Digital Image Processing with Canny Edge Detection

Junior Independent Study Thesis

Presented in Partial Fulfillment of the
Requirements for the Degree Bachelor of Arts in
the Department of Computer Science at The
College of Wooster

by
Tam Minh Nguyen

The College of Wooster
2019

Advised by:
DR Denise Byrnes

Abstract

This paper introduces basic concepts and features of digital image processing and devotes a majority of its content to defining and exploring the Canny Edge detection algorithm with its applications on a given dataset. There are many edge detection methods, which mainly use the changes in intensity to find the edges of an image. However, the intensity change can also be gradual where the edge detection operator then has to be modified for proper edge detection. Consequently, there are problems of lack of true edges, false edge detection, and high computational time. This is why a simple program is built by using Canny's method with other image processing modifications to find a process which correctly predicts edges and has a low computational time.

Contents

List of Figures

Chapter 1

Introduction to Image Processing

1.1 What is Image Processing

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it [11]. It is a type of signal processing in which the input is an image and the output may be another image or characteristics/features associated with that image. Nowadays, image processing is a rapidly growing technology. It also forms a core research area within engineering and computer science disciplines.

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools
- Analysing and manipulating the image
- Producing output in which the result can be an altered image or a report that is based on image analysis.

There are two types of methods used for image processing, namely analog and digital image processing [16]. Analog image processing can be used for hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in the manipulation of digital images by

using computers. The three general phases that all types of data have to undergo while using digital techniques are pre-processing, enhancement, and information extraction. This project mainly focuses on using a computer to perform digital image processing.

1.2 Image Processing and Its Application

Digital image processing, as a computer-based technology, carries out automatic processing, manipulation and interpretation of such visual information. It plays an increasingly important role in many aspects of our daily life, as well as in a wide variety of disciplines and fields in science and technology, with applications such as television, photography, robotics, remote sensing, medical diagnosis, and industrial inspection [16]. Below are some applications of image processing.

- Computerized photography (e.g., Photoshop) where a before and after example is shown in Figure 1.1

Figure 1.1: An example of image processing in computerized photography [13]

- Space image processing (e.g., Hubble space telescope images, interplanetary probe images)
- Medical/Biological image processing (e.g., interpretation of X-ray images, blood/cellular microscope images)as seen in Figure 1.2.

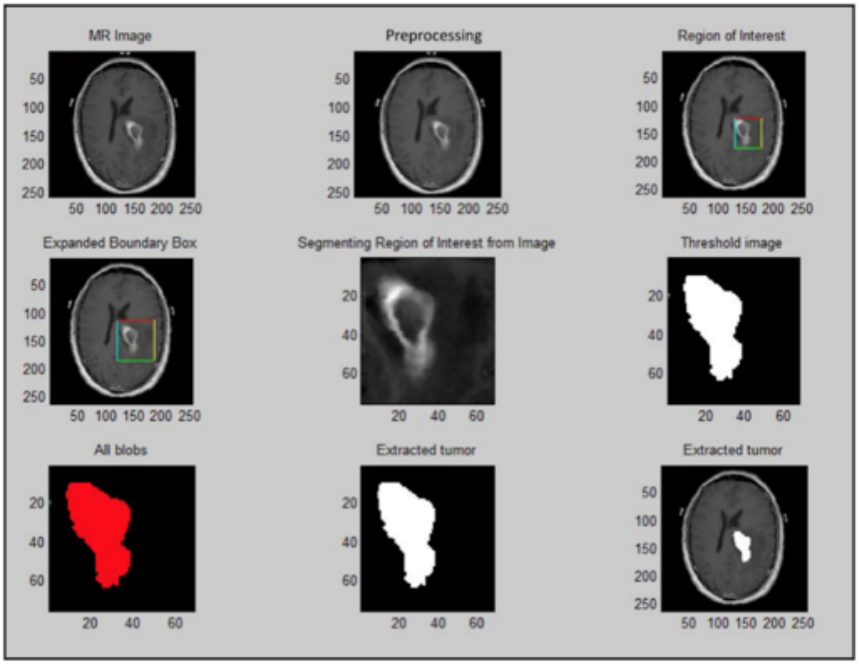


Figure 1.2: Using image processing in detecting tumors [11]

- Automatic character recognition (zip code, license plate recognition)
- Finger print/face/iris recognition
- Remote sensing: aerial and satellite image interpretations
- Industrial applications (e.g., product inspection/sorting)

1.3 Motivation and Outline of Approach

To study the complexity of the image processing field, this project gives a formal definition and applications of image processing and demonstrates its

features and filters of a digital image. Digital image processing is a multi-stage technique that requires many modifications to adapt to the data set's attributes and functionalities. Hence, the second aim of this project is to use Canny's algorithm with image processing steps to detect playing cards on different backgrounds and lighting settings. A program using Canny's algorithm is built to detect playing cards from a given image data set.

