ENPM 809T – Autonomous Robotics: Summer 2019

Master of Engineering Program in Robotics

Due Date Tuesday, June 25th, 2019

Submission Information

- This assignment explores the use of OpenCV on the Raspberry Pi to facilitate object identification and subsequent robotic control/command
- This assignment requires writing data to file and subsequent analysis in Matplotlib
- The codes developed in this assignment will be used to implement initial tracking and following algorithms on our robot ground vehicles
- Submit response to Questions #2 and #3 via Gradescope by 5:30 pm

Question #1 (nothing to submit)

General reminder to use your phone to record images and 30-60 second video clips throughout the course. These images/videos can then be stitched together towards the end of the summer as part of your submission for the Grand Challenge. It is always better to have to cut out extra material than to say "I wish I had an image of that…!" at the conclusion of the project.

In this spirit, download the *timelapse01.py* script from the course GitHub:

https://github.com/oneshell/enpm809T

The script utilizes raspistill( ) in conjunction with the os.system( ) to create a folder on the Raspberry Pi with a time-stamped name, record still images at a user-defined frequency, and write the images to a video file using cv2.VideoWriter( ).

https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspistill.md

https://docs.python.org/3/library/os.html#os.system

The script is provided as a template for recording and creating time-lapse videos this summer, both in preparation of and during the Grand Challenge. Students are encouraged to modify the script as they desire to meet their needs.

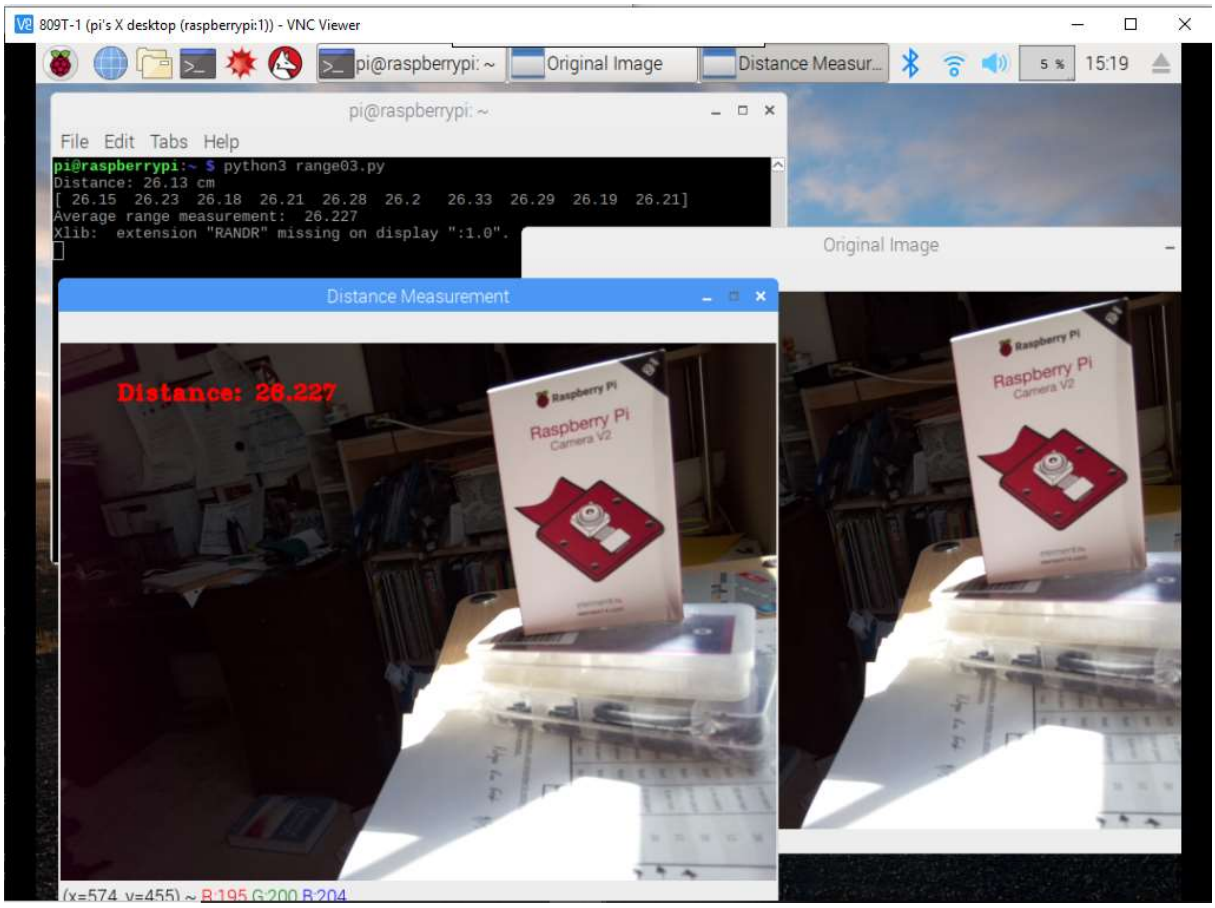
Question #2 (10 points)

