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,

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.

****WARNING: Camera Hardware****

The Camera connector cable is very fragile, and is easy to destroy. Take care not to twist, fold, or scratch the camera connector cable. The Camera module, additionally, has metal on it. **Do not rest the camera module on your Pi, or on another piece of metal, or you could permanently damage your Pi or your Camera module.**

Additionally, The Camera module needs to be plugged into the camera connector on the Pi. To do this, pull the gray retaining clip out on the board (it should travel ~1mm), insert the camera connector cable such that the metal pins on the connector line up with the metal pins inside the Pi's camera connector, and close the retaining clip.

Lastly, connect the camera while the power to the Pi is turned off to avoid damaging the Pi.

Taking Pictures

The following lines will take images using the camera using the python interpreter. The first few lines set up a PiCamera object. Note that the newer picamera2 library is being used [2]. The image taken has the default resolution, in the next module altering the resolution of the images can be explored. Smaller resolution images will also be smaller in size and thus faster to load into memory, save to the drive and capture initially. They will obviously not contain as much information, but that may be acceptable for a given application. The tradeoff in speed, size and resolution then becomes a design decision for the particular application.

```
>>> from picamera2 import Picamera2
>>> camera = Picamera2()
>>> camera.start()
>>> camera.capture_file('testing1.png')
```

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

****** Note: Verbose warnings ******

The `picamera2` module may send information and warnings to the screen, unless these are errors, they can be ignored. When running the python scripts, messages can be suppressed by they added the following. 2 refers to errors and warnings, > sends them to `/dev/null` which will be ignored by linux.

```
$ python myscript.py 2> /dev/null
```

Invoking python similarly will help suppress the warnings.

```
$ python 2> /dev/null
```

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 [3]. First, OpenCV must be installed [4].

```
$ sudo apt-get install python3-opencv
```

Running the command 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 image in the illustration below can be downloaded with a browser and copied to the directory repo, or the following command will also download it if the computer is connected to the internet. The curl command is often useful for transferring data over a network via the command line.

```
$ curl http://feherhome.no-ip.biz/temp/wustl/205/monarch.jpg > monarch.jpg
```

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.

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

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

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. If you have a monitor connected to the Pi, the following will display the image, while the second command will remove the image picture from the screen.

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

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

> ****** Hint: `imshow` & File Transfer ******
>
> The `imshow` method displays the image to the GUI for the Pi. When logged in via ssh, this will not work.
>
> It is often desirable to transfer files from the Pi to another machine, especially when using ssh to log into the Pi when doing image processing. Make sure to get comfortable using winscp in Windows or sftp from the terminal with a Mac.

Locations in Image

The previous image was stored in a 3D array that had 533 rows, 400 columns with the 3rd dimension used for RGB. One thing that often confuses students is that the rows, which often comes first when representing arrays actually are associated with the Y axis visually and the columns are associated with the horizontal/X axis.

The origin is in the upper left, with increasing column numbers moving right and increasing row numbers moving down in the image.

