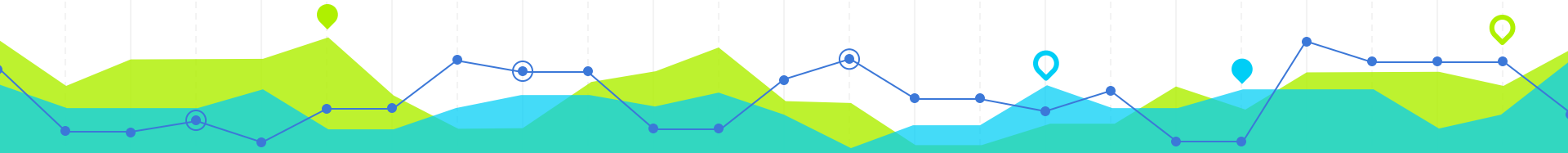
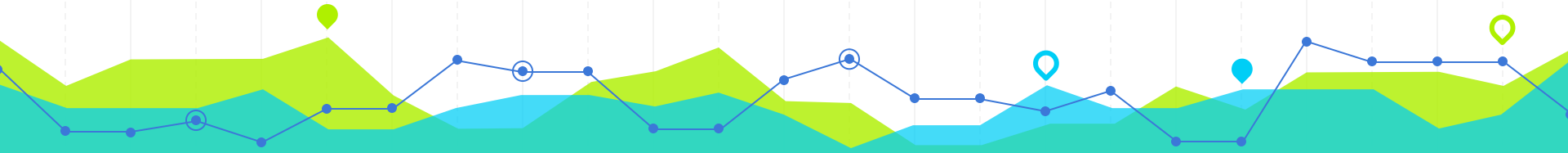


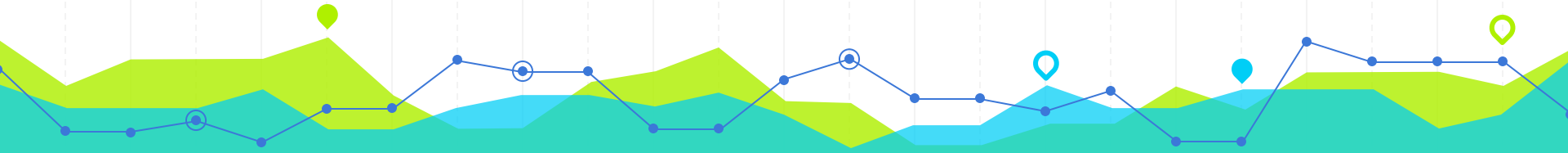
Finding the Effectiveness of

Repellent Substances on



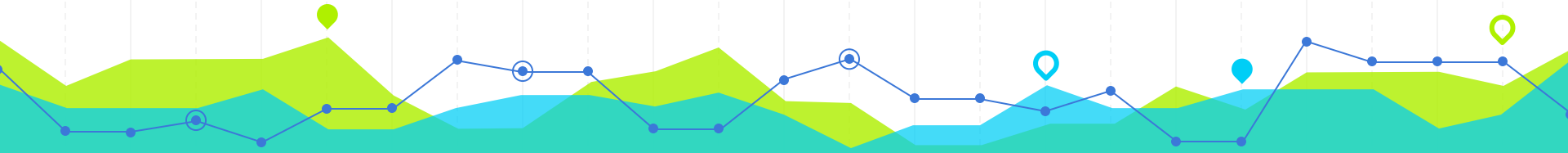
Ants w/ Computer Vision

& Raspberry Pi



THE PROCESS

The Code, Hardware, Procedure, and Materials



RESULTS

Analysis, Conclusion, Problems, and Implications

Hypothesis

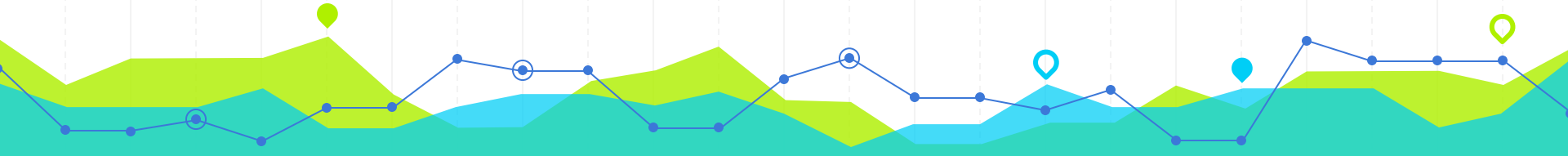
If Computer Vision is used, then it should be possible to accurately detect the amount of ants in an image to find the repellent success of different substances on those ants.

