CAP6665 - Project 1

Terrance Williams

June 21, 2023

Contents

1 Introduction **2**

2 Implementation **2**
2.1 Detection 2
2.1.1 Color Masking 2
2.1.2 Circle Detection 3
2.2 ROS Integration 4

3 Analysis **6**
3.1 Multiple Balls 6
3.2 JetHexa Parameters 7

4 Flowchart **9**

5 Conclusion **9**

1 Introduction

CAP6665 is a project-based graduate course where students learn the fundamentals of computer vision. The target project for this semester is the development and implementation of a tennis ball tracker: the platform used for implementation should be able to identify tennis balls, ascertain its relative angular position, and provide corrective movements to constantly adjust its orientation to face the abll. For Project 1, the ball detection method was implemented.

2 Implementation

Project 1 was segmented into two sub-components. The first was the detection of the target object, and the second was the integration of this detection method into the platform through the Robot Operating System (ROS).

2.1 Detection

The detection method chosen for this project uses a combination of color masking and circle detection. Tennis balls typically range along the yellow-green color spectrum, and are always spherical, which appears as circular in a 2D image. This detection model was chosen due to its approachable implementation and the presence of multiple computer vision techniques: color masking and feature detection.

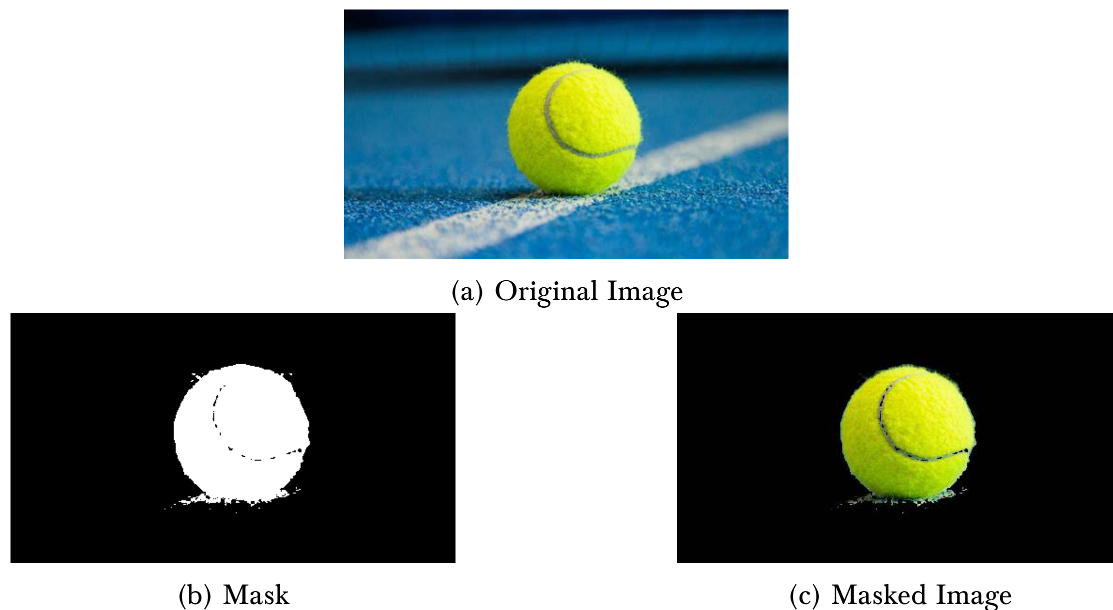
2.1.1 Color Masking

The detector filters the image for color first. This is done to minimize the number of circular features detected by the detector, working to obviate the need for additional filter work later in the process. As mentioned previously, tennis balls are typically along yellow or green in color, so these are the colors used for image color filtering. The filtering works as follows[1]:

First, the image is loaded into the program and converted to the HSV colorspace. This conversion is done because hue thresholding is considerably easier in this space than the traditional RGB space, with each hue along the spectrum (Red, Yellow, Green, Cyan, Blue, and Magenta) being represented in intervals of 30 unit values in OpenCV [1]. So, for example, yellow can be represented through hue values of $[30, 59]$ while blue would be $[120, 149]$. Naturally, the range of values used for color filtering is subject to the specific implementation conditions, including the camera model and environmental lighting.

[1]The color detection code is a modified version of the color detection code I used in the EML6805 course

The HSV image and the user-tuned color-threshold are then used to create a mask. This mask is then used on the original image in a bitwise_and() operation, effectively mapping all pixels in the original image that are outside of the threshold to black. The resulting image only includes the desired color.



(a) Original Image

(b) Mask

(c) Masked Image

Figure 1: Color Filtering Results[2]

2.1.2 Circle Detection

The next step in the process was the circle detection. The goal of this section was to be able to run an algorithm to detect circular contours on the masked image. Circular shapes on the masked images are then assumed to be tennis balls. Naturally, the efficacy of this approach depends on the surrounding environment, but this will be discussed more thoroughly in a later section.

To implement this step, the masked image is first converted to gray-scale. Next, OpenCV's **HoughCircles** function is used, which takes in various arguments for tuning, including minimum distance between detected circle centers, Hough Gradient parameters, and radii limits [2][3]. Finally, the detected circle centers and approximated radii are used to draw the circles onto the original image. An example may be seen in Figure 2.

[2]Image Source:
https://www.today.com/news/are-tennis-balls-yellow-or-green-roger-federer-enters-debate-t125444

Figure 2: Detected circle

2.2 ROS Integration

The second component of Project 1 was coding the algorithm into the robot platform. The robot used for this project is the JetHexa hexapod robot, which runs a ROS Melodic distribution on an Ubuntu Linux OS. To incorporate the tennis ball detector into the JetHexa, it needed to be "ROS-ified", or implemented in a ROS-idiomatic way. ROS facilitates asynchronous communication through the concept of message publishers and subscribers. This publisher-subscriber system was constructed as a ROS package for the ball detector. A flow diagram of the publisher-subscriber relationship is show in Figure 6.

The publisher script uses Python OpenCV to stream images from the JetHexa's camera. OpenCV uses a different image format than that of ROS, so a conversion package is used to convert the captured image into a ROS Image message [4][5]. This message is then published on the user-defined topic (currently named *image_hub*).

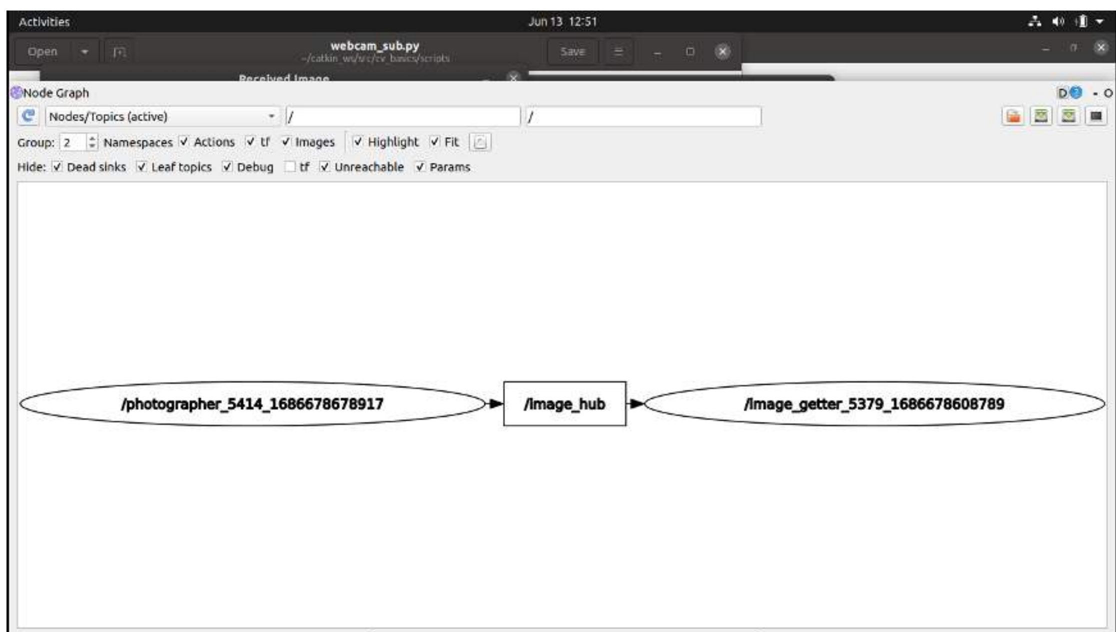


Figure 3: Resulting node communication (via rqt_graph)

The subscriber script connects to the *image_hub* topic and processes the Image messages. It uses the aforementioned *cv_bridge* package to convert the message into an OpenCV-compatible image. This image is then processed using the ball detection algorithm. Additionally, to minimize the effects of false positives, the subscriber keeps track of consecutive detection count. Once the detection count reaches a predefined threshold (currently defined as 10), the program can consider the ball to be "detected". This threshold will likely need to be adjusted in the near future.

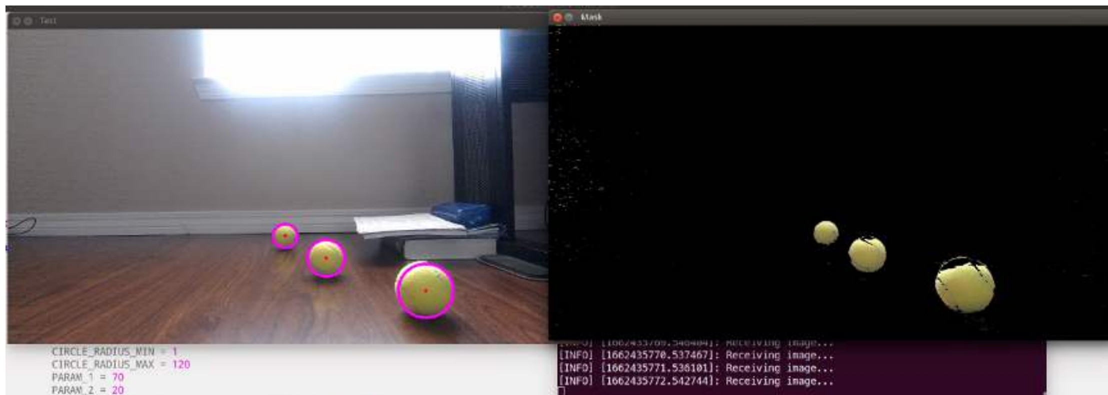


Figure 4: Results of the ROS implementation

3 Analysis

3.1 Multiple Balls

It was observed that multiple balls in a given image results in a combination of false positives (non-ball detected) and false negatives (balls not detected). After experimenting with parameters, it was concluded that this behavior is likely the result of the following:

1. minDist parameter - The minimum distance between detected circle centers. Increasing this distance curbs false positives, but can increase false negatives. As a compromise, I take the value to be the maximum value of a fraction of either the image height or width:

   $$minDist = \max\left(\frac{\text{height}}{4}, \frac{\text{width}}{4}\right)$$

   The denominator was derived by trial and error.

