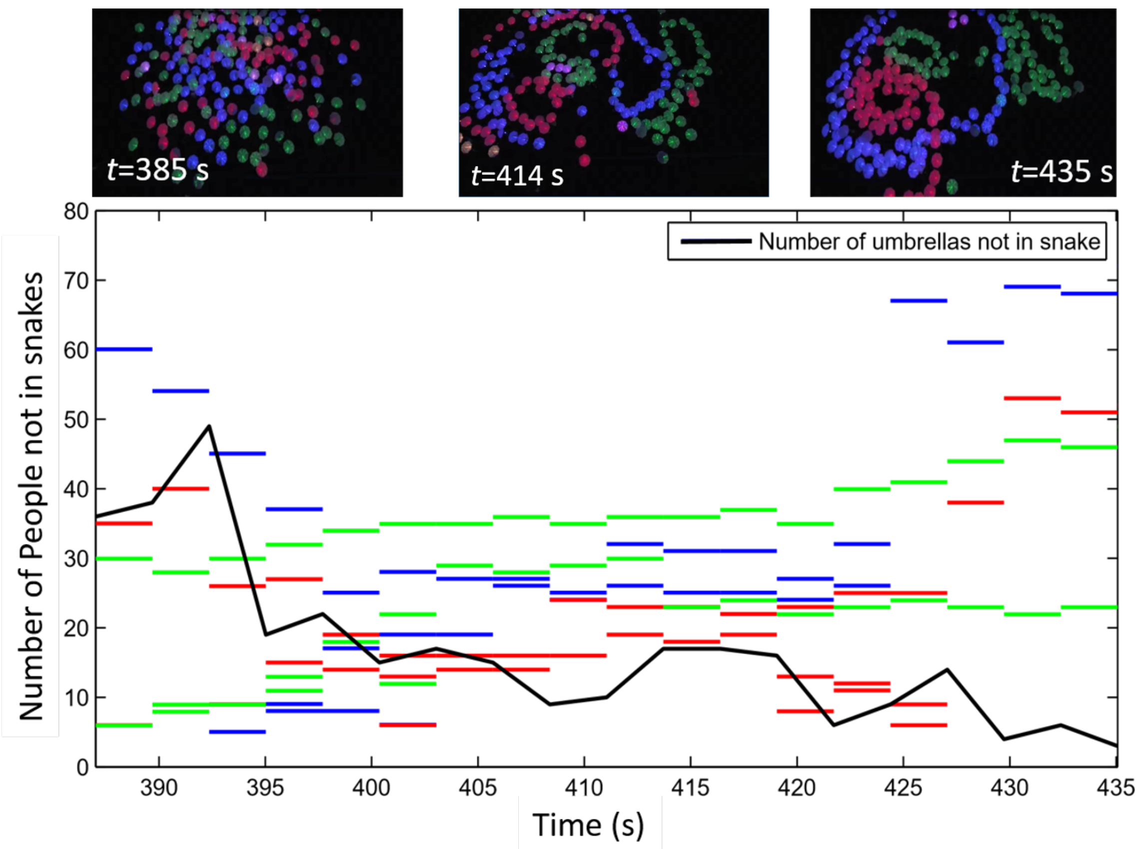


Figure 2.12: Time required to form snakes, y-axis shows how long the snake is, and the number of umbrellas are in (or not in) each snake.

There are three different colors of snakes in the video. For example, we know that there are around 200 umbrellas in the frame, let's see at time t=414s, there are two green snakes with 25, 35 people, one 19-member red snake, two 33, 28 blue snakes, and 18 undefined people who are not in any snake.

The human swarm began to form snakes at the moment they heard the vocal command, at *t*= 385 seconds, and this command completed at *t*=435 seconds. During this period, the number of people in snakes are increasing, while the number not in a snake decreased. At the end there are four snakes in the image.

Based on mentioned above, human swarm still able to perform a "snake" as required in the vocal command, and its performance is improving. So we can get the conclusion that human swarm is a good learner in this experiment, even it is harder than "color change" commands.

2.2.7 Accuracy of the "Bullseye" configuration

Human swarm is able to accomplish simple vocal command such as change a color or for a circle, how about a harder vocal command as "shape-matching", we evaluate the accuracy of performance too. In the overhead video, at *t =613s* the human swarm was directed to form a "Bullseye". To evaluate their performance, we have two standrads, first of all, circularity was again evaluated using the equation $C^2/4\pi$, and second, by calculating the mean distance between each circle's centers. In a perfect bullseye, all circles' are concentric.

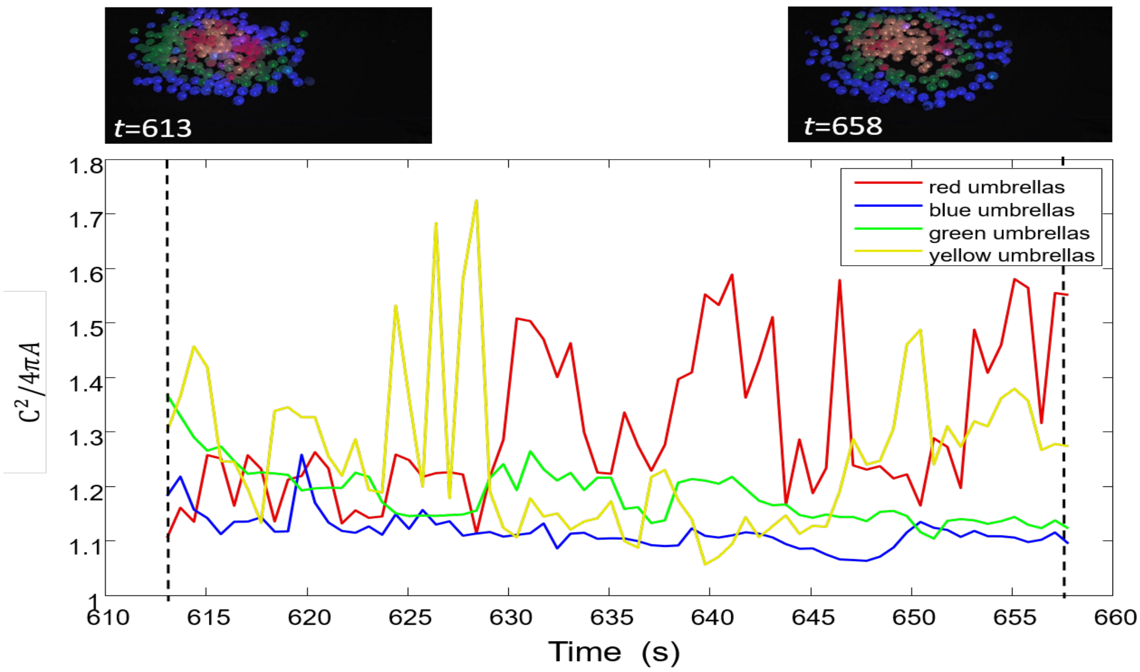


Figure 2.13: Circularity of each circle in a bullseye given the command, "*I would like to see three stripes, you know, like in the middle one color*".

As we can see from figure 2.13, there are total four circles, which green circle and blue circle is closer to be a real circle, red and yellow circle are not so good, while human swarm did make circles. Theory applied to evaluate human swarm's performance mentioned above maybe is not visual enough. To display the result we got more directly, we made "circle fit" for the "Bullseye". In the overhead video, human swarm are directed to form a "Bullseye" which should be consisted by four concentric circles which are blue, green, yellow and red. We find out the best fit circle human swarm formed to see whether human swarm did a good job.

In next image, we find out human swarm's best fit circle to see whether it's a good bullseye or not. TODO: CITE THE CODE SOURCE

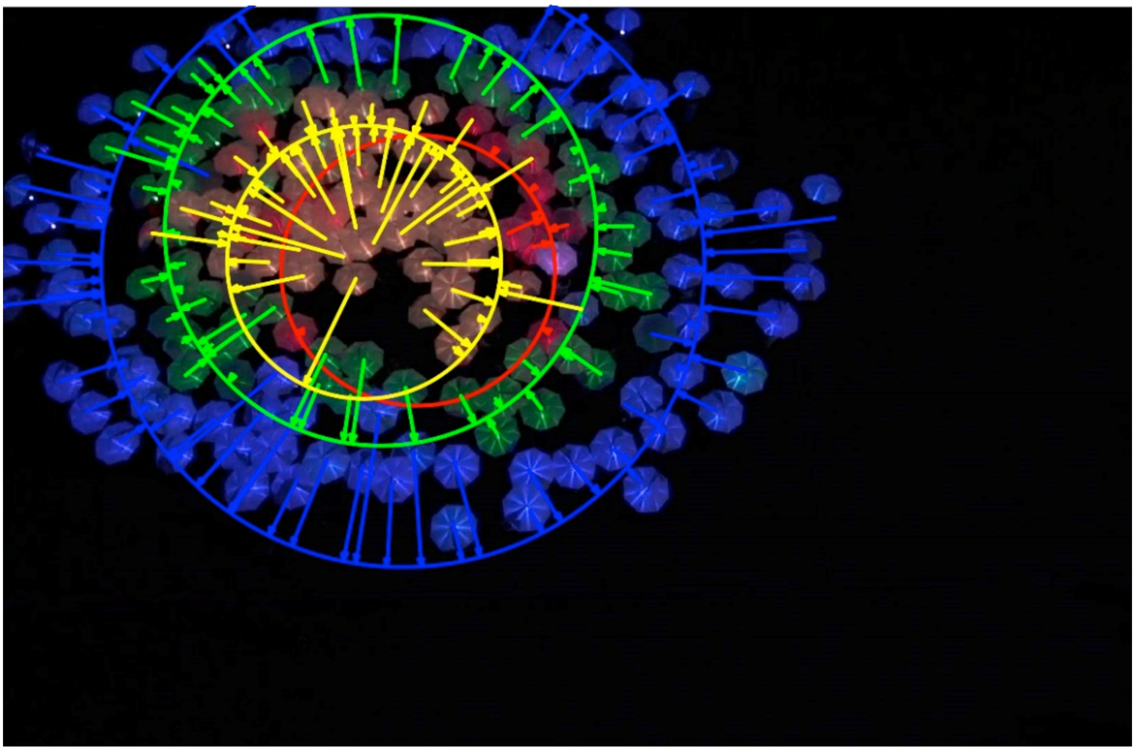
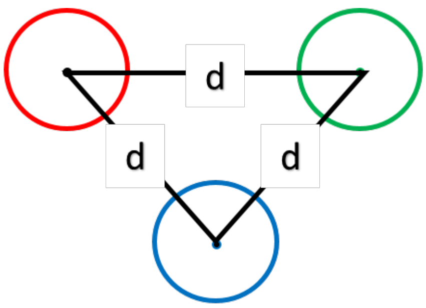