Scientific Question

Is it possible to create code that accurately identifies/detects ants in a picture to find the effectiveness of repellents on ants?

Purpose

The purpose of this project is to be able to measure the effectiveness of common household ant repellents through the detection of ants. This can be done by identifying the number of ants through computer vision techniques using the cheap Raspberry Pi and open source Python computer language with some of its packages.



Terms

1. Package -- Blocks of code with specific methods and purposes from various preprogrammed open libraries that can be imported and utilized.
2. Raspberry Pi -- A small, cheap computer.
3. Cv2/OpenCV -- The OpenCV package that contains code that can be imported from the OpenCV library, which runs through images to detect items.
4. Jupyter Notebook -- A Python IDE that can run and demonstrate results of code in an organized manner.
5. Sqlite -- A database management system in the C language
6. Scikit-image -- A package that processes images, and is normally paired with NumPy and SciPy to extract and realize data out of those images.

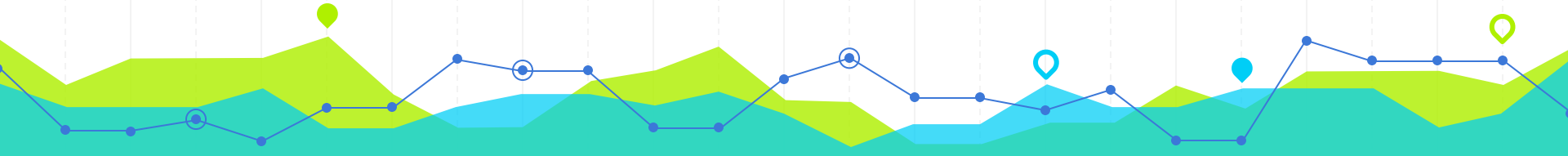
Building the Code

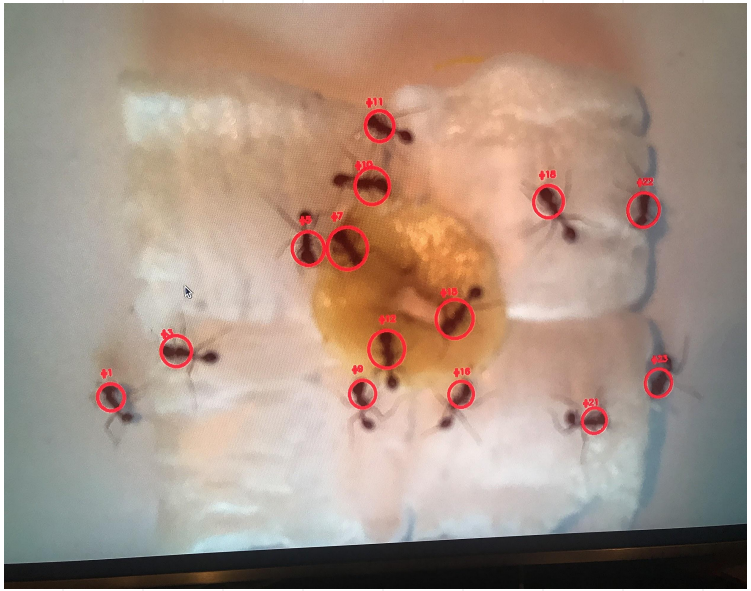
```
def count_ants(image):
    print("reading image: {}".format(image))
    image = cv2.imread(image)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # blurred = cv2.GaussianBlur(gray, (11, 11), 0)

    thresh = cv2.threshold(gray, GRAY_2ND_ARG, 255, cv2.THRESH_BINARY_INV)[1]
    #thresh = cv2.threshold(blurred, 80, 255, cv2.THRESH_BINARY_INV)[1]

    # perform a series of erosions and dilations to remove
    # any small blobs of noise from the thresholded image
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=4)
```

The first step was to develop the detection code with black dots on a white background to simulate ants. A Python environment was used on Raspberry Pi with a camera to enable computer vision to detect the ants. The detection process started by taking a picture and plugging that image into the computer program. The program would then detect dark spots of certain sizes.

