Barcode Detection Program

Katrina Mitzi A. Chua

Barcode Detection Program

A Project by

Katrina Mitzi A. Chua

Submitted to

Luisito L. Agustin
Instructor, CE 101

In Partial Fulfillment of the Requirements for the Course
CE 101: Techniques in Signal Processing

Department of Electronics, Computer and Communications Engineering
School of Science and Engineering
Loyola Schools
Ateneo de Manila University
Quezon City, Philippines

April 2019

Abstract

This project is a barcode detector program. It utilizes OpenCV as an extra library that carries out different signal processing functions and acts as a GUI, Visual Studio 2017 as the IDE and C++ as the programming language. It focuses on detecting a set of contours that make up the barcode. The project at first was using the Canny Edge-Detection Algorithm to process the image but since that did not give the desired output image type, a Sobel Edge-Detection Algorithm was used instead. The program would output each stage of the image processing, with the input being preferably a jpeg file, and the user would have to press enter to move on to each next stage. The only control the user would have of the stages is the thresholding as it will affect most of the latter processes. The final output would be a box around the barcode. The code was tested on three images two with noise (such as being on a surface like a jar) and another with no noise (just a barcode image).

Acknowledgements

The student would like to express her deepest gratitude to Alyssa Cuan for helping in finding a solution to the errors in the final stages of the project, Nymark Kho for helping the student find references to the harder parts of the code and for helping to find a solution on how to use OpenCV on wxDev-C++ and Visual Studio earlier in the project.

The researcher sincerely thanks all the contributors on the internet that have contributed their source codes as reference for those who need the help especially for OpenCV users in C++ because the source codes are usually in Python.

The researcher would also like to thank Dr. Luisito Agustin for monitoring and having mercy on the researcher as she was doing the group project alone and for letting the researcher use OpenCV and Visual Studio.

Lastly, the researcher would like to thank the reader and hopes that this project might help pave the way to pass the reader's DSP class.

Table of Contents

Introduction

The aim of this project is to be able to create a program that helps in identifying whether a certain image contains a barcode or not. The significance of the project depends highly on whether the researcher is able to use her knowledge of digital signal processing or not to create a project. In this project, several image processing algorithms are implemented to make the application work.

The objectives of this project are patterned on what the researcher has learned in term of digital signal processing, especially on image processing. The primary input needed for the program is any image with a barcode. This will be the first step to extracting the barcode. The next steps would be that the image would be made into a grayscale format and will be able to be manually thresholded on another window when the threshold button is pushed in the first window. Then the algorithms would be applied to extract the contours of the barcode.

Theoretical Background

2.1. Image Processing

There are two types of image processing: analog and digital. Analog image processing is applied on analog signals and it processes only two-dimensional signals. Images are manipulated by electrical signals and in analog image processing, analog signals can be periodic or non-periodic. An example analog images are television images, photographs, paintings and medical images etc. On the other hand, digital image processing is applied to digital images which is made up of a matrix of small pixels and elements. There is a number of software and algorithms that can be applied for manipulating the images to perform changes. Some examples of digital images are color processing, image recognition, video processing, etc.

2.2. OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning library. It was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. OpenCV has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.

2.3. Grayscale

A digital image usually contains both color information and luminance or grayscale. If the color information is removed, the grayscale black and white image is left. It is the only portion of images not removed otherwise only a pure black image would be seen despite the color information. Most digital imaging software, even the most basic ones, are able to convert images into their grayscale equivalents. Grayscale is also very important when printing since it only consumes black ink in comparison to printing in color which consumes all three ink colors (cyan, magenta, and yellow) along with black.

This code shows the grayscale function:

```
Mat grayscale(Mat src, int method) {
```

```
	Mat output(src.rows, src.cols, CV_8UC1);
	// basic grayscale
	if (method == GRAY_AVG) {
		for (int i = 0; i < src.rows; i++) {
			for (int j = 0; j < src.cols; j++) {
				//cout << "(" << to_string(i) << ", " <<
to_string(j) << ")" << to_string(img.at<Vec3b>(i, j)[0]) << " " <<
to_string(img.at<Vec3b>(i, j)[1]) << " " << to_string(img.at<Vec3b>(i,
j)[2]) << endl;
				int origBlue = src.at<Vec3b>(i, j)[0]; //blue
				int origGreen = src.at<Vec3b>(i, j)[1]; //green
				int origRed = src.at<Vec3b>(i, j)[2]; //red
				int grayscale = (origBlue + origGreen + origRed)
/ 3;
				output.at<uchar>(i, j) = grayscale;
			}
		}
	}
	// clearer grayscale
	else if (method == GRAY_WEIGHT) {
		for (int i = 0; i < src.rows; i++) {
			for (int j = 0; j < src.cols; j++) {
				//cout << "(" << to_string(i) << ", " <<
to_string(j) << ")" << to_string(img.at<Vec3b>(i, j)[0]) << " " <<
to_string(img.at<Vec3b>(i, j)[1]) << " " << to_string(img.at<Vec3b>(i,
j)[2]) << endl;
				int origBlue = src.at<Vec3b>(i, j)[0]; //blue
				int origGreen = src.at<Vec3b>(i, j)[1]; //green
				int origRed = src.at<Vec3b>(i, j)[2]; //red
				int grayscale = (0.3 * origRed) + (0.59 * orig-
Green) + (0.11 * origBlue);
				output.at<uchar>(i, j) = grayscale;
			}
		}
	}

	return output;
}
```

2.4. Thresholding

Image thresholding is a simple but effective way of partitioning an image into a foreground and a background. This type of image segmentation isolates objects by converting grayscale images into binary images. Image processing would be most effective with high levels of contrast. For this project, what will be used is set level thresholding to isolate the barcode.