2. Circle Radii - The minimum radius needs to be small enough to detect balls that are far away. The max radius, however, must be large enough to detect balls close to the camera, but not so large that false positives are detected.

3. Hough Gradient Params - These are rather sensitive parameters. Increasing Parameter 2 by even 1 unit can result in drastically different results, even to the point of detecting multiple circles for the same ball or not detecting the ball at all. Parameter 1 had a less drastic effect, but it was still noticeable, including false positives even on images with only one tennis ball present.

Figure 5 displays the effects of changing only the minDist parameter. For each image pair, the left image uses a lower distance minimum compared to the right image (by a factor of about 0.5).

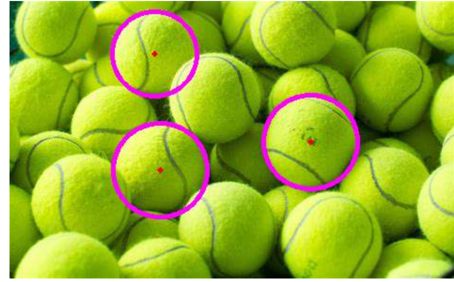
3.2 JetHexa Parameters

After testing on the JetHexa environment, the following parameter configuration was determined:

- $minDist = \min\left(\frac{\text{height}}{8}, \frac{\text{width}}{8}\right)$
- Radius Range: $[1, 120]$ px
- Parameter 1: 50
- Parameter 2: 25

[3] Image Sources:
https://www.freeimages.com/photo/tennis-racquets-and-balls-1414737
https://www.tasteofhome.com/collection/reasons-need-to-buy-more-tennis-balls/
http://7-themes.com/7007704-tennis-ball-photography.html

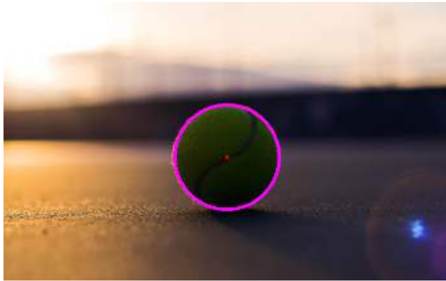
(a) Dist. 1



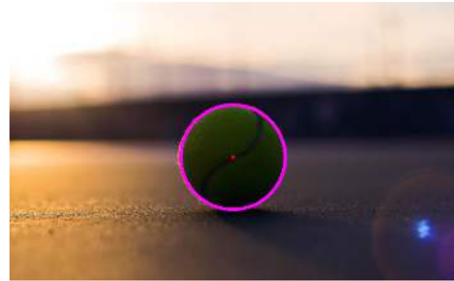
(b) Dist. 2

(c) Dist. 1

(d) Dist. 2



(e) Dist. 1



(f) Dist. 2

Figure 5: Effects of minimum distance. Notice the solo image result in the same detection compared to multi-balled images.[3]

4 Flowchart

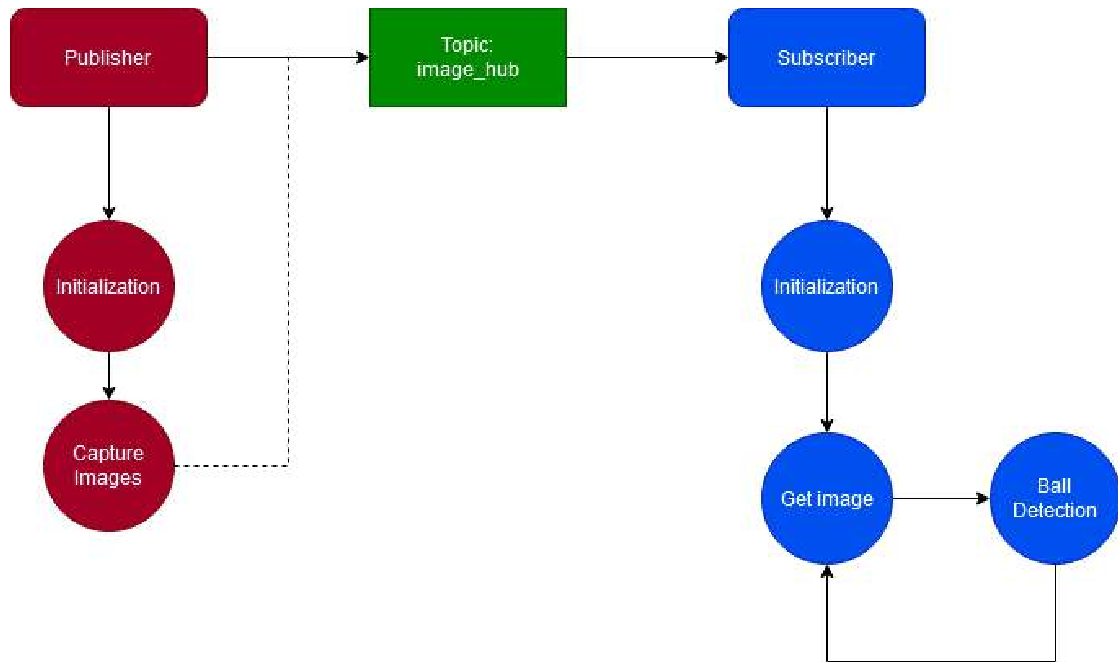


Figure 6: A flow diagram showing the communication path of the ROS publisher and subscriber nodes.

5 Conclusion

In this project, a tennis ball detector was programmed using OpenCV Python and implemented on a ROS-based robot. A publisher-subscriber relationship between nodes allows images to be captured from the robot and processed to determine the presence of a ball. Currently, the detection algorithm is not "specific" to tennis balls; any circular, green object can be detected by the program. To improve this functionality, a machine-learning approach may be a viable option, allowing for a more robust detection mechanism.

Additionally, the efficacy of the detection algorithm varies with lighting conditions. If possible, condition-based color thresholding would allow the user to determine which threshold values to use depending on whether the robot is inside or outside, improving the flexibility of the program.

For the next project, a "facing" behavior will need to be implemented. The aim is to use the centers of a detected ball to determine how the robot should rotate to constantly face the ball.

References

[1] *Changing Colorspaces*, https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html, Accessed: 15 June 2023, Open Source Computer Vision.

[2] *Feature Detection: HoughCircles()*, https://docs.opencv.org/4.6.0/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d, Accessed: 5 June 2023, Open Source Computer Vision.

[3] *Hough Circle Transform*, https://docs.opencv.org/4.6.0/d4/d70/tutorial_hough_circle.html, Accessed: 5 June 2023, Open Source Computer Vision.

[4] *Converting between ROS images and OpenCV images (Python)*, https://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython, Accessed: 13 June 2023, Open Robotics.

[5] *Working With ROS and OpenCV in ROS Noetic*, https://automaticaddison.com/working-with-ros-and-opencv-in-ros-noetic/, Accessed: 15 June 2023, Automatic Addison.

```python
#!/usr/bin/env python
# coding: utf-8

"""
Title: Photographer Node
Author: Terrance Williams
Date: 13 June 2023
Description: Creates a ROS Publisher to transfer images to the /image_hub topic

Credit: Addison Sears-Collins
https://automaticaddison.com/working-with-ros-and-opencv-in-ros-noetic/

NOTE: This is a Python 2 script
"""

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2 as cv
import sys

def publish_msg():
    PUB_RATE = 10 # Hz (same as FPS in this case?)
    # ROS setup
    pub = rospy.Publisher("image_hub", Image, queue_size=1)  # adjust the queue_size
    rospy.init_node("JH_camera", anonymous=False) # Only one camera
    rate = rospy.Rate(PUB_RATE)

    # Create ROS <--> OpenCV Bridge
    br = CvBridge()

    # OpenCV Image Capture
    cap = cv.VideoCapture(0)  # capture JetHexa camera

    if not cap.isOpened():
        rospy.signal_shudown("Could not open camera." )

    # Capture images and send
    while not rospy.is_shutdown():
        ret, frame = cap.read()
        if ret:
            #rospy.loginfo("Sending Image")
            # scale image down (16:9 ratio width:height)
            height, width, _ = frame.shape
            height, width = int(height/2), int(width/2)
            #height = 720
            #width = int(16*(height/9))
            down_frame = cv.resize(frame, (width, height), interpolation=cv.INTER_AREA)
            # send image
            msg = br.cv2_to_imgmsg(down_frame)
```

```
51            pub.publish(msg)
52
53        rate.sleep()
54
55
56  if __name__ == '__main__':
57      try:
58          publish_msg()
59      except rospy.ROSInterruptException:
60          pass
61
```

