CAP6665 - Project 1

Terrance Williams

June 21, 2023

Contents

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

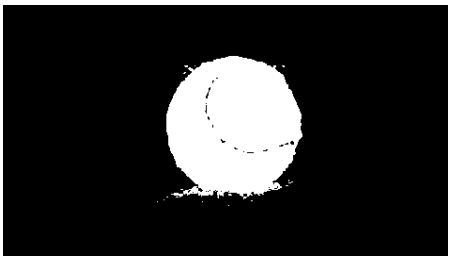
The detector filters the image for color first. This is done to minimize the number of circular features detected by the detector, working to obviate the need for additional filter work later in the process. As mentioned previously, tennis balls are typically along yellow or green in color, so these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

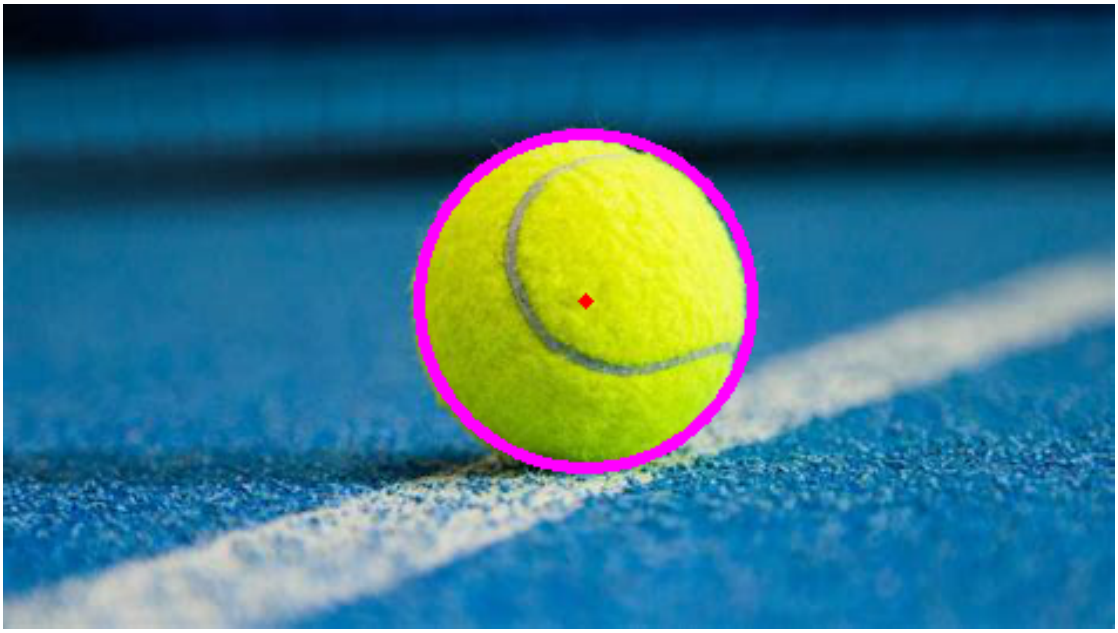


Figure 2: Detected circle

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*).