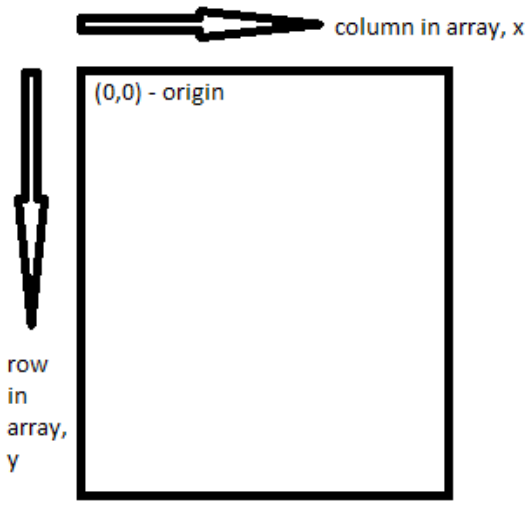

Illustration 2: Image Location

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 [5].

```
>>> 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.

Working with HSV Color

RGB (Red, Green, Blue) is probably one of the more familiar color models for images. However, in order to isolate specific shapes, it is often more advantageous to use, HSV. HSV stands for Hue, Saturation, and Value. Note that HSV is **not the same as HSL (which is another name for HSB)**. See reference [6] for an online HSV picker which can be used to see how manipulating a color manipulates its HSV values. In general:

- *Hue* specifies the color "family" (technically the wavelength) (i.e. red, purple, orange). Note that the colors wrap around, starting with red and ending with red as it transitions the spectrum of visible shades.
- *Saturation* specifies how much of that color is present. Note that it starts lighter and gets darker as the saturation is increased.
- *Value* specifies how much light is present or reflected. A lower value would correspond to a space with less light and a higher value would be a space with more light.

A good explanation for the color science behind HSV is provided in reference [7].

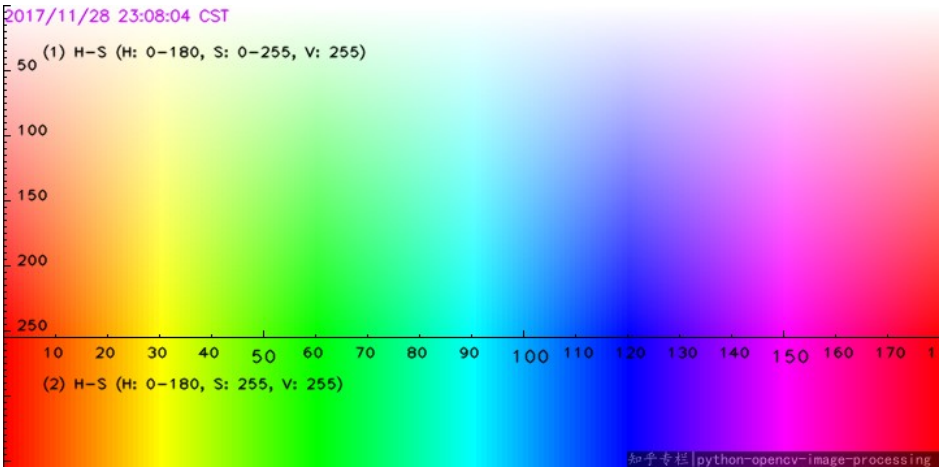

Illustration 3: HSV Color Map for OpenCV[6]

NOTE: Hue Range

While many color pickers allow Hue selection in a range between 0 and 360, OpenCV uses a Hue value between 0 and 180. To convert a 0-360 Hue to a 0-180 Hue, simply divide the Hue by 2 (and vice versa). Similarly, the saturation and value might also need to be properly scaled. OpenCV uses 0-255 for the ranges of Saturation and Value.

Isolating a Specific Color

It can be helpful for object tracking to detect a specific color in an image [8]. One way to do that is to convert the image from RGB (Red, Green, Blue) format to HSV (Hue, Saturation, Value). So, in order to isolate the yellow flowers in our picture, first determine the approximate value of the yellow in the image. Using a color picking tool or the chart provided on the previous link, it can be seen that the yellow has a low hue value around 30 (See highlighted values for `inRange` command). A mask can be created to highlight just the portions that are around that value that also have larger shift and saturation values. Again, note that some color pickers do not use the same scale as OpenCV. Lastly, one tricky color to isolate is red which has both low and high values. For that color a tip would be to mask the lower values and then the upper with two different masks and then combine the results.

Illustration 4: 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))
```

In this case, the saturation is high, indicating a more "dark" yellow and the value is also high, indicating a "bright" picture. These values may change depending upon the lighting in which the image is taken.

