Raspberry PI – II - OpenCV

Contents

DAY 1 - NUMPY MODULE DAY

REVIEW:

- Deep copy vs Shallow copy: C2Python.pdf
- Command Line Parser : see the C2Python.pdf .
- Slicing Technique : https://www.tutorialspoint.com/numpy/numpy_indexing_and_slicing.htm
-

Check to see if numpy module have been installed. In addition, we shall use python 3.6.2 as well.

To check:

- $ python -c "import numpy"
- If not, you will need to run the following:

pip3 install numpy or apt-get install python3-numpy. You may need to apt-get install python3-pip.

- To check version : in the script: import numpy np \ print ("numpy: ", np.__version__)

TRY OUT THE MAJOR FUNCTIONS AND METHODS:

IMPORTANT: (Must take the initiative to be Inquisitive. If you finish these within 30 minutes, that usually is a symptom of you not being proactive to test out various scenarios.)

- Go thru the list in the Numpy Functions in the C2Python.pdf
- Click on each one, try each of them out with your Python IDE

WORK ON NUMPY PROBLEMS – SAMPLE ANSWERS

Write a program to do each of the following problem.

1- Reverse an 1D array object a = np.arange(10).

Input : [0 1 2 3]	Output: [3 2 1 0]

2- Remove an Nth index character from an 1D character array object

Input : “Storming Robots” User wants to remove 9th character	Output: “StormingRobots”

3- Remove an Nth index element from an 1D integer array object.

SampleInput : [1 2 3 4 5 6 7 8 9] User wants to remove 4th element	Output: [1 2 3 5 6 7 8 9]

4- Reverse an 2D array (first row becomes last row).

Sample input : [[1 2 3] [4 5 6] [7 8 9]]	Output: [[7 8 9] [4 5 6] [1 2 3]]

5- Transpose a 2D array.

Sample Input : [[1 2 3] [4 5 6]]	Output: [[1 4] [2 5] [3 6]]

6- What is the difference between these two arrays :

Sample input:	Output:
a = np.array([1,2,3]) print (a.T, a.ndim) a = np.array([[1,2,3]]) print (a.T, a.ndim)	What are the dimensions ? What the output looks like

7- convert a list or tuple of 6 elements into 2x3 arrays. E.g.

Sample input:	Output:
theList = (1,2,3,4,5,6_ theTuple = [1,2,3,4,5, 6]	[[1 2 3] [4 5 6]]

8- convert the values of Centigrade degrees into Fahrenheit degrees. Centigrade values are stored into a NumPy array.

Sample Array [0, 12, 45.21 ,34, 99.91]

Expected Output:

Values in Fahrenheit degrees:

[0. 12. 45.21 34. 99.91]
Values in Centigrade degrees:
[-17.77777778 -11.11111111 7.33888889 1.11111111 37.72777778]

9- Combine a one and a two dimensional array together and display their elements.

Sample input:	Output:
A1: [0,1,2,3] A2: [[4,5,6,7], [8,9,10,11]]	[4 5 6 7 8 9 10 11 0 1 2 3]

10- Remove the nth row from a nonempty array.

Sample Input :	Output if you desire remove row 1
[[1. 1. 1.] [1. 1. 1.] [1. 1. 1.]]	[[1. 1. 1.] [1. 1. 1.]]

11- reverse an 2D array (first element becomes last).

Sample Input : [[1 2 3] [4 5 6] [7 8 9]]	Output: [[9 8 7] [6 5 4] [3 2 1]]

12- find common values between two arrays.

Sample input: Array1: [0 10 20 40 60] Array2: [10, 30, 40]	Output: [10 40]

13- Find the set difference of two arrays. The set difference will return the sorted, unique values in array1 that are not in array2.

Sample input: Array1: [0 10 20 40 60 80] Array2: [10, 30, 40, 50, 70, 90]	Output: [0 20 60 80]

14- find the set exclusive-or of two arrays. Set exclusive-or will return the sorted, unique values that are in only one (not both) of the input arrays.

Sample input: Array1: [0 10 20 40 60 80] Array2: [10, 30, 40, 50, 70]	Output: [0 20 30 50 60 70 80]

15- Find the union of two arrays. Union will return the unique, sorted array of values that are in either of the two input arrays.

Sample input: Array1: [0 10 20 40 60 80] Array2: [10, 30, 40, 50, 70]	Output: [0 10 20 30 40 50 60 70 80]

16- Construct an array by repeating.

Sample input: [1, 2, 3, 4]	Output: Repeating 2 times [1 2 3 4 1 2 3 4] Repeating 3 times [1 2 3 4 1 2 3 4 1 2 3 4]

17- Find the indices of the maximum and minimum values along the given axis of an array

18- Compare two arrays, a & b. Evaluate whether the following expressions are true or false.

a > b
a >= b
a < b
a <= b

Sample input:	Output:
Array a: [1 2]	a>b [False False]
Array b: [4 5]	a>=b [False False]
	a < b [True True]
	a <= b [True True]

19- Create a 3x5 numpy array, and add an extra row and column to it.

