# Project 4: OpenCV Basic Use

Daniel Sawyer U3363-7705
CAP4401 Image Processing Spring 2017
University of South Florida, Tampa

***Abstract*— Basic edge detection and histogram equalization.**

## I. Introduction and overall description

This assignment focuses on edge detection and histogram equalization using the OpenCV library with C++. OpenCV is a robust image processing library with many features. This project focuses on using Sobel and Canny 3x3 for edge detection as well as Histogram Equalization to improve contrast. In Section 2, the basic algorithms used in the assignment are described. Section 3 describes the implementation details of the assignment. In Section 4 the results for the assignment is presented and finally we conclude in Section 5.

## II. Description of algorithms

In this section the algorithms used in this assignment are described. The Sobel is first, followed by Canny, and finally Histogram Equalization.

### A. Sobel Edge Detector

Calculating the gradient and direction requires you to first calculate dx and dy. This is done by calling the Sobel() function twice, once for gradient x (dx) and once for gradient y (dy). Then you take the $\sqrt{dx^2 + dy^2}$ of every pixel and save that as your new gradient. Then threshold it based on users T value, which leaves you with just the edges. Here is the Sobel 3x3 kernel used for dx and transposed for dy:

$$dx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$dy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

### B. Canny Edge Detector

The Canny edge detector is nothing more than a simple function call in OpenCV. Although we have not went over Canny in class yet, I found some basic information on it from OpenCV's documentation. Basically it runs The Gaussian Filter on it to remove noise, then calculates Gx and Gy using a kernel, then G =$\sqrt{Gx^2 + Gy^2}$ and $\theta = tan^{-1}(Gy/Gx)$ for each pixel. Then it thresholds it depending on the users T value entered, leaving only the edges within that threshold T.

### C. Histogram Equalization

This is a simple function call in OpenCV. The OpenCV documentation states that it first calculates a cumulative distribution using a CDF(cumulative distribution function), then it normalizes it using a maximum value of 255, and finally it uses a simple remapping procedure to obtain the intensity values of the equalized image.

## III. Description of implementation

The entire code is developed in C++ language on Ubuntu 16.04.1 and USF's FSPrime linux server. The code for reading, writing the ppm and pgm image files, and conversion from HSI is provided as part of the Image Class file. The HSI Class contains conversion to HSI from the Image Class. The opencv class contains the opencv functions used in this project. The main is in project.cpp and reads the parameters file, which contains all the info on the input image, output image, and process to run on the image with their ROIs. It then saves the resulting image(s). Canny and Sobel take in a threshold value T and all processes also have an ROI that uses the following format that is put in after the process parameter(s): startX startY SizeX SizeY. The README includes a more detailed explanation.

## IV. Description and Analysis of Results

### A. Description of Results

This section illustrates the results of the algorithms used. The first group of images are the before and after of the Prewitt, Sobel, and Canny Edge Detectors, Figures 1-4. The second group shows the original image with Histogram Stretching we wrote in Project 2 and Histogram Equalization from OpenCV, Figures 5-7.