In lecture 4 we integrated and tested our robot distance sensors. The following was originally posted as an In-Class Exercise in the lecture notes, but should be completed and submitted here as part of Assignment #4.

To begin, place an object of your choosing roughly 0.5 meters away from the Raspberry Pi. Write a Python script to perform the following tasks:

1. Record a still image of the scene using either raspistill or picamera
2. Using the ultrasonic distance sensor, record 10 successive distance measurements between the Pi and the object
3. Calculate and print the average of the 10 distance measurements onto the image using OpenCV

Once complete, cut/paste the image into the .pdf uploaded to Gradescope. An example image is shown below:

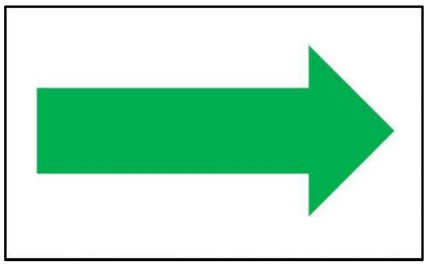


Question #3 (20 points)

In Assignment #2 we used the Raspberry Pi, Pi camera, and Python to record video files by exercising the cv2.VideoWriter() function in OpenCV. In Assignment #3 we implemented object tracking with the Raspberry Pi and OpenCV to simulate a camera used for navigation onboard an autonomous vehicle.

In this assignment, we build upon our object tracking algorithms to identify the presence and *orientation* of a green arrow. We will use this algorithm in the initial control and testing of our ground vehicles.

Step 1: Download image

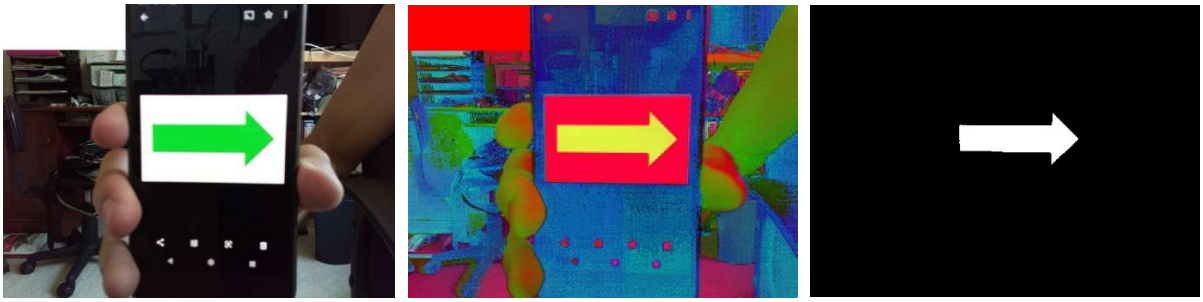


To begin, head over to ELMS > Assignments > HW #4 and download *hw4arrow.jpg* to your cell phone. This image depicts a green arrow on a white background and will be used as the input to robot tracking algorithms in future lectures.