Figure 2.14: Best fit circles for human swarm's performance

This plot show clearly how far away for human swarm to form a beautiful real "Bullseye". Blue, green, yellow and red circles are best fit circles, arrows represent the "error lines" means the distance between umbrellas center to best fit circles' boundary.

To evaluate the bullseye, besides calculate how round the circle is like shown in Fig. 14, as another standard, we calculate the distance between every two circles' center too. If the bullseye is good enough, the distance should be small enough.

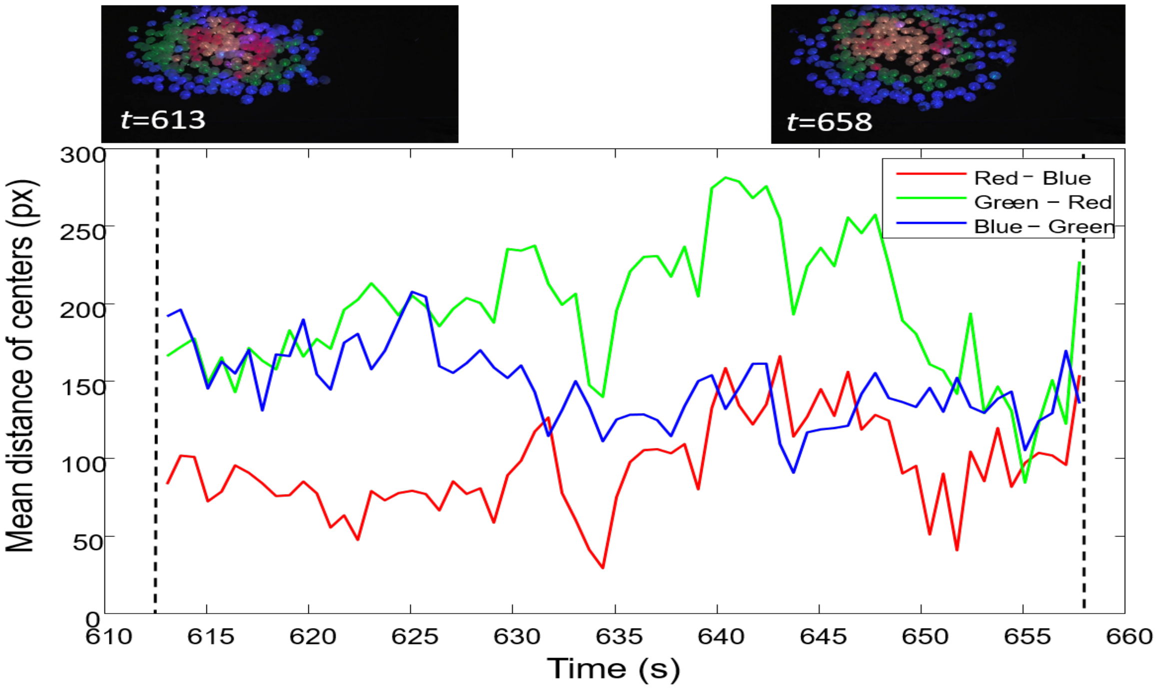
Figure 2.15, shows the theory we used, calculating the distance between centers,



$d = distance\ of\ every\ two\ circle$

Figure 2.15: Another equation applied to evaluate human swarm's performance

Uses the equation $\sqrt{(r_x - g_x)^2 + (r_y - g_y)^2}$ for example, r_x, r_y is x, y position of red circle, g_x, g_y is x, y position of green circle.



TODO: x = 0:0.1:5;p = plot(x,sin(x),'-b',x,sin(x),'--r'); set(p,'linewidth',4) Make the lines two colors
Figure 2.16: The distance between each two circles to evaluate the quality of bullseye as a function of time

The two figures above demonstrate that the human swarm was unable to improve their bullseye. At first, according to the vocal command, it was required to form three kind colors of circles, which are red, green and blue. However, as we can see from the figure, started at *t*= 610 seconds, there are four different colors: red, green, blue and yellow. This situation may be caused by the LED light itself, so it may not be human swarm's fault.

However, in the second figure, we did not count in the yellow circle. But the mean distance between each two circle centers did not reduce or stay constant, giving a second metric indicating the bullseye was not improving.

2.2.8 The accuracy of the human swarm's position memory

This experiment analyzes the accuracy of human swarm when was commanded to return back to known position. In this experiment, we want to see how good human swarm's memory is. How to evaluate their memory? We compare the original position, which is before the swarm moves to follow the vocal command, and the final position, when they finished the command that ordered them to move back. Results are shown in Fig 2.17.

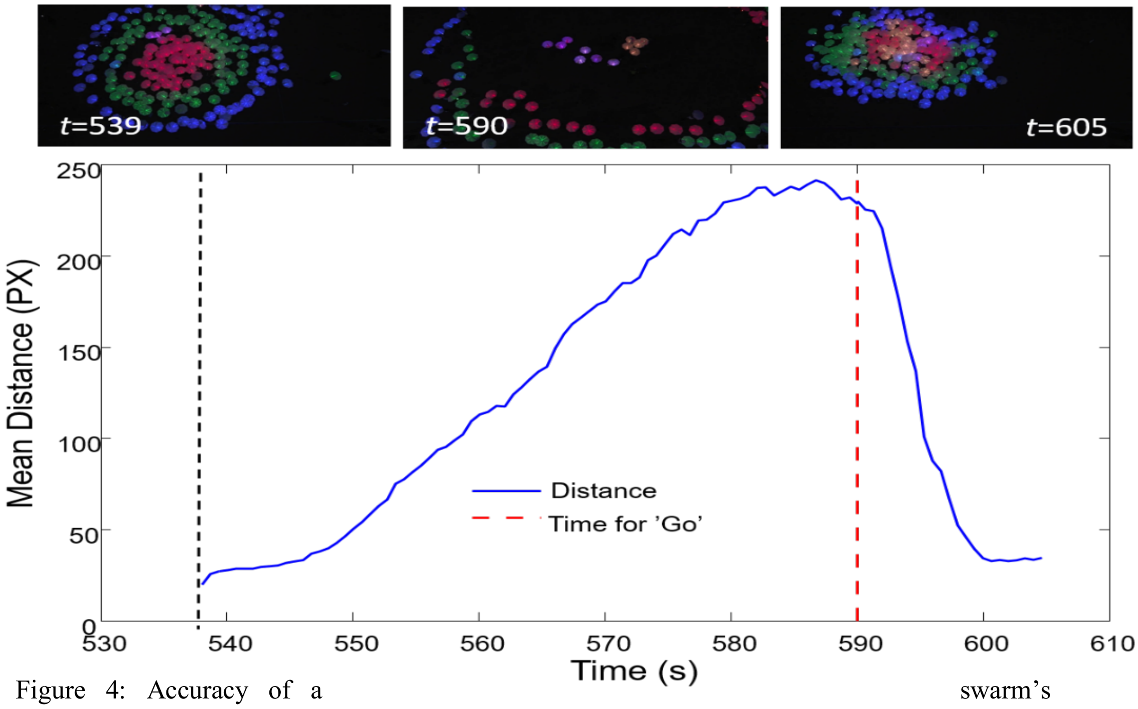


Figure 4: Accuracy of a swarm's position memory. It compares the mean distance between everyone's "*original position*" and the "*returned position*". Smaller distances indicate better memory. When they heard the vocal command "*When I say go I want you to move*"

In Fig 2.17, from t=539, the human swarm spread out slowly, and at *t*=590 they were commanded to return to their original position. During this period, the distance between current position and original position is increasing, which reflects the reality. After that moment, human swarm began to go back, so the distance is decreasing. Finally till t=605 before next command, the distance is almost same with original one.

CHAPTER 3

Simulation

Previous work aims at analyze how large quantity of people respond to vocal commands with local feedback and overhead camera video. We talked about how human swarm performed under instructions such as “color change”, “shape match” and “position memory.” This analysis proved that they human swarm is good at simple vocal commands. While it is obvious that people are able to learn, it is encouraging to know that a swarm of people can also improve in performance.

When it comes to harder vocal command, the human swarm did not perform as well as for simple commands. They may even fail to improve their performance, as demonstrated by the “bullseye test”. In this chapter, we ran many simulations to explore how swarm robots perform under distributed decision-making.

