ENPM 809T – Autonomous Robotics: Spring 2020

Master of Engineering Program in Robotics

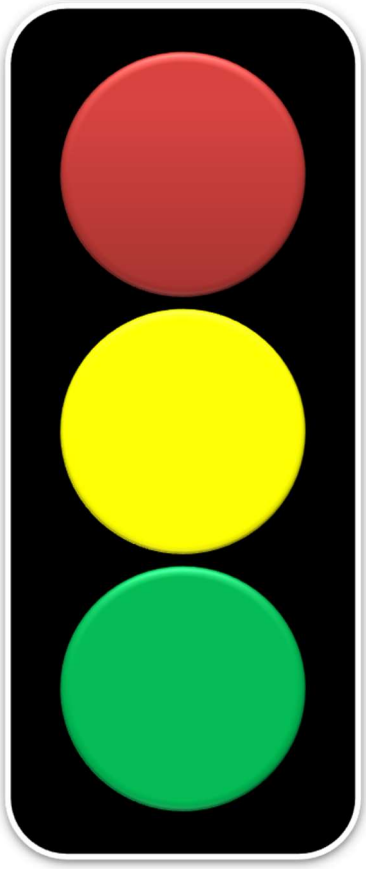
Due Date Friday, February 21st, 2020

Submission Information

- This assignment explores the use of OpenCV on the Raspberry Pi to facilitate object identification and tracking, including writing data to file and subsequent analysis in Matplotlib
- Submit response to Question #1 via Gradescope by 11:59 pm
- Steps 1-4 should be completed using your Raspberry Pi
- Step 5 should be completed using your laptop
- **Each student must complete their own submission for this assignment**

Question #1 (20 points)

The perception subsystem of autonomous vehicles typically consists of camera, radar, sonar, and lidar sensors. A primary advantage of cameras is the ability to sense color, while radar and lidar provide a 3-dimensional representation of the vehicle's environment. Sonar sensors are typically used for close-range applications including detecting vehicles in neighboring lanes (i.e. blind spot detection) and parking operations.



Cameras enable the vehicle to detect objects such as stop lights, road signs, and lane markings. As an example, checkout Civil Maps' "Cognition for Cars" demo:

https://www.civilmaps.com/

https://www.youtube.com/watch?v=lW9eVnk962U

In Assignment #2 we used the Raspberry Pi, Pi camera, and Python to record a minimum 30 second video clip by exercising the cv2.VideoWriter() function in OpenCV.

In this assignment, we explore the implementation of object tracking with the Raspberry Pi and OpenCV to simulate a camera used for navigation onboard an autonomous vehicle. Note that we will use a modified version of the script(s) developed in this assignment for the semester project.

Step 1: HSV masking

Using your Raspberry Pi, record an image of yourself holding the stoplight schematic from page 1, either as a print out on paper or using a tablet/phone/etc. This RGB image serves as a baseline image which we convert to HSV in order to define the upper and lower bounds of the HSV mask. Recall from lecture that OpenCV converts an RGB image to HSV using the cv2.cvtColor( ) function:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html

Learn OpenCV has a good review of the motivation behind converting from the RGB color space to HSV for object detection:

https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/

Once you have recorded an image of yourself holding the schematic, create a blank canvas of dimensions identical to the image recorded by the Pi. Set each pixel of the blank canvas to 0 on a 0-255 grayscale (i.e. black). This blank canvas will serve as our HSV mask.

Write a script that searches through the HSV image and sets each pixel of the canvas/mask to white (255) in the case where the HSV pixel values are between a lower and upper HSV mask bound that you define. The goal here is to mask the image of you holding the stoplight such that only those pixels corresponding to the green light are white (255) in the mask.

To complete this step, you may use a function of your own creation or a prebuilt function such as cv2.inRange( ). Note that this process likely requires iteration to arrive at the proper HSV mask bounds.

For convenience, show the RGB, HSV, and masked images to the screen using NumPy's hstack( ) function.

