Experiment No. 03:

Basics of Image Processing with Jetson Nano

By: John Ford

Instructor: Dr. Jafar Saniie

TA: Xinrui Yu

ECE 501

Lab Date: 07-20-2020

Due Date: 07-24-2020

Acknowledgement: I acknowledge all of the work (including figures and codes) belongs to me and/or persons who are referenced.

John Ford

# I. Introduction

## A. Purpose

In this lab, students are required to execute a number of Python image processing programs on Jetson Nano. The image processing techniques involved are the same ones in Experiment #1. The purpose of this lab is to introduce the following items:

* Python compatible image processing library - OpenCV
* Jetson Nano specific image processing toolkit - CUDA
* Realization of image processing techniques in Experiment #1 on Jetson Nano

After finishing this lab, students should have a better and refined understanding of the basic concepts of image processing, perform programming in Python using the functions provided by its image processing toolbox, and code one's own image processing program then execute it on Jetson Nano.

## B. Background

### Description of Python Compatible Image Processing Library - OpenCV

According to its official website <https://opencv.org/about/> [1], OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. Being a BSDlicensed (a family of permissive free software licenses)product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. In terms of Jetson Nano, it supports OpenCV with GPU acceleration. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

### Description of NVIDIA CUDA Toolkit for Jetson Nano (CUDA 10.0)

According to its official website <https://developer.nvidia.com/cuda-toolkit> [2], The NVIDIA® CUDA® Toolkit is used to develop, optimize and deploy GPU-accelerated apps on products with a NVIDIA GPU, including Jetson Nano. It provides a development environment for creating high performance GPU-accelerated applications. With the CUDA Toolkit, you can develop, optimize and deploy your applications on GPU-accelerated embedded systems, desktop workstations, enterprise data centers, cloud-based platforms and HPC supercomputers. The toolkit includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler and a runtime library to deploy your application.

GPU-accelerated CUDA libraries enable drop-in acceleration across multiple domains such as linear algebra, image and video processing, deep learning and graph analytics. For developing custom algorithms, you can use available integrations with commonly used languages and numerical packages as well as well-published development APIs. Your CUDA applications can be deployed across all NVIDIA GPU families available on premise and on GPU instances in the cloud. Using built-in capabilities for distributing computations across multi-GPU configurations, scientists and researchers can develop applications that scale from single GPU workstations to cloud installations with thousands of GPUs.

## C. Preliminary Assignment

1. Review course materials on OpenCV and familiarize yourself with its operations.
2. Review course materials on the basic concepts of GPU acceleration using CUDA on Jetson Nano, and make sure that you understand why this is beneficial for image processing.
3. Check one of the provided scripts and describe what the script does in general.

The script provided in Appendix A checks an image for shapes, traces the detected edge of the shape with a line by thresholding, and classifies each shape as a triangle, rectangle, pentagon, circle, or ellipse.

1. Prepare several images for image classification, so as to generate more results and have a better understanding when checking them.

# II. Lab Procedure and Equipment List

## A. Equipment

*Equipment*

* 1 x Jetson Nano single-board computer
* Internet connectivity
* I/O devices (USB keyboard/mouse, etc.)
* External Monitor with HDMI/DP port

## B. Procedure

### Run Image Segmentation Script and Edge Detection Script

In this part of the lab, you are required to execute the image segmentation program; you need to check its results with different operations and analyze the results. OpenCV is required for this program (preinstalled if you have the OS image from NVIDIA). Check [3] for more info.

The procedures are as follows:

1. Run the image segmentation source code with only the thresholding technique applied. [4]

The code is as follows:

import numpy as np

import cv2

from matplotlib import pyplot as plt

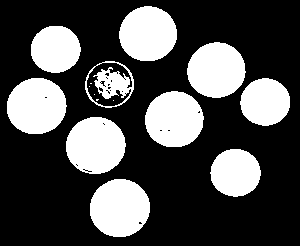
img = cv2.imread('coins.png')

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)

cv2.imwrite('out.png',thresh)

The program uses the demo image coin.png for image segmentation (same as in Experiment #1). The demo image and the thresholding result (right) are shown below.



After executing the program, you will see segmentation result in the out.png file. Check the result and analyze the effect of the image segmentation algorithm. Also, note the difference between Python result and MATLAB result.