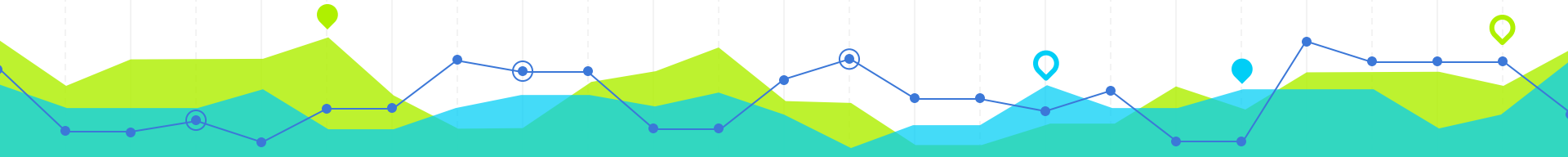




To single out the ants, the cv2 package was utilized. The color of the image would then be gray-scaled and checked with a threshold value of the color. If the color went over this threshold value, in this case 50, the color of the object becomes completely white, or 255. This was then inverted to identify white on black background. The second step was to label the areas that the code detected as white relative to the black background and count these spots. This was done by measuring the images through the skimage package and masking over only the “large” spots after running through the labels to eliminate possible extra noise by comparing them to a certain number of pixels.

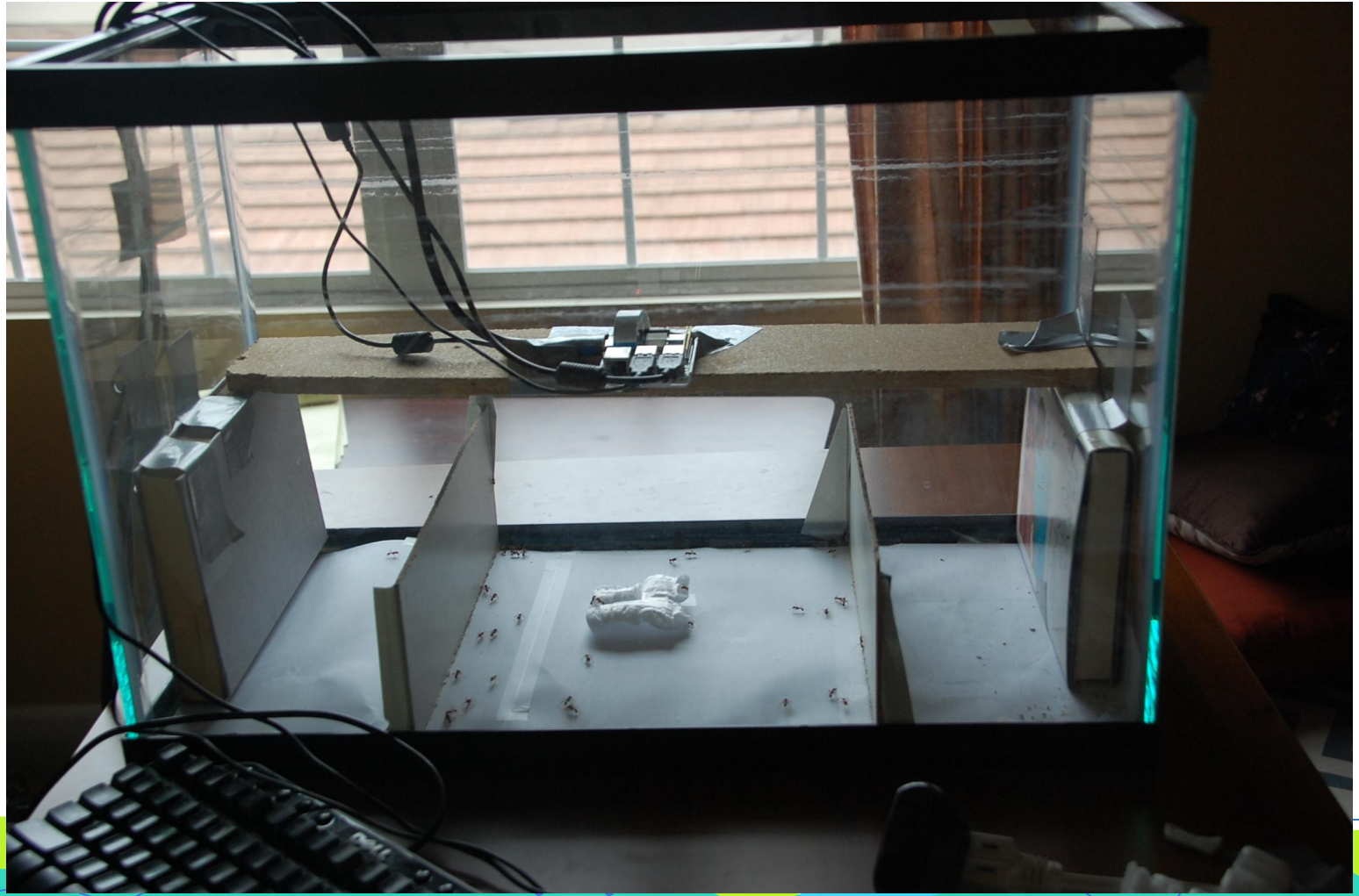
After this was sorted, the code ran through the spots in the masked image and takes the contours of the spots to get the smallest enclosing circle possible. This creates a circle around the spot and labels it, allowing for easy identification and counting of the spots.