In the simulation, there are 200 agents randomly placed on a 2D region, each agent initially selects a color from the red, green, and blue. The goal for this simulation is all these agents agree on one same color finally, but it does not matter what kind of color it is. At each turn, agents will decide whether to change color or not, or which color they need to change by checking their nearest k neighbors. This is a classic consensus problem for distributed agents. What makes it interesting is that, unlike a having a swarm agree on a leader or on a mean, there is no deterministic algorithm. Every agent is indistinguishable, each agent follows the same rules, and no agent is a leader.

Since no deterministic algorithm exists, in our algorithm each node selects its color from a probability distribution weighted according to the colors of its k-nearest neighbors, update their color based on neighbors, their own colors. There are a number of caveats: Turns are synchronized, and each robot must run the same algorithm, each robot has a unique random number generator, the set of potential colors is randomized, no algorithm such as “everyone choose red” is allowed.

In the simulation, we choose k-nearest neighbors, with k=7,8, 9, 10, 11 to see how good robots are at this simulation, comparing different values of k, see how it influences the simulation.

In figure 3.1, we set each point check 7 nearest neighbors’ color, 200 nodes select color from red, green, and blue randomly.

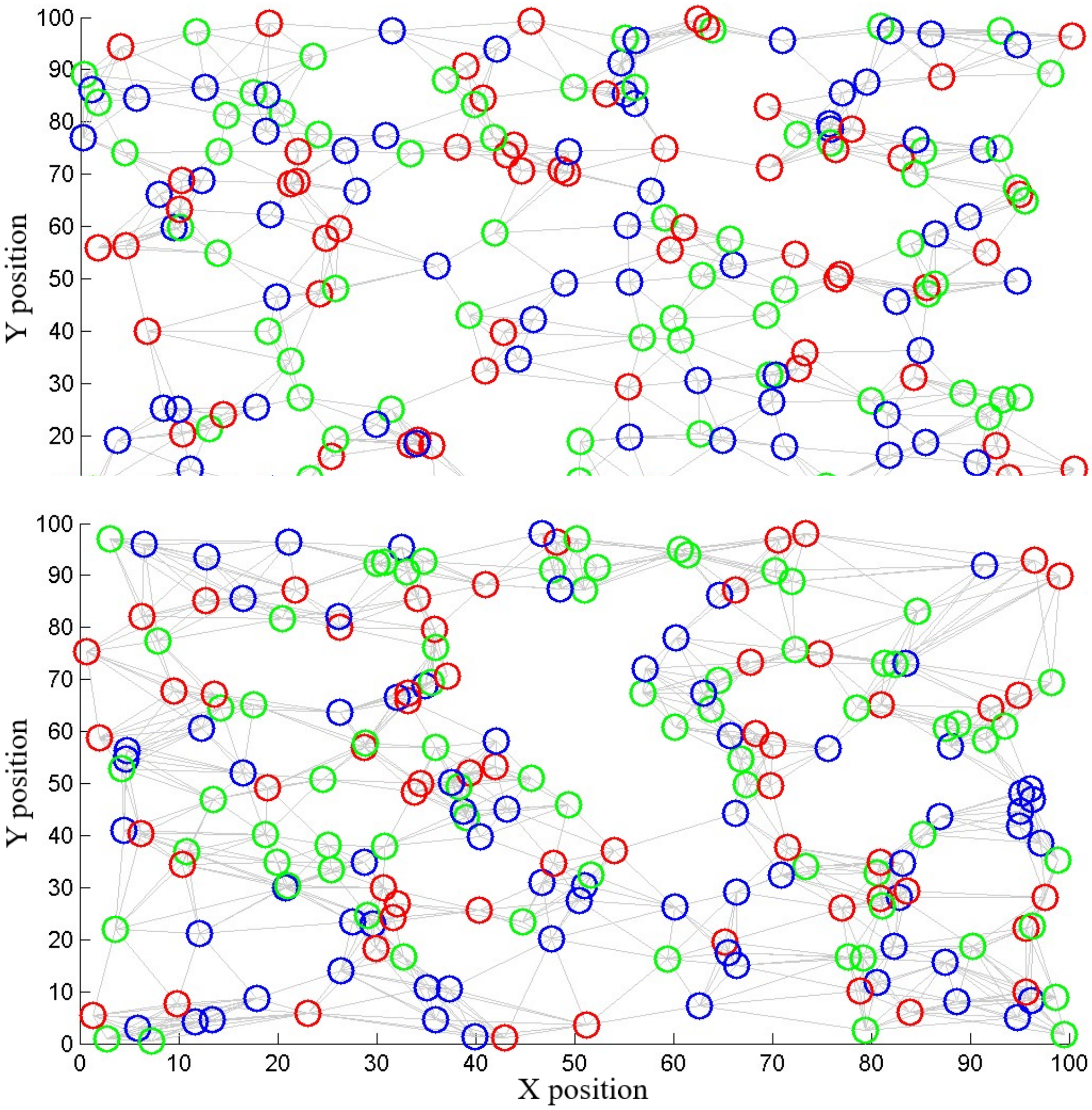


Figure 3.1: 200 nodes randomly initialized in a 2D space, each point selected a color from red, green, blue randomly. Each point turn color based on 7 nearest points

Figure 3.2: All nodes choose one same color finally, takes 205 iterations

In fig 3.1, we can see that each point is connected with its nearest points, After iterations, all 200 nodes will finally change to one same color, in figure 3.2, we show the result, show how many times iterations it took.

In figure 3.3, we change the value of nearest neighbors to 9, see how robots performed in this situation.

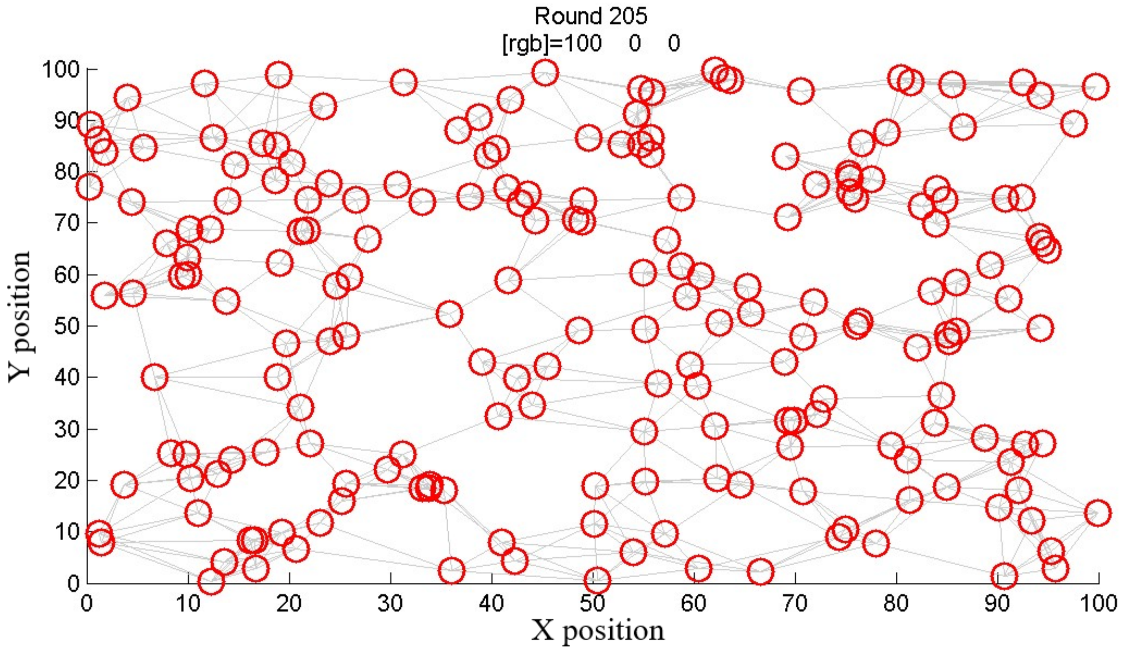
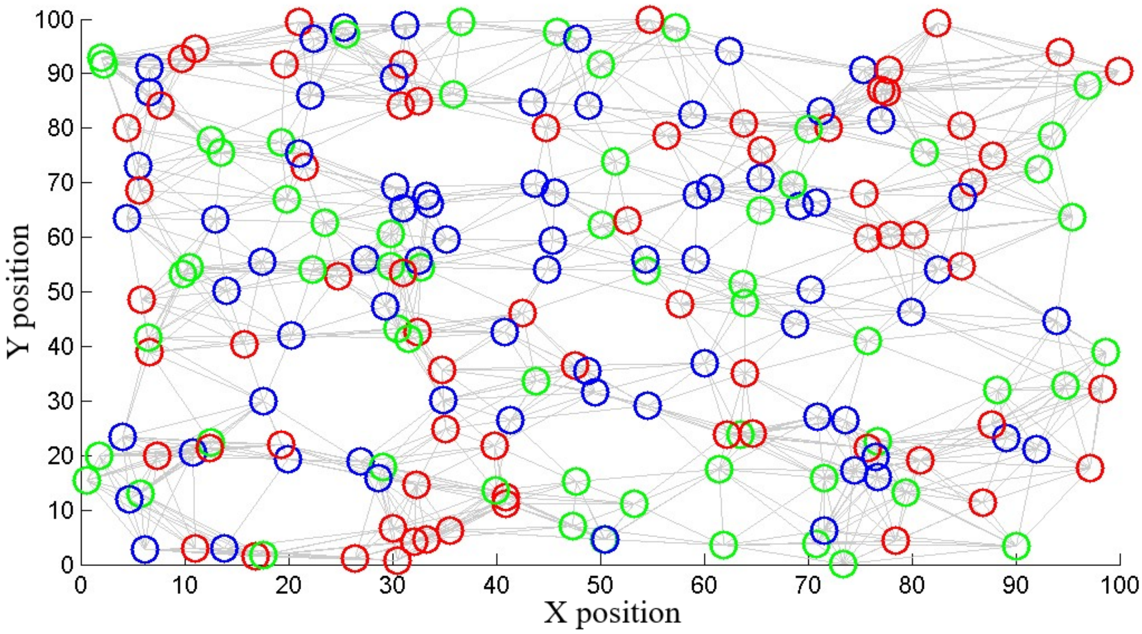


Figure 3.3: 200 nodes randomly initialized in a 2D space, each point selected a color from red, green, blue randomly. Each point turn color based on 9 nearest points.