2. Check results from the same script, this time using your own image. Put the image you would like to perform image segmentation in the same folder as the program and change the imread variable to be the filename of your own image. Execute the modified program again and check the results. Note: the portion with less intensity in the image is considered to be the background.

3. Create and run the edge detection script

You will need to create your own edge detection program for this procedure.

The example script is given as follows:

import cv2

import numpy as np from matplotlib

import pyplot as plt

img = cv2.imread('apple.jpg',0)

edges = cv2.Canny(img,100,200)

plt.subplot(121), plt.imshow(img,cmap = 'gray')

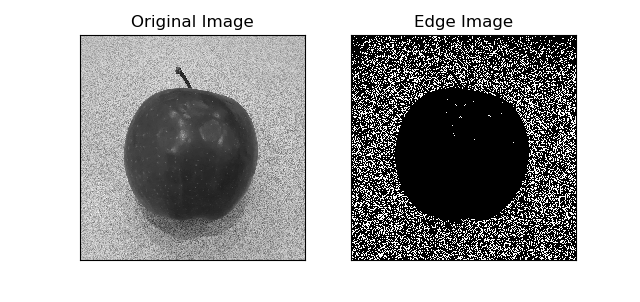
plt.title('Original Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122), plt.imshow(edges,cmap = 'gray')

plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()

Note: You can change the edge detection algorithm for different results. Run your program and check the results, analyze the difference from MATLAB one.



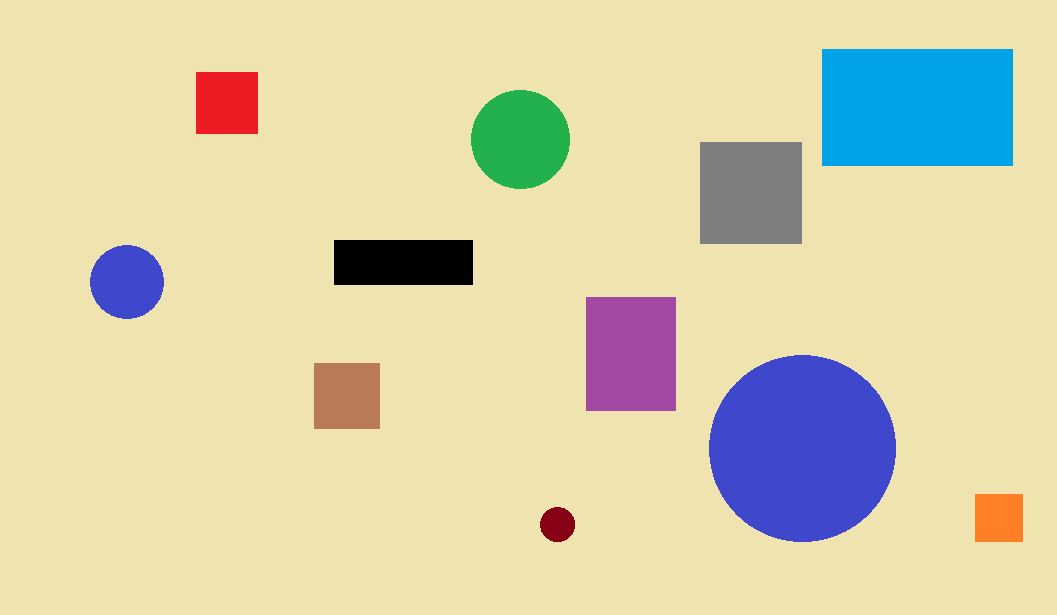
### Run Shape Extraction Script to Perform Object Recognition (OpenCV)

