MODULE 5: Using the Camera with the PI & Basic Image Processing

The Raspberry Pi comes with an interface to connect a camera module that is helpful in many different applications. The use of the camera will be explored along with image processing features available in python.

GOALS:

- Connect and use the camera on the Pi
- Use python to take pictures, save pictures
- Simple image manipulation: resizing images, determine properties of images and cropping with python
- More advanced image processing: find contours and shapes,
- More python features: writing functions and using include files

Camera Setup

Camera setup for the Pi is fairly straightforward [1]. Just make sure that the Pi configuration tool (Module 1) has been used to enable the camera. The default code provided by the link above will save still pictures to the desktop under the same name every time, so each new picture that is saved will overwrite the previous picture. It may be desirable to save the picture with with a different name by using the date or time as a parameter for the file name. Some caution should be used when doing this as it can be easy to use up significant memory if pictures are continually taken by a program without eventually deleting those that are not needed.

Simple Image Processing

While many ways exist to work with images on the Pi, OpenCV is one of the most well developed libraries for image processing that is currently available for python [2]. Before OpenCV can be used, it must be installed [3]. Some software first needs to be installed on the Pi before pip can install OpenCV.

```
$ sudo apt-get install libhdf5-serial-dev libatlas-base.dev libjasper-dev
$ sudo apt-get install libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5
$ sudo pip3 install opencv-python
```

Running the commands above from the command prompt should install OpenCV, assuming they are run on the Buster edition of Raspbian, but it may take some time.

The following commands serve as examples to illustrate the capabilities of OpenCV. It is suggested that the student follow along while reading this and also execute the commands on the Pi. First obtain the following image (or another image of your choice) and then open the python interpreter and run the following commands.

```
>>> import cv2
>>> import numpy as np
>>> img = cv2.imread("monarch.jpg")
>>> print (img.size)
639600
>>> print (img.shape)
(533, 400, 3)
```

Illustration 1: Monarch Catepillar
http://feherhome.no-ip.biz/temp/wustl/205/monarch.jpg

Notice that once the image is opened, an image object is created that may be accessed. This image is opened up as a 400 column (x visual direction) by 533 row (y visual direction) by 3 deep array and is using 639600 bytes of memory in python. The following lines will display the image, while the second command will remove the image picture from the screen.
NOTE: These commands will only work when running on the Pi itself and not a session where they are invoked after using ssh to login to the Pi remotely. If logged in remotely, transfer the files over to another system using scp, sftp, WinSCP or some other method as described in Module 3.

```
>>> cv2.imshow("Catepillar",img)
>>> cv2.destroyAllWindows()
```

Resizing and Saving Images

The image can be resized fairly easily and the new image saved. Notice, that `imwrite` method is smart enough to determine the file type from the file name provided. Often when using new commands it can be helpful to carefully check their specification so that you can fully understand how they work, for example resize method accepts three arguments, the image, its new (width, height), an interpolation method for how to calculate the pixels in the new image and then returns the resized image [4].

```
>>> new_width  = int(img.shape[1]/2)
>>> new_height = int(img.shape[0]/2)
>>> smaller = cv2.resize(img, (new_width, new_height), interpolation =
cv2.INTER_AREA)
>>> cv2.imwrite('small_monarch.png', smaller)
```

A new 3 dimensional matrix was formed with the appropriate colors for the new smaller image, called smaller and it was written to a file called small_monarch.png. After executing the command above, make sure to view both images.

Isolating a Specific Color

It can be helpful for object tracking to detect a specific color in an image [5]. One way to do that is to convert the image from RGB (Red, Green, Blue) format to HSV (Hue, Shift, Saturation). In the HSV, the hue corresponds to the color and the shift and saturation determine the brightness of the color, where as RGB are a the intensity of each of those three colors. So, in order to isolate the yellow flowers in our picture, we need to determine the approximate value of the yellow in the image. Using a color picking tool or the chart provided on the previous link, we can find that yellow has a low hue value around 30. So, a mask can be created to highlight just the portions that are around that value that also have larger shift and saturation values.

Illustration 2: Mask of Image

```
# Changing from BGR to  HSV
>>> hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# 1st set of values (27, 150, 150) form lower limits, the second the upper
>>> mask = cv2.inRange(hsv, (27, 150, 150), (33, 255,255))
```

Locating the Region of a Specific Color

The mask produced is a binary or gray scale image that is black and white only. Then the moments method can be used with this new two dimensional array to find the center of mass for this image. Of course because some yellow existed outside of the top left flowers, some other yellow bits can be seen. For now, we will ignore these smaller values, but will realize that they will shift the center of mass slightly.

```
>>> M = cv2.moments(mask)
>>> cX = int(M["m10"] / M["m00"])
>>> cY = int(M["m01"] / M["m00"])
>>> print ("Center: (%s , %s)" % (cX, cY))
Center: (103 , 66)
```

Regarding about black and white images, in an image that is stored with one byte per pixel, a full white pixel is the largest value and a totally black pixel is the smallest value or 0. It should also be noted that the origin is in the top left. So for this image that is 400 pixels wide and tall, the center of mass is about one fourth from the left side just a little below the top as expected, even accounting for the small yellow bits left over.