Step 2: Find contours and draw bounding circle

With the HSV image masked to identify those pixels corresponding to the green stoplight, use OpenCV to identify a bounding contour. Contours are simply a curve joining all the continuous points along a boundary. Use the cv2.findContours( ) function to retrieve the contours from the binary (i.e. masked) image from Step 1.

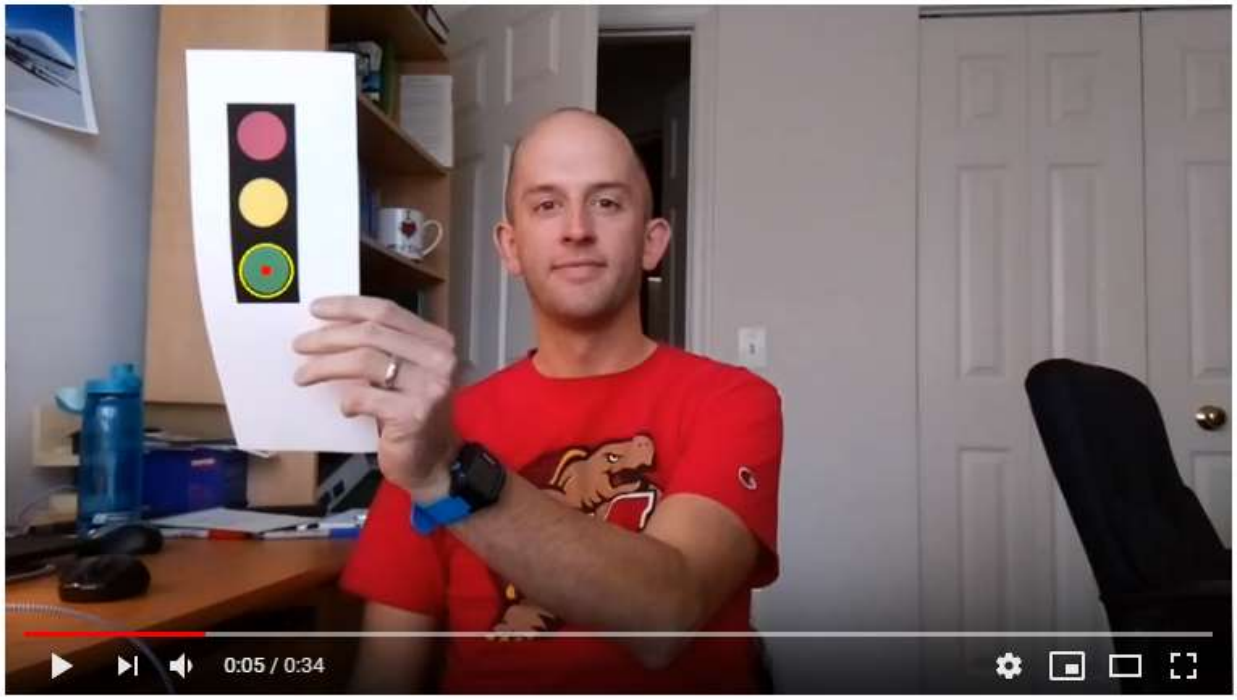
https://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html

In the event a contour was found in the image, use the cv2.minEnclosingCircle( ) and cv2.moments( ) functions in conjunction with cv2.circle( ) to draw the minimum enclosing circle around the green light in the original RGB image. Note: a bounding box is also acceptable if preferred.

Step 3: Apply to picamera video feed

Modify your code from Steps 1 and 2, in conjunction with your code from Assignment #2, to continuously locate and identify the green stop light in the Raspberry Pi video feed using the picamera module. Demonstrate your code can track the green stoplight as you translate the schematic across the Pi camera's field of view. A video demonstration is available here:

https://youtu.be/HLXRjiq7DXw



Step 4: Record video file & upload for submission

Using the Raspberry Pi, Pi camera, and code developed in Steps 1-3, record a minimum 30 second video clip of yourself tracking the green stoplight using the cv2.VideoWriter( ) function in OpenCV. Upload the video to your YouTube account then include a link to the video in the .pdf uploaded to Gradescope

Step 5: Analysis of hardware performance limitations

As discussed in lecture, although the Raspberry Pi is an affordable and convenient platform for robotics, it is not particularly optimized for image processing. While several alternative options exist, one example of a COTS platform for compute-intensive applications is the Jetson Nano from NVIDIA:

https://developer.nvidia.com/embedded/buy/jetson-nano-devkit

That said, the benefits of the Raspberry Pi for ENPM809T outweigh the performance limitations.