20- Replace all elements of numpy array that are greater than specified array.

21- File I/O exercise:

Ask users for input.e.g. *--f outFile -r 5 -c 4*
: you will create an 5 x 4 matrix filled with all ones on diagonal.
: create the transpose and write it to the outFile will contain this matrix.
: Read it back into your code and print out to ensure you have written it correctly.

DAY 2 - OPEN CV INTRO – WITH IMAGES

DO NOTE: OpenCV provides EXCELLENT online references. Carry proper learning habit, you will achieve a great deal from this workshop, i.e. be **inquisitive** and **resourceful.**

By the end of the day, you should be able to:

- Programmatically manipulate a static image
- Programmatically manipulate a live video feed
- Programming with slicing technic on both static images and videos

INSTALLATION

Building openCv modules will take hours. Therefore, we have installed OpenCV on your disc.

In code, you need to add in :

```
import cv2
import numpy
```

Before you get started:

1. create a work folder for opencv exercises. E.g.

```
$ mkdir /home/pi/cvWk
```

1. Check to see if the sample images folder exist. Either using the desktop folder menu, or just type in the terminal:

   ```
   $ ls -l /home/pi/Source/OpenCV/MyCode/Python/OpenCV/"My OpenCV Samples"/"My OpenCV Samples"/images
   ```

2. Create softlink in your Desktop folder to allow you to have quick access from desktop:
   You will use : ln -s target-source-path-to-link-to newLinkName . E.g.

   ```
   $ cd ./Desktop
   $ ln -s /home/pi/Source/OpenCV/MyCode/Python/OpenCV/"My OpenCV Samples"/"My OpenCV Samples"/images images
   ```

3. The icon for the images should show up at your desktop.
4. If you look at home Desktop folder, you should see:

```
$ cd /home/pi/Desktop
lrw-rw-rw- images → .. the long folder name
```

5. Build a stand to hold the cam to enable it to allow you to switch between facing forward to downward. Or, ultimately some pivot mount will be ideal.

To install the cam as image capture device type:

- You will need to load the media drivers and platform codecs so that you can use the same code for either using a usb-cam or the PI-cam.

- sudo modprobe bcm2835-v412 : to manually add, followed by reboot

  OR

```
$ sudo vi /etc/modules
```

- add this in : bcm2835-v4l2 . (beware! It it a with lower case "L")

- if you wish to do more fancy resize (do not do this now though) , etc. : sudo v4l2-ctl --set-fmt-video=width=1280,height=720,pixelformat="H264" -d /dev/video0.

BASIC OPENCV FUNCTIONS TO HANDLE STATIC IMAGE:

Most basic operations to show a static image file

Steps to read an image file:

Create a problem using the following steps:

	Code Sample
1- Import the modules	import cv2 import numpy
2- Read in the image file	img = cv2.imread(the filename string)
3- Show the image	cv2.imshow("Label of the window", img)
4- Handy wait for user key	cv2.waitKey(0)
5- Terminate the window	cv2.destroyAllWindows( )

Some useful expressions:

Check out the dimension of the file	img.shape img.size H = img.shape[0] W = img.shape[1]
a numpy.ndarray object represents the pixel. Gives you the color tuple (B, G, R)	px = img[shape[0] // 2, shape[1] // 2]
Numpy array of pixel color	(b,g,r) = px
Copy from one section to another	img[10:30, 20:50] = img[50:70, 100:130]
More on how you read in images	
Read in as color image with (B,G,R) Then, get the height, width and format type. Format type : 0 == gray scale, > 1 == B G R format < 0 == as it is	img = cv2.imread("myfile.jpg") (Height, Width, C) = cv.imread() #note: a tuple of 3
Read in as color image with grayscale	img = cv2.imread("myfile.jpg", 0) (Height, Width) = cv.imread() # note , a tuple of 2

EXERCISE 1 – YOUR MOST BASIC IMAGE OPERATION PROGRAM

Displaying and modifying an Image. For each step in this exercise call imshow( ) to display the changes taking place.

a) Using imread( ) and imshow( ) to display a jpg or png file. Use waitKey( ) to pause until the 'q' key is hit then call destroyAllWindows( ) to close all the windows.
b) What is the size of the image (width and length)?
c) Where is the x,y of the center pixel? What is its color (BGR) value?
d) Put some lines, rectangles, circles and text in your image.
e) use flip( ) to perform a vertical and/or horizontal transform
f) Use resize( ) to change the size of your image to ½ the width and ½ the height of the original
g) Save out your results with imwrite( )
h) Copy a 50x50 px section from the center of the image, and paste it to 4 corners