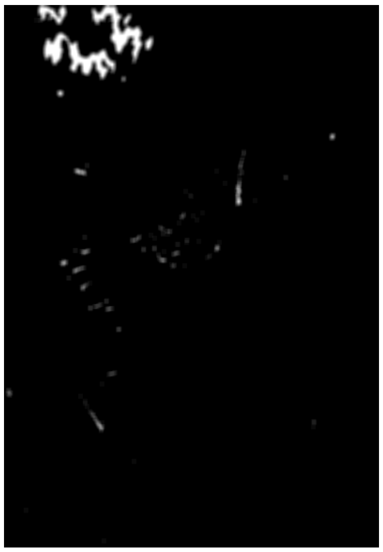
Illustration 3: Blurred Image

A note about the notation used for moments. The moment provides a way to determine the center of mass. In this 2 dimensional example, m00 is simply the mass of the object. m10 is the moment with respect to x and m01 is the moment with respect to y. m20 would be the second moment with respect to x and so on. To find the center of mass along the x axis, divide m10 by m00. In this case, mass is the color value of each pixel, with white carrying the largest weight and black the least.

With the origin in the upper left, this places the center of mass for the white portion of the grayscale image about a quarter of the way to the right and slightly below the top. As can be seen from Illustration 3, several portions of "white" still exist which can skew the result of our moment slightly. One way to remove "noise" or bits that are unwanted is to use some form of filter to remove it. In this case, let's assume that the desire it to locate the larger mass of yellow in the upper left corner. In this case, the smaller dots of yellow are not as important. One way to do this is smooth out the bits of the gray scale with a filter, several exist, but in this case the blur method will be used [6].

```
>>> mask = cv2.blur(mask,(5,5))
```

The blur method takes the average of a 5 by 5 area around a pixel to get the value of that one pixel. A fancy way of saying this is that it convolves a 5 by 5 matrix of ones with the image itself. Convolution is an important concept that will be used throughout signals and systems, and in this case it just means to take average that 5 by 5 square around each pixel.

```
>>> thresh = cv2.threshold(mask, 200, 255, cv2.THRESH_BINARY)[1]
>>> cv2.imwrite('monarch_thresh.png', thresh)
>>> M = cv2.moments(thresh)
>>> cX = int(M["m10"] / M["m00"])
>>> cY = int(M["m01"] / M["m00"])
>>> print ("Center: (%s , %s)" % (cX, cY))
Center: (94 , 37)
```

Now the threshold method will be used with this new blurred image, it is something similar to the inRange used before to pick the yellow regions. Remember, white is 255 and black 0, so with the new blurred image, we chose just the pixels with a very high white value of 200 or more to get the following image. Notice the new center of mass that is now not influenced by the "noise" from the other smaller yellow bits.

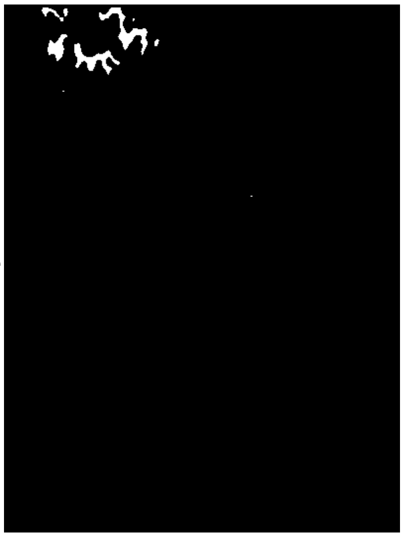
Illustration 4: With Threshold

Homework

Prelab – Have this completed prior to showing up in the lab

1. Briefly explain the purpose of the following methods provided in the OpenCV library.
   a. imread
   b. cvtColor
   c. blur
   d. threshold
   e. moments
2. Write python code that will open an image, resize it based upon a user input of the width while retaining the aspect ratio and then will save it as resized.jpg.
3. Given an image of width, w and height, h, calculate the angle from the center of the image if you are given a random pixel. Assume the origin is in the upper left and that the center is in the lower middle.
4. Write python code that will determine the size of the image and create a new image that is scaled such that the new image is thumbnail that fits within a 200 by 200 pixel square. Make sure to preserve the aspect ratio for the new image.
5. Define center of mass for a 2 dimensional object. The OpenCV moments method refers to m00 as the mass, m01 as the moment with respect to y and m10 as the moment with respect to x.
   a. Define the center of mass in terms of m00, m01 and m10.
   b. Sketch and determine the center of mass for
      i. A rectangle with corners at (2,1), (2, 6),(5,1),(5,6)
      ii. A triangle with corners at (2,2), (2,8), (5,8)

Lab

1. Connect the Pi camera and take some pictures.
2. Either create script or alter the script that comes with the camera module (commenting your changes) to save the picture as a jpg with the first part of the file being user specified.
3. Update the script to save both the first image and then a thumbnail that fits in a 200 by 200 pixel square, the thumbnail can be called <<user specified name>>`_thmb.jpg`.
4. Go through the steps used in the examples for this module. It may be easier to write a short script for this instead of running the commands in the python command line mode. NOTE: It may be necessary to

use file transfer to view the images as the `imshow` command will not work when using ssh to connect to the Pi and run scripts.

5. Adapt the previous script to find a racquetball in an image. Determine the angle from the camera position for the ball. Note the color change for the racquetball.

Bibliography

[1] Getting Started with Picamera, https://projects.raspberrypi.org/en/projects/getting-started-with-picamera, Accessed 7/10/19

[2] opencv-python, https://pypi.org/project/opencv-python/, Accessed 7/12/19

[3] Basic Operations on Images, https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_core/py_basic_ops/py_basic_ops.html, Accessed 7/12/19

[4] Geometric Image Transformations https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html, Accessed 7/12/19

[5] How to define a threshold value to detect only green colour objects in an image :Opencv, https://stackoverflow.com/questions/47483951/how-to-define-a-threshold-value-to-detect-only-green-colour-objects-in-an-image, Accessed 7/13/19

[6] Smoothing Images, https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_filtering/py_filtering.html#filtering, Accessed 7/13/19