Step 2: HSV masking

Using your Raspberry Pi, record an image of yourself holding the arrow image on your cell phone. 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. 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 arrow such that only those pixels corresponding to the green arrow are white (255) in the mask.



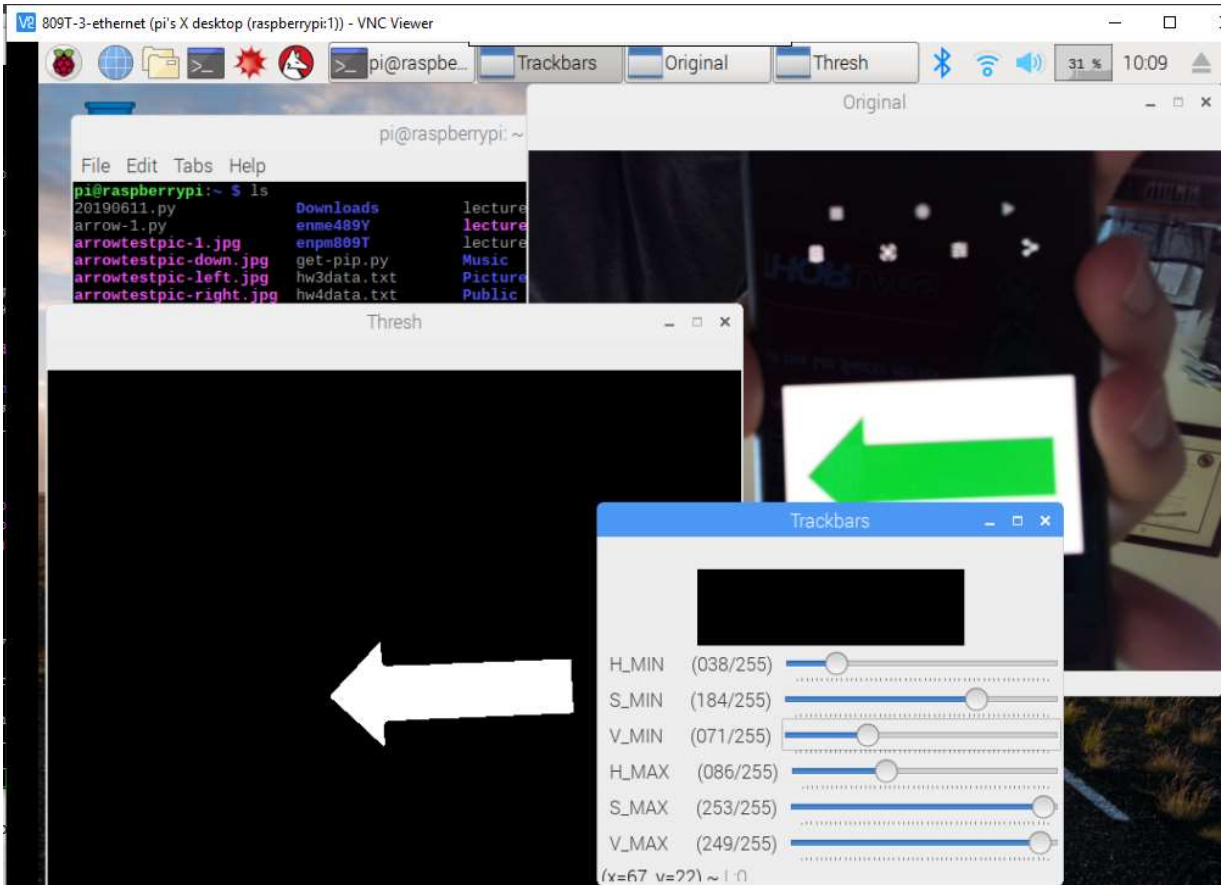
Original RGB image (left), RGB image converted to HSV (center), and HSV masked image (right)

To help facilitate the selection of HSV mask bounds, head to the course GitHub and download the Python script *colorpicker.py* to your Raspberry Pi:

https://github.com/oneshell/enpm809T

This script has been developed to (1) input a RGB image file specified by the user, and (2) identify the upper and lower bounds of a HSV color mask of the image dynamically, using a trackbar interface.