Locating the Region of a Specific Color

The mask produced is a binary or gray scale image that is black and white only. 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, ignore these smaller values, but 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 (f"Center: ({cX} , {cY})")
Center: (103 , 66)
```

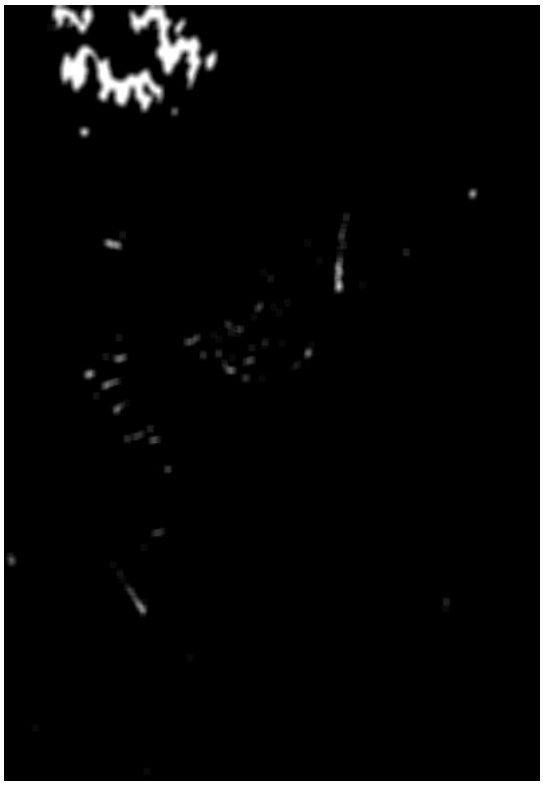

Illustration 5: Blurred Image

Regarding 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. Recall, the origin is in the top left. So for this image that is 400 pixels wide and 533 pixels 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.

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 "yellow" in the image. 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 4, 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 [9].

```
>>> mask_blur = 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. The blur performs a 2 dimensional moving average. Another way of explaining the blur would be to say 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_blur, 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 (f"Center: ({cX} , {cY})")
Center: (94 , 37)
```

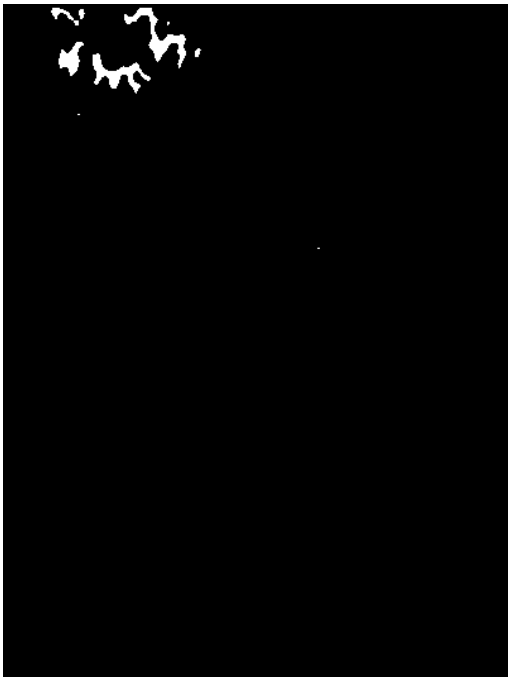

Illustration 6: With Threshold

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.