Fig. 1. Original Snow Boarder Gray Scale

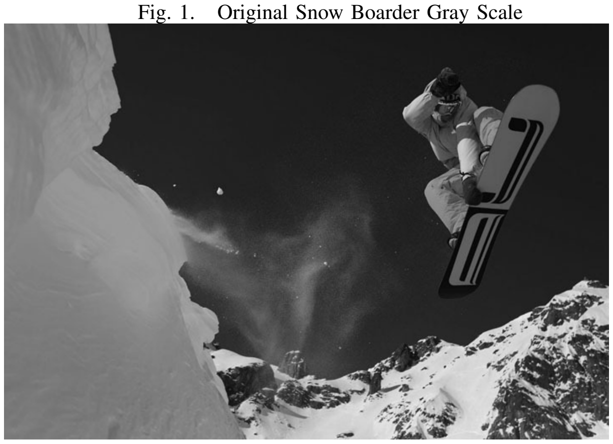


Fig. 2. Edges Prewitt 3x3

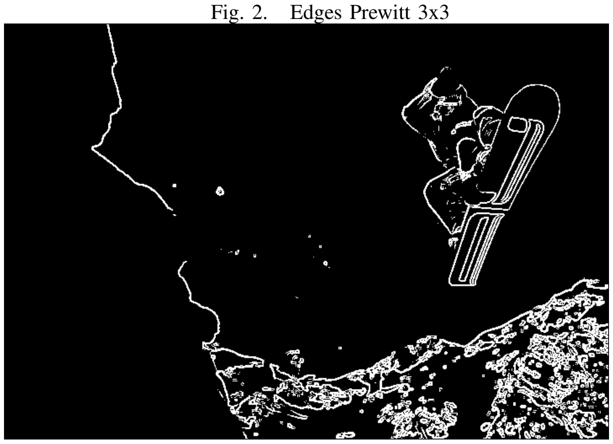


Fig. 3. Edges Sobel 3x3

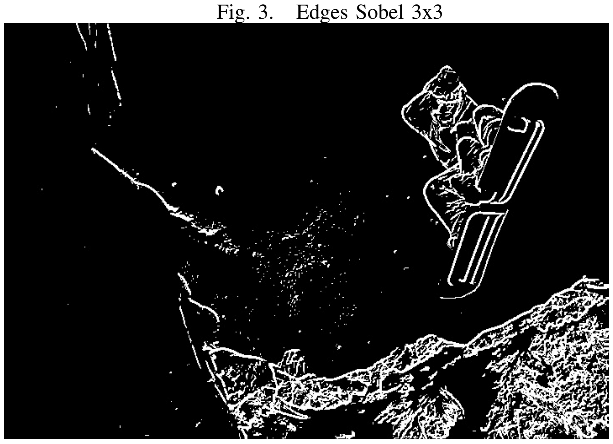


Fig. 4. Edges Canny 3x3

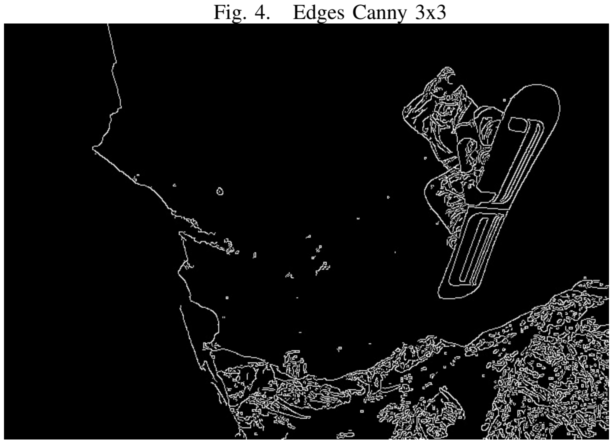


Fig. 5. Crayons Original Image

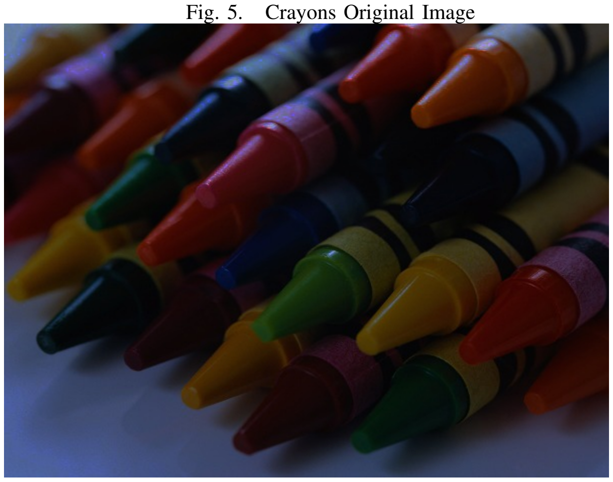


Fig. 6. Histogram Stretching

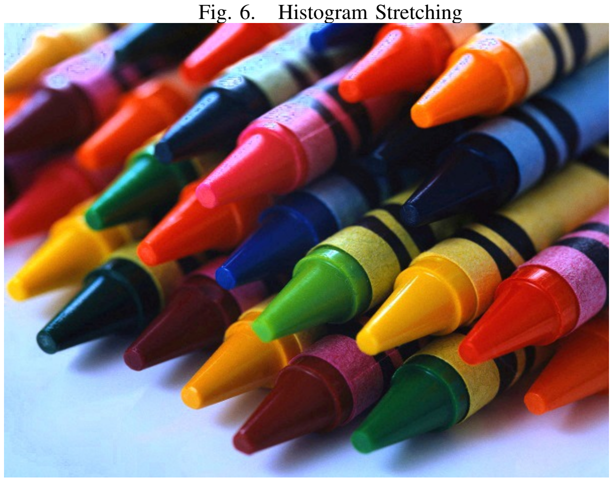
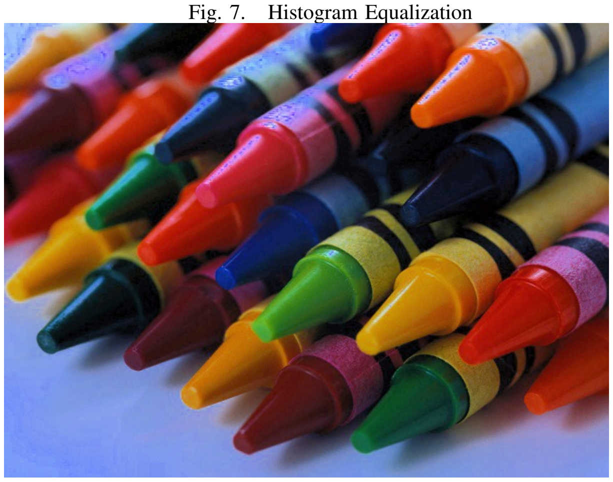


Fig. 7. Histogram Equalization



### B. *Performance Evaluation and Analysis of Results*

I am not sure about the exact complexity of OpenCV's Canny, Sobel, and Histogram Equalization because there website does not state the complexity. However, I did take the running times and compared them against our previous Prewitt and Histogram Stretching functions and OpenCV is much much faster. The Prewitt 3x3 ran at 92ms while Sobel ran at 2.3ms and Canny at 0.7ms for the same images, which are both substantially faster than Prewitt. Sobel and Prewitt have very similar end results but Canny is both faster and superior edge detection. Canny has finer and more precise edges than Sobel and Prewitt and is significantly faster than both. The previous Histogram Stretching implementation took 141ms where OpenCV's Histogram Equalization took only 3.9ms. The end result is slightly different depending on which A,B,C,D you choose for stretching but OpenCVs is much faster. Since the stretching takes in user input to focus on a certain range of intensity levels, it has more user precision but is far harder to implement over OpenCV's simple equalization function. As you can see, OpenCV has been highly optimized and is much faster than my previous implementations of similar functions. However, I am disappointed they do not include the complexity on their documentation site like Boost does.

## V. Conclusion

This assignment focuses on basic uses of the OpenCV library. It focuses mainly on the Sobel and Canny edge detectors as well as Histogram Equalization functions. OpenCV seems to be a very fast and robust image processing library that has so many built in processes its ridiculous. The imshow function is also really neat and helps with debugging because you can view the image in between steps. Overall, OpenCV is a great library and I can see many uses for it in the future.

## References

[1] Example Project Code and Example Report from TA website.
[2] My Project 1.
[3] My Project 2.
[4] My Project 3.
[5] OpenCV Documentation http://http://docs.opencv.org/2.4/