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  """
5  Title: Tennis Ball Detector Node
6  Author: Terrance Williams
7  Date: 13 June 2023
8  Description: Creates a ROS Subscriber to transfer images to the /image_hub topic
9
10 Credit: Addison Sears-Collins
11 https://automaticaddison.com/working-with-ros-and-opencv-in-ros-noetic/
12
13 NOTE: This is a Python 2 script
14 """
15
16 import rospy
17 from sensor_msgs.msg import Image
18 from cv_bridge import CvBridge
19 import cv2 as cv
20 import numpy as np
21
22
23 # Define Globals
24 SATURATION_LOWER = 50
25 SATURATION_MAX = 255  # Max 'S' value in HSV
26 BRIGHTNESS_LOWER = 20
27 BRIGHTNESS_MAX = 255  # Max 'V' Value in HSV
28 YELLOW_LOWER = 22
29 GREEN_UPPER = 85
30 TENNIS_THRESH = 10 # Number of consecutive detections needed to be a "true" detection.
31 count = 0
32
33 def img_mask(image):
34 # input raw image
35 # outputs array of masked images (red, yellow, green, blue, original)
36     img_hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
37
38     # Generate HSV Threshold (yellow to light blue)
39     color_thresh = np.array([[YELLOW_LOWER, SATURATION_LOWER, BRIGHTNESS_LOWER],
40                              [GREEN_UPPER, SATURATION_MAX, BRIGHTNESS_MAX]])
41
42     # Generate Mask
43     color_mask = cv.inRange(img_hsv, color_thresh[0], color_thresh[1])
44
45     img_masked = cv.bitwise_and(image, image, mask=color_mask)
46
47     return img_masked
48
49
50 def houghCircles(image):
```

```
51
52 # Gray image
53     imgray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
54     img = cv.medianBlur(imgray, ksize=5)
55
56     # Hough Params
57     rows, cols= img.shape[0:2]
58     DIST = min(rows / 8, cols/8)
59
60     CIRCLE_RADIUS_MIN = 1
61     CIRCLE_RADIUS_MAX = 120
62     PARAM_1 = 50
63     PARAM_2 = 25
64     circles = cv.HoughCircles(img, cv.HOUGH_GRADIENT, dp=1,
65                           minDist=DIST,
66                           param1=PARAM_1, param2=PARAM_2,
67                           minRadius=CIRCLE_RADIUS_MIN,
68                           maxRadius=CIRCLE_RADIUS_MAX)
69
70     return circles
71
72
73 def callback(msg):
74     global count
75     last_count = count
76     # ROS <--> OpenCV bridge
77     br = CvBridge()
78
79     # ball_found = False
80
81     # get message data
82     #rospy.loginfo("Receiving image...")
83     img = br.imgmsg_to_cv2(msg)
84
85     # Color filter the image
86     masked = img_mask(img)
87     # Apply Hough
88     '''https://docs.opencv.org/4.6.0/d4/d70/tutorial_hough_circle.html'''
89     circles = houghCircles(masked)
90     # Draw circles; Track consec. detections
91     if circles is not None:
92         count += 1
93         circles = np.uint16(np.around(circles))
94         for j in circles[0, :]:
95             center = (j[0],j[1])
96             # circle center
97             cv.circle(img, center, 1, (0, 0, 255), 3)
98             # circle outline
99             radius = j[2]
100            cv.circle(img, center, radius, (255, 0, 255), 3)
```