```
>>> img = cv2.circle(img, (103,66), 5, (255,0,0), 2) # blue circle
>>> img = cv2.circle(img, (94,37), 5,  (0,255,0), 2) # green circle
```

The last illustration shows the difference in calculating the center of mass for the yellow portion of the image with (red circle) and without the blur (green circle). The red circle can be drawn on an image with the following command and the green with the second.

The first argument in the two commands above is the image, the second the center of the circle (X,Y not Row, Column, which again could be confusing), the radius in pixes of the circle, the color in terms of B, G or R and the thickness of the perimeter. See how using the blur technique was able to filter out the other yellow components that were much smaller and not of interest.

Illustration 7: Center of Mass Comparison

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

Edge Detection

While, isolating colors can be an effective way to identify different objects that can assist with navigation, another technique is to determine the edges in a given image. And finding colors involves looking for groups of pixels with similar properties, finding an edge is done by examining where the image properties change. Hallways, sidewalks, roadways all generally have clear changes in image color that could be identified. These changes also usually correspond to the direction of the path of navigation.

Illustration 9: Hallway

Illustration 8: Sidewalk

Finding the change in properties is similar to using a derivative, look for the largest gradient or difference between pixels with neighboring pixels. The following steps are usually used in the process.

- Convert the image to a gray scale image. The gray scale image has a benefit of less data to process.
- Blur the image to remove some of the smaller edges and get them to blend with the surrounding areas and reduce the affect of smaller pixels that are different and look for more major changes.
- Run the edge detection on the new blurred image which locates regions where the values have the highest gradient or change which correspond to the edges.

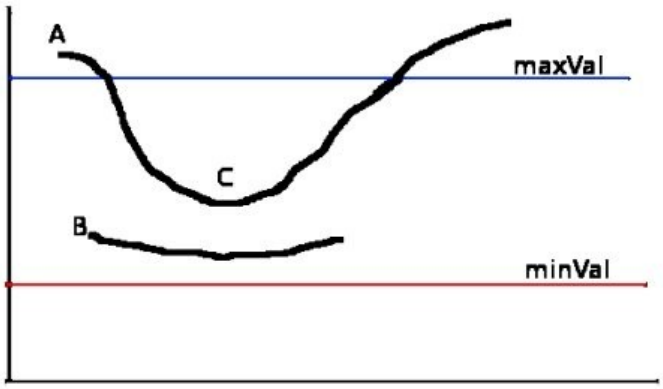

Illustration 10: Canny Hysteresis[10]

The Canny edge detection algorithm(developed by John Canny) first looks for areas of large gradient difference [10]. It then accepts two arguments, a maximum value

of the required difference to identify as an edge and a minimum value required below which the edge is discarded (maxVal and minVal in Illustration 10). Edges must have pixels above the maxVal otherwise they are not considered edges and if a portion of the edge falls below the maxVal but is above the minVal it is still considered part of the edge. Areas that fall below the minVal are not considered part of the edge.

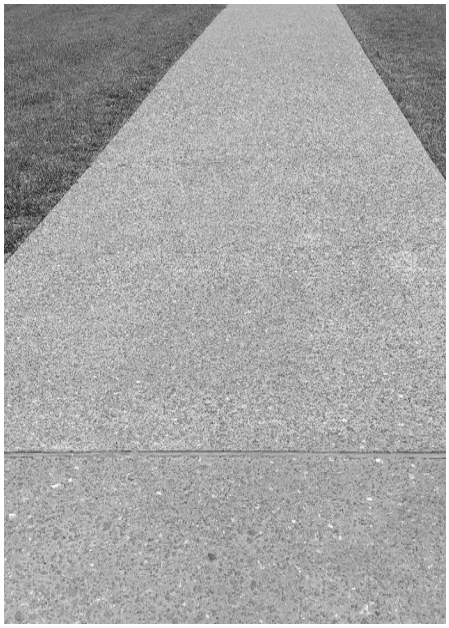

Illustration 13: Gray Scale Image

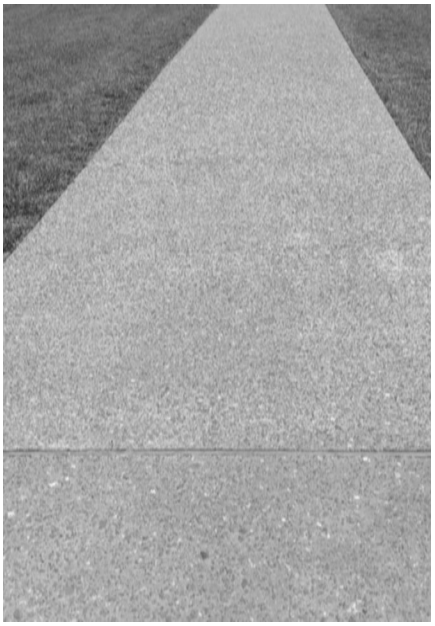

Illustration 11: Blurred Image

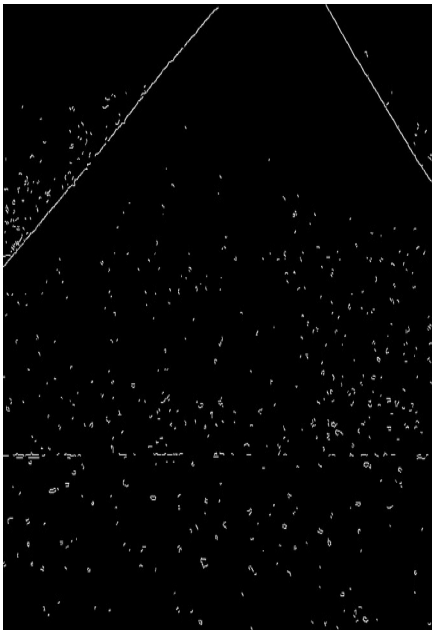

Illustration 12: Canny Detection Applied

The three images show the process on the image of the sidewalk from Illustration 9. The blurred image has a very subtle change with some of the grassy and "pebbley" areas blurred. The lines of the sidewalk are then shown with smaller edges that correspond to the grass or pebbles. The next step is to identify the lines that correspond to the edges that are desired, discarding the smaller spots and the non-linear edges. The Hough line detection algorithm can be applied to the edges discovered from the Canny Edge Detection [11]. The Hough Line Transform converts (or transforms) the lines over to a polar representation where it is easier to identify the lines. OpenCV provides the `HoughLines` and `HoughLinesP` methods, the first returns a radius and angle which represents the line found. The second know as the Probabilistic Hough Line transform returns the endpoints of the line. The HoughLineP method has the advantage of being more efficient and it also provides the endpoints which will lie on the actual image. Another Hough transform exists for identifying circles and ellipses in images.

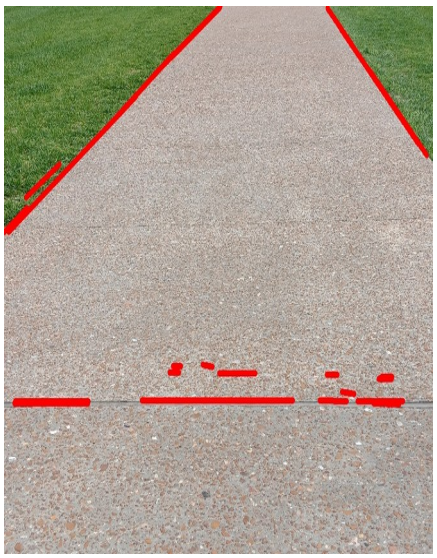

Illustration 14: HoughLinesP Applied

The lines detected using the Probabilistic Hough Lines approach are drawn in red on the original image. Some of those lines can be further eliminated (the horizontal ones for example) because they are clearly not in the direction of the sidewalk.

