University of Colorado Boulder

# ECEN 5763 EMVIA Exercise 2

Dhiraj Bennadi

June 22nd, 2022

Guided By: Professor Dr Sam Siewert

# Code Repository

**Github Link**

The repository contains the source files, makefiles, output images and the report

# 1 Problem 1

## 1.1 FFPMPEG

FFmpeg is a free and open-source software project consisting of a suite of libraries and programs for handling video, audio, and other multimedia files and streams

## 1.2 Command to find the information on the video

Using the following command, the information of the encoding of video was extracted
Command : **ffmpeg -i *.avi**


Figure 1: Encoded Details of the video

The frame rate was found out to be 24.The extraction of frames from the video was done at the same frame rate and the 100th image is as follows

Command : **ffmpeg -i big_buck_bunny_480p_surround-fix.avi -vf fps=24 %000d.jpeg**

The extracted frames were in .jpeg format, however any format can be specifed as the output files. The %000d specifies the frames to be named in the format (Similar to a for loop in C++)

Figure 2: 100th frame

The above method is a manual method where the whole video is encoded, to select an individual frame the following command can also be used.
Command : **ffmpeg -i big_buck_bunny_480p_surround-fix.avi -vf select='between(n 100 100)' -vsync 0 %000d.jpeg**

# 2 Problem 2

## 2.1 Input

Figure 3: Sobel Transformation Input Image

## 2.2 Output

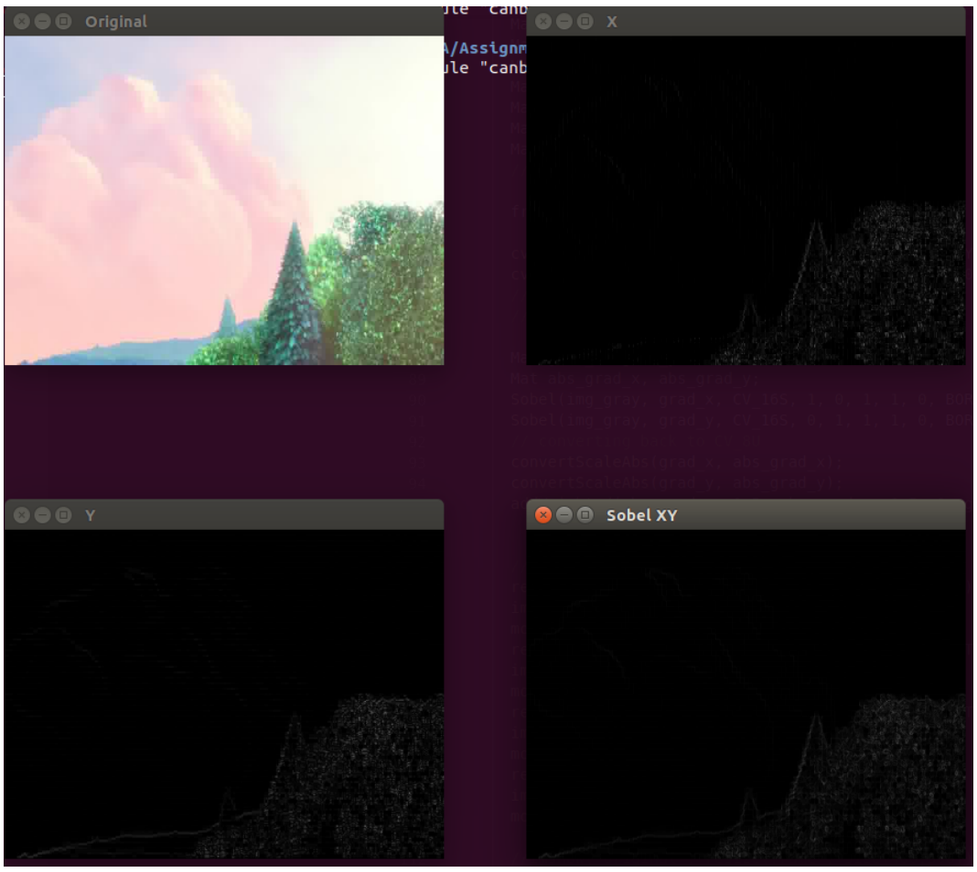


Figure 4: Sobel Transformation Output Image

## 2.3 Explanation of Sobel Operator

Sobel Operation is a type of kernel convolution. Sobel Edge Detection is a case of trying to find the regions in an image where the intensity changes in a significant manner.