```
101
102     # Thresholding
103     if last_count == count:
104         count = 0
105         last_count = count
106     else:
107         last_count = count
108     print "Detection Count: {}".format(count)
109     if count >= TENNIS_THRESH:
110         print("Ball Detected!")
111         ''' PUT BOOLEAN PUBLISH COMMAND HERE IF DESIRED'''
112     # Display images
113     cv.imshow("Mask", masked)
114     cv.imshow("Test", img)
115     cv.waitKey(1)
116
117
118 def img_sub():
119
120     # ROS Setup
121     rospy.init_node("ball_detector", anonymous=True)
122     name = rospy.get_name()
123     TOPIC = "image_hub"
124     sub = rospy.Subscriber(TOPIC, Image, callback)
125     rospy.loginfo("{}: Beginning listener.".format(name))
126
127     # Don't do anything until the exit
128     rospy.spin()
129     cv.destroyAllWindows()
130
131
132 if __name__ == '__main__':
133     '''TO-DO: Add conditional argument "inside":bool '''
134     img_sub()
135
```

CAP6665 Project 2 Report

Terrance Williams

July 27, 2023

Contents

1 Introduction **2**

2 Handling Noise **2**
- 2.1 Parameter Tuning . . . 2
- 2.2 k-Means Clustering . . . 4
  - 2.2.1 Justification . . . 4
  - 2.2.2 Determining the Amount of Clusters . . . 4
  - 2.2.3 Setting Initial Means . . . 5
- 2.3 Choosing the Correct Cluster . . . 6
  - 2.3.1 Metrics . . . 6
  - 2.3.2 Density Calculation . . . 6

3 Selecting the Winning Cluster **11**

4 ROS Implementation **16**
- 4.1 Structure . . . 16
  - 4.1.1 Program Structure . . . 16
  - 4.1.2 Communication . . . 16
- 4.2 Package Construction . . . 19

5 Results **20**

6 Flowchart **21**

7 Conclusion **22**

A Code **23**

B Acceleration Plots **23**

1 Introduction

Project 2 continues the work done in Project 1. In the previous project, a basic tennis ball detection algorithm was used to detect one or more tennis ball in a given camera image. The detection algorithm used a combination of color filtering and Hough Circle detection—which detects circle centers—to look for the characteristics of a tennis ball, namely its yellow color and spherical shape. As a result of the implementation method, uncertainty in the results were introduced to the system in the form of noise, making it more difficult to distinguish true detections from false positives.

This project aims to correct this behavior by fine-tuning the detection method and creating a way to differentiate the real tennis balls from the noise. Finally, once the real tennis ball is determined, the angle between the robot and the ball will be calculated and used to perform corrective rotation, resulting in the robot directly facing the ball.

2 Handling Noise

Phase One of the project aimed to reduce the effects of noise on the system, beginning with minimizing its overall presence. Throughout this project, initial tests were done in a testing environment[1] to then be incorporated into the robot after the concepts were proven to work. This section will discuss the results of the testing environment in order to separate the general concepts from the ROS-specific implementation details (which add more complexity).

2.1 Parameter Tuning

To reduce the total amount of noise in the system, I performed real-time parameter tuning. OpenCV provides the ability to create trackbars for various parameters of the image, and coupled with video streaming, one can get live feedback of how a given variable setting affects the performance of the algorithm, [1][2]. I tested four 'categories' of control variables: color masking (HSV values), Hough Circle parameters, image contrast, and image smoothing. Minor tweaks were made to the acceptable range of HSV values as well as the Hough parameters and contrast, however the greatest positive effect on noise reduction was how the image was smoothed. In Project 1, captured images were only filtered one time by using a median blur filter. What I found through tuning was that a combination approach has better performance, especially when done multiple times per image. The presence

[1] Testing environment included my laptop and office space.

of noise in the testing environment was reduced considerably by passing each image through a median+Gaussian filter three times. The result was a much smoother image that detected round shapes more accurately. Of course, the specific parameter values change when using the robot's camera, but the same result was observed from improved filtering.

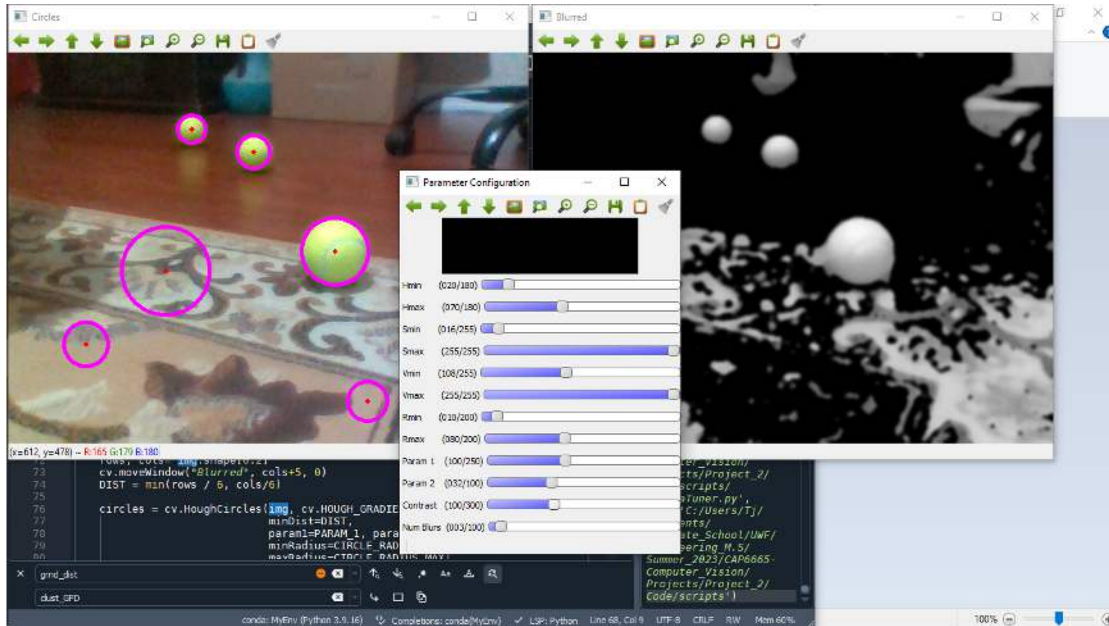


Figure 1: Trackbar Tuning Example (Test Environment)

2.2 k-Means Clustering

After adjusting the parameter values, the next step was to separate the remaining noise from the desired data. As seen in Figure 1, even with parameter tuning, undesired detections are still present in the system. To separate these outlier points from the tennis ball data, I used k-Means clustering.

2.2.1 Justification

The data's characteristics were the primary driving factor for choosing k-means clustering as the project's data-filtering method. Because the robot is meant to collect tennis balls, it can be assumed that the tennis balls of interest will be stationary. Since the robot will also remain motionless when scanning for balls, the Hough-detected circle centers corresponding to tennis balls should be positioned close together over the course of the detection period. Noise, however, tends to have a more widely-spread distribution. As a result, if the data can be segmented into clusters, those with small pixel area (high density) are more likely to be tennis balls.

2.2.2 Determining the Amount of Clusters

In order for k-means to perform properly, however, the correct number of clusters—the k value—must be chosen according to the data. The presence of system noise is dynamic, so the choice of k must be dynamic as well. Otherwise, if there is a detection period with very little noise for example, a constant k-value may result in multiple clusters whose points all belong to one tennis ball, effectively segmenting one tennis ball into multiple.