EXERCISE 2 – MANIPULATION OF THE IMAGE - SLICING

a. Open and display the original jpg file from Exercise 1
b. Take a slice of the image (say a 100x100 slice) and display it in another window
c. Take the same slice and reinsert it back into the original image but at another place (different x,y). This can be achieved by:

```
sliceimg = image[0:100,0:100]
dup = image.copy( )
dup[100:200,100:200] = sliceimg
```

Helpful Reference for Ex 1 and 2:

Basic Drawing Functions

1. line( )
2. rectangle( )
3. circle( )
4. putText( )
5. resize( )
6. Fonts available : find "InitFont" from the basic drawing line

EXERCISE 3 - BITWISE OPERATIONS -

a. Create a black 400x400 image and place a large white rectangle inside (Note: use np.zeros((400, 400), dtype = "uint8") to create the initial image.
b. Create another blank 400x400 image and place a large circle inside of it.
c. Create a new image using the rectangle and circle by calling bitwise_And( ). What do you see and why?
d. Do the same with bitwise_Or( ),bitwise_Xor( ),and bitwise_Not( ). Explain what you see.

EXERCISE 4 - ADD AND REDUCE INTENSITY –

Let's modify the intensity of all pixels in our image . We accomplish this by:

- constructing a NumPy array that is the same size of our matrix (filled with ones), then multiplying it by 100 to create an array filled with 100's.
  - E.g. np.ones(image.shape, np.uint8) * 100
- Then we simply add the images together.
- Watch what happens to the image.

```
e.g.
M = np.ones( image.shape, np.uint8 ) * 100
added = cv2.add(image, M)
cv2.imshow("Added", added)
```

- After that, you should try to “subtract” the value.

Helpful expressions for Ex.3 & 4

rect = np.zeros((300, 300, 3), np.uint 8)	create color object with max 300x 300 uint8
rect = np.zeros((300, 300), np.uint 8)	Create grayscale object with max 300x 300 uint8
cv.rectangle (rect, (20,20), (280, 280), red , -1) red = (0,0,255) ; -1 = fill up	create a rectangle from pt(20,20) to pt(280,280) filled up with color red
Red = (0,0,255) | blue = (255, 0, 0) | green = (0, 255,0)	
newImg = cv.bitwise_And(rect, circle)	assuming that you have created a circle and rect . newImg = bit-and between rect and circle
newImg = cv.bitwise_Or(rect, circle)	Bits-IOR between rect and circle image
newImg = cv.bitwise_Xor(rect, circle)	Bits-XOR between rect and circle image
newImg = cv.bitwise_Not(rect),copy	Bits-NOT of rect and shallow-copy to newImg

	rect:	11111111	00000000	00000000	
bit-and	circle	00000000	11111111	00000000	
		00000000	00000000	00000000	ie. black
	rect:	11111111	00000000	00000000	
bit-ior	circle	00000000	11111111	00000000	
		11111111	11111111	00000000	ie. yellow
	rect:	11111111	00000000	00000000	
bit-xor	circle	00000000	11111111	00000000	
		11111111	11111111	00000000	ie. yellow
	rect:	11111111	00000000	00000000	
NOT					
		00000000	11111111	11111111	ie. cyan

DAY 2 – BASIC OPENCV FUNCTIONS TO HANDLE VIDEO

CAPTURE VIDEO FOR DISPLAY AND BASIC CONTROLS

Do Your first video capturing Program:

Steps	Sample
1. Create an object of VideoCapture from the PI-Cam	vd = cv2.VideoCapture(0)
2. Check to see it is open first	vd.isOpened()):
3. Read a frame from the video	ret, Frame = vd.read()
4. Should get the return status as well.	ret, Frame = vd.read( ) if ret == False: do something about it. Such as throw a message before sys.exit()
5. Should free it when you are done	vd.release()
You will need to loop 3,4 in order to continue to display the frames	
Also, put in code to allow user to quite. Eg. Hit 'q' to quit	if cv2.waitKey(1) & 0xFF == ord('q') is true

NOTE: Running the Cam device can be cpu resource intensive. Do not leave your cam program constantly running for hours without a heat sink. Even with a heat sink, those which come with the retail kit does not do much at all. A good solid one is necessary if you do need to run it constantly for hours.

Exercise 5 – Open a Video File

a) Using the VideoCapture Object to open the provided video file. Display the frame (returned from the read( ) method) in a window with imshow( ) and keep playing until the video ends OR you hit the letter 'q' .
b) Notice that the frame that come back from the read( ) method of the VideoCapture object is a numpy array image. You can perform the same operations on it that you did the static image from exercise 1. Try putting in some lines/circles/rects/text like you did in earlier exercise, and display that in a separate window.