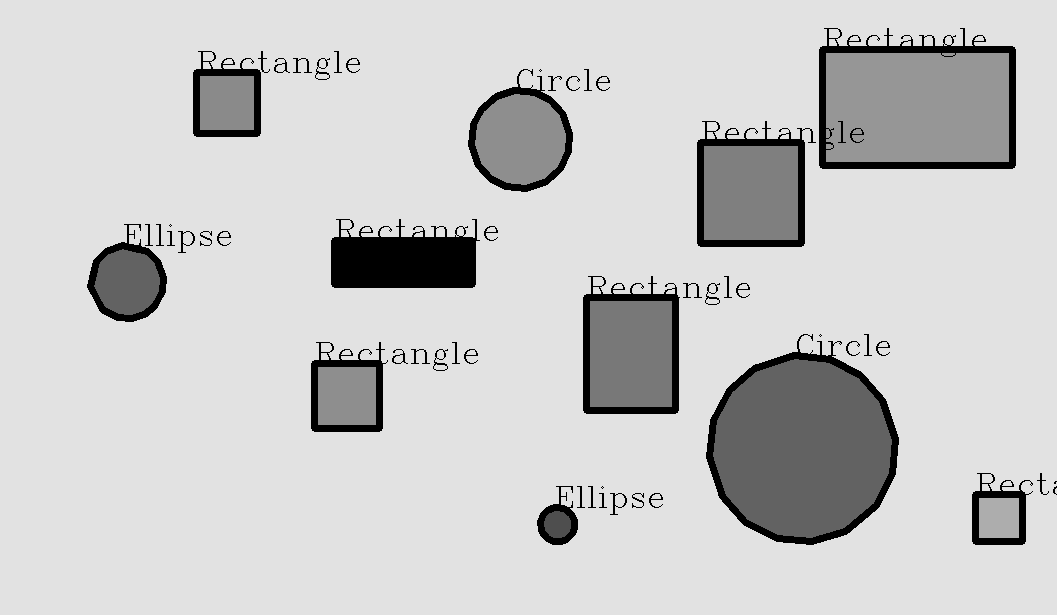
In this part of the lab, you are required to execute the shape recognition program; you need to check its result and analyze.

The procedures are as follows:

1. Run the shape extraction program with the test image

Execute the shape extraction program with the given test image and check its results. The shape extraction program is given in Appendix A. Note: the script is able to recognize rectangular, ellipses and circles.





2. Add noise to the test image

Add noise to the image by executing the following script:

import numpy as np

import cv2 from matplotlib

import pyplot as plt

img = cv2.imread('shapes.png')

row,col,ch = img.shape

mean = 0

var = 255

sigma = var\*0.0707

gauss = np.random.normal(mean,sigma,(row,col,ch))

gauss = gauss.reshape(row,col,ch)

noisy = img + gauss cv2.imwrite('out.png',noisy)

This program adds a gaussian noise of mean 0 and variance of 0.005. Note the difference in numbers.

3. Modify the shape extraction program to run with the image with added noise. Check the results.

4. Redo step 2 and 3, but with a variance of 0.01.

### Image Morphological Operation and Histogram Expansion with Python (OpenCV)

OpenCV have inbuilt functions for all four image morphological operations and histogram expansion. The procedures are as follows:

1. Try opening, closing, erosion, and dilation with Python (OpenCV)

The functions for the aforementioned four operations are respectively:

opening = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel)

closing = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)

erosion = cv2.erode(img, kernel, iterations = 1)

dilation = cv2.dilate(img, kernel, iterations = 1)

Where *img* is the original image matrix, and kernel is a structural element/matrix used to define direction and area of the function, generated by the following command:

kernel = np.ones((5,5), np.uint8)

Check the corresponding help document [6] for more information.

Use your own image to perform all four operations and compare the results with the example effect image in the background section.

2. Histogram expansion with Python (OpenCV)

The function for histogram expansion in OpenCV is:

equ = cv2.equalizeHist(img)

Where *img* is the original image. Automatic determination of histogram range is used in OpenCV.

Use your own image to perform histogram expansion and compare the result with the example effect image in the background section. Note: if your original image is balanced in intensity, the result may be indistinguishable from the original. The following is a comparison of before and after histogram expansion.



### Perform Video Processing with Python (OpenCV)

In this part of the lab, you are required to create and execute a video processing program in Python and generate a number of frames from the example video. Then, you need to use the script from part B to extract shapes from these frames and use the video creation function in video processing script to label the test video with shape markers. Refer to [7] for more information.

The procedures are as follows:

1. Create and run the video processing program with the test video

You need to create the video processing program to extract frames from the test video. Then, run the video processing program to make it generate the separate frames from the video. Check these frames (they should be stored as png images) to make sure you have the correct frames.

2. Create a program to use the shape extraction program in part B to process these frames iteratively

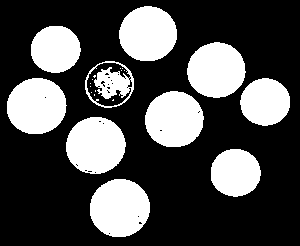
You will need to label each frame with annotations of shape and store these annotated frames.