Recall from Project 1 that the ball detection algorithm requires a number of consecutive detections, α, for a detection to officially occur. This α is the detection period (ex. 75 images). Throughout this period, there will be a frame that has the largest number of individual circle center detections. I use this image to determine the k value for a given clustering operation.

For example, imagine the environment has only one tennis ball. Ideally, all α images throughout the detection period would have only one detected circle center per image. However, due to the presence of noise, a given image may have multiple detected circle centers. Now imagine that over this detection period, the maximum number of individual detections was three. This means that at some point in the detection period there was an image with one detection that was legitimate and two that were noise. Therefore, we know there must be (at least) three clusters,

two for noise and one for the ball[2].

Figure 2 provides a visual depiction of this operation's result. The left image is the final image of the detection period (the αth image). There are five detections in this image: one detection is the tennis ball and the other four, noise. The image on the right is the graph-representation of the k-means operation. Note that there are seven clusters. This means that at some point in the detection period there was an image with seven detections. Also notice that the cluster belonging to the tennis ball, Cluster 4, is much more densely-concentrated than the other clusters, lending credibility to the method's ability to distinguish legitimate data from noise.

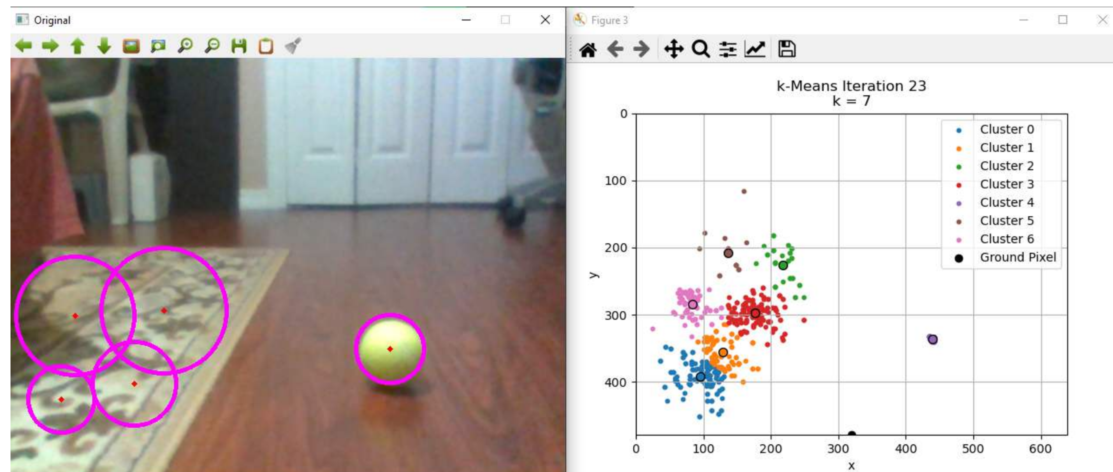


Figure 2: K-Means Clustering of the Detection Period

2.2.3 Setting Initial Means

Instead of selecting the initial cluster means arbitrarily, I decided to have the program select them systematically because, in the former case, there is the possibility that multiple points from the same tennis ball could be chosen as separate means, dividing the tennis ball into two or more clusters.

Fortunately, the same image that determines the k value also selects the initial means. For each detected circle on that image, the coordinate of the circle center is used as an initial mean. So, if we have a k of three, there are three matching initial mean points. Setting the means this way ensures that the noise points in the max detection image each belong to their own cluster.

[2]While noise points appear random from the perspective of the human observer, they are not from the perspective of the computer. If the program detects noise in some location, other noise points may be in the same general area, hence including the noise point in the cluster count. Even if only one noise point appears in the whole period, we still want to isolate it from the legitimate points.

2.3 Choosing the Correct Cluster

Along with being able to cluster the data, the program needs to be able to select the correct cluster. To do so, it considers three characteristics: the number of points the clusters have, point concentration (density), and the vertical distance of a given cluster from the bottom-center pixel (which I will refer to as the ground-pixel distance, GPD).

2.3.1 Metrics

Cluster point count is used as a metric primarily because density is used as a metric. Consider what happens if a noise cluster has only one point or two points that are very close together. The cluster would have a very high density (potentially infinite in the former case) which would skew the results. Tennis ball clusters should have a high number of points along with a high density, so removing the clusters with a low number of points will prevent the low-point, high-density issue.

The minimum number of points a cluster must have to be considered a 'candidate' is a proportion of the detection threshold α. Ideally, a legitimate tennis ball cluster would have α points, but the detector occasionally misses detections, so a candidate must have $p\alpha$ points where $p = 0.8$ at the time of writing. So, if the detection threshold is 100 consecutive detections, a candidate cluster must have at least 80 data points. Otherwise, the cluster is discarded.

Density is used as a metric because tennis ball cluster points are extremely close in proximity, so each point's distance from the cluster centroid should be small. Therefore, clusters with high density are more likely to be legitimate tennis balls than clusters with lower densities. Finally, ground-pixel distance (GPD) is used to help the case where there is more than one legitimate tennis ball present. In that case, a tennis ball collector would want to choose the closest ball to it, which means the balls closer to the bottom of the image. If two or more candidates have similar densities, GPD is a tie-breaker of sorts[3].

2.3.2 Density Calculation

The initial method used to calculate density was the blanket use of maximum intra-cluster distance. For a given candidate cluster (a cluster that meets the minimum point count), the centroidal distance of every point in the cluster is calculated. The maximum distance is then chosen as a radius of a cluster boundary circle. The

[3]However, density is still the primary factor in choosing a cluster as a legitimate ball. so, even if two real balls are present, if a ball further away from the camera has a much higher density, it will be chosen since the program has higher confidence that it is a tennis ball.

area of the boundary circle is then used in the density calculation:

$$\rho_i = \frac{n_i}{A_i} = \frac{n_i}{\pi r_i^2} = \frac{n_i}{(\text{max centroidal distance})_i^2}$$

where ρ_i, n_i, A_i are the density, point count, and area of the ith candidate cluster, respectively.

The problem with the above method is that it is sensitive to outliers. If a stray noise point happens to be placed in the tennis ball cluster, it ruins the density calculation, resulting in a density potentially multiple orders of magnitude smaller than it should be.

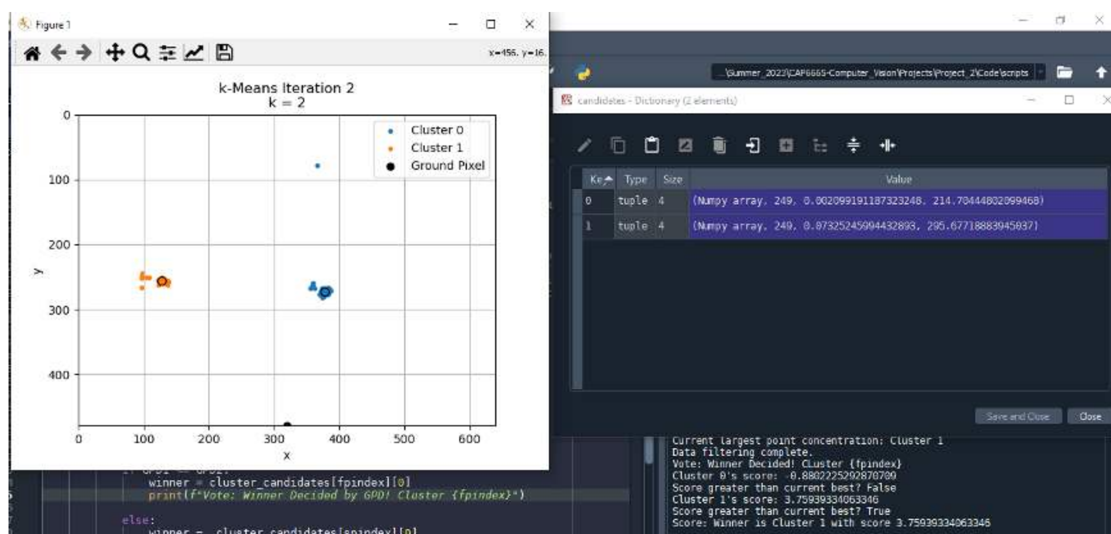


Figure 3: Effect of an Outlier on Cluster Density Calculation

Figure 3 shows the effect a single outlier has on the density calculation. Clusters 0 and 1 are both visually similar save an extreme outlier point for Cluster 0 (blue). In fact, without this point, Cluster 0 should be the winning cluster since it is slightly closer to the ground pixel. However, because of the outlier point, its density is calculated to be ≈ 0.0021 compared to Cluster 1's 0.073. As a result of this single outlier, Cluster 1 becomes the clear winner, which is not the desired result.