Chapter 2

Features and Filters of a Digital Image

2.1 What is a Digital Image

A computer software understands a digital image as a grid (matrix) of intensity values as shown in Figure 2.1. We can think of a (grayscale) image as a function, f(x,y), from R^2 to R (or a 2D signal), such that f(x,y) gives the intensity at position (x,y). An example of f(x,y) can be found in Figure 2.2. A digital image is a discrete (sampled, quantized) version of this function [1].

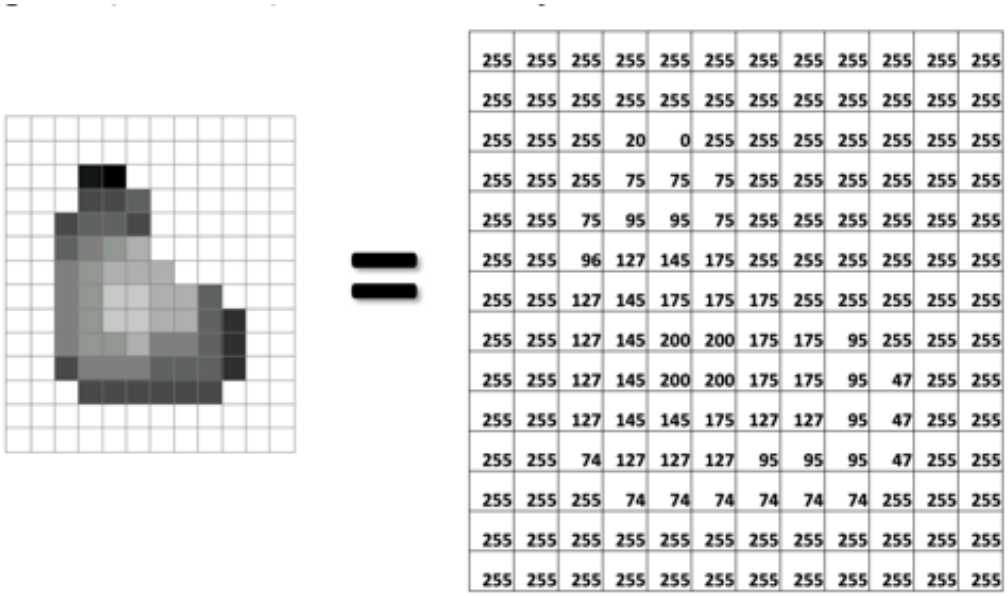
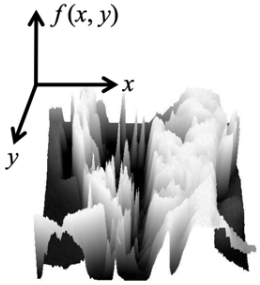


Figure 2.1: A digital image in matrix display [6]

(a) An example of a gray scale image



(b) Grayscale image displaying as an f(x,y) function

Figure 2.2: Displaying a grayscale image as a f(x,y) function [17]

2.2 Edge Detection

The early stages of vision processing identify features in images that are relevant to estimating the structure and properties of objects in a scene. Edges are one of such features. Edges are significant local changes in the image and are important features for analyzing images. Edges typically occur on the boundary between two different regions in an image. Edge detection is frequently the first step in recovering information from images. Due to its importance, edge detection continues to be an active research area.

- **Definition 1.1:** An **edge** in an image is a significant local change in the image intensity, usually associated with a discontinuity in either the image intensity or the first derivative of the image intensity [16].

Here are criteria for an optimal edge detector:

- **Good detection**: the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges).
- **Good localization**: the edges detected must be as close as possible to the true edges.
- **Single response**: the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge.