Exercise 6 – Capturing Video from the Pi-Cam

a) Create a Cam object and set the resolution to 640x480 with a frame rate of 15. Loop thru it to show each frame. Change the vflip and hflip to get a proper image and not a mirrored one. Note that this is how we do it on the Raspberry Pi.
b) Notice that the image (image = vd.read()) object is the same numpy image that one can get when they read a jpg/png file. So you can manipulate the image as if it was a jpg/png. Try putting in some lines/circles/rects/text like you did in exercise 1-d and display that on top of your live video
c) Capture portion of the image and duplicate them on all 4 corners.
d) Take a slice from the video and display it somewhere else in the video.
e) Split the image up into its Blue, Green and Red channels using: (blue, green, red) = cv2.split(image). Display each channel with imshow( ). Notice the channels are grayscale and not colored. You are seeing the intensity value for each color. Take a Red object aim the camera at it. How does it appear in the Red channel window? And how does it display in the Blue and Green channel?

Helpful expression samples

```
cap = cv.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH,  320)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT,240)
cap.set(cv2.CAP_PROP_FPS,4)      # frame rate 4 frames per second
```

Exercise 7 – Converting from Color

a) By default we grab BGR from the camera but often times we need to convert this to grayscale. In a color image each entry in the Numpy array is actually 3 intensity values (blue,green,red). Each intensity value ranges from 0 to 255 so a “pure” green pixel would have the values (0,255,0). A grayscale image is a numpy array that is made up if only one intensity value between 0 and 255. Using cvtColor( ) convert cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) and display it.

Helpful Reference for Ex.5 to 7:

Reading and Writing Images and Video Files - VideoCapture Object

1. VideoCapture(0)
2. read( )
3. isOpened( )
4. release( )
5. Resolution
6. Framerate
7. vflip/hflip
8. capture_continuous( )
9. shutter_speed
10. exposure_mode
11. iso

Smoothing Images

1. blur( )
2. medianBlur( )
3. bilateralBlur( )
4. GaussianBlur( )

More on Common Array Operations Functions

5. split( )
6. merge( )

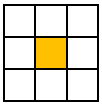
Misc Image Transformation:

1. cvtColor( )
2. threshold( )
3. adaptiveThreshold( )

IMAGE FILTERS (FOR IMAGE SMOOTHING)

Smoothing images with Noise Reduction (blurring) - One of the problems with performing image processing is noise. Noise can 'distract' our image processing to look at things we really don't want to. Noise reduction is normally achieved through slight blurring of our images. There are different types of blurs in Open CV each with their own benefits.

Mathematically speaking, convolution operation is done on an image with a kernel.



About Kernel: a fixed size array of pixels used for calculating image smoothing and filtering effect. They are used as numerical coefficients anchoring along the center point.

Here list four options:

Technique Option 1: Standard "averaging" blurring: This is also called homogeneous smoothing. Average blurring (as the name suggests), takes the average of all pixels in the surrounding area and replaces the center element of the output image with the average. Thus, in order to have a central element, the area surrounding the center must be odd. Here are a few examples with varying kernel sizes. Notice how the larger the kernel gets, the more blurred the image becomes

```
blurred = cv2.blur(image, (3, 3))                    # kernel size 3x3
```

Technique Option 2: - Gaussian blurring: uses a weighted kernel. GaussianBlur() is similar to blur() except it uses a weighted kernel. Pixels closer to the center have a higher weight associated with them whereas regular blur() uses equal weight. It is good for getting rid of camera noise. Like blur() it does tend to smooth out the edges but not as much as blur() does. It also tends to run slower than blur().

```
blurred = cv2.GaussianBlur(image, (3, 3), 0)   # last argument == standard deviation from x, y. 0 == use kernel (3,3) instead.
```

Technique Option 3: - Median Blur : this function is mainly used for removing what is called "salt-and-pepper" noise. Unlike the Average method mentioned above, the median method (as the name suggests), calculates the median pixel value amongst the surrounding area.

```
blurred = cv2.medianBlur(image, (3, 3)
```

Technique Option 4: Bilateral filtering : You may have noticed that blurring can help remove noise, but also makes edge less sharp. In order to keep edges sharp, we can use bilateral filtering. We need to specify the diameter of the neighborhood (as in examples above), along with sigma values for color and coordinate space. The larger these sigma values, the more pixels will be considered within the neighborhood.

```
blurred = cv2.bilateralFilter (image, 5, 21, 21),   # note (image, 5, 21) will be sufficient as sigma space == sigma color by default
blurred = cv2.bilateralFilter (image, 7, >=150),      # image starts to look cartoony
blurred = cv2.bilateralFilter (image, 9, <=10),       # no visual effect to the edges
```

Exercise 8 –Noise Reduction

I) Do the following with various blurring effect:

a) Open the noise.png file, and display it in a window. Make a copy of the image (image.copy()) and call cv2.blur(img,(3,3)). Now increase the Kernel size to (5,5), (9,9) and (11,11). What do you notice about the change in Kernel size?

note: While blur() is very fast, it isn't the best blurring because it tends to blur the edges of objects (which we often don't want).

b) Note: The Kernel size should always be odd numbers. Why?

c) Now add in the code to do the various blurring methods and observe the differences closely. Perform all the various methods with various kernel's size. E.g.:

II) Put all blur, GaussianBlur, and bilateralFilter images side by side to help you to visualize the differences.