These functions show the thresholding:

```
// Threshold function
Mat thresholding(Mat sobel, int method) {
	Mat output = Mat(sobel.rows, sobel.cols, CV_8U);
	if (method == THRESH_MOD) {
		if (sobel.channels() != 1) {
			cout << "The input image must be single-channeled!" <<
endl;
```

```
                  system("EXIT");
            }
            double minVal = 0, maxVal = 0;
            for (int i = 0; i < sobel.rows; i++) {
                  for (int j = 0; j < sobel.cols; j++) {
                        //Mat kernel = Mat::zeros(9, 9, output.type());
                        // Implement logic to fill the 9x9 kernel with
                        // values from the gray Mat, respecting bounda-
ries.

                        //Scalar avg_intensity = mean(kernel);
                        //minMaxLoc(kernel, &minVal, &maxVal);

                        if (sobel.at<uchar>(i, j) <= 255 && so-
bel.at<uchar>(i, j) > threshold_value) {
                              output.at<uchar>(i, j) = 255;
                        }
                        else {
                              output.at<uchar>(i, j) = 0;
                        }
                  }
            }
      }
      return output;
}

void thresholding_call(int, void*) {
      threshold_value = (int)threshold_value;
      thresh = Mat(blurred.rows, blurred.cols, CV_8U);
      thresh = thresholding(blurred, THRESH_MOD);
      imshow("Thresholded Image", thresh);
}
```

Project Overview

3.1 Objectives

The goal of this project is to develop an OpenCV application that is able to detect a barcode within an uploaded image. The image should be able to be read by the program then the program should be able to first convert the image into a grayscale version of the image and then should transition to a Sobel Edge-Detected image and then a gradient should be applied then a blur and then the user should be able to let the user toggle with the threshold of the image. The program should then be able to close the contours, erode the image, dilate the image, find the contours again and get the biggest one and the draw a box based on the contour.

3.2 Significance of the Project

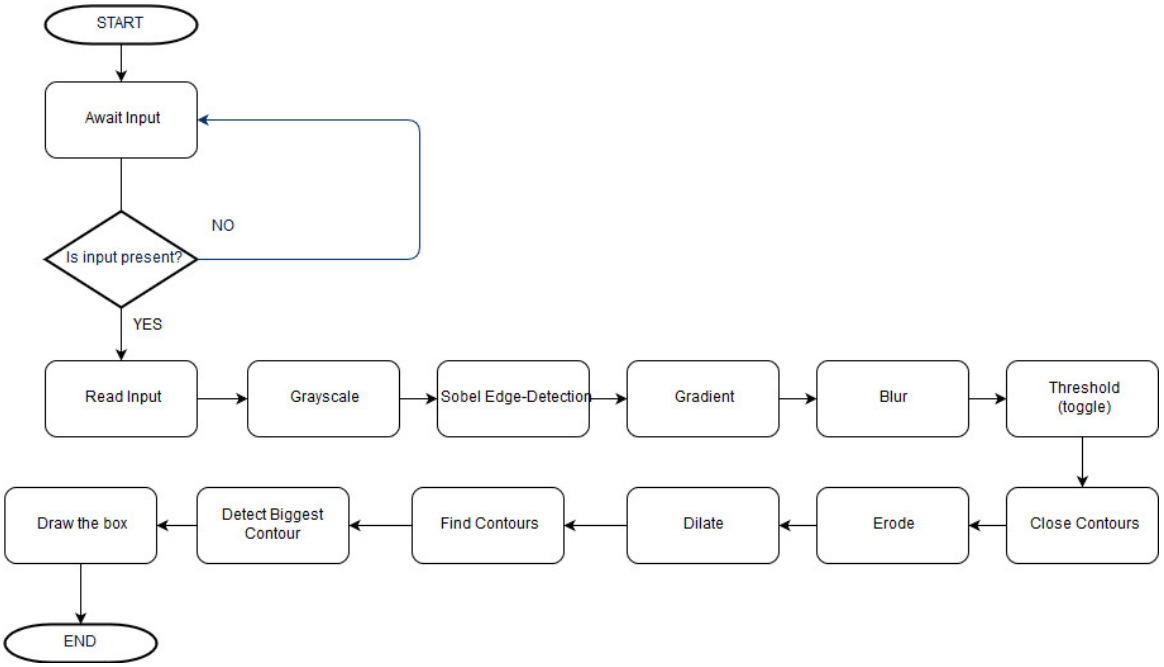
Aside from the fact that this project is made to supplement the student's knowledge in digital signal processing, and it is in partial fulfillment of her CE 101 Techniques in Signal Processing course, this project would help add to the current small database of C++ OpenCV code. This would also help OpenCV C++ developers have a basis for further implementation of a barcode scanner using only the OpenCV library.

As barcodes are very popular today to track products, this knowledge would be very useful to C++ developers who are developing barcode scanners and readers or even just detectors.

3.3 Scope and Limitations

The scope of the project is detecting a barcode from the image. Its limitations include the usage of a 64-bit OpenCV library so a 32-bit barcode reader library cannot be included as some functions are not compatible with 64-bit. The next limitation is that the barcode detector can only filter out some degree of noise from an image from an uncontrolled environment (e.g. a barcode on a jar). The next limitation is due to the library not being able to detect all kinds of images usually the processing stops at the thresholding function. This is most likely due to the weird way the trackbar was called inside the thresholding window. The final limitation has to do with the interface. The interface is limited to the capabilities of the GUI options of OpenCV. Therefore, the interface is not very sophisticated.

3.4 Project Flowchart



Algorithm and Program Discussion

4.1. Library Installation

Since the OpenCV library has environment dependencies, the library must therefore be installed and included properly. The requirement for that is to first have Visual Studio 2017 installed in the system then go to OpenCV's official download page and download OpenCV 4.0.1 and extract it in the desired destination. Open the project solution (.sln) using Visual Studio 2017. Then go on to the Project tab -> FinalProject Properties…. Then when a window called FinalProject Property Pages appears, select the All Configurations as the configuration and select C/C++ -> General from Configuration Properties. On the Additional Include Directories row, browse and add the include folder from the build folder in the opencv file. Next, select Linker -> General and under the row for the Additional Library dependencies, add the lib folder from the vc14 in the x64 folder from the build folder in the opencv file. Next, select Linker -> Input and under the Additional Dependencies row, add the line "opencv_world401.lib". Finally, selecting the configuration to Debug and under the same category and row, instead of "opencv_world401.lib", replace it with "opencv_world401d.lib". Click the apply button for every change made. The project is then ready to run.

4.2. Image Upload

The image is uploaded using OpenCV's imread() function. The function will read the image based on the image name. The image will be uploaded and then shown using the imshow() function on the first window along with a button that leads to the window where you manually threshold the grayscale of the image.