The following RPi screenshot illustrates the *colorpicker.py* script in action, in which the lower and upper boundaries of the RGB color green in the HSV color space permit detection of (only) the green arrow. The original RGB image ("Original") is shown at top right behind the HSV color mask ("Thresh"), with the adjustable Trackbars in front.



Recall from Assignment #2 that the HSV mask used to track the green stoplight incorporated similar lower and upper HSV boundaries of the form:

masklower = (H_MIN, S_MIN, V_MIN)

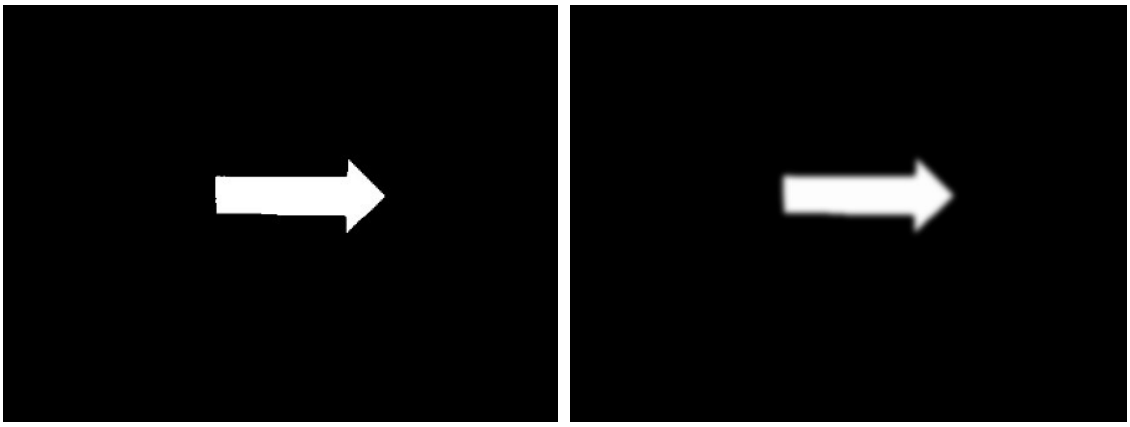
maskupper = (H_MAX, S_MAX, V_MAX)

Ensure you are able to run and utilize the *colorpicker.py* script, as this will facilitate efficient creation of HSV mask bounds throughout the course. The command line call is provided below as well as commented in the code:

```
python colorpicker.py -f HSV -i name_of_image_file.jpg
```

Step 3: Image blurring

As discussed in lecture, OpenCV has built-in functions for image blurring. Update your Python script to blur the masked image.



HSV masked image (left) and Gaussian blurred image (right)

Step 4: Corner identification

The goal of this assignment is to identify the presence and ***orientation*** of the green arrow. There are multiple approaches to tackling this problem. One is to identify the corners, or some subset of the corners, of the blurred arrow image. This can be accomplished using the Harris Corner Detection algorithm via the cv2.cornerHarris( ) function in OpenCV:

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html

as well as the Shi-Tomasi Corner Detection algorithm via the cv2.goodFeaturesToTrack( ) function in OpenCV:

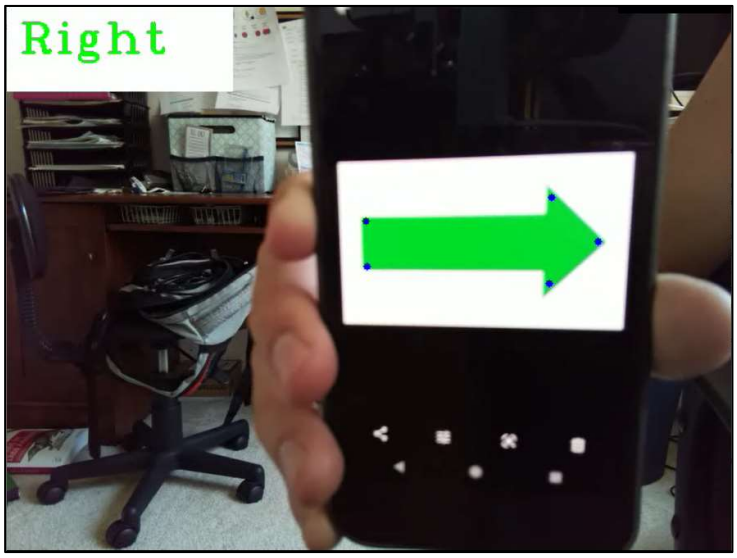
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html