Sample:

```
blurred = np.hstack([
        cv2.blur(image, (5,5)),
        cv2.GaussianBlur(image, (5, 5), 0),
        cv2.bilateralFilter(image, 5, 21, 21) ]   )
```

Or using np.vstack (…)

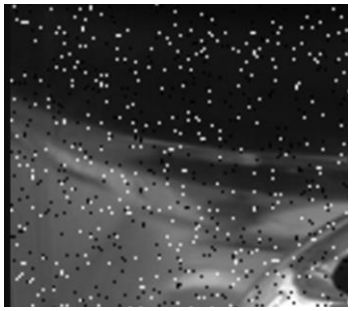
Exercise 9 – More on image smoothing

a. Now add to the code from 7a and call to GaussianBlur(img,(3,3)). Now change the Kernel sizes to (5,5), (9,9) and (11,11). GaussianBluring uses kernel just like blur() does but instead of using a kernel of all 1's

   GaussianBlur() is similar to blur() except it uses a weighted kernel. That is the values closer to the center have a higher weight associated with them whereas with regular blur() they are all equal.

   GaussianBlur is good for getting rid of camera noise. Like blur() it does tend to smooth out the edges but not as much as blur() does. It also tends to run slower than blur().

b. Add medianBlur(). It also takes a kernel size but not as a tuple but as a single k value (it creates a k x k kernel). It is good for reducing "salt and pepper" noise which are random white and black pixels.



c. Add a call to bilateralFilter() . BilateralFilter basically applies a Gaussian Filter to an image but then applies another form of Gaussian Filter but which is a function of intensity (actually the difference in intensity). This has the effect of blurring out noise while **preserving the edges** and is a good filter (though computationally expensive to use).

Keanu Reeves a bit overly bilateral filtered from 'A Scanner Darkly'

The bilateralFilter() takes the image as the first argument and the neighbor diameter size as the second. The neighbor size isn't the same as the kernel size as we are computing the actual diameter around a pixel instead of a k x k kernel. The larger the neighbor size the **slower** this algorithm runs. For live video you probably don't want to go above 5. For static images the diameter can be set to 9.

The third and fourth arguments are the sigmaColor and sigmaSpace. These arguments state how much influence pixels farther away can affect the blurring. If these values are too small (less than 10) then there will be little influence. Going too big (such as 150) can have not so great effects on your image.

Note: If you apply bilateralFilter on an image multiple times you can create a cartoonish effect. For example try the following

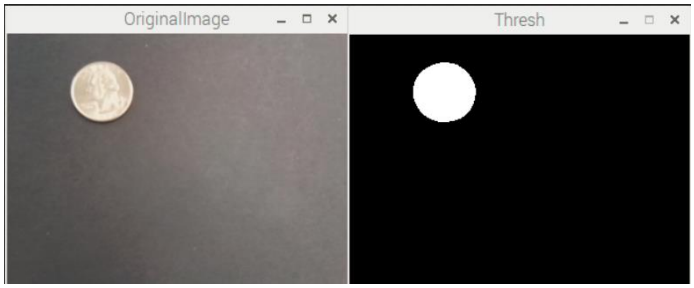
Now, you should capture an image of yourself. Then, you should apply bilateral filtering and generate a cartoony image of yourself.

```
for i in range(9):
    image = cv2.bilateralFilter(image,9,41,41)
cv2.imshow("bilateralCrazy",image)
```

DAY 3 - THRESHOLDING & FINDING CONTOUR (COIN RECOGNITION)

IMAGE TRESHOLDING

1. Threshold – Say you have a gray scale image (each pixel is a value from 0 – 255) and you want to convert it pure black and white (all pixels are either 0 or 255). You could take look at each pixel and any that are above or equal to 127 (255/2) would have a value of 255 (white) and any below 127 would have a value of 0 (black). This is a form of thresholding called a *binary threshold*. In this case our threshold value is 127. Below is an example:



One of the main reasons for using creating a threshold is to determine the region of interest (ROI) of an image. Often times we will combine a thresholded image with the OpenCV findContours() function which returns back a region in our image we are interested in.

Exercise 10 – Thresholding exercises

There are a number of different types of thresholding algorithms: (note: Canny edge detection and the inRange() function also create thresholded images but we will cover them in a later exercise)

a. Basic threshold – Open an image (or the video camera) and convert the image/frames to gray scale. Apply a GaussianBlur with a k=5. Then call the threshold() function (say 127), the max value (normally 255) and the type of threshold which would be cv2.THRESH_BINARY. Also create a threshold image passing cv2.THRESH_BINARY_INV. What do you see and why?