1. The Operation has 2 different kernels of the X and Y direction.
2. The X kernel is represented as follows: $\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$
   When this kernel is applied to the image, this kernel creates a difference between the pixel intensity on the X-axis
3. The Y kernel is represented as follows: $\begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}$
   When this kernel is applied to the image, this kernel creates a difference between the pixel intensity on the Y-axis
4. At each pixel, the gradient of the x and y is combined and an output image with edge detection highlighted in the output image
5. The Sobel operator is inherently very noise and might not produce great results with color images as the edge detection is performed on the 3 color pixels. To counter this a Gaussian Blur filter and conversion of image into grayscale is performed to get better output.

6. The values of pixel of the output image is kept in the range of 0 to 255 to avoid any negative values during the transformation using an absolute function

The output is more visibly defined in the below 2 images (Figure [5] and and Figure[6] ) as the input image is a Zebra cross which has significant difference in most regions of the pixels with respect to the intensity. Now when the X-gradient transformation is applied on the input image, if the image is held horizontally, the edge detection can be seen in the X-window and comparatively, the Y-gradient does not perform well. If the input image is held vertically, then the Y-gradient performs better edge detection and X doesn't. The X gradient and Y gradient are combined with mathematical equations using Pythagoras theorem and orientation using arc tan property gives a better result.