The image read function is shown in this function:

```
// read the image
Mat ReadImage(string imageName) {
	Mat img = imread(imageName, IMREAD_COLOR);
	if (img.empty()) {
		cout << "Cannot read image: " << imageName << std::endl;
		system("EXIT");
	}
	return img;
}
```

4.2. Sobel Edge Detection

Edge detection is a case of trying to find the regions in which the image has sharp changes in intensity or color. A high value in color change is a steep change and a low value in color change is a shallow change. The Sobel operator is a common operator to do this. The Sobel operator is an approximation to a derivative of an image. It is separated into the x and y directions. Essentially, the Sobel operator is trying to find out the amount of the difference by placing a gradient matrix over each pixel of the image. It is very quick to execute since it produces the same output image every time it is called. It also makes for a stable edge detection technique.

This function shows the Sobel Edge Detection:

```
// Sobel Edge Detection function
Mat SobelDetect(Mat gray) {
	int dx[3][3] = { {1, 0, -1},{2, 0, -2},{1, 0, -1 } };
	int dy[3][3] = { {1, 2, 1},{0, 0, 0},{-1, -2, -1} };

	Mat output = Mat(gray.rows, gray.cols, CV_8U);
	Mat kernel = Mat(3, 3, CV_8U);

	int max = -200, min = 2000;

	for (int i = 1; i < gray.rows - 2; i++) {
		for (int j = 1; j < gray.cols - 2; j++) {
			// apply kernel in X and Y directions
			int sumX = 0;
			int sumY = 0;
			uchar ker;
			for (int m = -1; m <= 1; m++) {
				for (int n = -1; n <= 1; n++) {
					// get the (i,j) pixel value
					kernel.at<uchar>(m + 1, n + 1) =
gray.at<uchar>(i + m, j + n);
					sumX += kernel.at<uchar>(m + 1, n + 1) *
dx[m + 1][n + 1];
					sumY += kernel.at<uchar>(m + 1, n + 1) *
dy[m + 1][n + 1];
				}
			}
			int sum = abs(sumX) + abs(sumY);
			//cout << sum << endl;
			output.at<uchar>(i, j) = (sum > 255) ? 255 : sum;
			//output2.at<uchar>(i, j) = kernel.at<uchar>(i, j);
		}
	}
	return output;
}
```

4.3. Blur

When an image gets blurred, the image that looked sharp and detailed and is very clear that gets its sharpness from the edges inside the picture would have the edge content reduced to make the transition from one color to another very smooth.

This code shows the implementation of a blur function within the program:

```
// Blur function 9x9 kernel
Mat blurImage(Mat gradient) {
	Mat output(gradient.rows, gradient.cols, CV_8U);
	int total = 0;
	//blur
	for (int i = 0; i < gradient.rows; i++) {
		for (int j = 0; j < gradient.cols; j++) {
			int ksize = 9;
			total = 0;
			for (int x = -ksize / 2; x <= ksize / 2; x++) {
				for (int y = -ksize / 2; y <= ksize / 2; y++) {
					int tx = i + x;
					int ty = j + y;
					if (tx > 0 && tx < gradient.rows && ty >=
0 && ty < gradient.cols) {
						total += gradient.at<uchar>(tx, ty);
					}
				}
			}
			output.at<uchar>(i, j) = total / ksize / ksize;
		}
	}
	return output;
}
```

4.4. Erode

Erosion works when you take a blob, for example a white blob, and erode it based on the number of pixels the user wants to erode it. The number of pixels would determine how many pixels around the edges to trim. This function was formulated only to trim 1 pixel around the designated blob and it was also designed to trim rectangularly.

This code shows the implementation of an erosion function within the program:

```
// Erode function
Mat erodeImage(Mat closed) {
	Mat output(closed.rows, closed.cols, CV_8U);
	// dilate
	for (int i = 1; i < closed.rows; i++) {
		for (int j = 1; j < closed.cols; j++) {
			if (closed.at<uchar>(i, j) == 0) {
				if (i > 0 && closed.at<uchar>(i - 1, j) == 255)
{
					closed.at<uchar>(i - 1, j) = 1;
				}
```

```
                        if (j > 0 && closed.at<uchar>(i, j - 1) == 255)
{
                              closed.at<uchar>(i, j - 1) = 1;
                        }
                        if (i + 1 < closed.rows && closed.at<uchar>(i +
1, j) == 255) {
                              closed.at<uchar>(i + 1, j) = 1;
                        }
                        if (j + 1 < closed.cols && closed.at<uchar>(i, j
+ 1) == 255) {
                              closed.at<uchar>(i, j + 1) = 1;
                        }
                        if (i > 0 && j > 0 && closed.at<uchar>(i - 1, j
- 1) == 255) {
                              closed.at<uchar>(i - 1, j - 1) = 1;
                        }
                        if (i > 0 && j + 1 < closed.cols &&
closed.at<uchar>(i - 1, j + 1) == 255) {
                              closed.at<uchar>(i - 1, j + 1) = 1;
                        }
                        if (i + 1 < closed.rows && j > 0 &&
closed.at<uchar>(i + 1, j - 1) == 255) {
                              closed.at<uchar>(i + 1, j - 1) = 1;
                        }
                        if (i + 1 < closed.rows && j + 1 < closed.cols
&& closed.at<uchar>(i + 1, j + 1) == 255) {
                              closed.at<uchar>(i + 1, j + 1) = 1;
                        }
                  }
            }
      }
      for (int i = 0; i < closed.rows; i++) {
            for (int j = 0; j < closed.cols; j++) {
                  if (closed.at<uchar>(i, j) == 1) {
                        closed.at<uchar>(i, j) = 0;
                  }
                  output.at<uchar>(i, j) = closed.at<uchar>(i, j);
            }
      }
      return output;
}
```

4.5. Dilate

Dilation works when you take a blob, for example a white blob, and dilate it based on the number of pixels the user wants to dilate it. The number of pixels would determine how many pixels around the edges to expand. This function was formulated only to expand 1 pixel around the designated blob and is designed to add to the image rectangularly. This and the erosion function were designed to further filter out the small noise around a larger blob.