3. Use generate video from frame function to create a video processing program to generate a video with labeled shapes.

You will need to re-generate the video from the annotated frames..

# III. Results and Analysis

**\*** See attached text files for code

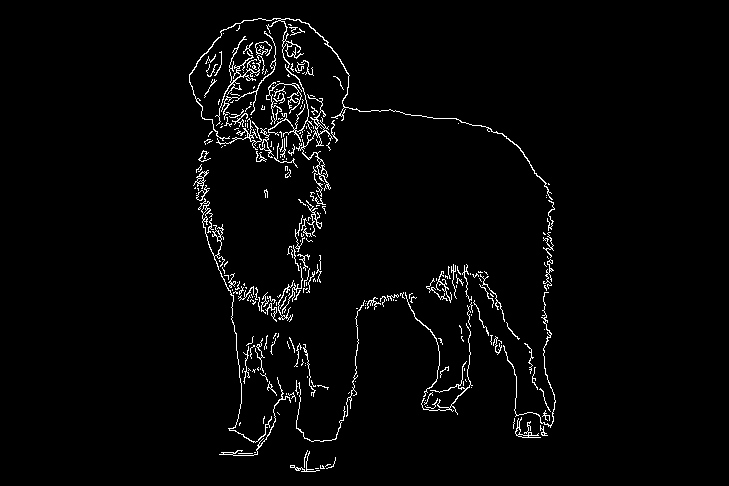