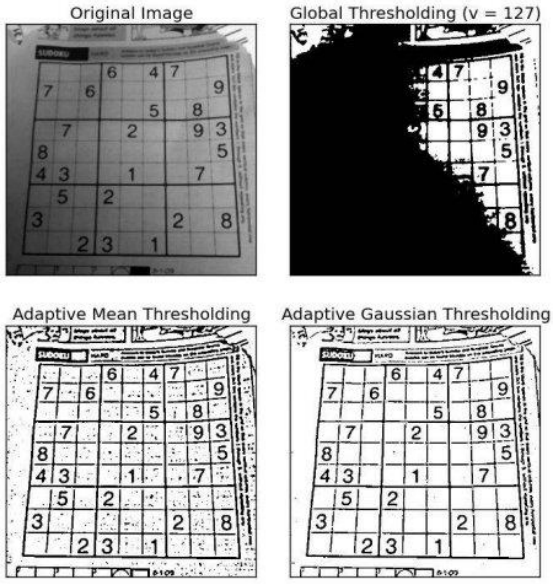
b. Adaptive Threshold – Basic threshold has problems with images that may have different illumination. Adaptive threshold looks at the surrounding region of a pixel and makes a smarter determination. Like the blurring functions this surrounding region size is based on a kernel size (so it must be both odd and a positive number)

There are two types of adaptive thresholding:

i. Adaptive Mean – the threshold value is the mean of k x k area. None of the values are weighted. This is faster but can still produce 'noise'.

In the same code from (a) call adaptiveThreshold() on the image. Try a k size of 11 and a C=2 (C is just a constant that gets subtracted from the mean value). You can try different values of K (and C as well) to see the effect.

ii. Gaussian – the threshold value is the *weighted* mean of k x k area smoothing effect . Call adaptiveThreshold() passing ADAPTIVE_THRESH_GAUSSIAN_C with a k size of 11 and a C value of 2 (C is just a constant that is subtracted from the weighted value). You can try different values of K (and C as well) to see the effect.



c. (optional) Otsu's Binarization Threshold – Otsu is a thresholding algorithm similar to the basic threshold (THRESH_BINARY) above but it determines the best threshold value for you. It works best if the image you pass it has two peaks in its histogram. To call Otsu just call threshold() with the following arguments: cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

Try running Otsu on the image

Functions

```
cv.imshow("Original", img)
blurImg = cv.GaussianBlur(img, (5, 5), 4)
cv.imshow("blurred ", blurImg)

blurImg = cv.cvtColor(blurImg, cv.COLOR_BGR2GRAY)      # set to grayscale Color Space
blurImg = cv.adaptiveThreshold( blurImg, 255,
      cv.ADAPTIVE_THRESH_GAUSSIAN_C, cv.THRESH_BINARY, 11, 2)
cv.imshow("New window adapt", blurImg)
return blurImg
```

CONTOURS

Contours are contiguous areas that have the same pixel values. They help us greatly in finding out regions of interest. We normally want to operate on binary images when finding contours which is why we needed to learn thresholding.

Exercise 11 – Finding Contours Exercises

1. Create a BGR image with a black background and add three yellow circles:
   color = (0,255,255)

```
image = np.zeros((300,300,3), dtype="uint8")
cv2.circle(image,(150,150),130,color,15)
cv2.circle(image,(150,150),90,color,15)
cv2.circle(image,(150,150),40,color,-1)

call imshow() to display the results
```

2. Now call cvtColor() to create a grayscale image.
3. Now make a binary image using threshold() on the gray scale image. Use a threshold of 127 and a max value of 255. Use imshow to display the results.
4. Call findContours() on the **copy** of the thresholded image (thresh.copy()) using the Contour retrieval mode argument cv2.RETR_EXTERNAL and the mode argument as cv2.CHAIN_APPROX_SIMPLE. Make a copy of the original image (call image.copy()) and pass the copy of the image to drawContours(). Display the results of drawContours(). Your code should look like the following:

```
(ret, contours, h) = cv2.findContours(
        thresh.copy(),
        cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
```

find the contours from the "thresholded" (grayscaled) image.

H : hierarchy (a form of linked list structure to represent how each of the contours set of data being linked together.)

CHAIN_APPROX_SIMPLE : It removes all redundant points and compresses the contour, thereby saving memory.

```
print len(contours)
imagecopy = image.copy()

contourcolor = (255,0,255)

cv2.drawContours(imagecopy, contours,
            -1, contourcolor, 3)

cv2.imshow("RETR_EXTERNAL",imagecopy)
```

Show the # of contours found
Make a deep copy == just copy the value, not reference.
Assign the color of the contour

Draw contours onto a copy of the original image
-1 : all contours are drawn
3 : thickness of the line

5. Copy the code from above and paste it underneath. Change the Contour retrieval mode argument from cv2.RETR_EXTERNAL argument in findContours() to cv2.RETR_TREE. Re-run the code and notice the changes in the contours drawn.
   i. What changed in the image?
   ii. Notice that in the code sample we print out the number of contours using len(contours). The number should have changed.
   iii. Why did the change in Contour retrieval mode create a change in the number of contours?