```
lines = cv2.HoughLinesP(
     edge,                    # image 2 process
     1,                       # rho, resolution in pixels
     np.pi/180,               # theta, resolution of angles
     threshold=100,           # votes required
     minLineLength=20,        # shorter lines rejected
     maxLineGap=10)           # max gap between lines to be same
```

In the above command `lines` is an array of coordinates that correspond to the endpoints of the lines that have been identified. Both color identification and edge detection are both valuable techniques for autonomous navigation. Selecting the correct parameters is a challenge and requires an understanding of how the various algorithms work:

- proper thresholds
- size of region to blur
- minimum line length
- size of the image to process

These will all be discussed further in the next module.

Picking Appropriate Parameters

Finding the right hue, threshold, MaxVal/MinVal for the Canny algorithm or line lengths or gaps for the Hough algorithm can almost feel daunting. Ways to determine them automatically exist by examining the characteristics of the pixels. For example, examine a histogram of the various pixel properties and look for spikes that relate to values of interest. Interesting problems such as this are examined in the field of imaging processing.

Below is a program that employs two trackbars to choose the X and Y value of a given pixel with the HSV values shown on the top bar of the window. When the program stops, an image will be saved with the final pixel chosen along with the HSV values for that pixel.

- Lines 2-3: Get the file name
- Lines 6-9: Convert the image to HSV
- Lines 11-12: Create a null function for the trackbar
- Lines 14-19: Create trackbar, name the window (`namedWindow`)with HSV values for given pixel
- Lines 22-25: Put small red circle on pixel chosen
- Lines 26-28: get HSV values at that point
- Lines 30-32: show image with circle and change title of window with HSV values
- Lines 34-43: Break out of while loop with spacebar and save various images

This program can be adapted to look for other parameters besides the color. Just keep in mind that as environmental conditions change, the parameters may need need to adjust so more sophisticated techniques may need to be employed in those cases.