This code shows the implementation of a dilation function within the program:

```
// Dilate function
Mat dilateImage(Mat eroded) {
```

```
	Mat output(eroded.rows, eroded.cols, CV_8U);
	// dilate
	for (int i = 1; i < eroded.rows; i++) {
		for (int j = 1; j < eroded.cols; j++) {
			if (eroded.at<uchar>(i, j) == 255) {
				if (i > 0 && eroded.at<uchar>(i - 1, j) == 0) {
					eroded.at<uchar>(i - 1, j) = 254;
				}
				if (j > 0 && eroded.at<uchar>(i, j - 1) == 0) {
					eroded.at<uchar>(i, j - 1) = 254;
				}
				if (i + 1 < eroded.rows && eroded.at<uchar>(i +
1, j) == 0) {
					eroded.at<uchar>(i + 1, j) = 254;
				}
				if (j + 1 < eroded.cols && eroded.at<uchar>(i, j
+ 1) == 0) {
					eroded.at<uchar>(i, j + 1) = 254;
				}
				if (i > 0 && j > 0 && eroded.at<uchar>(i - 1, j
- 1) == 0) {
					eroded.at<uchar>(i - 1, j - 1) = 254;
				}
				if (i > 0 && j + 1 < eroded.cols &&
eroded.at<uchar>(i - 1, j + 1) == 0) {
					eroded.at<uchar>(i - 1, j + 1) = 254;
				}
				if (i + 1 < eroded.rows && j > 0 &&
eroded.at<uchar>(i + 1, j - 1) == 0) {
					eroded.at<uchar>(i + 1, j - 1) = 254;
				}
				if (i + 1 < eroded.rows && j + 1 < eroded.cols
&& eroded.at<uchar>(i + 1, j + 1) == 0) {
					eroded.at<uchar>(i + 1, j + 1) = 254;
				}
			}
		}
	}
	for (int i = 0; i < eroded.rows; i++) {
		for (int j = 0; j < eroded.cols; j++) {
			if (eroded.at<uchar>(i, j) == 254) {
				eroded.at<uchar>(i, j) = 255;
			}
			output.at<uchar>(i, j) = eroded.at<uchar>(i, j);
		}
	}
	return output;
}
```

4.6. Closing Contours

This function was formulated from the ground up by the student. This involves the rectangular closing of nearby contours by adding an integer parameter called pixel_spacing to the function. This pixel spacing is used to determine whether to fill in the spaces between the different blobs or not. It processes per line and will fill the space in between two white

pixels in with white if the number of pixels in between the two white pixels is less than pixel_spacing.

This code shows the implementation of a rectangular closing function within the program:

```
// rectangular morph
Mat closeContours(Mat thresh, int pixel_spacing) {
	Mat output(thresh.rows, thresh.cols, CV_8U);
	for (int i = 0; i < thresh.rows; i++) {
		for (int j = 0; j < thresh.cols; j++) {
			if (thresh.at<uchar>(i, j) == 255) {
				for (int a = 0; a < pixel_spacing + 1; a++) {
					if (thresh.at<uchar>(i, j + a) == 255) {
						for (int b = 0; b < a; b++) {
							thresh.at<uchar>(i, j + b) =
255;
						}
					}
				}

			}
		}
	}
	for (int i = 0; i < thresh.rows; i++) {
		for (int j = 0; j < thresh.cols; j++) {
			output.at<uchar>(i, j) = thresh.at<uchar>(i, j);
		}
	}
	return output;
}
```

4.7. Finding Contours

This function was also made by the student from the ground up and is used to find the contour/s of the function needed to draw the box around the barcode. This was made to detect all the blobs and check until where the white blobs were. Then it would proceed to draw a box over the original image over the barcode in the main function.

This code shows the implementation of finding the contours inside function within the program:

```
// find all contours
queue<int> find_contours(Mat dilated) {
	int max_x = 0, max_y = 0, min_x = dilated.cols, min_y = di-
lated.rows;
	//Mat output = Mat(dilated.rows, dilated.cols, CV_8UC3);
	queue<int> output;
	vector<Point> all_contours;

	// find all white px and store in vector
	for (int i = 0; i < dilated.rows; i++) {
		for (int j = 0; j < dilated.cols; j++) {
```

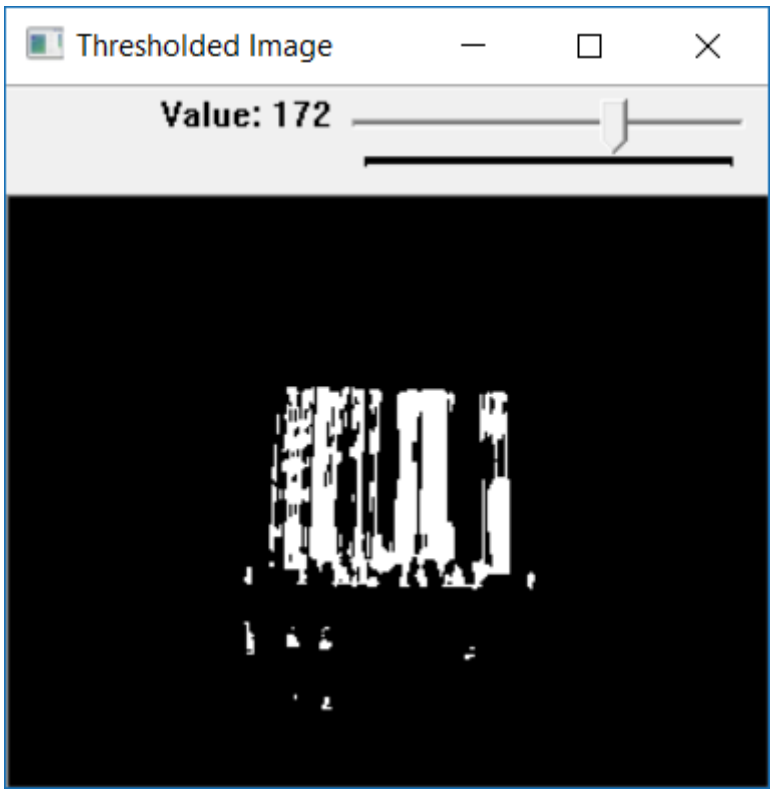
```
            if (dilated.at<uchar>(i, j) == 255) {
                  all_contours.push_back(Point(j, i));
            }
      }
}
// search for max and min x,y
for (int cont = 0; cont < all_contours.size(); cont++) {
      if (all_contours.at(cont).x > max_x) {
            max_x = all_contours.at(cont).x;
      }
      if (all_contours.at(cont).y > max_y) {
            max_y = all_contours.at(cont).y;
      }
      if (all_contours.at(cont).x < min_x) {
            min_x = all_contours.at(cont).x;
      }
      if (all_contours.at(cont).y < min_y) {
            min_y = all_contours.at(cont).y;
      }
}

// detect in min and max x,y for lines more than 80% white
//top to bottom
for (int i = min_y; i <= max_y; i++) {
      float count = 0.0;
      for (int j = min_x; j < max_x; j++) {
            //if white
            if (dilated.at<uchar>(j, i) == 255) {
                  count++;
            }
      }
      float percentage = (count / (max_x - min_x)) * 100;
      if (percentage < 80.0) {
            break;
      }
      else {
            min_y++;
      }
}

//bottom to top
for (int i = max_y; i >= min_y; i--) {
      float count = 0.0;
      for (int j = min_x; j < max_x; j++) {
            //if white
            if (dilated.at<uchar>(j, i) == 255) {
                  count++;
            }
      }
      float percentage = (count / (max_x - min_x)) * 100;
      if (percentage < 80.0) {
            break;
      }
      else {
            max_y--;
      }
}
output.push(max_x);
output.push(min_x);
output.push(max_y);
output.push(min_y);
```

```
        return output;
    }
```