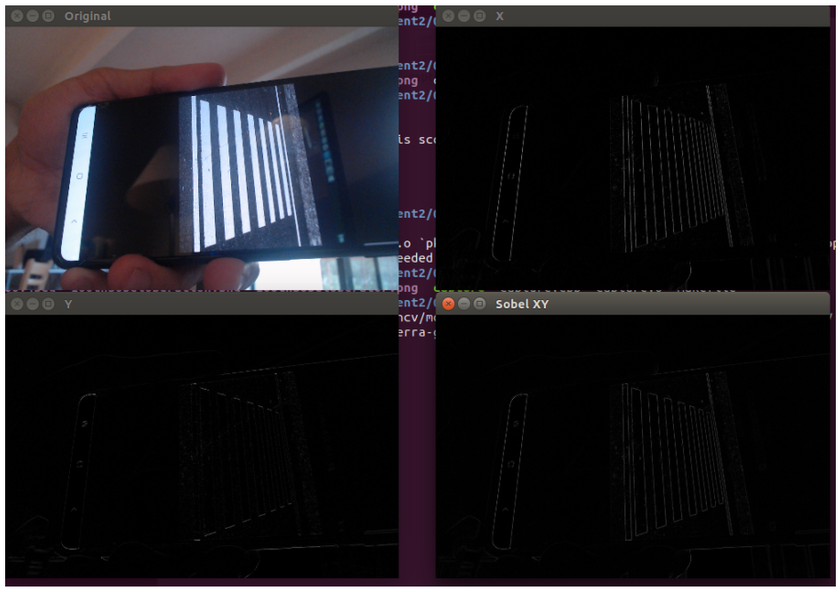


Figure 5: Zebra Crossing X Axis

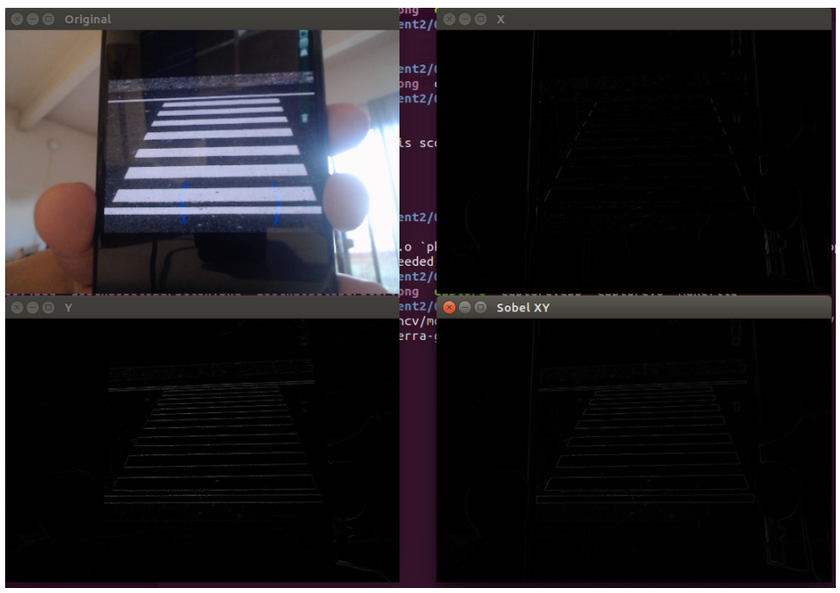


Figure 6: Zebra Crossing Y Axis

# 3 Problem 3

## 3.1 Input

Figure 7: Canny Transformation Input Image

## 3.2 Output

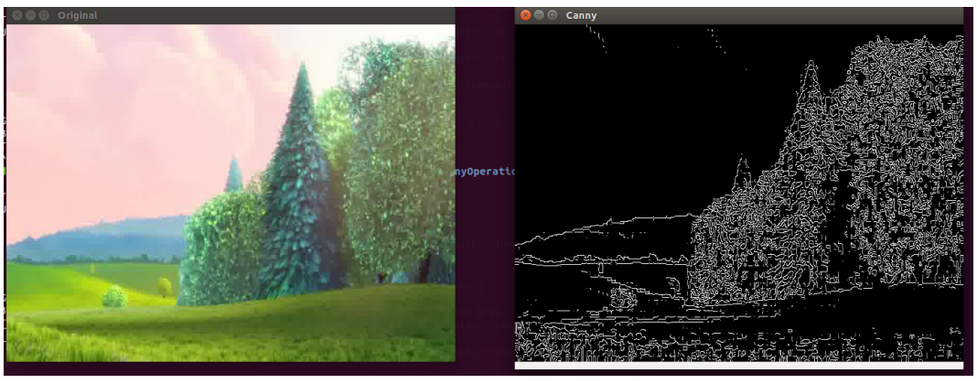


Figure 8: Canny Transformation Output Image

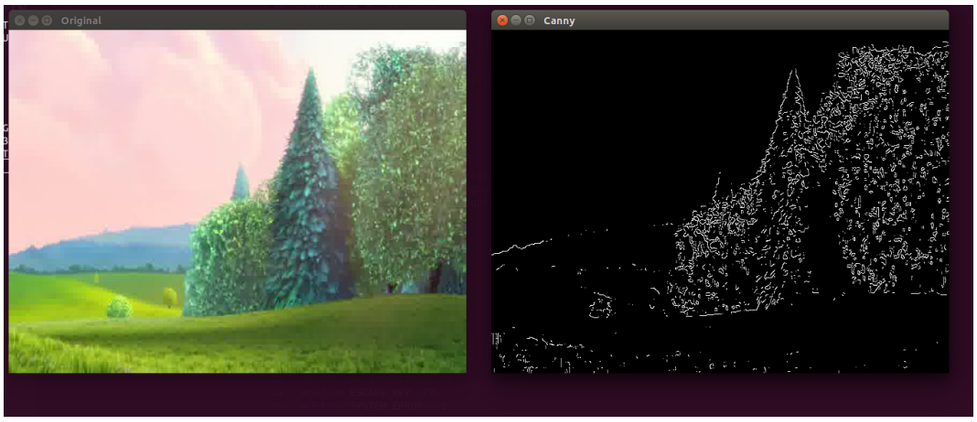


Figure 9: Canny Transformation Output Image Threshold value 100

### 3.3 Explanation of Canny Edge Detection

The Canny Edge detector is a multistage algorithm to detect wide range of edges in images. The algorithms works on the principle of identifying the regions that change in the intensity of pixel values.

1. An Gaussian filter is first applied to the image to remove noise.
2. The Intensity gradients in the image are identified using kernel transformation
3. Another transformation is applied to thin out edges namely Non-Maximum Suppression Thresholding. This transformation removes the regions that are not identified as true edges
4. A Double Thresholding is applied determine proper edges
5. The edges are tracked by Hysteresis and regions that are no longer above a certain level are ignored and the regions that are potential edges are accepted only if there are close to the threshold levels
6. For the Double Thresholding, the API takes 2 values, Threshold1 and Threshold2 based upon