Before discussing about important considerations in edge detection operators, some terms must be carefully defined:

- **Definition 1.2** An **edge point** is a point in an image with coordinates [i,j] at the location of a significant local intensity change in the image
- **Definition 1.3** An **edge fragment** corresponds to the i and j coordinates of an edge and the edge orientations θ, which maybe the gradient angle
- **Definition 1.4** An **edge detector** is an algorithm that produces a set of edges (edge points or edge fragments) from an image.

- **Definition 1.5** A **contour** is a list of edges or the mathematical curve that models that list of edges.
- **Definition 1.6 Edge linking** is the process of forming an ordered list of edges from an in-ordered list. By convention, edges are ordered by traversal in a clockwise direction.
- **Definition 1.6 Edge following** is the process of searching the (filtered) image to determine contours.

A classical method of edge detection involves the use of operators, a two dimensional filter. An edge in an image occurs when the gradient is greatest. The operator works by identifying these large gradients to find the edges. There is a vast number of operators designed to detect certain types of edges. The operators can be configured to search for vertical, horizontal, or diagonal edges. One major problem with edge detection is when noise is present in images. It is not enough to simply reduce the noise, because the image will be either distorted or blurred. Fortunately, the operator can average enough data to discount localized noisy pixels. However, the change in intensity is not always a step change. The intensity change can also be gradual where the operator then has to be modified for proper edge detection. Consequently, missing true edges, false edge detection, and high computational time are the main problems while operating edge detection.

2.3 Image Gradient

The gradient of an image is the measurement of change in an image function, F(x,y) in x (across 1 column) and y (down 1 row) [16]:

$$\Delta F = \frac{\partial F}{\partial x}\frac{\partial F}{\partial y} \tag{2.1}$$

It provides two pieces of information: the magnitude of the gradient tells us how quickly the image is changing, while the direction of the gradient tells us the direction in which the image is changing most rapidly. The image

gradient is important in boundary detection because images often change most quickly at the boundary between objects.

An image can be considered to be an array of samples of some continuous function of image intensity. By analogy, significant changes in the gray values in an image can be detected by using a discrete approximation to the gradient.

2.4 Convolution

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of 'multiplying together' two arrays of numbers, generally of different sizes, but of the same dimension, to produce a third array of numbers of the same dimension [11]. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

In an image processing context, one of the input arrays is normally just a gray level image. The second array is usually much smaller, and is also two-dimensional (although it may be just a single pixel thick), and is known as the kernel or mask. The figure below shows an example image and kernel that will illustrate convolution.

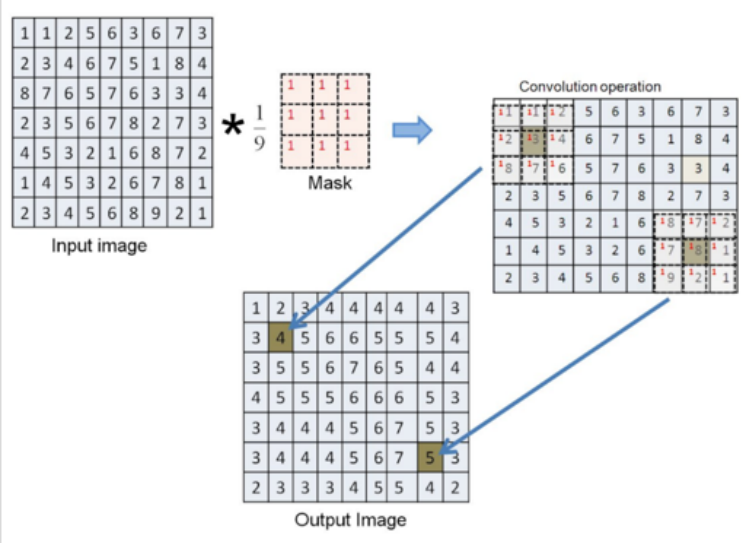


Figure 2.3: Example of convolution given 3*3 mask [11]

The convolution is performed by sliding the kernel over the image, generally starting at the top left corner, so as to move the kernel through all the positions where the kernel fits entirely within the boundaries of the image. Each kernel position corresponds to a single output pixel, the value of which is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adding all these numbers together [16].