Graphical User Interface

The Graphical User Interface of the program was designed using the built-in OpenCV library. This was done out of convenience of the code since it is integrated in the library.

The usual interface would be a window opening per step of the way and just awaiting the press of enter or any button. This is true for all except the threshold image window where the blurred image is shown along with a threshold trackbar. The trackbar can be used to adjust the threshold of the blurred image as this step is important to the pre-processing of the image for detection. The navigation in the cases of after thresholding and when the user is done is by pressing enter or any key.

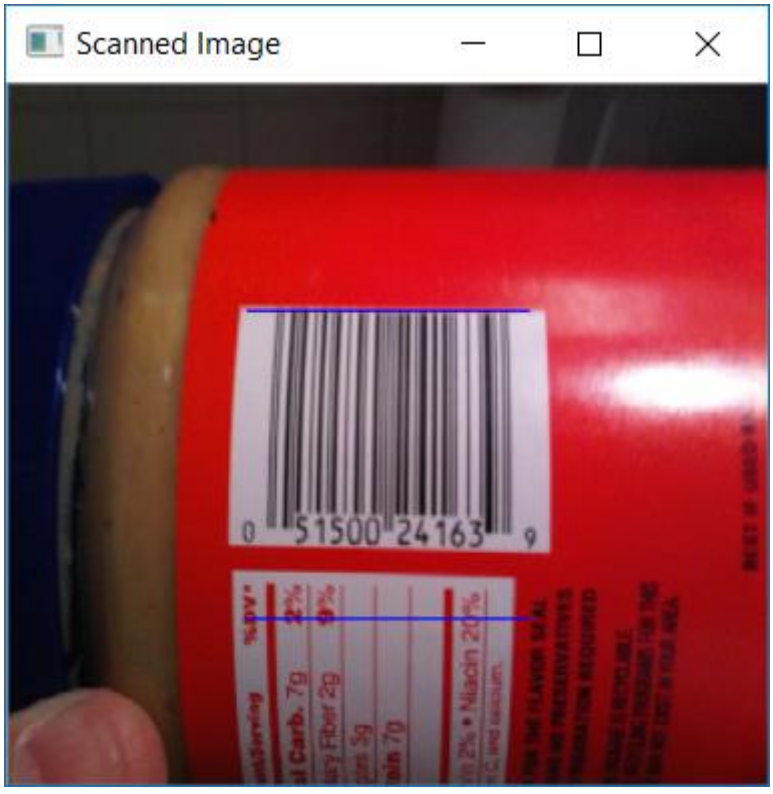


Test Results

6.1. Image with Noise (Barcode with an Uncontrolled Environment)

The initial image picked is shown below:

When the image is processed, the detection of the contours of this image is as shown below:



This image shows the contours of the barcode and some noise that was hard to filter out. The amount of noise would have to depend on the image that was used. This one had

white spots that were hard to get rid of due to the reflection of the light on the side of the jar.

Conclusion and Recommendations

In conclusion, the student was able to create a program that successfully detects a barcode or something similar to a barcode. The objectives stated that the student should be able to detect where the barcode or something similar is, and the program successfully does that.

As a future suggestion to improve the project, the student recommend that a different way of filtering be used for the program to further filter out the noise. The limited OpenCV C++ sources might not help and the future researcher might want to either try Python or know how to translate Python to further improve on the results.