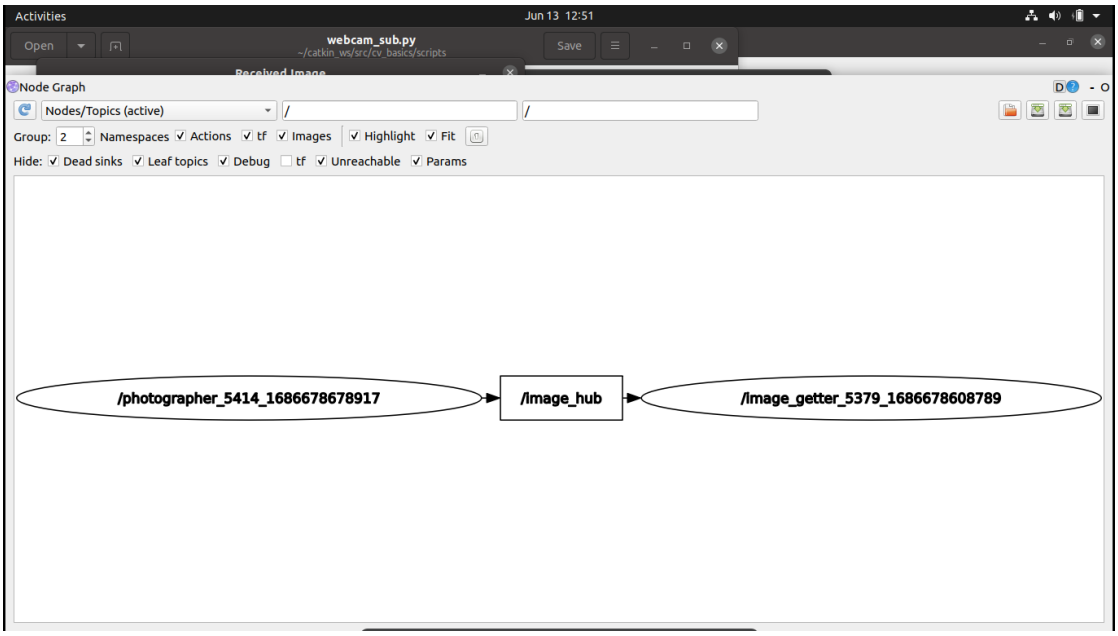


Figure 3: Resulting node communication (via rqt_graph)

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.

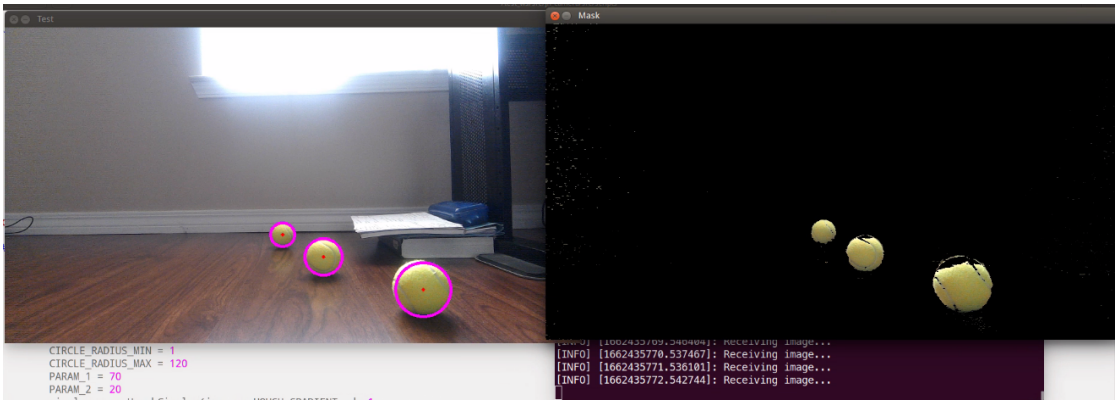


Figure 4: Results of the ROS implementation

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.

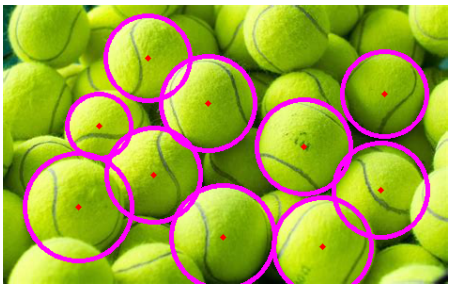
Figure 5 displays the effects of changing only the minDist parameter. For each image pair, the left image uses a lower distance minimum compared to the right image (by a factor of about 0.5).

3.2 JetHexa Parameters

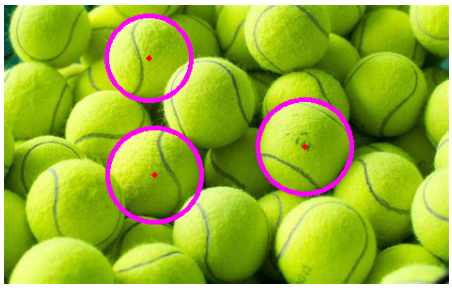
After testing on the JetHexa environment, the following parameter configuration was determined:

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

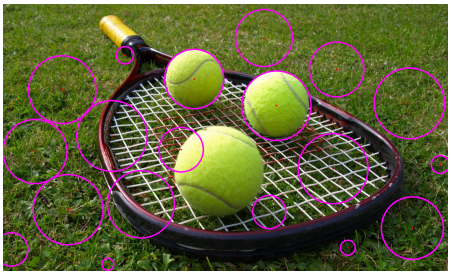
[3]Image Sources:
https://www.freeimages.com/photo/tennis-racquets-and-balls-1414737
https://www.tasteofhome.com/collection/reasons-need-to-buy-more-tennis-balls/
http://7-themes.com/7007704-tennis-ball-photography.html