2.5 Understanding Image Filters

2.5.1 Gaussian Blur

The Gaussian smoothing operator is a 2-D convolution operator that is used to 'blur' images and remove detail and noise [7].

The Gaussian distribution in 1-D has the form:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-x^2}{2\sigma^2}} \tag{2.2}$$

where σ is the standard deviation of the distribution.

The distribution is assumed to have a mean of zero (i.e. it is centered on the line x=0). The distribution is illustrated in Figure 2.4 below:

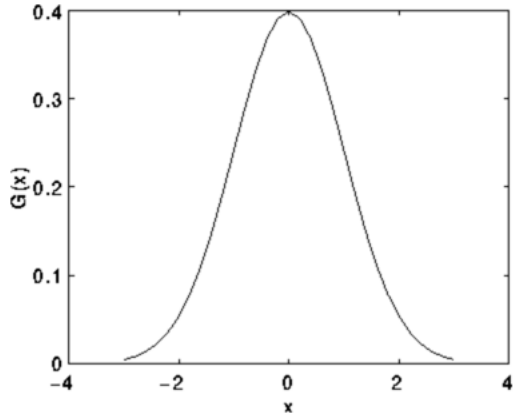


Figure 2.4: 1-D Gaussian distribution with mean 0 and σ=1 [6]

In 2-D, an isotropic (i.e. circularly symmetric) Gaussian distribution has the form:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.3}$$

where σ is the standard deviation of the distribution. The distribution of Gaussian in 2D is illustrated in Figure 6 below:

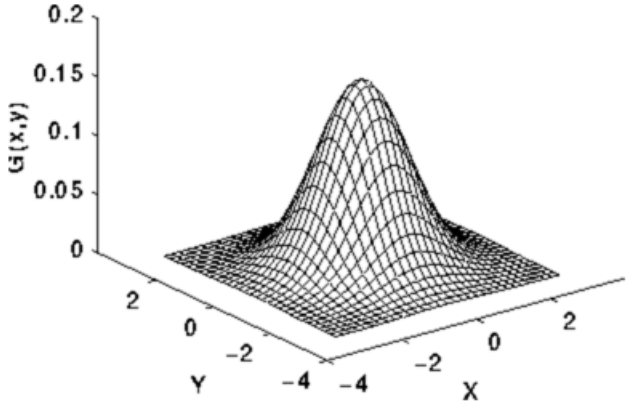


Figure 2.5: 2-D Gaussian distribution with mean 0 and σ=1 [6]

The Gaussian filter works by convoluting the input image with a Gaussian kernel. The convolution with such a kernel performs a weighted average of the current pixels neighborhoods in a way that distant pixels receive lower weight than these at the center. The result of this is a blurry image with better edges than other uniform smoothing algorithms. In theory, the Gaussian function at every point on the image will be non-zero, meaning that the entire image would need to be included in the calculations for each pixel. In practice, when computing a discrete approximation of the Gaussian function, pixels at a distance of more than 3σ have a small enough influence to be considered effectively zero. Thus, contributions from pixels outside that range can be ignored [12]. Gaussians smoothing as a separately filter can be used it as a separate 1D kernel and can process rows and columns separably. The use of weighted average masks makes it better for edge detection than some uniform blurring filters.

Figure shows different result of image by changing σ value.