Figure 3.4: All nodes choose one same color finally, takes 643 iterations.

In figure 3.4, show the result when all points get same color. In figure 3.5, we



change the value of nearest neighbors to 11, see how robots performed in this situation.

Figure 3.5: 200 nodes randomly initialized in a 2D space, each point selected a color from red, green, blue randomly. Each point turn color based on 11 nearest points.

In figure 3.6, show the result when all points get same color.

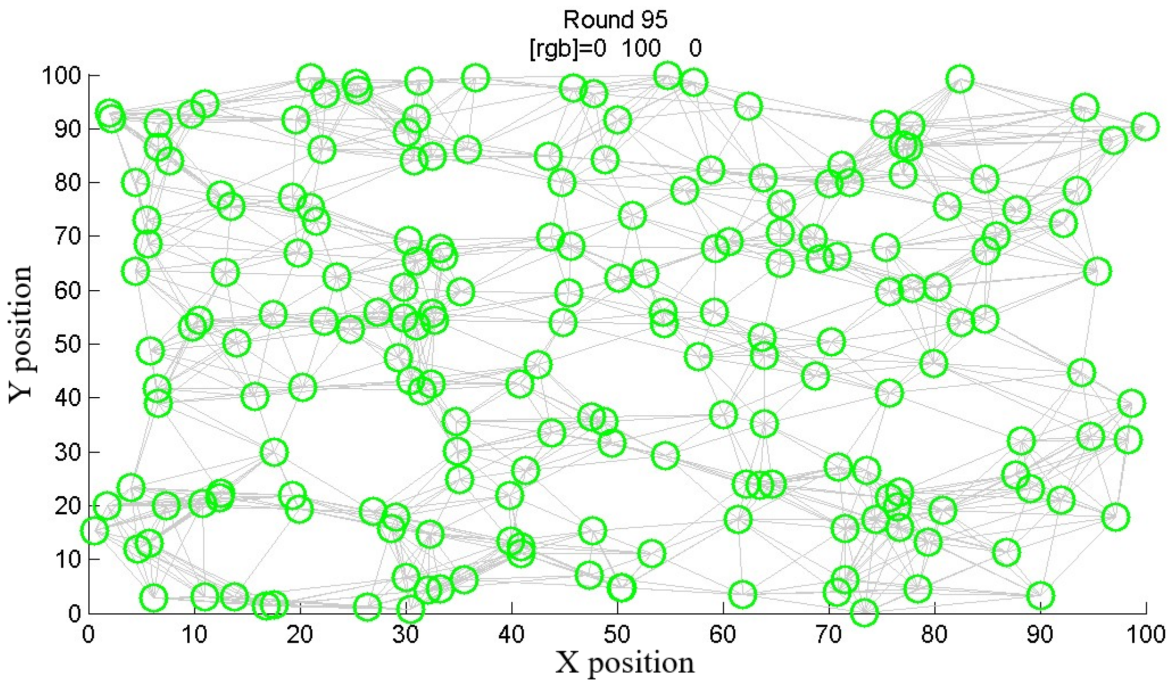
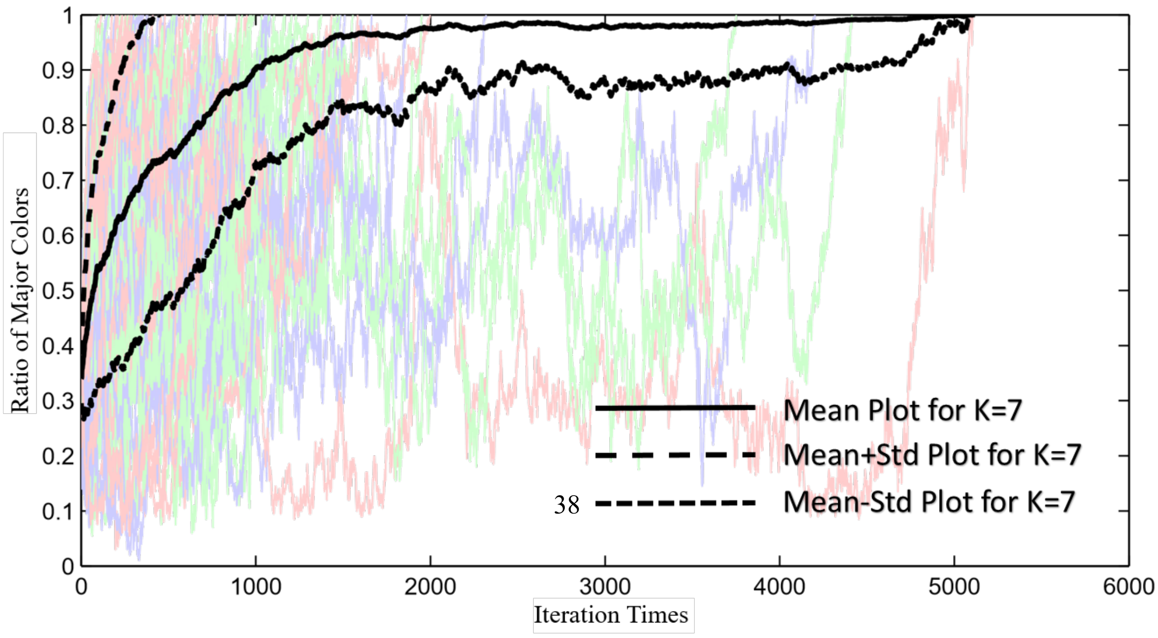


Figure 3.6: All nodes choose one same color finally, takes 95 iterations.

As figures shown above, different value of nearest neighbors may influence the simulation, however only one experiment cannot help us draw a certain conclusion. Then we run the experiment 100 times with 7, TODO: fix 9, and 11 nearest neighbors.

In figure 3.7, we plot the result of when each point check their 7 nearest neighbors



color, and run 100 times, see the ratio change of major colors as a function of iteration time.

Figure 3.7: Result of 7 nearest neighbors run simulation 100 times, plot ratio of major colors as a function of iteration times, also plot the "mean", "mean + std", "mean – std."

We can see in the figure 3.7, in the background there are light green, blue, and red lines which represent the major color in each simulation, and there are totally 100 times simulations.

Besides, we also get other three plot lines. The black solid line show the "Mean Plot for K = 7" which means the average ratio of major colors as a function of iteration times. And there are other two dash lines. The top one is "Mean + Std Plot for K = 7", the bottom one is "Mean - Std Plot for K = 7", these two dash lines represent the range of major colors' ratio as a function of iteration times.

We change the value of k nearest neighbors and get other two plots show ratio of major colors as a function of iteration times.

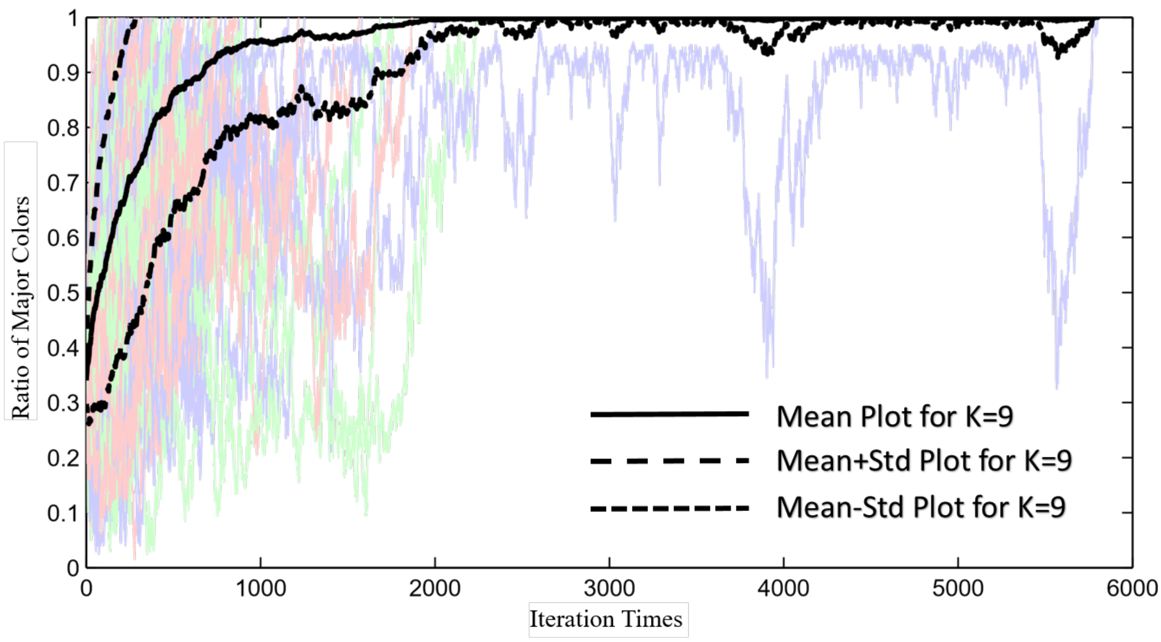


Figure 3.8: Result of 9 nearest neighbors run simulation 100 times, plot ratio of major colors as a function of iteration times, also plot the "mean", "mean + std", "mean – std."