To fix this behavior, I had to determine how humans are able to immediately see the outlier. Where is the boundary between inliers and outliers? Consider the following table of values:

Sample	y
0	2
1	4
2	8
3	14
4	22
5	87
6	95
7	101

Sample 5 hopefully attracted the reader's attention because the jump in value from Sample 4 to 5 is so much greater than the others. I applied the same idea to the visual example. We can see the outlier so clearly because it is so much further away from the centroid than its fellow points.

To find the inlier/outlier boundary, I calculate how the data "accelerates," or how the *change* in data values changes. Going back to the table:

Sample	y	y'	y''
0	2	-	-
1	4	2	-
2	8	4	2
3	14	6	2
4	22	8	2
5	87	65	57
6	95	8	-57
7	101	6	-2

we can see that Sample 5, the first outlier point, has the greatest positive acceleration compared to the other points. Sample 5 is boundary between inliers and outliers of this set of data. I use this concept to find the boundary for the distance values of a given candidate cluster by doing the following:

1. Calculate centroidal distances for every point in a cluster.
2. Sort the distances from least to greatest to get a sorted list of distances, d
3. Subtract every $d[n-1]$ from $d[n]$ to find velocity v, which represents how the data is changing w.r.t each sample.
4. Subtract every $v[n-1]$ from $v[n]$ to find acceleration a, which represents how the velocity is changing w.r.t each sample.

5. Find the maximum acceleration. If it is above a specified value, outliers exist in the data starting at boundary, Sample x. Choose instead the distance two samples away Sample $(x-2)$[4].

Plots of the above method are included in the Figures 4 and 5 for a visual aid. More can be found in the Appendix. It was observed that "good" clusters typically had points max accelerations of about two units. To be safe, the threshold for outlier boundary is five. If a cluster has an acceleration above five, the outlier points are ignored when choosing the cluster radius.

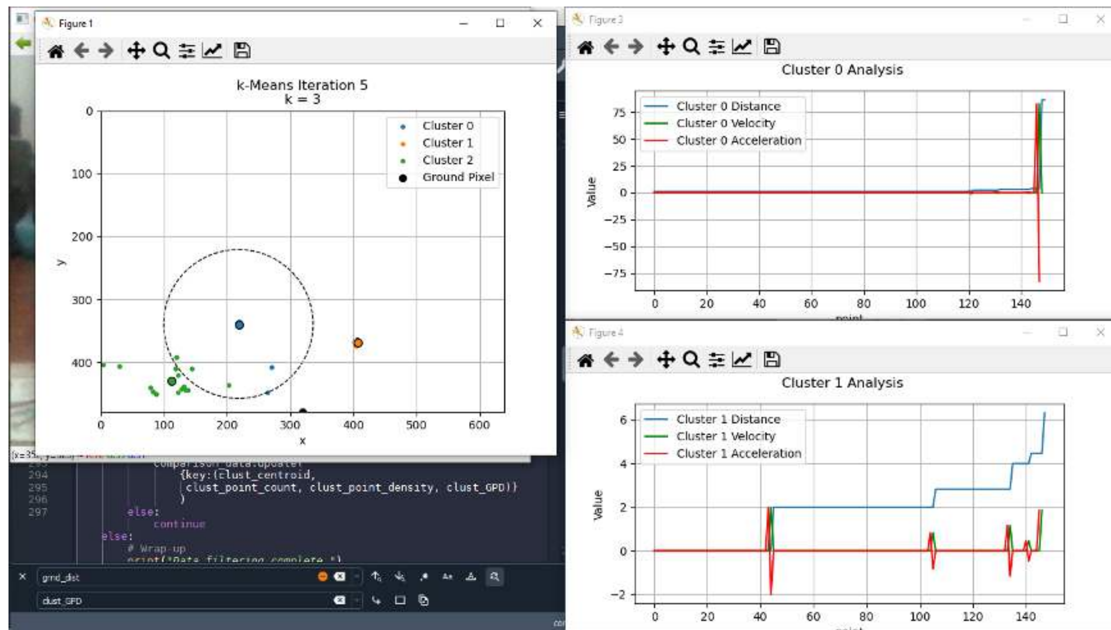
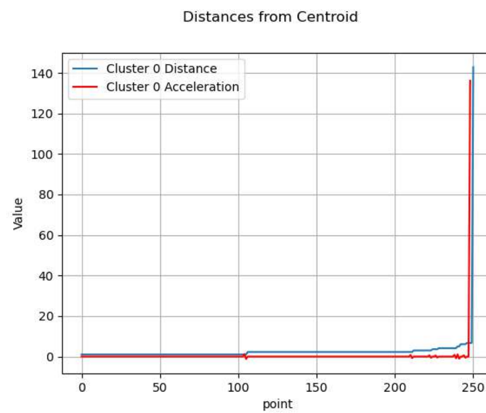
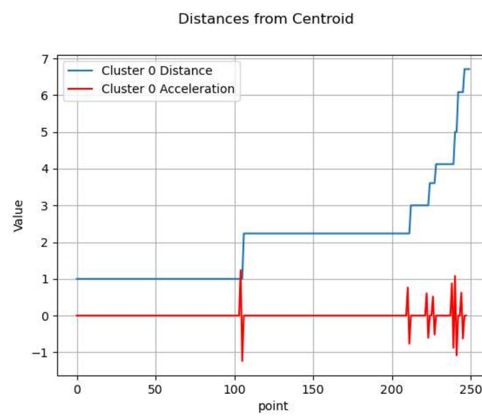


Figure 4: Acceleration Plot for two clusters. Cluster 1 had low acceleration while Cluster 0 peaked at 75.

[4]This is a result of how this method is implemented programmatically. If $max(a)$ has index j, the corresponding index in d is $j+2$: $a[j] \leftrightarrow d[j+2]$. Therefore, $d[j]$ is the distance value two samples away from the inlier-outlier boundary sample.



(a) Acceleration with Outlier



(b) Outlier Removed

Figure 5: Acceleration Plot of a Test Cluster with and without the outlier point.

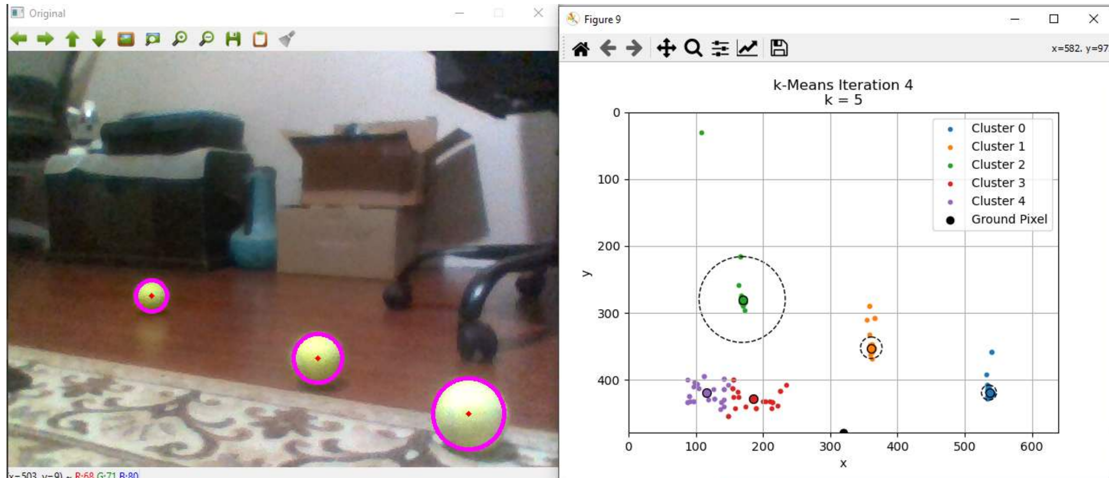
To test the method, I introduced acceleration into the system by (gently) jostling my laptop screen during a detection period, causing the camera to oscillate vertically. the results are shown in Figure 6. The clusters with the highest accelerations had their outliers removed. This is especially observable in Cluster 2 (green). Without this method, the enclosing circle would have been enormous.

3 Selecting the Winning Cluster

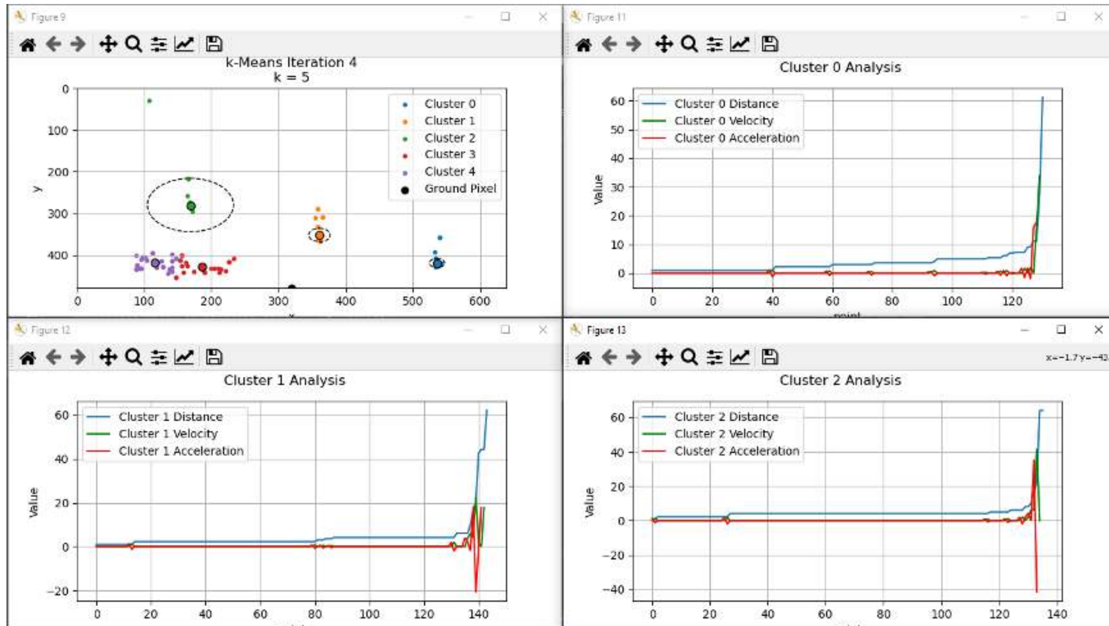
With the completion of the preceding steps, the detected ball data could then be judged by the computer to decide on the 'winning' cluster. I approached the selection method two ways: scoring and voting. For scoring, I attempted to create a linear scoring function that gave weight to each of the three metrics, but the results were not satisfactory compared to the voting method due to the arbitrary nature of the scoring function. This may be a path worth pursuing again in the future, but for this project, the voting method was chosen.

The vote is a filtering process that proceeds in three stages. The first stage is the aforementioned minimum point count. Any cluster not meeting this count is disqualified. The remaining contenders move to density comparison. I iterate through the remaining candidates, tracking which clusters have the highest and second-highest densities. These two contenders become the "first-place" and "second-place" clusters. Once these are found, I calculate a ratio between the clusters' densities:

$$R = \frac{\rho_{1st}}{\rho_{2nd}}$$



(a) Given Clustering Example



(b) Acceleration Plot

Figure 6: Effects of the Intra-cluster Outlier Removal Method

If $R \geq 1.5$, the current first place cluster is selected as the true winner because its density is notably higher (meaning it's more likely to be a tennis ball). If the ratio falls under this value, the vote proceeds to stage three, a GDP comparison. The cluster closer to the ground pixel is selected as the winner. In the event two clusters have the exact same density and exact same GDP, the program selects the cluster it encountered first. Upon selection of a winner, the voting function returns the centroid coordinates of the winning cluster for the adjustment angle calculation.

Figure 7 shows an example of the voting (and scoring) functions for two very similar candidates. The detection threshold for this test was $\alpha = 150$, so contenders needed $0.8\alpha = 120$ data points to enter the "contest". The two contending clusters had very similar densities with a result ratio of $R \approx 1.076$). As a result, the winner was decided by GDP with Cluster 0 being closer to the ground pixel than Cluster 1, exactly the desired behavior.

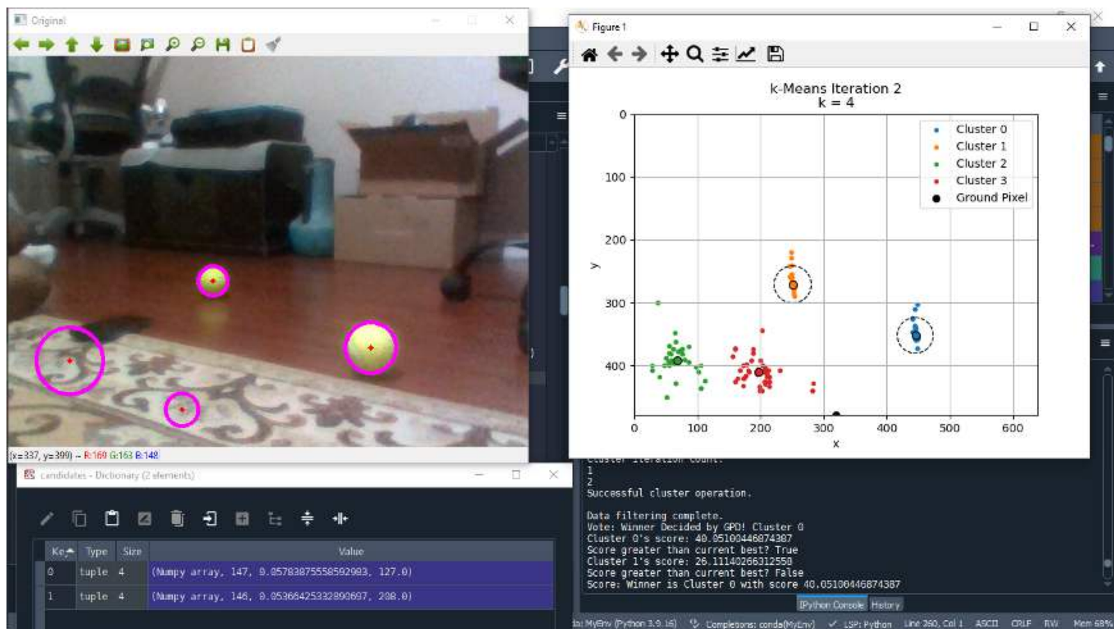


Figure 7: Voting Results - Cluster 0 wins.

The voting function returns the coordinates of the winning cluster's centroid, which are then used to calculate the angle between the vector made from the ground pixel to the centroid $\vec{v}$ and the unit vector from the ground pixel in the vertical direction $\hat{u}$. So, a given cluster has an angle range of $[-\frac{\pi}{2}, \frac{\pi}{2}]$, where CCW (left half of the image) from the vertical is positive and CW from the vertical (right half of the image) is negative. Because the angle θ is measured from the vertical, it can be calculated by using the components of $\vec{v}$. Given centroid (i, j) and ground pixel a, b where a = image height and $b = \frac{\text{image width}}{2}$, the angle θ is found by:

$$\vec{v} = (i, j) - (a, b) = \begin{bmatrix} i - a \\ j - b \end{bmatrix} = \begin{bmatrix} v_y \\ v_x \end{bmatrix}$$

$$\theta = \boxed{\arctan\left(\frac{j - b}{i - a}\right) = \arctan\left(\frac{v_x}{v_y}\right)}$$

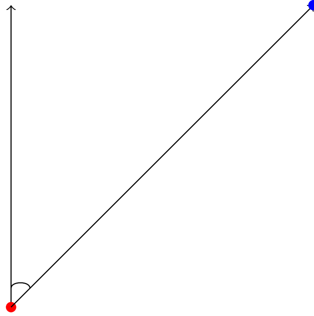


Figure 8: Angle Diagram; Centroid point is in blue. The ground pixel is red.

As a sanity check, I used OpenCV to rotate the image about the calculate angle during a test and received the results in Figure 9. The detected tennis ball (center) is aligned with the image center-vertical, which is the desired test result.
With the completion of the parameter tuning, cluster selection, and angle calculation in the test environment, I then moved to implement the program in ROS.

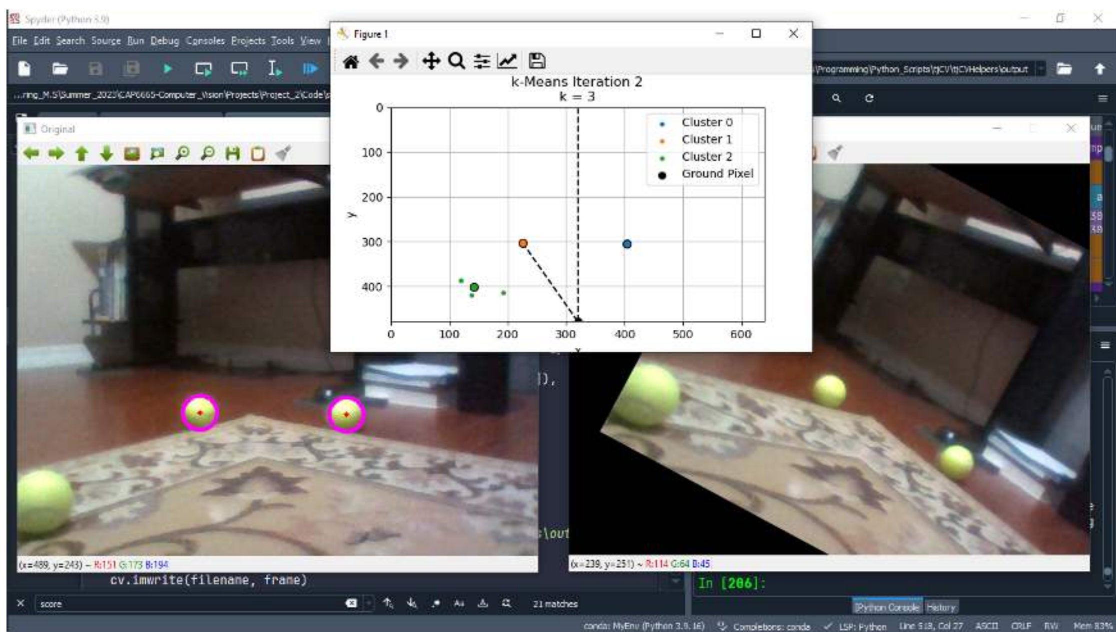


Figure 9: Angle Correction - Middle ball becomes aligned with the center vertical.

4 ROS Implementation

The primary problem to solve for the ROS implementation was organization. What data needs to be sent to which node(s) and by what channel the data should be sent? How many nodes should the program use? Ultimately, the project was structured into two ROS packages. One package, *tw_tennis*, was the primary package that held the code for the nodes, node configuration, and actions, while the other package, *numpy_msgs*, defined helpful ROS messages for data transfer and helper functions to interact with those message types.

4.1 Structure

4.1.1 Program Structure

The project is structured into four ROS nodes. Each node represents a different subsystem of the tennis ball detection program. Sub-system 1 (ss01) is the photographer node. This is the node that captures images from the robot's camera, publishes these images to a topic for subscribers to access, and also displays the images in a live-video feed. The second subsystem (ss02) runs the ball detection algorithm outlined in Project 1. It also serves as the main communication coordinator in the sense that it has some form of communication with all other nodes in the project. If the nodes were all functions in a program, ss02 would be the 'main' function.

Sub-system 3 (ss03) is the data processing node. This node does all of the data filtering, k-means clustering, cluster voting, and angle calculation defined in the previous section of this report. Sub-system 4 (ss04) is responsible for moving the robot through the angle it is supplied.

Finally, the camera tuner was implemented to be ROS-compatible, so the parameters are exported to the ROS Parameter Server on completion. The relevant node(s) may then access those values.

4.1.2 Communication

In terms of communication, the project utilizes many of ROS's data transfer structures. Specifically, this project uses Publishers, Subscribers, SimpleActionServers, SimpleActionClients, and the ROS Parameter Server. First, just as in Project 1, ss01 publishes its images to the dedicated ROS Topic, and ss02 accesses these images as a subscriber. During its subscriber callback function, ss02 performs its ball detection, stores all of the valid detections into a container, and sends the data to ss03 as a SimpleActionClient. ss03 is one of the SimpleActionServers. After processing the data and calculating the rotation angle, ss03 sends this angle back

to ss02 as the action result. From here, ss02 then passes the action result to ss04 as a SimpleActionClient. ss04 receives the angle as a SimpleActionServer, and moves the robot through the angle. It sends a Boolean to ss02 as a result to communicate the move's success.

ss02 acting as an intermediary between ss03 and ss04 was a deliberate design choice. ss01, the photographer node, continuously captures images throughout the duration of the program. Because ss02 is a subscriber to the image topic, it could keep gathering detections and sending new data for processing before the robot completes a move, an undesired behavior. By assigning ss02 as the client to the actions of both ss03 and ss04, it is forced to block until its current action server replies with a result, preventing it from collecting more data.

Structuring the communication this way is preferable because with this method only ss02 has two inter-node communication roles. The first design was a nested structure in which ss02 communicated with ss03 which then communicated with ss04. This required ss03 to be both SimpleActionClient and SimpleActionServer, and it required the ss03/ss04 communication to occur during the ss02/ss03 interaction, which could cause issues on failure such as memory leaks. The current organization is more robust.

Finally, in terms of the ROS Parameter Server, each node accesses the server to retrieve information relevant to their roles. They access their respective topic names to perform communication, camera parameters, and/or the detection threshold if needed for their job. A visual representation of the node communication structure is shown in Figure 10.

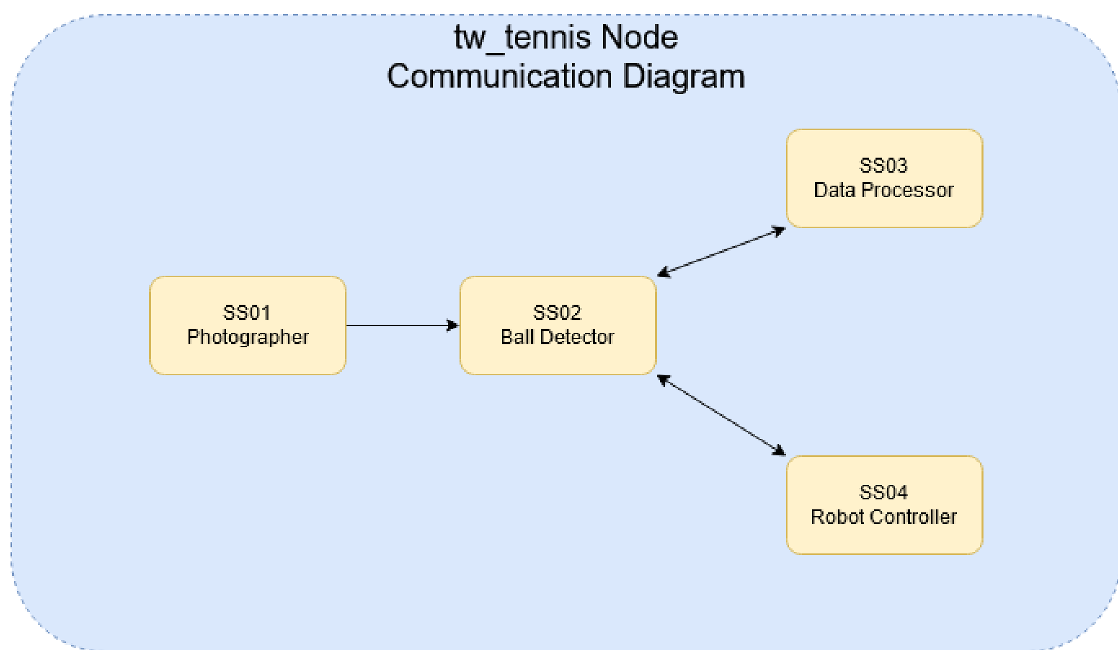


Figure 10: Project 2 node communication structure

4.2 Package Construction

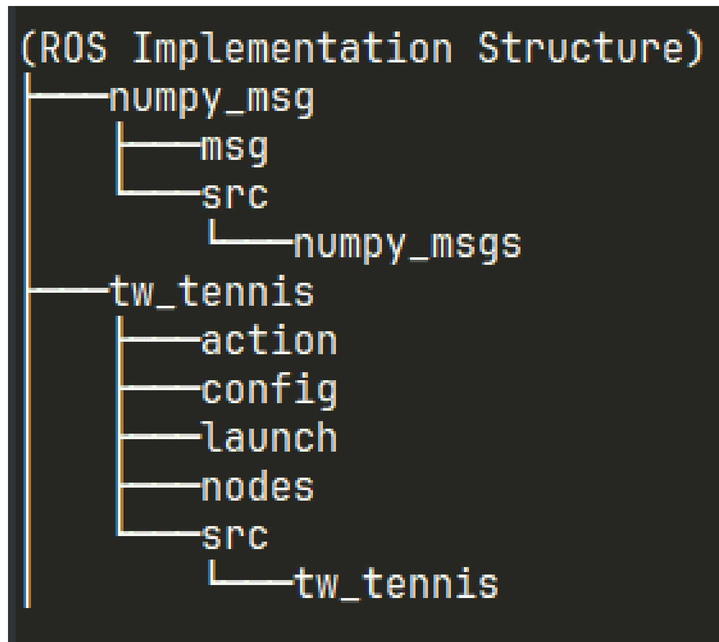


Figure 11: File Structure for the ROS Implementation

Figure 11 shows the file structure for the ROS project. The outer levels show two packages: *numpy_msgs* and *tw_tennis*. *numpy_msgs* is a message package that defines two messages, ROSNumpy and ROSNumpyList, that make sending Numpy arrays and lists of Numpy arrays easier. Since the circle center data is a Python list of Numpy arrays, these messages are essential to transfer data from ss02 to ss03.

The ROSNumpy uses information provided by a Numpy array to facilitate deconstruction and reconstruction of the array. Though Numpy arrays provide an abstraction to view arrays in multiple dimensions, the arrays are implemented as a contiguous block of memory [3]. Each array has an attribute called 'shape' which is a tuple that stores the array's dimensions. For example, a 3x4 array has a shape (3, 4).