Update your Python script for this assignment to apply the Shi-Tomasi Corner Detection algorithm to the blurred image.

Step 5: Arrow orientation

Write an algorithm that automatically determines the orientation of the arrow based on the coordinates output by cv2.goodFeaturesToTrack( ). Students are free to incorporate additional algorithms/metrics to increase the confidence of your estimate of direction. An image showing example output is provided on the following page.

Note: only the four cardinal directions are required: North, South, East, West (or correspondingly up, down, left, right) in the image frame. In future lectures, these directions will be translated to the drive direction of the robot ground vehicle (forward, reverse, left, right).



Step 6: Apply to picamera video feed

Modify the Python script generated in Steps 1-5, in conjunction with your codes from Assignments #2 and #3, to continuously locate and identify the orientation of the green arrow in the Raspberry Pi video feed using the picamera module. Demonstrate your code can track/identify the green arrow as you translate and rotate the image across the Pi camera's field of view. A video demonstration is available here:

https://youtu.be/I_AjyAYm6IM

Step 7: Record video file & upload for submission

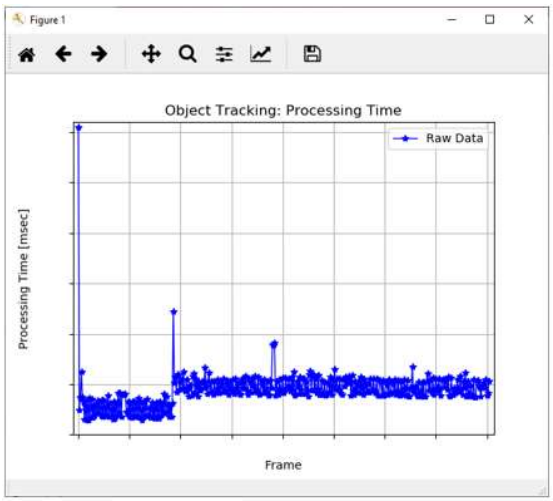
Using the Raspberry Pi, Pi camera, and code developed in Steps 1-6, record a minimum 30 second video clip of yourself tracking the green arrow using the cv2.VideoWriter( ) function in OpenCV. Upload the video to your YouTube account then include a link to the video in the .pdf uploaded to Gradescope.

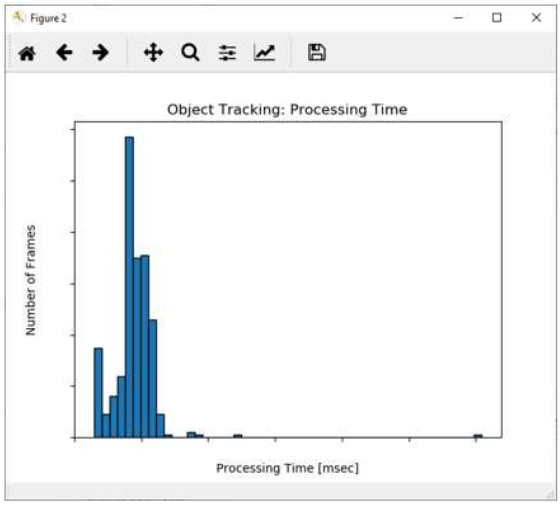
Step 8: Analysis of hardware performance limitations

As in Assignment #3, here we evaluate the time required to process and write each frame to video in Step 5. Repeat Step 6 (if necessary) and record a video such that your data .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. 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/orienting.