## 4 Problem 4

### 4.1 Output of Code


Figure 10: Output of OpenCV Code


Figure 11: Output of V4L2 Camera Capture Code

## 4.2 Syslog Output for 640*480 Resolution

```
Jun 21 04:38:17 dhiraj-emvia capture: Difference of Capture sec = 64 , msec = 626
Jun 21 04:38:17 dhiraj-emvia capture: Average time per frame in ms = 35
```

Figure 12: Syslog of OpenCV Code

```
Jun 21 20:28:54 dhiraj-emvia capture: Difference of Capture sec = 245 , msec = 890
Jun 21 20:28:54 dhiraj-emvia capture: Average time per frame in ms = 136
```

Figure 13: Syslog of V4L2 Camera Capture Code

## 4.3 Timing Analysis for 640*480 Resolution

| **Parameter** | **OpenCV** | **V4L2** |
|---|---|---|
| Total Time (ms) | 64626 | 245890 |
| Average Time (ms) | 35 | 136 |

## 4.4 Comparision of the OpenCV and V4L2 code

1. The average time for capture of image is pretty large value in V4L2 compared to the OpenCV code. OpenCV code is optimized to handle camera operations and the V4L2 code considers the camera as a general IO operation and hence OpenCV performs better in average frame rate

2. In terms of memory, OpenCV gives options to choose the compression ratio while storing the image with less degradation of the image quality. In the current example, for a 640*480 resolution images taken with both codes, the size of the image on OpenCV is approximately 20KB while that of V4L2 is 920 KB. When working with systems with smaller memory footprint, OpenCV performs better.

3. OpenCV reduces development time since the APIs have good documentation and ease of use. In commercial projects where resources are not restricted by speed and memory OpenCV can make a great case.