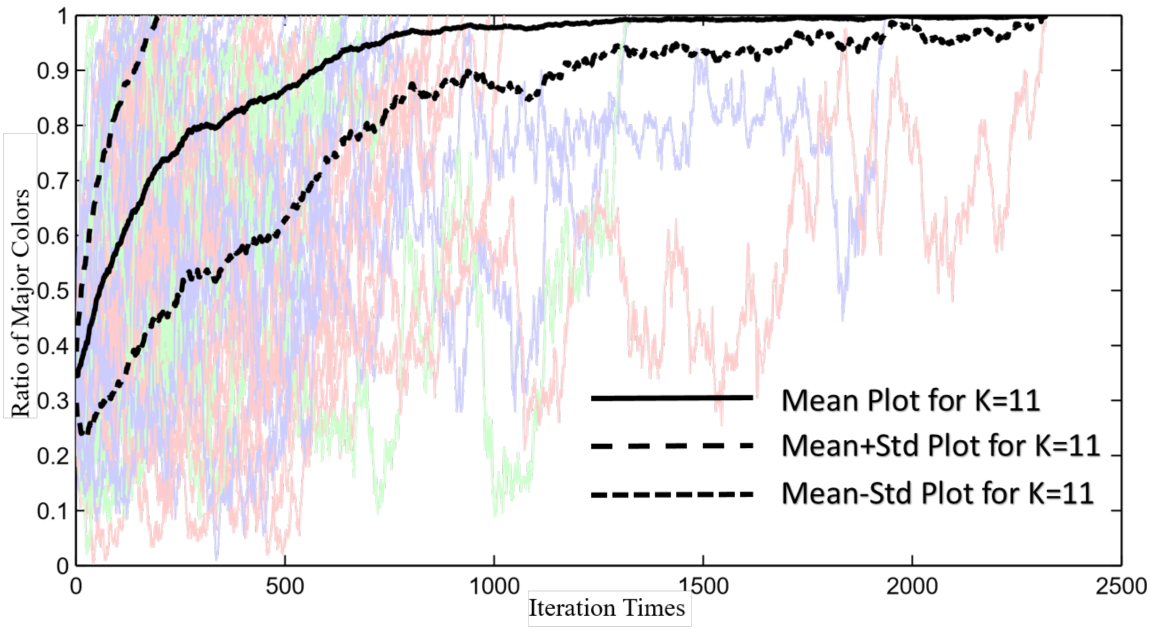


Figure 3.9: Result of 11 nearest neighbors run simulation 100 times, plot ratio of major colors as a function of iteration times, also plot the "mean", "mean + std", "mean – std."

After all, in figure 3.10 we comparing the mean plot after 100 simulations with k = 7, 9, and 11.

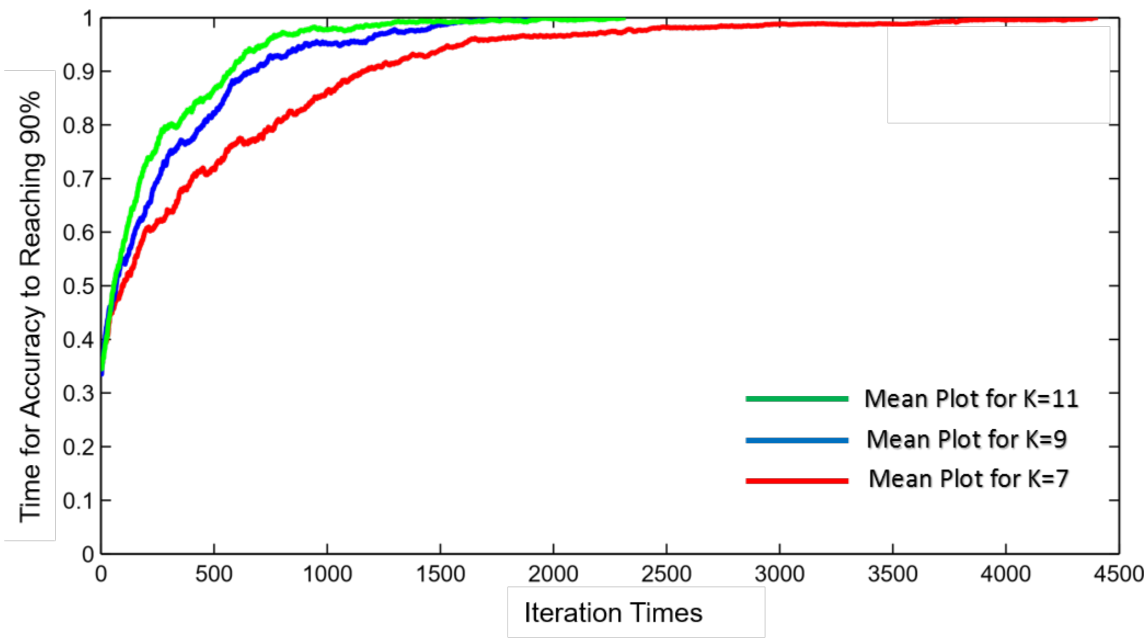


Figure 3.10: With k = 7, 9, and 11, mean plots after 100 times simulations

As shown in figure 3.10, we get the time constant for each k, which are $\tau_{11} = 1.03\ s$, $\tau_9 = 1.05\ s$, $\tau_7 = 1.1\ s$.

CHAPTER 4

Conclusions

Because the main contribution of this thesis is data analysis, the accuracy of data collecting is very important. In the first part, tracking, if the observed data matches the estimated data, this model for tracking umbrella seems good. While tracking the objects, the initial state and noise covariance influence a lot, maybe more than that, we need to tune the estimation functions to speed up our tracking system cause when tracking many objects, speed up is important. *Tuning* of the Kalman filter refers to estimation of covariance matrix, if it is not tuned properly, it leads to divergence of expected value from the actual value [13] In this project, the number of umbrellas are not constant all the time, it may change, and we need to track all the umbrellas.

Kalman Filter has wide applications in many fields, including object tracking. Our tracking system can track multiple objects which have similar appearance, more than two hundred umbrellas were tracked simultaneously. The system detected umbrellas disappearing or adding quickly and continued tracking.

Data analysis revealed the human swarm was learning and that the human swarm performed well at forming snakes, but poorly at forming concentric circles.

Appendix

The appendix provides specific algorithms and equations we applied as math tools when analyzing data during the thesis project. It also provides part of codes used during analyzing the thesis project.

Appendix A: K-means algorithm support

The *K*-means method uses *K* prototypes, the centroids of clusters, to characterize the data [8]. Data clustering is widely used in many fields, including data mining, pattern recognition, decision support, machine learning and image segmentation. *K*-means seeks to minimize:

$$J_{K-means} = \sum_{k=1}^{K} \sum_{i \in C_k} (x_i - m_k)^2 \tag{1}$$

Here $(x_1, \ldots, x_n) = X$ is the data matrix, $m_k = \sum_{i \in C_k} \frac{x_i}{n_k}$ is the centroid of clusterC_k, and n_k is the number of points in C_k.

K-means, has two steps: **assignment** and **update**. The first assignment step uses observed data to assign data points to the cluster which yields the minimum within-cluster sum of squares. The sum of squares is squared Euclidean distance, so this is the nearest mean [4]:

$$S_i^{(t)} = \left\{x_p : \left\|x_p - m_i^{(t)}\right\|^2 \ll \left\|x_p - m_j^{(t)}\right\|^2 \forall j, 1 \ll j \ll k\right\} \tag{2}$$

$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j \tag{3}$$

Equation (2) is used for assign objects, each x_p is assigned to exactly one $S_i^{(t)}$. Equation (3) is used to calculate new means to be the new centroids of the observations in the new clusters.

Appendix B: Kalman filter algorithm support

In this section, describes tracking umbrellas using the Kalman Filter algorithm. It is a recursive algorithm so that new measurements can be processed when they arrived, then a new round of calculating begin [14]. It can filtering out the noise during the time finding out the best estimate data, and a Kalman Filter not only just clean up the data measurements, but also projects those measurements onto the state estimate.

The Kalman Filter maintains both an estimate of the state:

$$X(n|n) \text{ Estimate of } X(n) \text{ given measurements } Z(n), Z(n-1)$$

$$X(n+1|n) \text{ Estimate of } X(n+1) \text{ given measurements } Z(n), Z(n-1)$$

And the error covariance matrix P of the state estimate:

$$P(n|n)\text{-covariance of } X(n) \text{ given} Z(n), Z(n-1)$$

$$P(n+1|k)\text{-estimate of } X(n+1) \text{ given} Z(n), Z(n-1)$$

The Kalman Filter recursive processing is separated into several stages. The first part consists of two equations is called *"Time Update (Predict)"*:

$$X(n+1|n) = AX(n|n) + Bu(n+1) \tag{4}$$

$$P(n+1|n) = AP(n|n)A' + Q \tag{5}$$

Equation (4) represents the predicted state, (5) represents the error covariance ahead. And the second part can be seen as "*Measurement Update (Correct)"*:

$$K(n+1|n) = P(n+1|n)H'(n+1)[R(n+1) + H(n+1)P(n+1|n)H'(n+1)]' \tag{6}$$

$$X(n+1|n+1) = AX(n+1|n) + K(n+1[Z(n+1) - H(n)X(n+1|n)]) \tag{7}$$

$$P(n+1|n+1) = [I - K(n+1)H(n+1)]P(n+1|n) \tag{8}$$

Equation (5) represents the Kalman Gain computed, (7) means update the estimate with measurement$Z(n+1)$, (8) represents the update of error covariance.