**Figure 1: Thresholding performed on images coins.png (top left), cat.jpg (top right), and rabbit.jpg (bottom)**

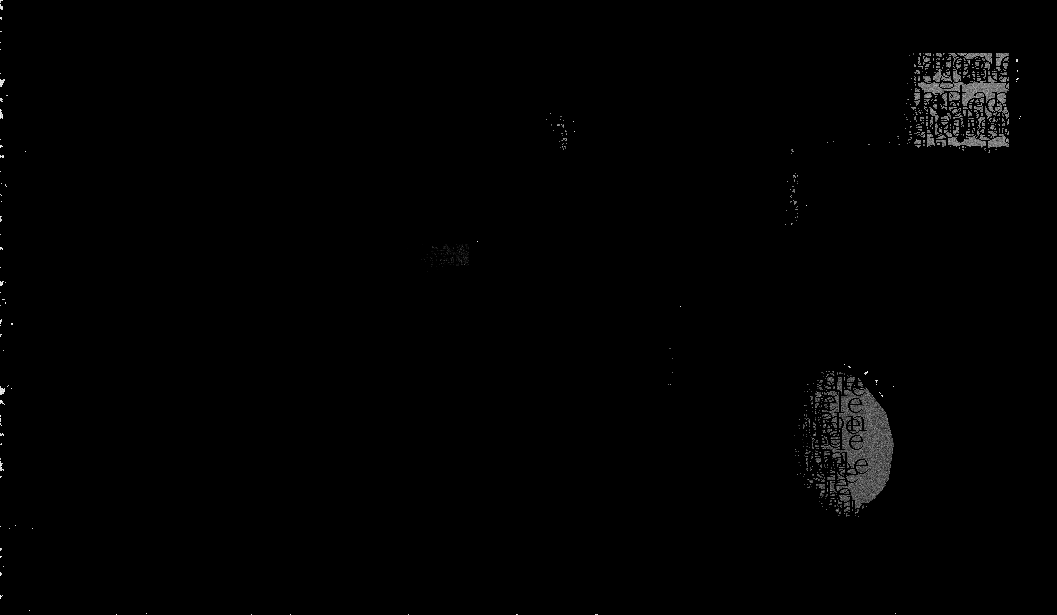
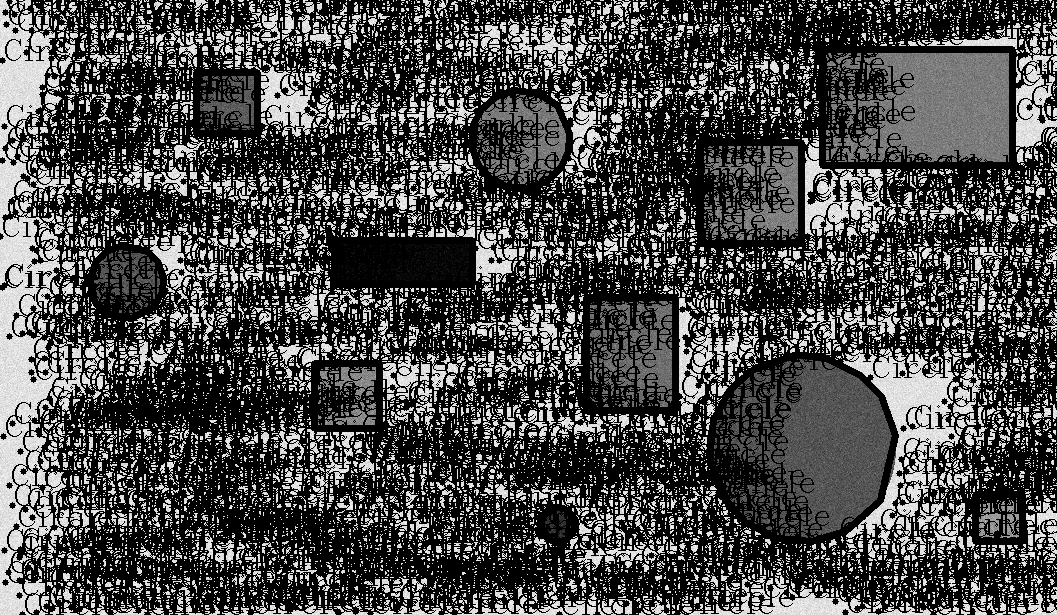
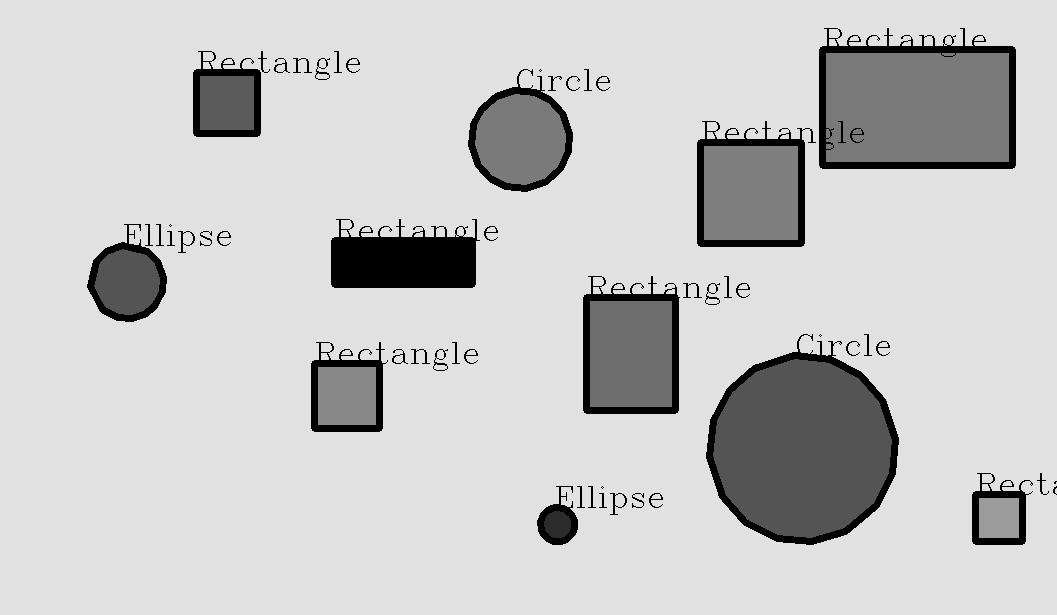
\* See attached file segment.py