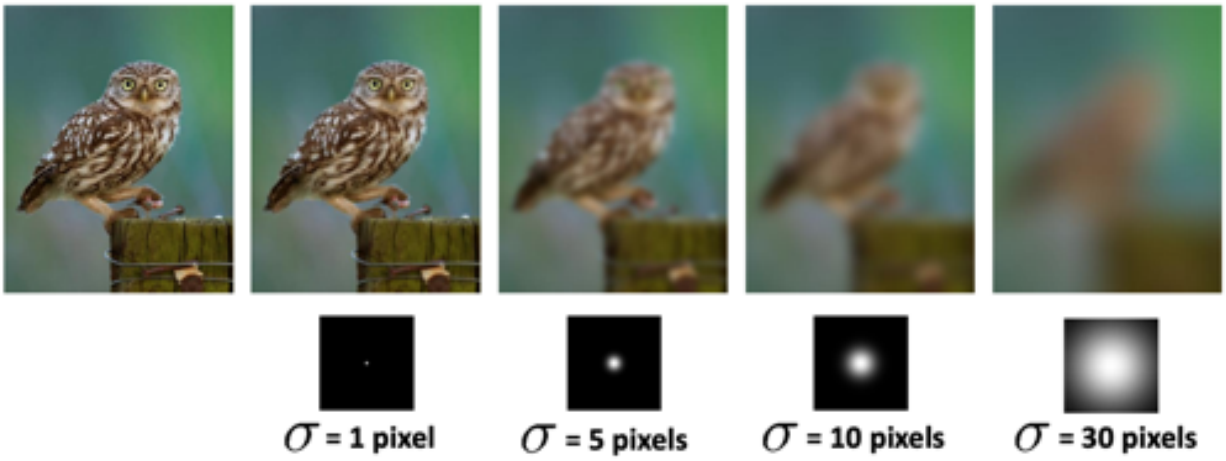


Figure 2.6: Performance of Gaussian filter with different σ value [6]

2.5.2 Image Thresholding

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. Image thresholding is most effective in images with high levels of contrast [11].

If g(x, y) is a thresholded version of f(x, y) at some global threshold T,

$$g(x,y)\begin{cases} = 1 \text{ if } f(x,y) \geq T \\ = 0 \text{ otherwise} \end{cases} \tag{2.4}$$

2.5.3 Hough Transformation

The Hough Transformation is a feature extraction technique used in image analysis, computer vision, and digital image processing [5]. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transformation. The classical Hough transformation is concerned with the identification of lines in the image, but later the Hough transform has been extended to identifying positions of arbitrary shapes, most commonly circles or ellipses.

Presenting a line as a function: $r = x\cos\theta + y\sin\theta$, where r is the distance from the origin to the closest point on the straight line, and θ is the angle between the x axis and the line connecting the origin with that closest point [5]. Figure 2.7 illustrates the f(x,y) function.

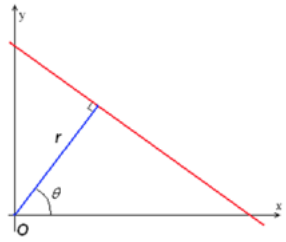


Figure 2.7: Example of line in Hough Transformation σ value [5]

In an image analysis context, the coordinates of the point(s) of edge segments (i.e.(x_i, y_i)) in the image are known and therefore serve as constants in the parametric line equation, while e and θ are the unknown variables. If ploting the possible (r, θ) values defined by each (x_i, y_i), points in cartesian image space map to curves in the polar Hough parameter space. This point-to-curve transformation is the Hough transformation for straight lines. When viewed in Hough parameter space, points which are collinear in the cartesian image space become readily apparent as they yield curves which intersect at a common (r, θ) point [5].

The Hough Tranformation is an elegant method for edge detection and 2D object recognition. Edges in an image don't need to be connected. Hough's method can also complete an object which doesn't need to be clearly visible.

Chapter 3

The Canny Edge Detection Algorithm

3.1 Background

The Canny operator is widely known as the optimal detector, developed by John F. Canny in 1986 [9]. There are multiple steps to implement the Canny operator. First, a Gaussian filter is used to smooth and to remove noise in an image. Second, the gradient magnitude is computed. Third, non-maximum suppression is applied in which the algorithm removes pixels that are not part of an edge. The final step involves the use of hysteresis thresholding along edges. Hysteresis uses two thresholds, upper and lower. If a pixel gradient is higher than the upper threshold, then the pixel is marked as an edge. If a pixel gradient is below the lower threshold, then the pixel is discarded. Finally, if the pixel gradient is between the two thresholds, then only the pixel that is connected to the upper threshold is marked as an edge.

3.2 Pseudo-code

The Canny Edge Detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. There are four main steps in Canny's algorithm [1]:

Step 1. The image is filtered with the derivative of Gaussian function

The Gaussian blur removes some of the noise before further processing the image. The algorithm goes through the image, which is a collection of discrete pixels, to perform the blur.