```
1>    import cv2
2>
3>    filename  = input("Enter name of file to process: ")
4>    orig_img  = cv2.imread(filename)
5>
6>    height, width = orig_img.shape[:2]
7>    gray      = cv2.cvtColor(orig_img, cv2.COLOR_BGR2GRAY)
8>    hsv_img   = cv2.cvtColor(orig_img, cv2.COLOR_BGR2HSV)
9>    new_img   = orig_img.copy()
10>
11>   def blank(x):   # needed for trackbar
12>      pass;
13>
14>   vertstr = (f'vert 0: {height-1}')
15>   horzstr = (f'hort 0: {width-1}')
16>   cv2.namedWindow('window', cv2.WINDOW_NORMAL)
17>   cv2.resizeWindow('window', width, height)
18>   cv2.createTrackbar(vertstr, 'window', 0, height-1, blank)
19>   cv2.createTrackbar(horzstr, 'window', 0, width-1,  blank)
20>
21>   while True:
22>      lc = cv2.getTrackbarPos(vertstr, 'window')
23>      hc = cv2.getTrackbarPos(horzstr, 'window')
24>      new_img = cv2.circle(new_img, (hc, lc), radius=int(width*0.005),
25>                           color=(0,0,255), thickness=int(width*0.002))
26>      hval = hsv_img[lc, hc, 0]
27>      sval = hsv_img[lc, hc, 1]
28>      vval = hsv_img[lc, hc, 2]
29>
30>      cv2.imshow('window', new_img)
31>      outstring = (f'H: {hval}\tS: {sval}\tV: {vval}')  # print at top
32>      cv2.setWindowTitle('window', outstring)
33>
34>      if cv2.waitKey(1) == 32:     # stop when space bar hit
35>         cv2.destroyAllWindows()
36>         outstring = (f'({lc}, {hc})  H:{hval} S:{sval} V:{vval}')
37>         # put circle on final spot
38>         orig_img = cv2.circle(orig_img, (hc, lc),int(width*0.005),
39>                        color=(0,0,255), thickness=int(width*0.002))
40>         orig_img = cv2.putText(orig_img, outstring, (height//10,
41>                  width//10), cv2.FONT_HERSHEY_SIMPLEX, 2, (0,0,255), 5)
42>         cv2.imwrite("new_img.jpg", orig_img)
43>         break;
```

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

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. inRange
   d. blur
   e. threshold
   f. moments
   g. imwrite
   h. createTrackbar
   i. Canny
   j. HoughLinesP
2. Write python code that will open an image, resize it based upon a user input of the desired width while retaining the aspect ratio (the aspect ratio is ratio of the width and height). Save the new image as `resized.jpg`. Call this program1.py.

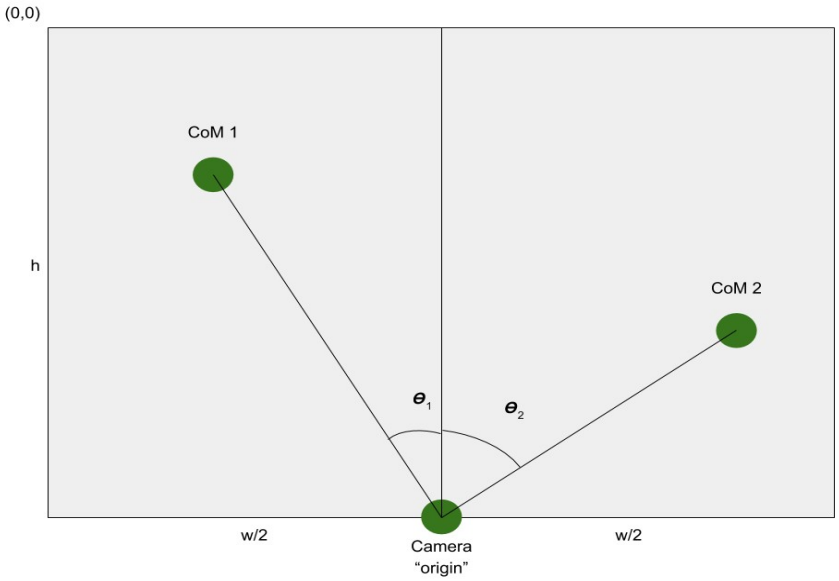

Illustration 14: Image Angle Calculations

3. Given an image of width, w and height, h, calculate the angle from the reference, which is the camera origin, of the image if you are given a random pixel. Assume the origin of the image is in the upper left. An example of setup is provided in illustration 14, where CoM 1 and CoM 2 are sample points corresponding to θ_1 (which should be negative) and θ_2 (which should be positive) respectively.
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. Call this program2.py.
5. Write the OpenCV code needed to:
   a. Open an image
   b. Create a copy of the image in HSV format
   c. Create a mask that identifies the green dots in the image
   d. Saves a copy of the masked image as <image_name>_mask.jpg.
   e. Call this program3.py.
6. Assume the image from problem 5 is 300 rows by 500 columns.
   a. What is the size in bytes of the RGB image in python?
   b. What is the size of the mask created from the image in python?
   c. If the RGB version of the image were resized by the program from problem 4, what would be the size of the new resized RGB image in python?
7. 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)

8. How would you change the color identifying program provided on page 10 so that the sliders would identify the MaxVal and MinVal parameters used in the Canny function in OpenCV?