The third step was to clean up and store the data. After the labels and circles were created, larger and smaller circles interfered and added to the count. To eliminate this, a limit was set on the radius to skip creating circles that were too large or too small to stop interfering circles. Once the proper circle count was found, a connection to a sqlite database file that had a table of the repellent, image file name, and ant count in the image, was created to store the data. After this was successfully created, the final step was to take pictures on minute intervals through the Raspberry Pi. With the picamera package and the camera that was connected to the Raspberry Pi, a filename with the directory path, and datetime down to the seconds was created to specify and be able to differentiate the images. After the camera captured the image it would be held on preview for another minute for 40 images to allow ant movement.



Hardware

Setting up the Raspberry Pi unexpectedly became another part of the experiment. Due to the focal length being unchanging and the numerous wires that had to be connected to the Raspberry Pi, a plank had to be able to hold the Raspberry Pi in place inside of the case for long periods of time. Without proper available material to sturdily stay inside of the lengthy case, a wooden piece had to be cut from a larger piece. With this piece, the Raspberry Pi could be duct-taped into place, with the HDMI, keyboard, and mouse cables all being taped to the outside of the case to stay in place while hanging over the edge. Then, books were kept on the outside to keep the Raspberry Pi at the right height with its focal length.



Materials

- Raspberry Pi
- Computer
- Wood
- Cardboard
- Stimulating food (banana)
- Stands (books)
- White styrofoam
- Printer paper
- Glass case/environment
- Ants
- Brush
- Repellent Material
  - Vinegar, Cayenne, Peppermint, Lime, Limespray

Procedure

1. Get a glass tank and place printer paper on the bottom
2. After the code and hardware involving the Raspberry Pi is finished, place two sliding cardboard pieces to cut off the far sides for the ants in the environment.
3. Place the plank with the Raspberry Pi on stands that are outside the middle area and the proper height for good detection
4. Place the ants inside the middle area
5. Create a small table with white styrofoam pieces and place it under the camera
6. Place the ants in the area
7. After 30 minutes without anything, place food on top of the styrofoam table
8. Turn on the camera for 40 minutes
9. After the camera is done, remove the table with the food and scrape off any ants with a brush
10. Repeat this with repellent surrounding the food

Using Jupyter to Compile Data

In order to efficiently analyze the data Jupyter provided an excellent platform and toolset. Jupyter enable the use of a Notebook (Jupyter Notebook) and built graphs after analyzing the data. First, data was read from the database and converted to a dataframe. Then, data was cleaned by deleting extra rows and correcting missing data. After this, some basic statistical insights (min, max, count, etc.) were achieved by describing data with pandas & numpy packages. Then matplotlib and bokeh graphed the data for visualization. The data was subgrouped by each repellent and with number of ants as dependent variable, created a box and whisker plot, density graph, and line graph to represent the data.