6. Enumerate the contours and display the area of each one using `cv2.contourArea(contour)`

7. Yet another thresholding function is called inRange(). It will take a color image and a low end color range and a high end color range.
   i. Use the code for capturing video from earlier exercise.
   ii. Take two tuples defined as the lower and upper red values. E.g.

```
redLower = (0,0,90)
redUpper = (50,50,255)
```

or

```
redLower = np.array([0, 0, 40], dtype = "uint8")
redUpper = np.array([90, 60, 255], dtype = "uint8")
```

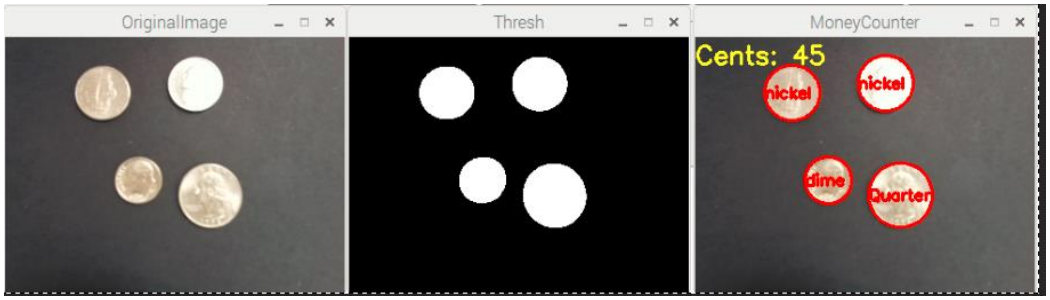
   iii. Pass the image from the camera to inRange() and pass redLower and redUpper. Display the returned image while holding up a red ball. What is displaying? If you see some noise you should probably perform a GaussianBlur() prior to calling inRange(). E.g.

```
threshImage = cv.inRange(colorImage, redLower, redUpper)  # return a thresholded colorImage
cv.imshow("InRange Image", threshImage)
```

   iv. Now pass the image from inRange to findContours. Which Retrieval mode flag makes the most sense to use? Will it matter?
   v. Create a minimum enclosing circle using the cv2.minEnclosingCircle() and then use the (x,y) and radius to display the circle in the original image.

8. Finding Coins
   a. Find silver coins (quarter, nickel, dime) on a black background (black paper) using threshold(). Start out with a threshold value of 127 and modify it to get a good value(call imshow() on the output). Don't forget to perform a blur first (either choose Gaussian() or BilateralFilter)
   b. Call findContour() on the output and call drawContours(). Make sure to use RETR_EXTERNAL as your retrieval mode.

9. Can you think of a way to determine the coins you are looking at? Do so and then display on the screen the coins (and their sums) like:



Helpful Sample and reference:

```
ret,contours, h = cv2.findContours(threshImage, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
                    # get your region of interests

sortedcontours = sorted(contours, key = cv2.contourArea, reverse = True)

print ("{} contours".format(len(contours)))

coins = image.copy()

cv2.drawContours(coins, contours, -1, (0, 0, 255), 3)          # draw red line all found contour with thickness of 3

for (i, contour) in enumerate(sortedcontours):          # go thru all regions of interest to do work (your application
    (x,y),radius = cv2.minEnclosingCircle(contour)
Etc.
```

Structural Analysis and Shape Descriptors

1. findContours( )
2. drawContours( )
3. minEnclosingCircle( )
4. contourArea( )
5. boundingRect( )

Online Samples about finding Contours

CANNY EDGE DETECTION

It was developed by John F. Canny in 1986. It is a multi-stage algorithm.