Each array also has an attribute called 'dtype' which store the data type of elements stored in the array [4]. So, by flattening the array to one-dimension and storing the flattened array, its shape, and its dtype in a message, the receiver can then reconstruct the array upon receipt. If multiple arrays are to be sent, each array can be composed into a ROSNumpy message, stored in a list, and converted to a ROSNumpyList message (whose only field is a variable-sized array of ROS-Numpy messages). Doing this allows data to be sent back and forth during the project's run-time. To keep the project's code organized, two helper functions, **con-**

struct_rosnumpy and **open_rosnumpy** create and open ROSNumpy messages for the user, respectively.

The *tw_tennis* package is the main package for the project that houses the data generation, processing, and robot-interfacing functions. The four subsystem nodes, the action messages, helper functions, configuration file, and launch file are all defined within this package.

5 Results

After iterating through the project design, I implemented the structure outlined in the previous section and began testing. At the time of writing, the robot is able to capture images, run the ball detection algorithm, process the data to calculate a rotation angle, and use ROS infrastructure provided by the robot manufacturer to rotate through that angle. In other words, the subsystems are able to work together as expected.

However, there are some issues that need correction. First, though the ball detection subsystem is now much more robust against noise, the system is still sensitive to lighting changes. The vast majority of the system testing has been in an office with natural lighting, and changes from the outside such as clouds moving obscuring the Sun or even whether the robot camera is facing the Sun or not can drastically affect the efficacy of the Hough Circle detection algorithm. Ideally, an adaptive solution could be pursued in which the surrounding environment affects the ball detection parameters in real-time.

Secondly, and more importantly, the angle calculated during the process, while correct in terms of the image, is not the angle the robot needs to robot through to align with a detected ball. The image angle calculated corresponds to a roll angle about the x axis, but the robot rotates about its z axis for alignment. To correct this angle inaccuracy, a method to calculate the yaw angle from the image's angle is required. Currently, an angle convergence method is used that allows the robot to converge to face the ball directly after multiple detections by reducing the calculated angle by a factor of about 5. After, say, three rounds, the robot faces the ball directly and oscillates between $\pm 1°$ angle movements for successive detections. More work can be done to converge more quickly, but ideally a more direct solution can be found.

6 Flowchart

Voting Strategy:

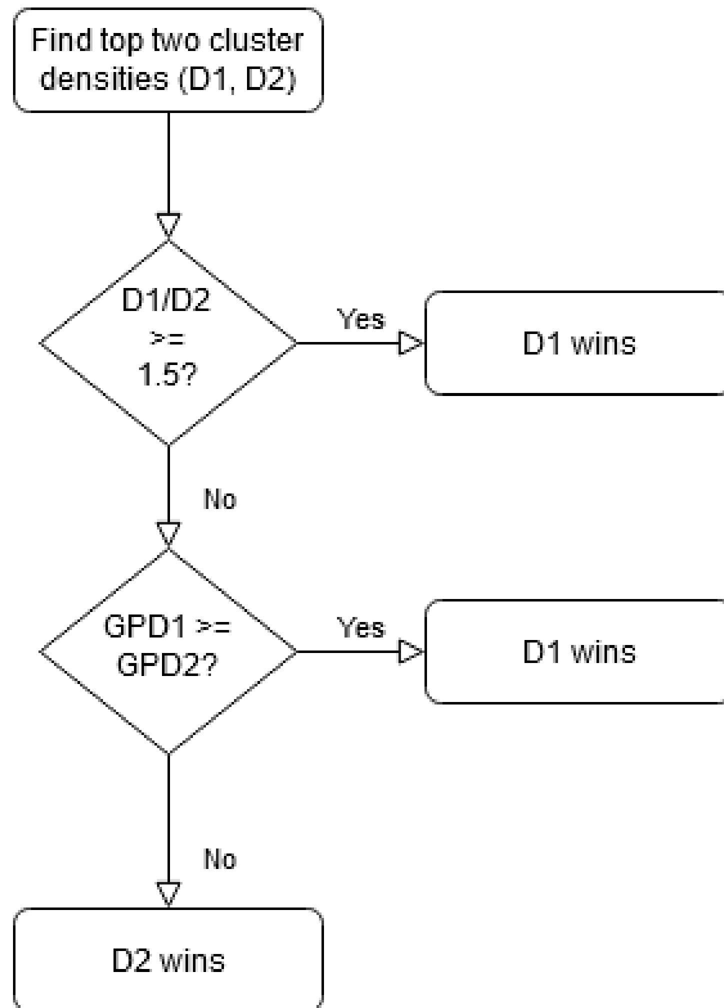


Figure 12: Voting process after finding the top two cluster densities.

7 Conclusion

In this project, I worked to reduce the effect of noise on the ball detection system and implemented the complete program into a ROS package. The lessons learned in terms of dealing with data outliers as well as project management, organization, and time management are invaluable. I look forward to continuing the work on the ball detector by pursuing the paths outlined in the *Results* section.

References

[1] SaifRehman, *Real-Time RGB color filtering with Python*, https://github.com/SaifRehman/Real-Time-RGB-Color-Filtering-with-Python, Accessed: 5 July 2023.

[2] *Adding a Trackbar to our applications!* https://docs.opencv.org/3.4/da/d6a/tutorial_trackbar.html, Accessed: 5 July 2023, OpenCV.

[3] *The N-dimensional array (ndarray)*, https://numpy.org/doc/stable/reference/arrays.ndarray.html, Accessed: 3 July 2023, NumPy Developers.

[4] *Data type objects (dtype)*, https://numpy.org/doc/stable/reference/arrays.dtypes.html, Accessed: 3 July 2023, NumPy Developers.

A Code

The code for this project may be found at the following GitHub link:
https://github.com/tjdwill/TennisBallDetector

B Acceleration Plots

This section presents more acceleration plots that would have cluttered the report but are very helpful for understanding.

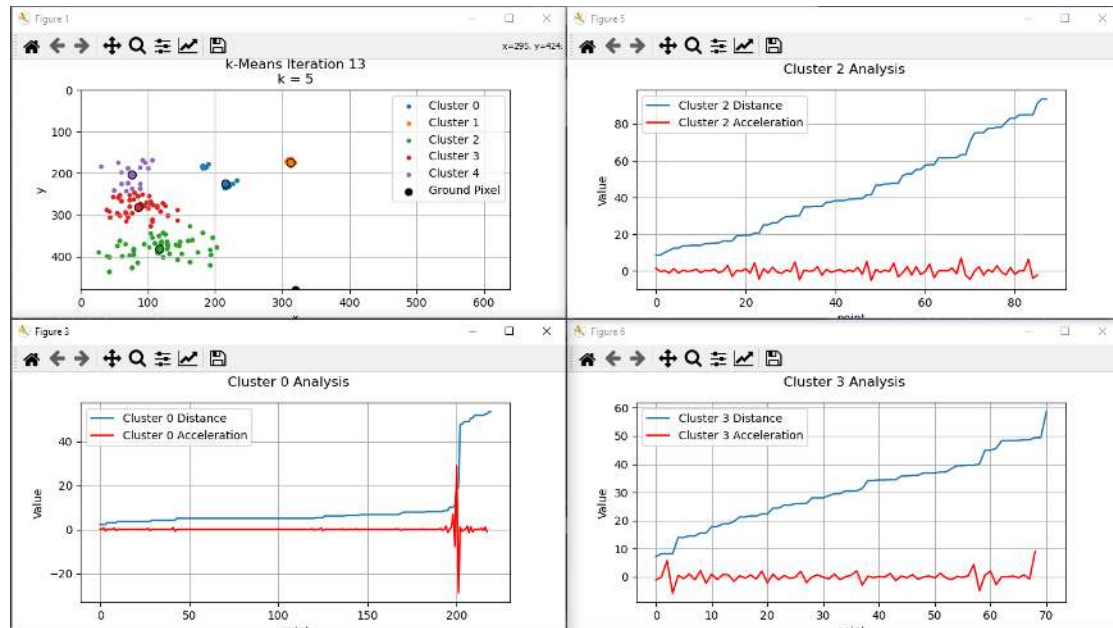


Figure B.1: Showing the plots for three clusters. Useful to show that clusters with a lot of spread (noise clusters) also have low acceleration, adding validity to the method for seeking legitimate clusters.

Figure B.2 shows the acceleration plots for three clusters. The 'good' cluster, Cluster 0, has a max acceleration of approximately two. Cluster 2, a bad cluster due to it being noise, also has high acceleration, but it missed the minimum point count, so it was not a candidate for consideration. Even if it did enter candidacy, it would lose because even after adjusting for outlier points, its minimum centroidal distance is 20, which would result in low density. Cluster 1, however, is a candidate for consideration and it has high acceleration, meaning there is at least one outlier point.

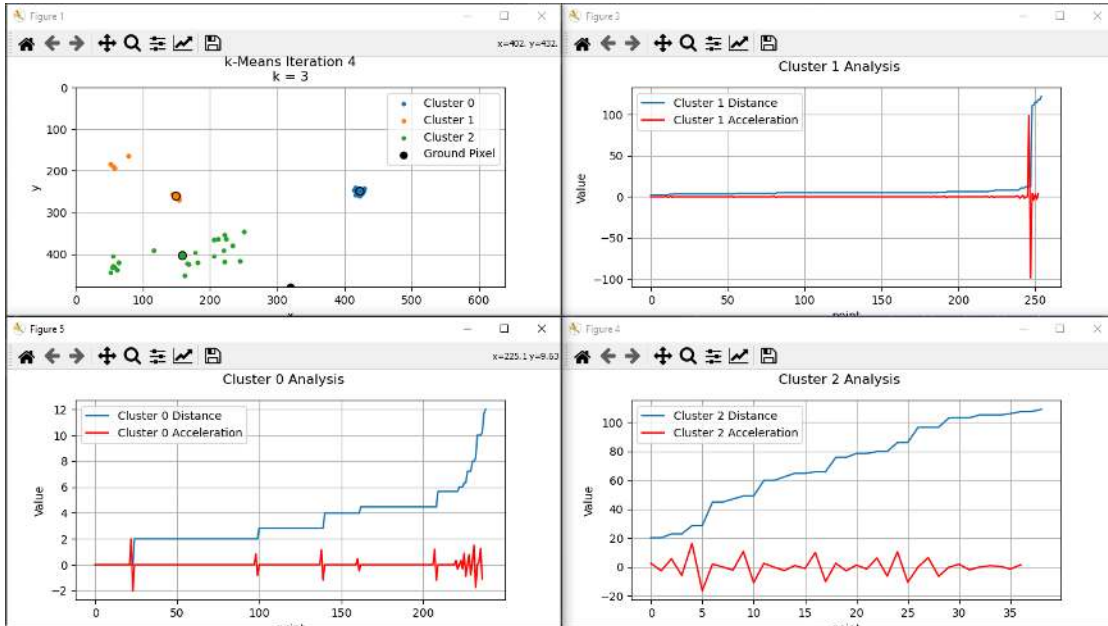
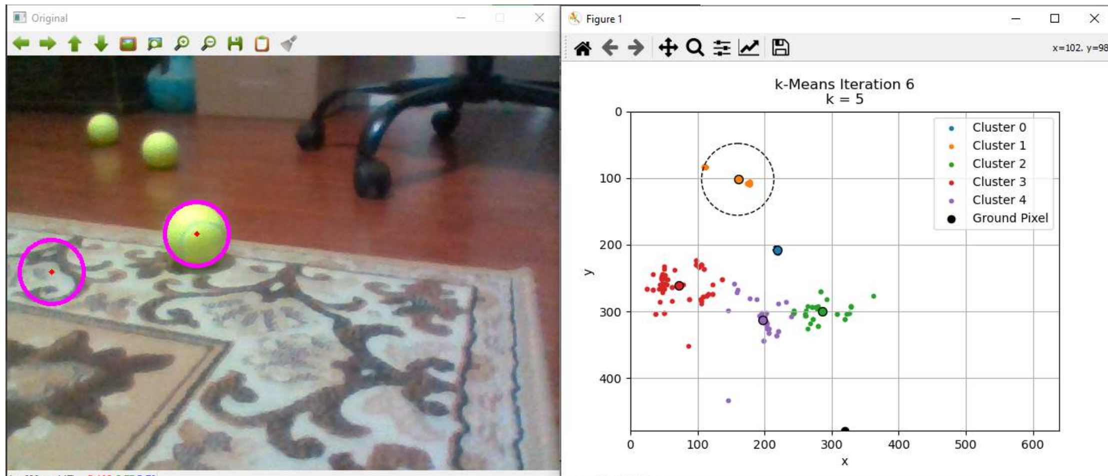


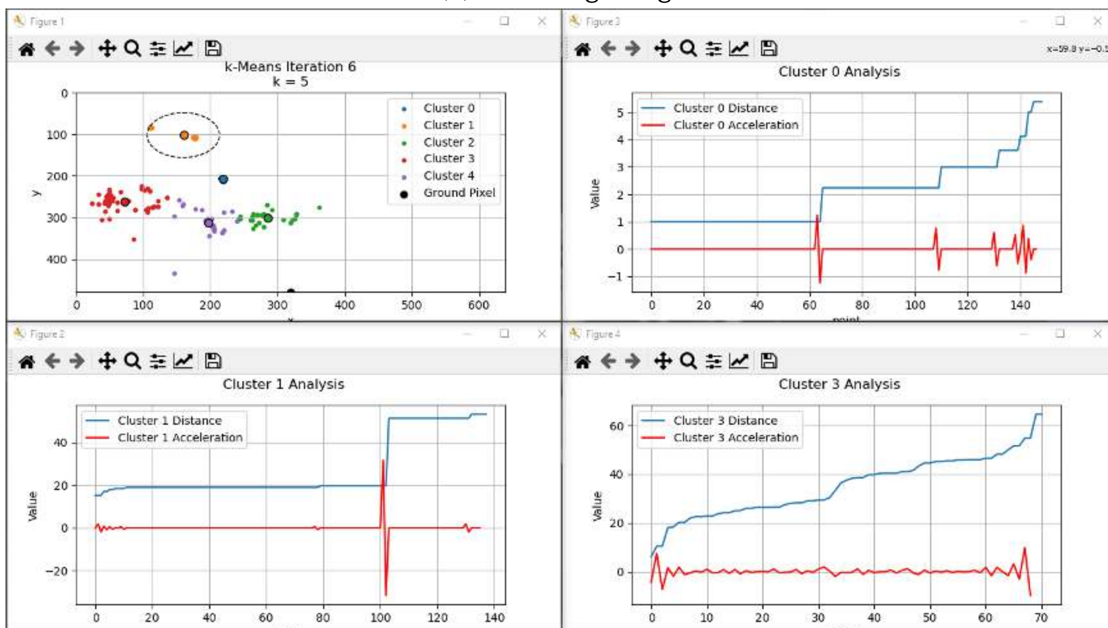
Figure B.2: Showing the plots for three clusters. Cluster 1 has high acceleration while Clusters 0 does not.

Figure B.3 shows an image along with the acceleration plots for selected clusters. Cluster 1 is the middle tennis ball. The outlier points of this cluster are the result of the left-most ball's occasional detection over the detection period. As it did not appear on the max detections frame, it was included in the middle ball's cluster. This is not an issue since Cluster 0 is the most valid ball to choose and the left-most ball is far away in terms of GPD.

One can also see that as a result of the outliers, Cluster 1 has a high acceleration, but Cluster 3, which is much more spread out due to being a noise cluster, does not.



(a) Matching Image



(b) Acceleration Plots

Figure B.3: Another Acceleration Plot. Notice the spread in Cluster 3 and its low acceleration compared to Cluster 1.