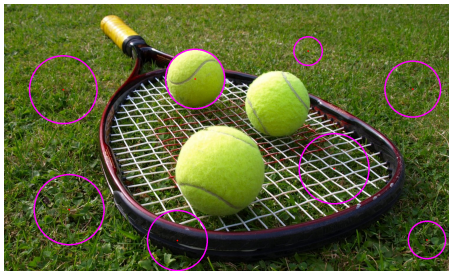
(a) Dist. 1



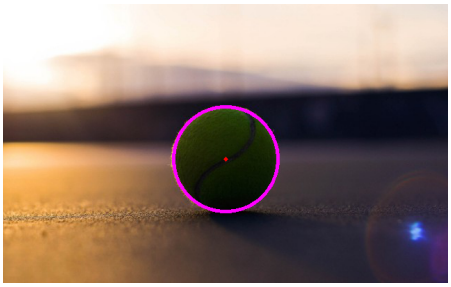
(b) Dist. 2



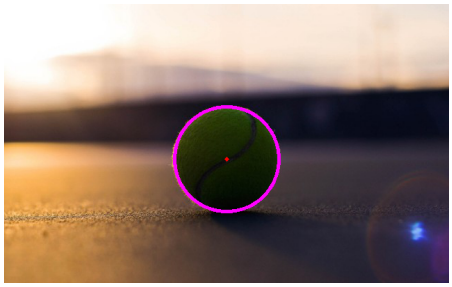
(c) Dist. 1



(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

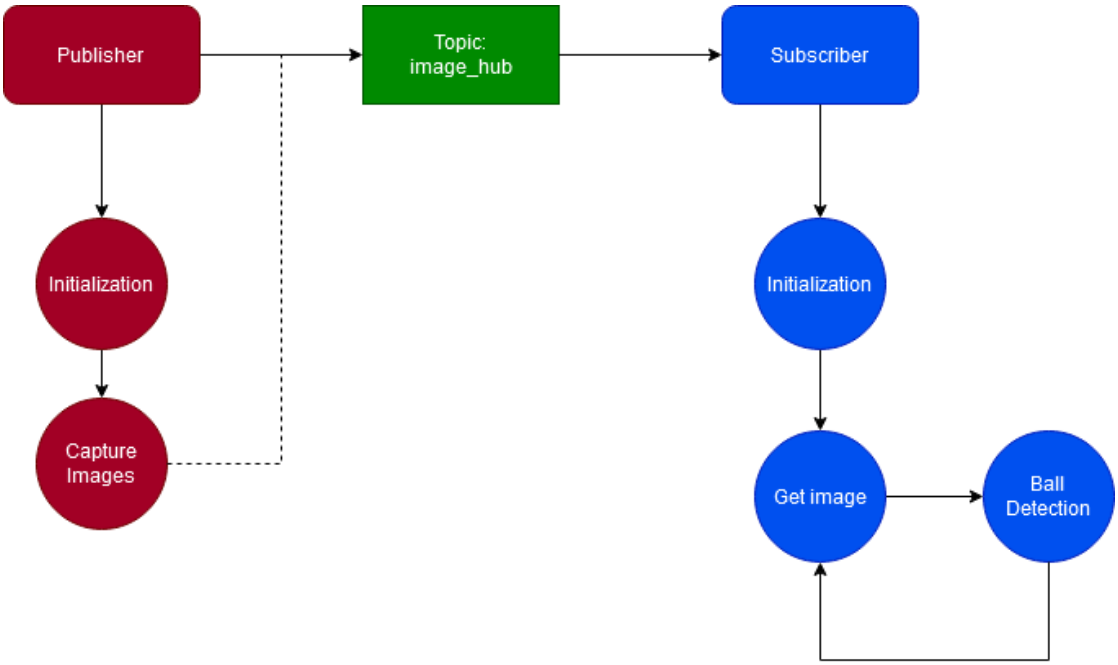


Figure 6: A flow diagram showing the communication path of the ROS publisher and subscriber nodes.

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
```

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

"""
Title: Tennis Ball Detector Node
Author: Terrance Williams
Date: 13 June 2023
Description: Creates a ROS Subscriber 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 numpy as np


# Define Globals
SATURATION_LOWER = 50
SATURATION_MAX = 255  # Max 'S' value in HSV
BRIGHTNESS_LOWER = 20
BRIGHTNESS_MAX = 255  # Max 'V' Value in HSV
YELLOW_LOWER = 22
GREEN_UPPER = 85
TENNIS_THRESH = 10 # Number of consecutive detections needed to be a "true" detection.
count = 0

def img_mask(image):
# input raw image
# outputs array of masked images (red, yellow, green, blue, original)
    img_hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)

    # Generate HSV Threshold (yellow to light blue)
    color_thresh = np.array([[YELLOW_LOWER, SATURATION_LOWER, BRIGHTNESS_LOWER],
                             [GREEN_UPPER, SATURATION_MAX, BRIGHTNESS_MAX]])

    # Generate Mask
    color_mask = cv.inRange(img_hsv, color_thresh[0], color_thresh[1])

    img_masked = cv.bitwise_and(image, image, mask=color_mask)

    return img_masked


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
```