Let's explore this idea further by evaluating the time required to process and write each frame to video in Step 4.

Using the **datetime** module (https://docs.python.org/2/library/datetime.html), create a variable that defines the beginning of each iteration the Pi runs through the loop (i.e. a "start" value). At the end of each loop, create a second variable that defines the "end" of each loop iteration. Each time through the loop, print the delta between these two variables to the screen then save the delta to a .txt file for subsequent analysis in NumPy/Matplotlib on your laptop.

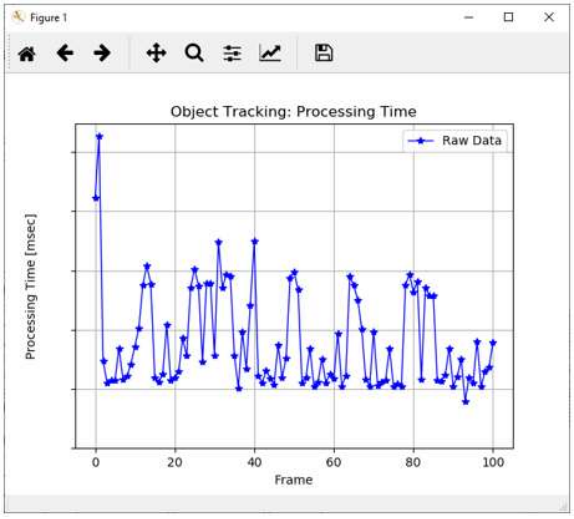
Example snippets of code are provided below to provide guidance on saving data to .txt file on the Pi:

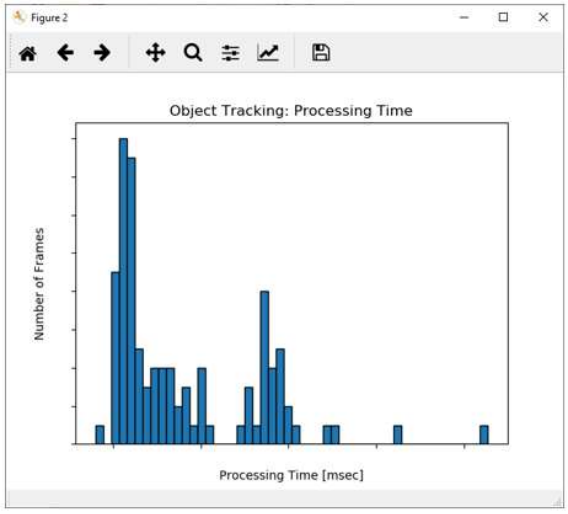
```
# open .txt file to save data
f = open('hw3data.txt','a')

        # print time to run through loop to the screen & save to file
        now = stop-start
        outstring = str(now.total_seconds()) + '\n'
        f.write(outstring)
        print(now.total_seconds())
```

Repeat Step 4 and record a video such that your .txt file contains a minimum of 100 delta time measurements.

Transfer the .txt data file from your Raspberry Pi to your laptop. Using Python on your laptop, read in the delta time data (hint: leverage the code written in Assignment #1). Using Matplotlib, create two plots, examples of which are provided below:

1. An ***x/y plot of the data***, where the x-axis is in units of frames and the y-axis is delta time in units of milliseconds
2. A ***histogram of the data***





Copy/paste both plots into the .pdf uploaded to Gradescope. Be sure to include a few sentences describing the results of your analysis of the Raspberry Pi's performance limitations in regards to object tracking.