4. The file format exchange is pretty easy in OpenCV compared to V4L2

# 5 Problem 5

## 5.1 Canny

### 5.1.1 Output

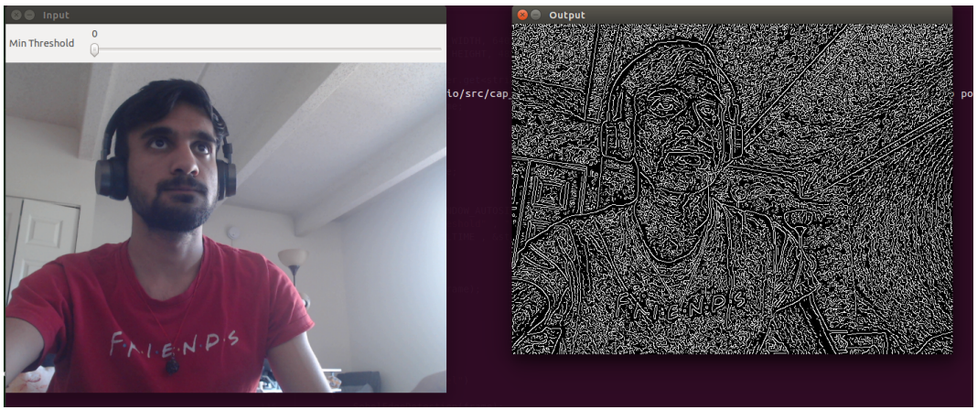


Figure 14: Canny Output 0 Threshold

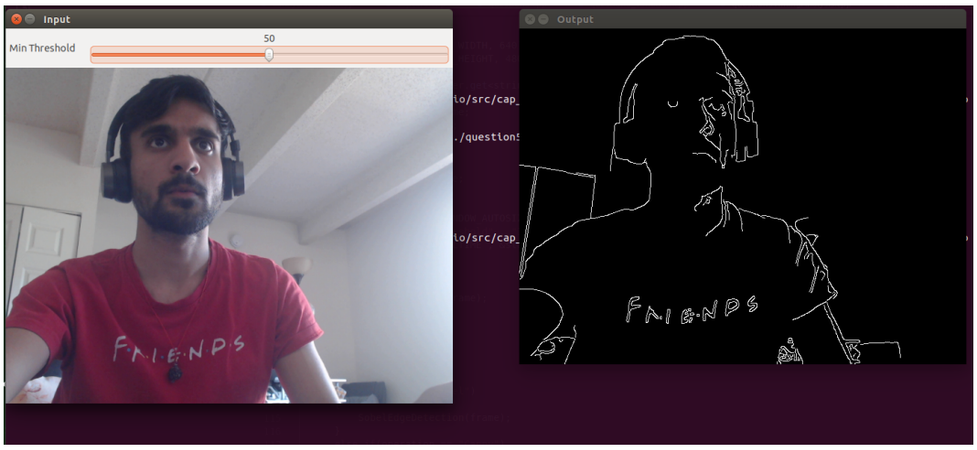


Figure 15: Canny Output 50 Threshold

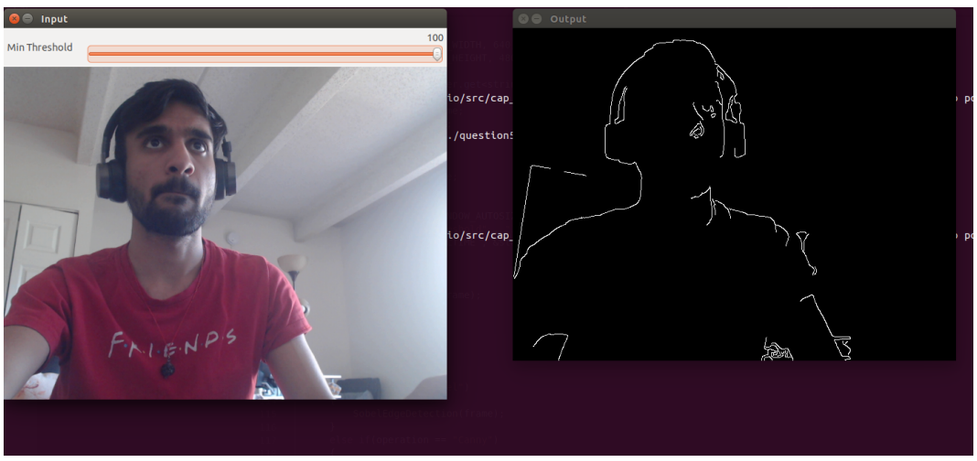


Figure 16: Canny Output 100 Threshold

## 5.2 Sobel

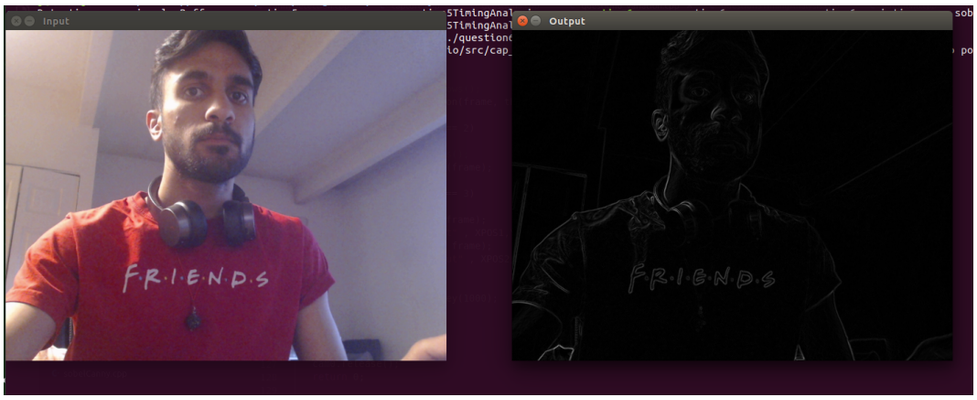


Figure 17: Sobel Operator Output

## 5.3 Average Frame Rate

For a resolution of 640*480, the average frame rate obtained is as follows:

| Parameter | Canny | Sobel |
|---|---|---|
| Average Time (ms) | 33 | 33 |

# 6 Syslog Outputs

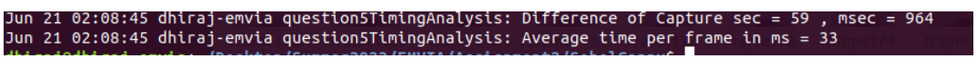


Figure 18: Canny Syslog

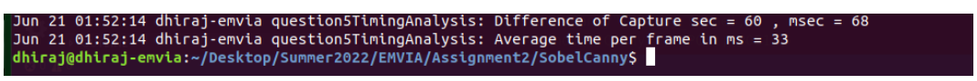


Figure 19: Sobel Syslog

## References


[1] OpenCV Documentation

[2] Paper on Cloud Analytics

[3] Video on explanation of HAAR Classifiers

[4] Repository provided by Dr. Sam Siewert