9. The HoughLinesP algorithm returns a set of endpoints that correspond to the lines found. Write a loop that will iterate over these endpoints and draw a line on the original image that corresponds to that line.

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

Lab

1. Connect the Pi camera (again, with the power off).
   a) Install OpenCV
   b) Additionally, on bookworm, libcamera-apps must be installed before the Picamera2 class will work.
   `$sudo apt-get install libcamera-apps`
   c) Take some pictures and view them.
2. Use the code from problem 2 of the prelab to resize some images that were taken in the previous problem. Save the program as `program1.py`. Also include one of the original images and its resized version in your repo. *(Original.jpg and Original_resized.jpg)*
3. Use the code from problem 4 of the prelab to resize your image from the first part of the lab. The newly created thumbnail image shoiuld be called `Orginal_thmb.jpg`. The program should be `program2.py`.
4. Determine the size of the original, resized and thumbnail of the original image. Save this information in a text file: `results.txt`
5. Go through the steps used in the examples for this module. Adapt the code from problem 5 in the prelab to create 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. Name the resulting script `program3.py`
6. Run the color identifier program provided to examine the HSV values for the blue areas
7. Adapt the previous script to find a section of blue painter's tape in an image.
   a) Run the color identifier program provided to examine the HSV values for the blue areas, comment on these in the results.txt file.
   b) Place a small red dot at the center of mass of the tape. Determine the angle from the camera position for the ball. Note: you will need to change the threshold colors for the tape. List the angle in the results.txt file.
   c) Name this script `track_tape.py`. Save an image of the ball as *tape.jpg* and the image resulting from your script as *tape_filtered.jpg.*
8. Implement your slider program for using the Canny edge detection from problem 8 in the prelab, call this program4.py. Take picture of the hallway with your camera, call it hallway.jpg. Comment on the best parameters found for identifying the edges in the image. Save the image when the program ends as hallway_edges.jpg.
9. Write a program that now identifies where the wall meets the floor using the HoughLinesP function in OpenCV, call this program5.py. The output should save the new image to hallway_lines.jpg.

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

Deliverables (these are expected in your repository):

- images/ (images is a directory containing the following files)
  - original.jpg
  - original_resized.jpg
  - original_thmb.jpg
  - tape.jpg
  - tape_filtered.jpg
  - hallway.jpg
  - hallway_edges.jpg
  - hallway_lines.jpg
- program1.py
- program2.py
- program3.py
- track_tape.py
- program4.py
- program5.py
- results.txt

Bibliography

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

[2] The Picamera2 Library, https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf, Accessed 12/28/23

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

[4] 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

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

[6] HSV Color Picker, https://alloyui.com/examples/color-picker/hsv.html, Accessed 12/14/19

[7] Hue, Value, Saturation, http://learn.leighcotnoir.com/artspeak/elements-color/hue-value-saturation/, Accessed 12/14/19

[8] 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

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

[10] Canny Edge Detection, https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html, accessed 7/1/24

[11] Hough Line Transform, https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html, accessed 7/1/24