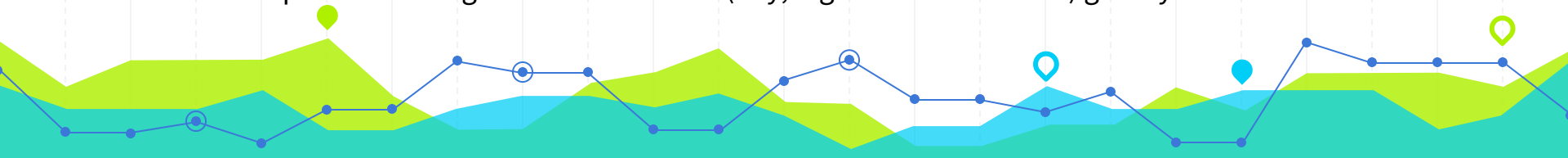
Data Analysis

The data clearly shows that there are differences between the substances used as repellents. As the box-and-whisker plot shows, vinegar had an lowest median value of number of ants present around the food, while the cayenne had the most ants present on average, ignoring the banana by itself. The decreased variability of the vinegar is shown through the smaller range between the higher quartile and lower quartile, further supporting the fact that vinegar had the most effectiveness as a repellent for ants compared to the other substances. The line graph also shows that, over time, cinnamon and vinegar track similarly. The box plot shows this as well, suggesting that cinnamon and vinegar may have something in common that repels ants, as ants communicate and identify things largely through pheromones. The density graphs help show how often amounts of ants were detected for the different solutions. Vinegar and cinnamon were once again close, but vinegar was clearly shown to have less general ants in the area.

Problems

There were many problems that needed to be worked out throughout this project. For example:

1. Coding
    a. This involved building the code to fit the fixed focal length of the camera. Since the image quality would be different at different distance, program need to be built to handle with thresholds.
    b. Shadows also became a problem with lights and time of experiment casting shadows which would affect image analysis that looked for black or gray spots on white background.
2. Handling the Ants
    a. At the beginning the ants were impossible to work with. They scattered to many different areas, particularly corners. The original experiment involved a bridge or cantilever to the food with repellent between, but it did not work at all. The experiment had to be changed to a more general area that the ants were confined in, with the food in the middle and repellent around the food.
    b. Some time ants died within few days of getting them shipped from North Carolina. It seemed even temperature changes inside the house (day, night and other times) greatly affected them.



Conclusion

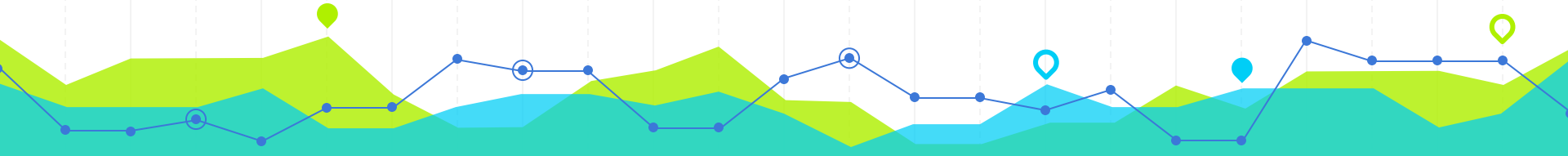
In the end, it was possible to build code that was able to accurately identify the location of ants through contrasts in color of the images given. While there certainly could have been some errors, such as the clumping of ants to cause the code to miscount or undercount by identify all of the clumped ants as one, the overall data was consistent and the code was able to detect ants that were little bit separated. It could also remove noisy spots, enabling accurate count of the ants. However, this could definitely become more accurate, especially through a better camera that has adjustable focal point. Simply put, the code was very effective in detecting ants and gathering data for analyzing the effectiveness of different substances in repelling ants. With the detection and counting of ants, it became possible to recognize that, out of all of the substances, vinegar was the most effective in repelling the ants with cinnamon coming close.

Errors

With fixed camera focal point, and software algorithm dependency on pixel count there were possibility of introducing some errors. However, one error that affected the experiment but manually corrected was with the clumping of dark spots that the code interpreted as one large spot. This meant that 5 ants were being considered as 1 in some cases, mostly during the food (banana) only segment of the experiment because ants were gathering on top of the banana without moving, while this did not occur in the other trials. Another error could be caused by the camera, which was blurry at times and gave corrupted data. These specific cases were filtered out in the data, but it still leaves room for error. Unaccounted for variables like the room temperature, scents of the room, or ant biology itself (such as eating behaviors, lifespan, etc.) could also have played a role.

Implications

While seemingly small in scale this could have large implications. Being able to detect ants in a radius could lead to the identification of the number of insects or objects in a source, especially clear one, like water at a given moment, just as a specific example. The idea behind detecting different objects could help at any scale where there is an image, as quick detection of anomalies, outliers, or even mass data could quicken processes to a very large scale. No longer is manual collection of data needed then; everything can be left up to computers. Just as how the effectiveness of several substances as repellents for ants could be measured through the simple detection and counting of ants in an image, many more experiments could utilize the detection of foreign and different objects at a given moment relative to its surrounding. Extracting information from images could be applied to almost anything now.



All Pictures Taken by self. All observations and data are from this experiment.