Appendix

```
#include <opencv2/opencv.hpp>
#include <opencv2/objdetect.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace std;
using namespace cv;

enum GrayscaleMethods {
	GRAY_AVG = 1,
	GRAY_WEIGHT = 2
};

enum ThresholdMethods {
	THRESH_MOD = 1
};

// Matrices used
Mat src, gray, sobel, gradient, blurred, thresh, closed, dilated,
eroded;

int threshold_value = 0;

// read the image
Mat ReadImage(string imageName) {
	Mat img = imread(imageName);
	if (img.empty()) {
		cout << "Cannot read image: " << imageName << std::endl;
		system("EXIT");
	}
	return img;
```

```
}


// Grayscale function
Mat grayscale(Mat src, int method) {
	Mat output(src.rows, src.cols, CV_8UC1);
	// basic grayscale
	if (method == GRAY_AVG) {
		for (int i = 0; i < src.rows; i++) {
			for (int j = 0; j < src.cols; j++) {
				//cout << "(" << to_string(i) << ", " << to_string(j) << ")" << to_string(img.at<Vec3b>(i, j)[0]) << " " << to_string(img.at<Vec3b>(i, j)[1]) << " " << to_string(img.at<Vec3b>(i, j)[2]) << endl;
				int origBlue = src.at<Vec3b>(i, j)[0]; //blue
				int origGreen = src.at<Vec3b>(i, j)[1]; //green
				int origRed = src.at<Vec3b>(i, j)[2]; //red
				int grayscale = (origBlue + origGreen + origRed) / 3;
				output.at<uchar>(i, j) = grayscale;
			}
		}
	}
	// clearer grayscale
	else if (method == GRAY_WEIGHT) {
		for (int i = 0; i < src.rows; i++) {
			for (int j = 0; j < src.cols; j++) {
				//cout << "(" << to_string(i) << ", " << to_string(j) << ")" << to_string(img.at<Vec3b>(i, j)[0]) << " " << to_string(img.at<Vec3b>(i, j)[1]) << " " << to_string(img.at<Vec3b>(i, j)[2]) << endl;
				int origBlue = src.at<Vec3b>(i, j)[0]; //blue
				int origGreen = src.at<Vec3b>(i, j)[1]; //green
				int origRed = src.at<Vec3b>(i, j)[2]; //red
				int grayscale = (0.3 * origRed) + (0.59 * origGreen) + (0.11 * origBlue);
				output.at<uchar>(i, j) = grayscale;
			}
		}
	}
```

```
	return output;
}

// Threshold function
Mat thresholding(Mat sobel, int method) {
	Mat output = Mat(sobel.rows, sobel.cols, CV_8U);
	if (method == THRESH_MOD) {
		if (sobel.channels() != 1) {
			cout << "The input image must be single-channeled!" <<
endl;
			system("EXIT");
		}
		double minVal = 0, maxVal = 0;
		for (int i = 0; i < sobel.rows; i++) {
			for (int j = 0; j < sobel.cols; j++) {
				//Mat kernel = Mat::zeros(9, 9, output.type());
				// Implement logic to fill the 9x9 kernel with
				// values from the gray Mat, respecting
boundaries.

				//Scalar avg_intensity = mean(kernel);
				//minMaxLoc(kernel, &minVal, &maxVal);

				if (sobel.at<uchar>(i,j) <= 255 &&
sobel.at<uchar>(i,j) > threshold_value) {
					output.at<uchar>(i, j) = 255;
				}
				else {
					output.at<uchar>(i, j) = 0;
				}
			}
		}
	}
	return output;
}
```

```
void thresholding_call(int, void*) {
	threshold_value = (int)threshold_value;
	thresh = Mat(blurred.rows, blurred.cols, CV_8U);
	thresh = thresholding(blurred, THRESH_MOD);
	imshow("Thresholded Image", thresh);
}

// Sobel Edge Detection function
Mat SobelDetect(Mat gray) {
	int dx[3][3] = { {1, 0, -1},{2, 0, -2},{1, 0, -1 } };
	int dy[3][3] = { {1, 2, 1},{0, 0, 0},{-1, -2, -1} };

	Mat output = Mat(gray.rows, gray.cols, CV_8U);
	Mat kernel = Mat(3, 3, CV_8U);

	int max = -200, min = 2000;

	for (int i = 1; i < gray.rows - 2; i++) {
		for (int j = 1; j < gray.cols - 2; j ++) {
			// apply kernel in X and Y directions
			int sumX = 0;
			int sumY = 0;
			uchar ker;
			for (int m = -1; m <= 1; m++) {
				for (int n = -1; n <= 1; n++) {
					// get the (i,j) pixel value
					kernel.at<uchar>(m+1, n+1) =
gray.at<uchar>(i + m, j + n);
					sumX += kernel.at<uchar>(m+1, n+1) * dx[m
+ 1][n + 1];
					sumY += kernel.at<uchar>(m+1, n+1) * dy[m
+ 1][n + 1];
				}
			}
			int sum = abs(sumX) + abs(sumY);
			//cout << sum << endl;
			output.at<uchar>(i, j) = (sum > 255) ? 255 : sum;
```

```
                  //output2.at<uchar>(i, j) = kernel.at<uchar>(i, j);
            }
      }
      return output;
}

// Blur function 9x9 kernel
Mat blurImage(Mat gradient) {
      Mat output(gradient.rows, gradient.cols, CV_8U);
      int total = 0;
      //blur
      for (int i = 0; i < gradient.rows; i++) {
            for (int j = 0; j < gradient.cols; j++) {
                  int ksize = 9;
                  total = 0;
                  for (int x = -ksize / 2; x <= ksize / 2; x++) {
                        for (int y = -ksize / 2; y <= ksize / 2; y++) {
                              int tx = i + x;
                              int ty = j + y;
                              if (tx > 0 && tx < gradient.rows && ty >=
0 && ty < gradient.cols) {
                                    total += gradient.at<uchar>(tx, ty);
                              }
                        }
                  }
                  output.at<uchar>(i, j) = total / ksize / ksize;
            }
      }
      return output;
}

// Erode function
Mat erodeImage(Mat closed) {
      Mat output(closed.rows, closed.cols, CV_8U);
      // dilate
      for (int i = 1; i < closed.rows; i++) {
```

```
            for (int j = 1; j < closed.cols; j++) {
                  if (closed.at<uchar>(i, j) == 0) {
                        if (i > 0 && closed.at<uchar>(i - 1, j) == 255)
{
                              closed.at<uchar>(i - 1, j) = 1;
                        }
                        if (j > 0 && closed.at<uchar>(i, j - 1) == 255)
{
                              closed.at<uchar>(i, j - 1) = 1;
                        }
                        if (i + 1 < closed.rows && closed.at<uchar>(i +
1, j) == 255) {
                              closed.at<uchar>(i + 1, j) = 1;
                        }
                        if (j + 1 < closed.cols && closed.at<uchar>(i, j
+ 1) == 255) {
                              closed.at<uchar>(i, j + 1) = 1;
                        }
                        if (i > 0 && j > 0 && closed.at<uchar>(i - 1, j
- 1) == 255) {
                              closed.at<uchar>(i - 1, j - 1) = 1;
                        }
                        if (i > 0 && j + 1 < closed.cols &&
closed.at<uchar>(i - 1, j + 1) == 255) {
                              closed.at<uchar>(i - 1, j + 1) = 1;
                        }
                        if (i + 1 < closed.rows && j > 0 &&
closed.at<uchar>(i + 1, j - 1) == 255) {
                              closed.at<uchar>(i + 1, j - 1) = 1;
                        }
                        if (i + 1 < closed.rows && j + 1 < closed.cols
&& closed.at<uchar>(i + 1, j + 1) == 255) {
                              closed.at<uchar>(i + 1, j + 1) = 1;
                        }
                  }
            }
      }
      for (int i = 0; i < closed.rows; i++) {
            for (int j = 0; j < closed.cols; j++) {
```

```
                  if (closed.at<uchar>(i, j) == 1) {
                        closed.at<uchar>(i, j) = 0;
                  }
                  output.at<uchar>(i, j) = closed.at<uchar>(i, j);
            }
      }
      return output;
}

// Dilate function
Mat dilateImage(Mat eroded) {
      Mat output(eroded.rows, eroded.cols, CV_8U);
      // dilate
      for (int i = 1; i < eroded.rows; i++) {
            for (int j = 1; j < eroded.cols; j++) {
                  if (eroded.at<uchar>(i, j) == 255) {
                        if (i > 0 && eroded.at<uchar>(i - 1, j) == 0) {
                              eroded.at<uchar>(i - 1, j) = 254;
                        }
                        if (j > 0 && eroded.at<uchar>(i, j - 1) == 0) {
                              eroded.at<uchar>(i, j - 1) = 254;
                        }
                        if (i + 1 < eroded.rows && eroded.at<uchar>(i +
1, j) == 0) {
                              eroded.at<uchar>(i + 1, j) = 254;
                        }
                        if (j + 1 < eroded.cols && eroded.at<uchar>(i, j
+ 1) == 0) {