When apply the Kalman Filter in this project trying to track the umbrellas' motion path, we need to do is initialize the algorithm at first, and also need to define the main variables that will be used in the equation. According to the practical situation, umbrellas are moving in the whole video with an un-constant velocity, the noise should be considered. Here I define the measurement noise R in the horizontal direction both x axis and y axis, and the process noise covariance Q, the estimate of initial umbrella position variance. Then we defined the update equations which also is the coefficient matrices, can be seen as a physics based model, so that we can make an estimation where the umbrella will be for the next moment.

We have another figure to further explain how Kalman filter works.

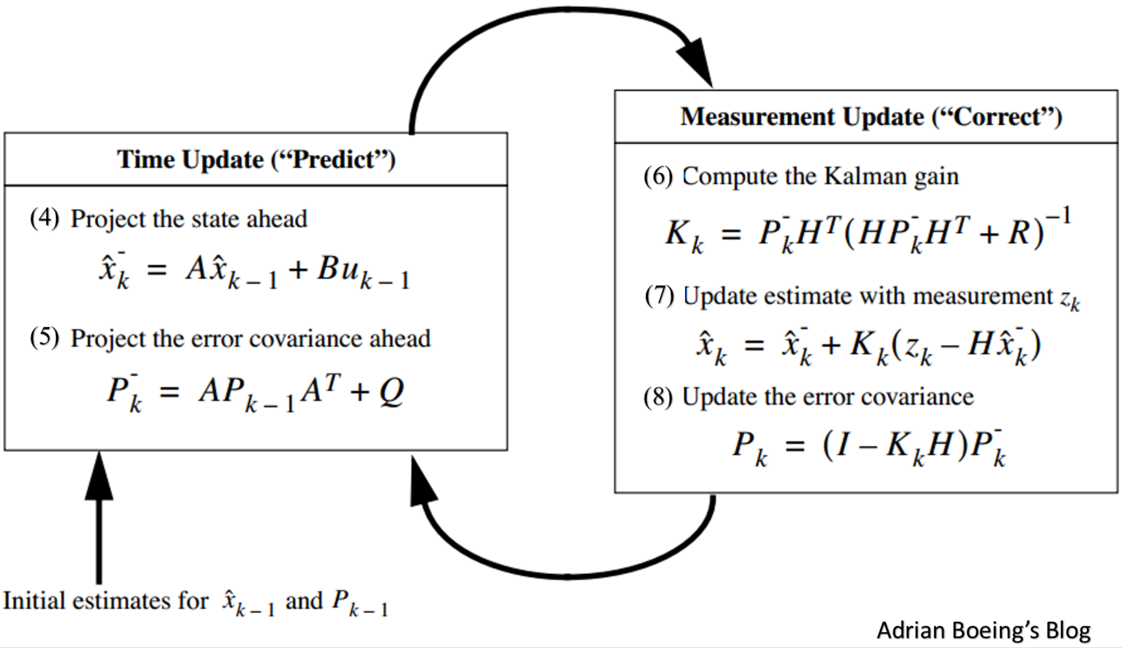


Figure A.1: Constituent part of Kalman Filter, how time update and measurement update work together.

In the update equations, all matrix variables need to be defined:

Initialize A represents the state transition matrix; B represents the input matrix, which is optional; H represents the observation matrix, K represents the Kalman Gain. After that, we can call the Kalman Filter. As mentioned before, each iteration of Kalman Filter will update the estimate of state vector of a system based upon the information in a new observation. In this project, the data which had already been collected is the x, y location of each umbrella at each frame. Although these location data have some error but they are reliable enough and they are used as measurement data. To track the motion path of the umbrella, we set an empty matrix "centroids" to store the x, y locations of each umbrella, so this matrix can represents the real locations of umbrella.

Appendix C: Other equations applied

To predict where the umbrella is in the next frame, my thought is to calculate the distance between two centroids of the umbrella, the first centroid is the observed location of umbrella at this frame, the second centroid is the estimated x, y location of umbrella, which will be updated. What we need is to calculate the distance between two central points, pick up the one which is nearest with observed umbrella's position, then this is the next position of the umbrella in the next frame. Because in a very short time between each frame, umbrella's moving could be seen as move towards a straight line and with a constant velocity. After that, the estimated umbrella's position at this frame can be used as observed position to estimate the next position of umbrella at next frame.

$$Distance = \sqrt{\left[\left(c(i,1) - c(x)\right)\right]^2 + \left[\left(c(i,2) - c(y)\right)\right]^2} \tag{9}$$

Appendix D: Codes

Appendix D.1: Classify umbrellas codes

```
function colorizeUmbrellaData

% Aaron T. Becker

% 7/29/2015

%

% Takes data of umbrella positions
```

```
% Save([dataFileName,num2str(frameNumber,'%07d')],
'pointLocations','imsz','frameNumber');
% 1. Get list of files that have data
% 2. For each data file:
% 3. Load the x, y locations of the umbrellas
% 4. Load the corresponding image from the video
% 5. Call k-means to get all pixels associated with each x, y location
% 6. Get the mean color of these pixels for each x, y location
% 7. Save the data [x, y, color, num pixels]
% 8. Save the image to make a movie
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% constants:
vidName = 'First10Min.mp4'; %much shorter!  Is 30 fps.  I want high resolutio0n
data from 1:20 to 2:20.  (frame 2400 to 4200)
dataFileName = 'manualPointsLowRes/';  %'manualPoints/';
meanGreen = 2.577;
meanGreen2 = -3.14;
meanRed = -0.4808;
meanBlue = -2.094;
meanPurple =-1.544;
meanOrange =-0.05;
%meanBlack = -2.13;
meanCyan = -2.50; %MAYBE BIGGER
```

```
meanColors =
[meanGreen,meanGreen2,meanRed,meanBlue,meanPurple,meanOrange,meanCyan];
colorNames = ['g','g','r','b','m','y','c','k'];
% setup instructions (call this at the beginning)
 MOVIE_NAME = 'ProcessedUmbrella';
    G.fig = figure(1);
    clf
    set(G.fig,'Units','normalized','outerposition',[0 0 1
1],'NumberTitle','off','MenuBar','none','color','w');
    writerObj =
VideoWriter(MOVIE_NAME,'MPEG4');%http://www.mathworks.com/help/matlab/r
ef/videowriterclass.html
    set(writerObj,'Quality',100,'FrameRate', 30);
    open(writerObj);
% 1. Get list of files that have data
% 1.a: Try to load points from a data file.
filenames = dir([dataFileName,'*.mat']); % s is structure array with fields name,


% 1.b: Load the video:
tic  %record the start time
display(['Loading video: ',vidName]) %about 4 seconds
vidBirdseye = VideoReader(vidName);
toc %display how long it took to load.  My mac takes 4 seconds.  My PC takes 16s


colorcount = NaN(numel(filenames),numel(colorNames));
```

```
frameNums = NaN(numel(filenames),1);


% 2. For each data file:
for i = 1:numel(filenames)


    % 3. load the xy locations of the umbrellas
    fileStr = filenames(i).name;
    data = load([dataFileName,fileStr], 'pointLocations');
    xy = data.pointLocations;
    % 4. load the corresponding image from the video


    frameNumber = str2double(fileStr(1:end-4));
    cdata = read(vidBirdseye,frameNumber);
    % convert rgb to YCbCr color space
    YCBCRim = rgb2ycbcr(cdata);
    Ythreshim = YCBCRim(:,:,1)>32;
    bw = bwareaopen(Ythreshim,100);  %for high resolution, use 400 px as threshold.
    % 5. call k-means to get all pixels associated with each xy location
    [xcoord,ycoord] = ind2sub( size(bw), find(bw>0));
    nonBackgroundPx = [xcoord,ycoord];
    nonBackgroundPx = [nonBackgroundPx(:,2) nonBackgroundPx(:,1)]; %TODO
make matrix in one step.
```

```
    % 6. get the mean color of these pixels for each xy location
    [aveHue, numPixels,colors,imageClassified] = measureColor( xy,
nonBackgroundPx, cdata); %#ok<ASGLU>



    % 7. save the data [x,y,color, num pixels]
    imsz = size(cdata); %#ok<NASGU>
    save([dataFileName,'/Hue/Hue',num2str(frameNumber,'%07d')],
'xy','aveHue','numPixels','colors','imsz','frameNumber');
    % 8. save the image (to make a movie?)


    indx = numPixels>5;                                           %Remove empty ones.
    colorcount(i,:) = sum( bsxfun(@eq, colors(indx),1:numel(colorNames)) );
    colorcount(i,2) = colorcount(i,1)+colorcount(i,2);
    frameNums(i) = frameNumber;


    %display the image
    figure(1)
    subplot(2,2,1)
    imshow(cdata)
    %Title(num2str(frameNumber))
    subplot(2,2,2)
    imshow(imageClassified)
    subplot(2,2,3:4)
```

```
    set(gca,'FontSize',16)
    for ik = 2:numel(colorNames);

plot(frameNums(1:i),colorcount(1:i,ik),'color',colorNames(ik),'linewidth',2,'Linewidth
',1.5);
        hold on
    end
    hold off
    title('umbrella colors')
    xlabel(['frame ', num2str(frameNumber)])
    ylabel('count of each color')
    axis([0,2000,0,220])
    makeMovie()
    drawnow
end
close(writerObj);
title('FINISHED')




    function makeMovie()
        % (for each frame)


         figure(G.fig)
```

```
        set(gcf,'renderer','painters')   %optional line to remove antialiasing
        tfig = myaa;   %optional line 2
        F = getframe(tfig);
        writeVideo(writerObj,F.cdata);
        close(tfig)
    end


    function rgbC = getRGBfromName(charN)
      rgbC = bitget(find('krgybmcw'==charN)-1,1:3);
    end
    function [aveHue, numPixels,colors,imageClassified] = measureColor( xy, data,
cdata)
      % x, y is the locations of the center of each umbrella
      % data is the x, y locations of the non-background pixels.
      % cdata is the color image r*c*3
      %find the pixels associated to each x, y location
      % returns the average hue 'aveHue' for each x, y location, numPixels: the number
of
      % associated pixels, the classified 'colors', and an rgb image 'imageClassified'
with
      % all the classified objects recolored.


      %convert the image to HSV
      ImageHSV = rgb2hsv(cdata);
      imHUE = ImageHSV(:,:,1);
```

```
imVAL = ImageHSV(:,:,3);
hueAngle = imHUE*2*pi;
imageClassified = 0.2*ones(size(cdata));


num = size(data,1);


k = size(xy,1);
aveHue = zeros(k,1);
aveVal = zeros(k,1);
numPixels = zeros(k,1);
colors = zeros(k,1);


tempx = repmat(data(:,1),1,k) - repmat(xy(:,1).',num,1);
tempy = repmat(data(:,2),1,k) - repmat(xy(:,2).',num,1);
distance = (tempx.^2 + tempy.^2);
[~,cluster_index] = min(distance.');
for ii = 1:k
    thisUmbrellaxy = data(cluster_index == ii,:);
    % figure out the average color
    linearInd = sub2ind(size(imHUE), thisUmbrellaxy(:,2), thisUmbrellaxy(:,1));
    hueSin = sum(sin(hueAngle(linearInd)));
    hueCos = sum(cos(hueAngle(linearInd)));
    aveHue(ii) = atan2(hueSin,hueCos);
    aveVal(ii) = mean(imVAL(linearInd));
```

```
            % count number of pixels associated with this mean
            numPixels(ii) = numel(thisUmbrellaxy(:,1));
            % classify the color
            [~,colors(ii)] = min(abs(meanColors - aveHue(ii)));
            if  ( colorNames(colors(ii)) == 'b'|| colorNames(colors(ii)) == 'c') &&
aveVal(ii) < 0.5
                colors(ii) = numel(colorNames); %black
            end
            rgbVal = getRGBfromName(colorNames(colors(ii)));
            for iii = 1:numPixels(ii) %TODO: fix this loop to be fast
                imageClassified(thisUmbrellaxy(iii,2), thisUmbrellaxy(iii,1),:) = rgbVal;
            end
        end
        figure(2)
        %for debugging:
        imshow(imageClassified)
        for ii = 1:k
            text(xy(ii,1), xy(ii,2),num2str(aveHue(ii),'%.2f'),'color','w')
        end
        %set(texth,'color','w')
    end
set(gcf,'PaperPositionMode','auto','PaperSize',[8,4], 'PaperPosition',[0,0,8,4] );
print(gcf, '-dpdf', 'fig1.pdf');
end
```