**Figure 2: Canny edge detection performed on coins.png (top left), cat.jpg (top right), and dog.jpg (bottom)**

\* See attached file edge\_detect.py



**Figure 3: Annotate shapes with various levels of noise**

**none (top), variance 0.005 (middle), variance 0.01 (bottom)**

\* See attached files noise.py and shapes.py



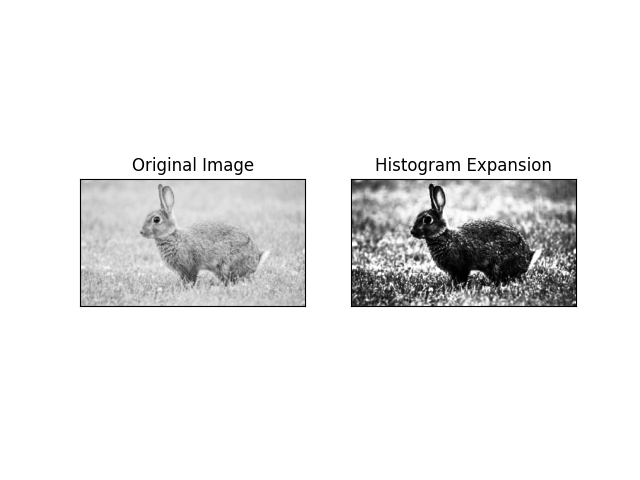




**Figure 4: Morphological operations performed on rabbit.jpg**

**original (top), opening (middle left), closing (middle right), erosion (bottom left), dilation (bottom right)**

\* See attached file morphology.py



**Figure 5: Histogram expansion performed on rabbit.jpg**

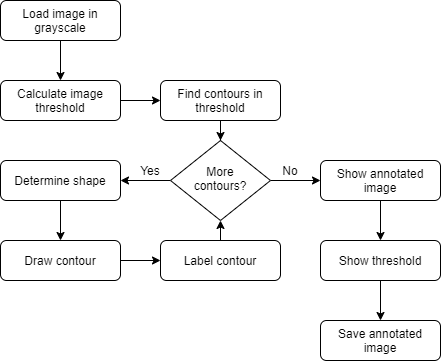
\* See attached file morphology.py

## A. Discussion

1. Comment on the advantages and disadvantages of programming with MATLAB and Python, specifically on the image segmentation script/program. (10 pts)

|  | **Advantages** | **Disadvantages** |
| --- | --- | --- |
| **MatLab** | Targeted towards mathematics behind image processing and contains a wider variety of algorithms to choose from for various tasks such as edge detection | Code for image operations is generally less intuitive and requires more lines of code than in Python. |
| **Python** | Most image operations require fewer lines of code due to the presence of built-in libraries | Some functions, such as the noise generation function, may be less intuitive (uses standard deviation instead of variance), but this is personal choice |

1. Draw a flow chart and explain how the shape extraction program in Appendix A works. (10 pts)



First, the image is loaded and the threshold obtained. For each contour in the image, the program determines the shape it represents and draws the contour, labelling it with its calculated shape. After all the contours are properly added and annotated, the image and threshold are displayed and the annotated image is saved to a file.

1. Why are the numbers different between the MATLAB and Python noise generation script/program? (10 pts)

In each language, a different function is used to compute the amount of noise to generate. In the case of MatLab, the parameter passed into the noise matrix generation function is the variance, whereas in the case of Python, it is the standard deviation (square of the variance). The numbers provided in the Python script have also been modified so as to generate a sufficient amount of noise comparable to that generated by the MatLab script.

1. Why is GPU acceleration useful/beneficial to image/video processing in general? (10 pts)

Because the GPU can perform a greater number of image/video processing calculations simultaneously, it significantly reduces the time required to perform the processing as compared to the CPU.

1. In the video processing part, explain why we need to generate separate frames to perform video processing. Assume the video is encoded. (10 pts)

Provided that the video is encoded in a compressed format, information on the deviation between frames is stored in the video file rather than the frames themselves. However, to process the frames by the image analysis program, they must be in the form of complete image files. To accomplish this, the video is processed into discrete images prior to analysis and then converted back into compressed video format after each frame is adequately processed.

# IV. Conclusions

The purpose of this experiment was accomplished. By analysing the given Python code snippets and using them and modifying them to perform operations on images and videos, understanding of the basic image manipulations was imparted on the student. All image manipulations desired were successfully achieved. These image manipulations include: image segmentation, object detection, morphological operations, histogram expansion, edge detection, and thresholding. Comparisons were then made to similar operations using MatLab to determine the advantages and disadvantages posed by the Python frameworks.

## References

[1] Experiment 3 Lab Manual

[2] Python documentation

## Attachments

1. *segment.py*
2. *edge\_detect.py*
3. *noise.py*
4. *shapes.py*
5. *morphology.py*
6. *histogram.py*
7. *process\_vid.py*
8. *graphics.avi* ⇒ original video obtained from YouTube
9. *graphics\_processed.avi* ⇒ video processed with process\_vid.py script
10. *graphics\_processed\_slower.avi* ⇒ processed video saved at a slower framerate