                              eroded.at<uchar>(i, j + 1) = 254;
                        }
                        if (i > 0 && j > 0 && eroded.at<uchar>(i - 1, j
- 1) == 0) {
                              eroded.at<uchar>(i - 1, j - 1) = 254;
                        }
                        if (i > 0 && j + 1 < eroded.cols &&
eroded.at<uchar>(i - 1, j + 1) == 0) {
                              eroded.at<uchar>(i - 1, j + 1) = 254;
                        }
```

```
                        if (i + 1 < eroded.rows && j > 0 &&
eroded.at<uchar>(i + 1, j - 1) == 0) {
                              eroded.at<uchar>(i + 1, j - 1) = 254;
                        }
                        if (i + 1 < eroded.rows && j + 1 < eroded.cols
&& eroded.at<uchar>(i + 1, j + 1) == 0) {
                              eroded.at<uchar>(i + 1, j + 1) = 254;
                        }
                  }
            }
      }
      for (int i = 0; i < eroded.rows; i++) {
            for (int j = 0; j < eroded.cols; j++) {
                  if (eroded.at<uchar>(i, j) == 254) {
                        eroded.at<uchar>(i, j) = 255;
                  }
                  output.at<uchar>(i, j) = eroded.at<uchar>(i, j);
            }
      }
      return output;
}

// rectangular morph
Mat closeContours(Mat thresh, int pixel_spacing) {
      Mat output(thresh.rows, thresh.cols, CV_8U);
      for (int i = 0; i < thresh.rows; i++) {
            for (int j = 0; j < thresh.cols; j++) {
                  if (thresh.at<uchar>(i, j) == 255) {
                        for (int a = 0; a < pixel_spacing + 1; a++) {
                              if (thresh.at<uchar>(i, j + a) == 255) {
                                    for (int b = 0; b < a; b++) {
                                          thresh.at<uchar>(i, j + b) =
255;
                                    }
                              }
                        }
```

```
                }
            }
        }
        for (int i = 0; i < thresh.rows; i++) {
            for (int j = 0; j < thresh.cols; j++) {
                output.at<uchar>(i, j) = thresh.at<uchar>(i, j);
            }
        }
        return output;
}

// find all contours
queue<int> find_contours(Mat dilated) {
        int max_x = 0, max_y = 0, min_x = dilated.cols, min_y =
dilated.rows;
        //Mat output = Mat(dilated.rows, dilated.cols, CV_8UC3);
        queue<int> output;
        vector<Point> all_contours;

        // find all white px and store in vector
        for (int i = 0; i < dilated.rows; i++) {
            for (int j = 0; j < dilated.cols; j++) {
                if (dilated.at<uchar>(i, j) == 255) {
                    all_contours.push_back(Point(j, i));
                }
            }
        }
        // search for max and min x,y
        for (int cont = 0; cont < all_contours.size(); cont++) {
            if (all_contours.at(cont).x > max_x) {
                max_x = all_contours.at(cont).x;
            }
            if (all_contours.at(cont).y > max_y) {
                max_y = all_contours.at(cont).y;
            }
            if (all_contours.at(cont).x < min_x) {
```

```
            min_x = all_contours.at(cont).x;
      }
      if (all_contours.at(cont).y < min_y) {
            min_y = all_contours.at(cont).y;
      }
}

// detect in min and max x,y for lines more than 80% white
//top to bottom
for (int i = min_y; i <= max_y; i++) {
      float count = 0.0;
      for (int j = min_x; j < max_x; j++) {
            //if white
            if (dilated.at<uchar>(j,i) == 255) {
                  count++;
            }
      }
      float percentage = (count / (max_x - min_x)) * 100;
      if (percentage < 80.0) {
            break;
      }
      else {
            min_y++;
      }
}

//bottom to top
for (int i = max_y; i >= min_y; i--) {
      float count = 0.0;
      for (int j = min_x; j < max_x; j++) {
            //if white
            if (dilated.at<uchar>(j, i) == 255) {
                  count++;
            }
      }
      float percentage = (count / (max_x - min_x)) * 100;
```

```
            if (percentage < 80.0) {
                  break;
            }
            else {
                  max_y--;
            }
      }
      output.push(max_x);
      output.push(min_x);
      output.push(max_y);
      output.push(min_y);
      return output;
}

// compare contours
bool compareContourAreas(vector<Point>contour1, vector<Point>contour2) {
      double i = fabs(contourArea(Mat(contour1)));
      double j = fabs(contourArea(Mat(contour2)));
      return (i < j);
}

// point vector to mat function
void vector_Point_to_Mat(std::vector<Point>& v_point, Mat& mat)
{
      mat = Mat(v_point, true);
}


// main function
int main() {
      String imageName;
      cout << "Enter the image file name (ex. barcode.jpg): " << endl;
      cin >> imageName;
      // read the image
      src = ReadImage(imageName);
      // show original image
```

```
	namedWindow("Original Image", WINDOW_NORMAL);
	imshow("Original Image", src);
	waitKey(0);
	destroyWindow("Original Image");
	// grayscale the image
	Mat gray = grayscale(src.clone(), GRAY_WEIGHT);
	namedWindow("Grayscale", WINDOW_NORMAL);
	imshow("Grayscale", gray);
	waitKey(0);
	destroyWindow("Grayscale");
	// Sobel Edge Detection
	sobel = Mat(src.rows, src.cols, CV_32F);
	sobel = SobelDetect(gray);
	namedWindow("Sobel Edge Detection", WINDOW_NORMAL);
	imshow("Sobel Edge Detection", sobel);
	waitKey(0);
	destroyWindow("Sobel Edge Detection");
	// Gradient
	convertScaleAbs(sobel, gradient);
	namedWindow("Gradient", WINDOW_NORMAL);
	imshow("Gradient", gradient);
	waitKey(0);
	destroyWindow("Gradient");
	// blur the image
	blurred = blurImage(gradient);
	namedWindow("Blurred Image", WINDOW_NORMAL);
	imshow("Blurred Image", blurred);
	waitKey(0);
	destroyWindow("Blurred Image");
	namedWindow("Thresholded Image", WINDOW_NORMAL);
	// create trackbar
	const char* trackbar_value = "Value";
	createTrackbar(trackbar_value, "Thresholded Image",
&threshold_value, 254, thresholding_call);
	thresholding_call(0, 0);
	// threshold the image
```

```
thresh = thresholding(blurred, THRESH_MOD);
imshow("Thresholded Image", thresh);
waitKey(0);
destroyWindow("Thresholded Image");
// constructing a closing kernel
//Mat kernel = getStructuringElement(MORPH_RECT, Size(21, 7));
// applying the kernel to the thresholded image
closed = closeContours(thresh, 100);
//morphologyEx(thresh, closed, MORPH_CLOSE, kernel);
namedWindow("Closed Contoured Image", WINDOW_NORMAL);
imshow("Closed Contoured Image", closed);
waitKey(0);
destroyWindow("Closed Contoured Image");

// perform erosions
eroded = Mat(closed.rows, closed.cols, CV_8U);
eroded = erodeImage(closed);
for (int i = 0; i < 3; i++) {
      eroded = erodeImage(eroded);
}
namedWindow("Eroded Image", WINDOW_NORMAL);
imshow("Eroded Image", eroded);
waitKey(0);
destroyWindow("Eroded Image");

// perform dilations
dilated = Mat(eroded.rows, eroded.cols, CV_8U);
dilated = dilateImage(eroded);
for (int i = 0; i < 3; i++) {
      dilated = dilateImage(dilated);
}
namedWindow("Dilated Image", WINDOW_NORMAL);
imshow("Dilated Image", dilated);
waitKey(0);
destroyWindow("Dilated Image");
```

```
        // make the output the same size as the image by equating them
        queue<int>minmax = find_contours(dilated);
        int max_x, min_x, max_y, min_y;
        max_x = minmax.front();
        minmax.pop();
        min_x = minmax.front();
        minmax.pop();
        max_y = minmax.front();
        minmax.pop();
        min_y = minmax.front();
        minmax.pop();

        Rect box = Rect(min_x, min_y, max_x - min_x, max_y - min_y);
        rectangle(src, box, Scalar(255, 0, 0),1,8,0);

        namedWindow("Scanned Image", WINDOW_NORMAL);
        imshow("Scanned Image", src);
        waitKey(0);
        return 0;
}
```