Appendix D.2: Distribution consensus simulation codes

```
function [colorHist, color] = colorconsensusRand(k,IndexRepeat)
% this version tries to find the maximum among the others.  If there are
% ties, it randomly assigns.
%
% InitializationL
%   N agents (N=200) are randomly placed on a 2D region.  Each agent
%   initially selects a color from the set {R,G,B}.
%
% Goal: all agents to select the same color
%
% Process:
Turns are synchronized.  At each turn the robots check the current color of their k-nearest neighbors. Update their current color based on the neighbors, their own color, and (perhaps) generating a random number Caveats:  each robot must run the same algorithm, each robot has a unique random number generator, all turns are synchronized, the set of potential colors is randomized -- no algorithm "everyone choose red" will work.
L = 100; %size of workspace
N = 200;%number of nodes
if nargin<1
   k = 7; %number of nearest neighbors
end
maxIter = 10000; %number of iterations to try to get consensus
colorHist = zeros(maxIter,3); %record the ratios of different colors
```

```
bShowNN = true;


Xpos = rand(200,2)*L;

Xcol = randi(3,N,1);




%set up figure

figure(1); clf;

IDX = knnsearch(Xpos,Xpos,'K',k);


% This code draws the nearest neighbors

if bShowNN

    for i = 1:N

        for j = 2:k

            hl = line([Xpos(IDX(i,1),1) Xpos(IDX(i,j),1)],[Xpos(IDX(i,1),2)
Xpos(IDX(i,j),2)]);

            set(hl,'color',[0.8,0.8,0.8],'LineWidth',1);

        end

    end

end

hold on

h = scatter(Xpos(:,1),Xpos(:,2),ones(N,1)*140,Xcol);

set(h,'marker','o','LineWidth',1.5)

hold off
```

```
%simulate
for i = 1:maxIter
    Xcoli = Xcol;
    for j = 1:N % loop over each node
        vc = histc(Xcol(IDX(j,:)),[1,2,3])/k;
        %randomly assign with probability proportional to most likely color
        r= rand(1);
        if r<vc(1)
            Xcoli(j) = 1;
        elseif r<vc(1)+vc(2)
            Xcoli(j) = 2;
        else
            Xcoli(j) = 3;
        end
    end
    Xcol = Xcoli;
    vc = histc(Xcol,[1,2,3])/N*100;
    colorHist(i,:) = vc;
    title({['Round ',num2str(i)];['[bgr]=',num2str(vc')]})
    %update the figure
    set(h,'CData',Xcol);
    drawnow
    if max(vc) >= 100
%        Xcol
        %IndexTmp = find(vc == max(vc));
```

```
        %find the major color and save the ratios of the major color
        colorHist = colorHist(:,vc == max(vc));
        color = find(vc == max(vc));
        %save the picture
        saveas(h,['colorconsensus/myfig_',num2str(k),'_',num2str(IndexRepeat),'.fig']);
        break
    end
    %pause(0.1)
end
```

Appendix D.3: Mean plot of simulation codes

Appendix D.3.1: k = 7

```
k = 7;
RepeatTime = 100;
maxIter = 10000;
colorRecord = zeros(RepeatTime,maxIter);%record the ratios of the major color
colorArray = zeros(1,RepeatTime);
MaxIndex = 0;
for IndexRepeat = 1:RepeatTime
    [colorRecord(IndexRepeat,:), colorArray(IndexRepeat,:)] =
colorconsensusRand(k,IndexRepeat);
    colorRecord(IndexRepeat,:) = colorRecord(IndexRepeat,:)/100;
    LastIndex = find(colorRecord(IndexRepeat,:)>0);
    LastIndex = LastIndex(end);
    MaxIndex = max(MaxIndex,LastIndex);
end
```

```
for IndexRepeat = 1:RepeatTime
    LastIndex = find(colorRecord(IndexRepeat,:)>0);
    LastIndex = LastIndex(end);
    colorRecord(IndexRepeat,LastIndex + 1:MaxIndex) = 1;
end
colorRecord = colorRecord(:,1:MaxIndex);
MeanValueArray = mean(colorRecord,1);
StdArray = std(colorRecord,1,1);
for IndexRepeat = 1:RepeatTime
    if colorArray(IndexRepeat,:) == 1
        IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
        IndexTmp = IndexTmp(end) + 1;
        plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[0.8 0.8
1],'linewidth',0.5);
%         plot(1:MinIndex, NumberCount(1:MinIndex,2)/Total,'color',[1 0.8
0.8],'linewidth',0.5);
    else
        if colorArray(IndexRepeat,:) == 2
            IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
            IndexTmp = IndexTmp(end) + 1;
            plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[0.8 1
0.8],'linewidth',0.5);
        else
            IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
            IndexTmp = IndexTmp(end) + 1;
```

```
            plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[1 0.8
0.8],'linewidth',0.5);
%            plot(1:MinIndex, NumberCount(1:MinIndex,2)/Total,'color',[0.8 1
0.8],'linewidth',0.5);
        end
    end
    hold on
end
MeanMinusStd = MeanValueArray - StdArray;
MeanPlusStd = MeanValueArray + StdArray;
plot(1:MaxIndex, MeanValueArray(1:MaxIndex),'k','linewidth',2);
hold on
plot(1:MaxIndex, MeanMinusStd(1:MaxIndex),'k--','linewidth',2);
hold on
plot(1:MaxIndex, MeanPlusStd(1:MaxIndex),'k--','linewidth',2);
hold on
set(gcf,'color','w');
set(gca, 'YLim', [0,1]);
xlabel('time(s)');
% set(gca, 'YLim', [min(cr(:)) - 5, max(cr(:)) + 5]);
ylabel('ratio of the major colors');
% legend('Mean for 7','Mean + std for 7','Mean - std for 7');
% set(h,'interpreter','latex')
saveas(gcf,'myfig7.fig');
save('MeanValueArray7.mat','MeanValueArray');
```