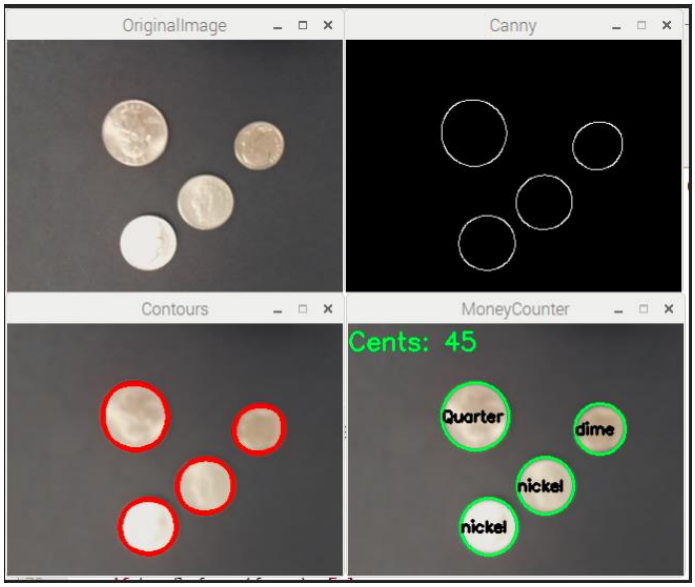
1) Noise Reduction
2) Finding Intensity Gradient of the Image
3) Remove unwanted pixel which may not constitute the edge. Each pixel is checked if it is within a specific maximum in its neighborhood in the direction of gradient.
4) Use two threshold value, min and max for Hysteresis Thresholding algorithm.
   - edges with intensity gradient > max == wanted edge
   - edges with intensity gradient < min == discarded
5) Based on connectivity, those who lie between these two thresholds are classified edges.

Reference

- Canny( )
- Object Tracking sample

Exercise 12 – Coin Recognition:

1. Another thresholding function is Canny Edge detection.
   Find silver coins (quarter, nickel, dime) on a black background (black paper) using Canny(). Start out with a minvalue of 40 and a maxValue of 120 but make it so you just detect the edges of the coin (call imshow() on the output)
   a. Call findContour() on the output from Canny() and call drawContours(). Make sure to use RETR_EXTERNAL as your retrieval mode.
   b. Try and figure out a way to determine the coins you are looking at and sum the values like so:



See sample code segment next page…

Sample:

```
image = cv2.GaussianBlur(image, (11,11), 0)          # decide your kernel size
edges = cv2.Canny(image,50,110)                      # with min and max value

ret,contours, h = cv2.findContours(edges.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
                    # get your region of interests

sortedcontours = sorted(contours, key = cv2.contourArea, reverse = True)

print ("{} contours".format(len(contours)))

coins = image.copy()

cv2.drawContours(coins, contours, -1, (0, 0, 255), 3)  # draw red line all found contour with thickness of 3

for (i, contour) in enumerate(sortedcontours):         # go thru all regions of interest to do work (your application
    (x,y),radius = cv2.minEnclosingCircle(contour)
Etc.
```

2. Track a Red Ball
    a. Using the inRange() function code from exercise 10 to track the red ball. Put a text said “red ball” wherever the ball is found.

More reading on Image Processing

http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html

DAY 4 - 5 BALL CHASING (WITH ADAFRUIT MOTOR HAT)

SUMMARY

		Open methods
1.	Read in an image	cv.imread( ) cv.imshow( )
2.	Preform some image smoothing (depends on application) Note : kernel size always positive odd numbers	4 methods: - cv2.Blur(image, kernel size) - cv2.GaussianBlur(image, kernel size, deviation in X, • deviation in Y direction) - cv2.medianBlur(img,kernel size) - cv2.bilateralFilter(img, distance pixel distance used for filter, deviation in the color space deviation in the coordinate space
4.	Perform Thresholding Note: if you read in the image in grayscale already, you should not do cvtColor	Two methods: Method 1: e.g. - cv.cvtColor(… .COLOR_BGR2GRAY) - cv.threshold(… cv.THRESH_BINARY) or cv.adaptiveThreshold, etc.. OR Method 2 : using inRange, e.g. - redLower = np.array([*b1, g1, r1*], dtype = "uint8") - redUpper = np.array([*b2, g2, r2*], dtype = "uint8") - cv.inRange(img, redLower, redUpper) - note that this method does grayscale for you.
5.	Find Contour	cv.findContours(…) - With either .RETR_TREE or RETR_EXTERNAL, or both ; depending on your applications. (note: it is always a good idea to view them too: cv.drawContours(…))
6.	Creating bounding circles, boxes around the contour.	- sortedcontours = sorted(…) - numerate thru each circle/box and get each center X, Y and radius from minEnclosingCircle or boundingRect
Now, you get your possible regions of interest. Then, you can proceed with your application, such as tracking a color object.		

DISCUSSION:

Which one of the Image Smoothing Techniques to use:

	Blur	Gaussian Blur	Median Blur	Bilateral Blur
speed	Fastest as it is the simplest form of averaging on the kernel pixels	Ok, depending on the standard deviation of x, y space given		Slowest
Noise removal	ok	Better	Better	Best
Linear / non-linear	Linear	Linear	Non-linear	Non-linear
Edge preserving	No	No	Yes (averaging across image edges)	yes
With low light				
Edge sharpening	No	No	No	yes

Misc :

- HSV vs RGB
- Letter recognition

FINAL CHALLENGES:

1. Soccer : the soccer ball will be in a specific color, such as an orange or red ball. Goal will be in either yellow or blue. You will use the Adafruit PI Motor hat for your robot.

2. Coin Calculation: A program to tell the total value of coins you have (just lay them out flat on a black surface.