Bibliography

[1] https://www.javatpoint.com/analog-image-processing-vs-digital-image-processing

[2] https://opencv.org/about.html

[3] https://www.techopedia.com/definition/7468/grayscale

[4] https://www.mathworks.com/discovery/image-thresholding.html

[5] http://aishack.in/tutorials/canny-edge-detector/

[6] https://stackoverflow.com/questions/13144229/opencv-cvgetstructuringelement-with-cv-shape-ellipse-produces-a-black-square

[7] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[8] https://www.pyimagesearch.com/2014/11/24/detecting-barcodes-images-python-opencv/

[9] https://stackoverflow.com/questions/13495207/opencv-c-sorting-contours-by-their-contourarea

[10] https://stackoverflow.com/questions/31458566/copy-mat-in-opencv

[11] https://stackoverflow.com/questions/33937800/how-to-make-a-simple-window-with-one-button-using-opencv-highgui-only

[12] https://docs.opencv.org/3.1.0/d2/df8/group__core__c.html

[13] https://docs.opencv.org/4.0.1/d7/dfc/group__highgui.html

[14] https://stackoverflow.com/questions/8842901/opencv-closing-the-image-display-window

[15] https://www.techiedelight.com/convert-vector-to-array-cpp/

[16] http://answers.opencv.org/question/81831/convert-stdvectordouble-to-mat-and-show-the-image/

[17] https://github.com/opencv/opencv/blob/master/modules/java/generator/src/cpp/converters.cpp

[18] http://answers.opencv.org/question/100382/convert-vectorpoint-to-mat-with-2-columns/

[19] https://stackoverflow.com/questions/19083775/convert-vector-of-points-to-mat-opencv

[20] https://stackoverflow.com/questions/26681713/convert-mat-to-array-vector-in-opencv/43669431

[21] https://github.com/opencv/opencv/issues/6701

[22] http://answers.opencv.org/question/21700/clustring-white-pixels-in-binary-image/

[23] https://stackoverflow.com/questions/30757273/opencv-findcontours-complains-if-used-with-black-white-image

[24] http://answers.opencv.org/question/4423/how-to-create-a-binary-image-mat/

[25] https://www.tutorialspoint.com/opencv/opencv_canny_edge_detection.htm

[26] http://answers.opencv.org/question/32140/draw-largestrect-contour-on-this-image/

[27] https://harismoonamkunnu.blogspot.com/2013/06/opencv-find-biggest-contour-using-c.html

[28] https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html

[29] https://www.researchgate.net/post/What_is_the_algorithm_and_concept_behind_finding_contour_in_openCV

[30] http://opencvexamples.blogspot.com/2013/09/find-contour.html