Appendix D.3.2: k = 9

```
k = 9;
RepeatTime = 100;
maxIter = 10000;
colorRecord = zeros(RepeatTime,maxIter);%record the ratios of the major color
colorArray = zeros(1,RepeatTime);
MaxIndex = 0;
for IndexRepeat = 1:RepeatTime
    [colorRecord(IndexRepeat,:), colorArray(IndexRepeat,:)] =
colorconsensusRand(k,IndexRepeat);
    colorRecord(IndexRepeat,:) = colorRecord(IndexRepeat,:)/100;
    LastIndex = find(colorRecord(IndexRepeat,:)>0);
    LastIndex = LastIndex(end);
    MaxIndex = max(MaxIndex,LastIndex);
end
for IndexRepeat = 1:RepeatTime
    LastIndex = find(colorRecord(IndexRepeat,:)>0);
    LastIndex = LastIndex(end);
    colorRecord(IndexRepeat,LastIndex + 1:MaxIndex) = 1;
end
colorRecord = colorRecord(:,1:MaxIndex);
MeanValueArray = mean(colorRecord,1);
StdArray = std(colorRecord,1,1);
for IndexRepeat = 1:RepeatTime
```

```
    if colorArray(IndexRepeat,:) == 1
        IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
        IndexTmp = IndexTmp(end) + 1;
        plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[0.8 0.8 1],'linewidth',0.5);
%         plot(1:MinIndex, NumberCount(1:MinIndex,2)/Total,'color',[1 0.8 0.8],'linewidth',0.5);
    else
        if colorArray(IndexRepeat,:) == 2
            IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
            IndexTmp = IndexTmp(end) + 1;
            plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[0.8 1 0.8],'linewidth',0.5);
        else
            IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
            IndexTmp = IndexTmp(end) + 1;
            plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[1 0.8 0.8],'linewidth',0.5);
%             plot(1:MinIndex, NumberCount(1:MinIndex,2)/Total,'color',[0.8 1 0.8],'linewidth',0.5);
        end
    end
    hold on
end
MeanMinusStd = MeanValueArray - StdArray;
```

```
MeanPlusStd = MeanValueArray + StdArray;
plot(1:MaxIndex, MeanValueArray(1:MaxIndex),'k','linewidth',2);
hold on
plot(1:MaxIndex, MeanMinusStd(1:MaxIndex),'k--','linewidth',2);
hold on
plot(1:MaxIndex, MeanPlusStd(1:MaxIndex),'k--','linewidth',2);
hold on
set(gcf,'color','w');
set(gca, 'YLim', [0,1]);
xlabel('time(s)');
% set(gca, 'YLim', [min(cr(:)) - 5, max(cr(:)) + 5]);
ylabel('ratio of the major colors');
% legend('Mean for 9','Mean + std for 9','Mean - std for 9');
% set(h,'interpreter','latex')
saveas(gcf,'myfig9.fig');
save('MeanValueArray9.mat','MeanValueArray');
```

Appendix D.3.2: k = 11

```
k = 11;
RepeatTime = 100;
maxIter = 10000;
colorRecord = zeros(RepeatTime,maxIter);%record the ratios of the major color
colorArray = zeros(1,RepeatTime);
MaxIndex = 0;
```

```
for IndexRepeat = 1:RepeatTime
    [colorRecord(IndexRepeat,:), colorArray(IndexRepeat,:)] = colorconsensusRand(k,IndexRepeat);
    colorRecord(IndexRepeat,:) = colorRecord(IndexRepeat,:)/100;
    LastIndex = find(colorRecord(IndexRepeat,:)>0);
    LastIndex = LastIndex(end);
    MaxIndex = max(MaxIndex,LastIndex);
end
for IndexRepeat = 1:RepeatTime
    LastIndex = find(colorRecord(IndexRepeat,:)>0);
    LastIndex = LastIndex(end);
    colorRecord(IndexRepeat,LastIndex + 1:MaxIndex) = 1;
end
colorRecord = colorRecord(:,1:MaxIndex);
MeanValueArray = mean(colorRecord,1);
StdArray = std(colorRecord,1,1);
for IndexRepeat = 1:RepeatTime
    if colorArray(IndexRepeat,:) == 1
        IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
        IndexTmp = IndexTmp(end) + 1;
        plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[0.8 0.8 1],'linewidth',0.5);
%         plot(1:MinIndex, NumberCount(1:MinIndex,2)/Total,'color',[1 0.8 0.8],'linewidth',0.5);
    else
```

```
        if colorArray(IndexRepeat,:) == 2
            IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
            IndexTmp = IndexTmp(end) + 1;
            plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[0.8 1
0.8],'linewidth',0.5);
        else
            IndexTmp = find(colorRecord(IndexRepeat,:) < 1);
            IndexTmp = IndexTmp(end) + 1;
            plot(1:IndexTmp, colorRecord(IndexRepeat,1:IndexTmp),'color',[1 0.8
0.8],'linewidth',0.5);
%            plot(1:MinIndex, NumberCount(1:MinIndex,2)/Total,'color',[0.8 1
0.8],'linewidth',0.5);
        end
    end
    hold on
end
MeanMinusStd = MeanValueArray - StdArray;
MeanPlusStd = MeanValueArray + StdArray;
plot(1:MaxIndex, MeanValueArray(1:MaxIndex),'k','linewidth',2);
hold on
plot(1:MaxIndex, MeanMinusStd(1:MaxIndex),'k--','linewidth',2);
hold on
plot(1:MaxIndex, MeanPlusStd(1:MaxIndex),'k--','linewidth',2);
hold on
set(gcf,'color','w');
```

```
set(gca, 'YLim', [0,1]);

xlabel('time(s)');

% set(gca, 'YLim', [min(cr(:)) - 5, max(cr(:)) + 5]);

ylabel('ratio of the major colors');

% legend('Mean for 11','Mean + std for 11','Mean - std for 11');

% set(h,'interpreter','latex')

saveas(gcf,'myfig11.fig');

save('MeanValueArray11.mat','MeanValueArray');
```

Appendix D.3.2: Comparing k = 7, 9, 11

```
MeanValueArray7 = load('MeanValueArray7');

MeanValueArray7 = MeanValueArray7.MeanValueArray;

MeanValueArray9 = load('MeanValueArray9');

MeanValueArray9 = MeanValueArray9.MeanValueArray;

MeanValueArray11 = load('MeanValueArray11');

MeanValueArray11 = MeanValueArray11.MeanValueArray;

MaxIndex = length(MeanValueArray7);

plot(1:MaxIndex, MeanValueArray7,'r','linewidth',2);

hold on

MaxIndex = length(MeanValueArray9);

plot(1:MaxIndex, MeanValueArray9(1:MaxIndex),'b','linewidth',2);

hold on

MaxIndex = length(MeanValueArray11);

plot(1:MaxIndex, MeanValueArray11(1:MaxIndex),'g','linewidth',2);

hold on

set(gcf,'color','w');
```

```
set(gca, 'YLim', [0,1]);

xlabel('time(s)');

% set(gca, 'YLim', [min(cr(:)) - 5, max(cr(:)) + 5]);

ylabel('ratio of the major colors');

legend('Mean for 7','Mean for 9','Mean for 11');

% set(h,'interpreter','latex')

saveas(gcf,'myfig_ratio.fig');

set(gcf,'PaperPositionMode','auto','PaperSize',[8,4], 'PaperPosition',[0,0,8,4] );

print(gcf, '-dpdf', 'meanplot.pdf');
```

References

[1] *https://www.dropbox.com/s/0zvompid7dnzbxf/detectupdate1121_3.m?dl=0*.

[2] https://vimeo.com/56647019

[3] M. Betke et al., Tracking Large Variable Numbers of Objects in Clutter, IEEE CVPR, 2007.

[4] Yang, Hanxuan, et al. "Recent advances and trends in visual tracking: A review." *Neurocomputing* 74.18 (2011): 3823-3831.

[5] *https://www.dropbox.com/s/97h3jp4ist5tlvv/First10Min.mp4?dl=0*

[6] Jeong, Jong-Min, Tae-Sung Yoon, and Jin-Bae Park. "Kalman filter based multiple objects detection-tracking algorithm robust to occlusion." *SICE Annual Conference (SICE), 2014 Proceedings of the*. IEEE, 2014.

[7] Cuevas, Erik V., Daniel Zaldivar, and Raul Rojas. "Kalman filter for vision tracking." (2005).

[8] Jain, Anil K., M. Narasimha Murty, and Patrick J. Flynn. "Data clustering: a review." *ACM computing surveys (CSUR)* 31.3 (1999): 264-323.

[9] Kanungo, Tapas, et al. "An efficient k-means clustering algorithm: Analysis and implementation." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.7 (2002): 881-892.

[10] Hargrave, P.J, "A tutorial introduction to Kalman filtering", Kalman Filters: Introduction, Applications and Future Developments, IEE Colloquium on, 1989, pp. 1-6

[11] *https://www.dropbox.com/s/n0xiimufsp552pl/colorizeUmbrellaData.m?dl=0*

[12] *https://www.dropbox.com/s/e3f22ouoqj5r17p/ProcessedUmbrella.mp4?dl=0*

[13] Wei, Lai, Weiming Zeng, and Hong Wang. "K-means Clustering with Manifold." Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on. Vol. 5. IEEE, 2010.

[14] Esteves, Rui Máximo, Thomas Hacker, and Chunming Rong. "Competitive k-means, a new accurate and distributed k-means algorithm for large datasets." *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. Vol. 1. IEEE, 2013.

[15] Rodriguez, M, Ali, S, and Kanade, T, "Tracking in Unstructured Crowded Scenes", Computer Vision, 2009 IEEE 12th International Conference on, 2009, pp.1389-1396.

[16] J. Hartigan, M. Wang, A K-means clustering algorithm, Applied Statistics 1979, 28:100-108

[17] Zhao, Tao, and Ram Nevatia. "Tracking multiple humans in complex situations." Pattern Analysis and Machine Intelligence, IEEE Transactions on 26.9 (2004): 1208-1221.

[18] Y.Cai et al., Robust Visual Tracking of Multiple Targets, ECCV, 2006.

[19] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT press, 2005.

[20] Ramakoti, Nimmakayala, Ari Vinay, and Ravi Kumar Jatoth. "Particle swarm optimization aided Kalman filter for object tracking." Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT'09. International Conference on. IEEE, 2009.