Pseudo-code for step 1

Compute f_x and f_y as

$$\begin{aligned} f_x &= \frac{\partial}{\partial x}(f * G) = f * \frac{\partial}{\partial x}G = f * G_x, \\ f_y &= \frac{\partial}{\partial y}(f * G) = f * \frac{\partial}{\partial y}G = f * G_y. \end{aligned} \tag{3.1}$$

G(x, y) is the Gaussian function

$G_x(x, y)$ is the derivative of G(x,y) with respect to x: $G_x(x, y) = \frac{-x}{\sigma^2}G(x, y)$

$G_y(x, y)$ is the derivative of G(x,y) with respect to y: $G_y(x, y) = \frac{-y}{\sigma^2}G(x, y)$

Step 2. The magnitude and orientation of the gradient is determined as

$$magn(i, j) = \sqrt{f_x^2 + f_y^2} \tag{3.2}$$

Each pixel in a gray blurred image is known as a (x,y) point in f(x,y) function. Calculating the magnitude and orientation of a gradient tells us how quickly the image is changing and provides the fastest changing direction of the image. The higher the magnitude is, the more likely there is an edge.

Step 3. The non-maximum suppression is applied to thin out the edges from step 2:

In step 2, the size of the real edges has been magnified to separate the noise. Hence, strong edges have a larger value than weak edges. However, this step produces thick edges which would make the algorithm locate the edges incorrectly. In step three, maximum suppression is applied to reduce the size of the thick edges. The actual edges are then easier to be located by the algorithm.

Non maximum suppression works by finding the pixel with the maximum value in an edge. In Figure 3.1, the suppression occurs when pixel q has an intensity that is larger than both p and r where pixels p and r are the pixels in

the gradient direction of q. If this condition is true, the pixels are connected, otherwise the pixels are set to zero (make it a black pixel).

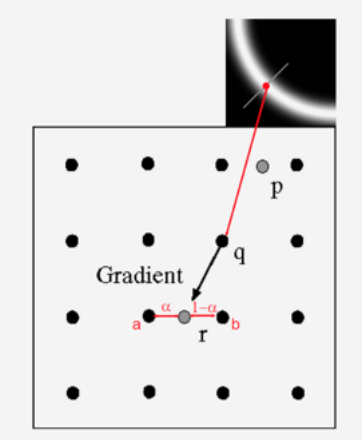


Figure 3.1: Gradient vector q with pixels in its direction [6]

Pseudo-code for step 3

For each pixel's magnitude $I_N(i, j)$ at point (x,y) do:

if magn(i, j) < magn(i_1, j_1) or magn(i, j) < $magn(i_2, j_2)$

then $I_N(i, j) = 0$ // Set the false pixel to black

else $I_N(i, j) = magn(i, j)$

Step 4. Double thresholding and edge linking are applied to fully detect real edges:

The result from non maximum suppression is likely not perfect. Some edges may not actually be edges and there is some noise in the image. To solve this, double thresholding sets two thresholds to classify pixels in the image. As discussed above, pixels with high value are most likely to be edges. Pixels that are larger than the high threshold are considered as strong edges. Pixels that are less than the lower threshold are not edges. All the pixels in between low and high thresholds are weak edges. The algorithm performs edge linking by incorporating the weak pixels that are 8-connected to the strong pixels.

The range of low and high thresholds is based on the type of image. Normally, an RGB image has a range from 0-255. If the image is normalized, the range is from 0-1. Usually, the high threshold (t_h) is double the low

threshold (t_l): $t_h = 2t_l$ [15]. According to Kerry Wong, the simplest way to choose low and high thresholds is to use the mean value of the gray scale image pixel values. As a rule of thumb, the low threshold is approximately to 0.66*[mean value] and the high threshold is double the low threshold [6].

Pseudo-code for step 4

1. Define low and high thresholds.

2. Apply two thresholds on the non maxima-suppressed image, N[i,j], to produce two thresholded images $I_1(i, j)$ and $I_2(i, j)$.

3. Link the edges in $I_2(i, j)$ into contours

    3.1 Look in $I_1(i, j)$ when a gap is found.

    3.2 By examining the 8 neighbors in $I_1(i, j)$, gather edge points from $I_1(i, j)$ until the gap has been bridged to an edge in $I_2(i, j)$. An illustration is shown below:

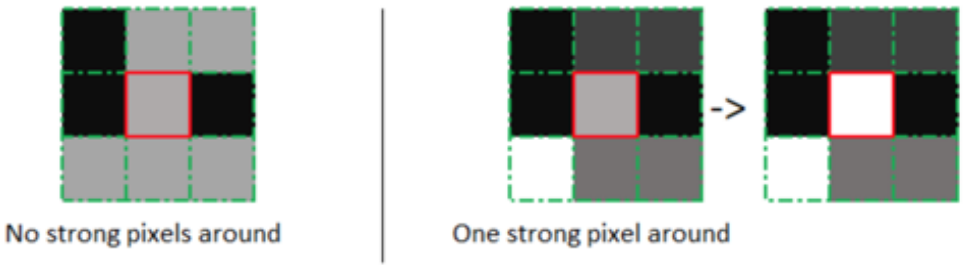


Figure 3.2: Examining the 8 neighbors of a week pixel [15]

Figure 3.3 shows an example of applying double thresholds.

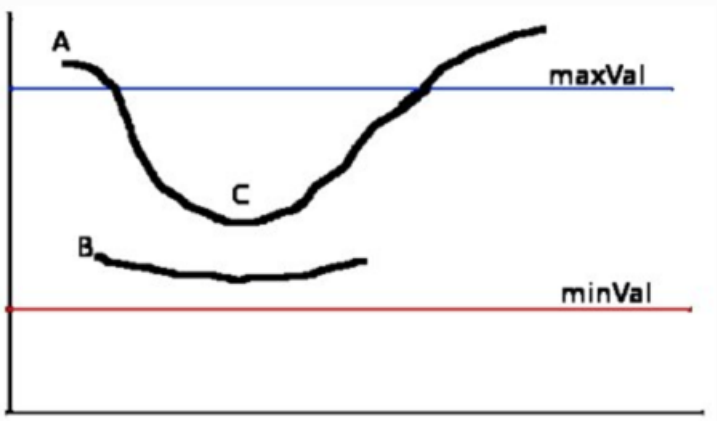


Figure 3.3: Edge A, B, C in double thresholding [1]

The edge A is above the maxVal, so it is considered as a sure-edge. Although edge C is below maxVal, it is connected to edge A, so that it is also considered as a valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, is not connected to any sure-edge, so that is discarded. The minVal and the maxVal, as known as low and high thresholds, should be carefully selected to get the correct result.

3.2.1 A Summary of Canny Method

The Canny Edge Detector is probably the most widely used edge detector in computer vision. Canny has shown that the first derivative of the Gaussian function closely approximates the operator that optimizes the product of signal-to-noise ratio and localization. His analysis is based on ”step-edges” corrupted by ”additive Gaussian noise.” With different image data sets, the process using the Canny Edge Detection is modified to adapt to the data set’s attributes and functionalities.

Chapter 4

Case study: Playing Card Detection with Canny's Method

4.1 Dataset General Information

In order to test the efficiency of Canny's Algorithm, an image data set was created by taking photos of different combinations of cards in a set of 52 playing cards. Each of the cards has a different design and texture, which can be a good candidate to test Canny's Algorithm. There are a total of 10 photos which were taken by the same phone under different settings for angles, lightings, and backgrounds. The combinations of playing cards were selected randomly in a set from 1 to 10 in order to increase the difficulty level of the problem. Here are some examples of photos that were taken:

Figure 4.1: Photo was taken under low light condition with a clear background

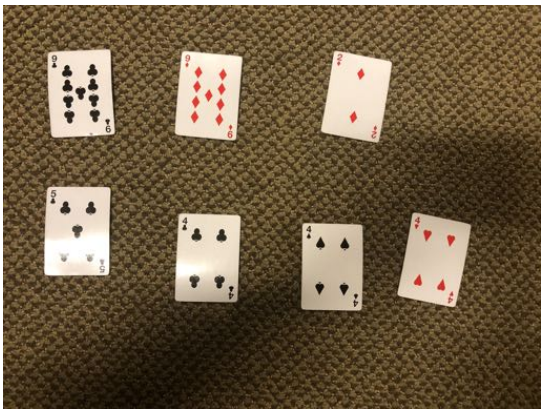


Figure 4.2: Photo was taken under good light condition with textured background

Figure 4.3: Photo was taken under good light condition with white background

4.2 Cleaning Image Dataset

Purpose:

Perform different image operations to clean as much noise as possible before using Canny method.

Language and Program Library:

Written in Python 3.6, this program uses Open CV library and its built-in image processing attributes. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.

A visualization of image processing step:

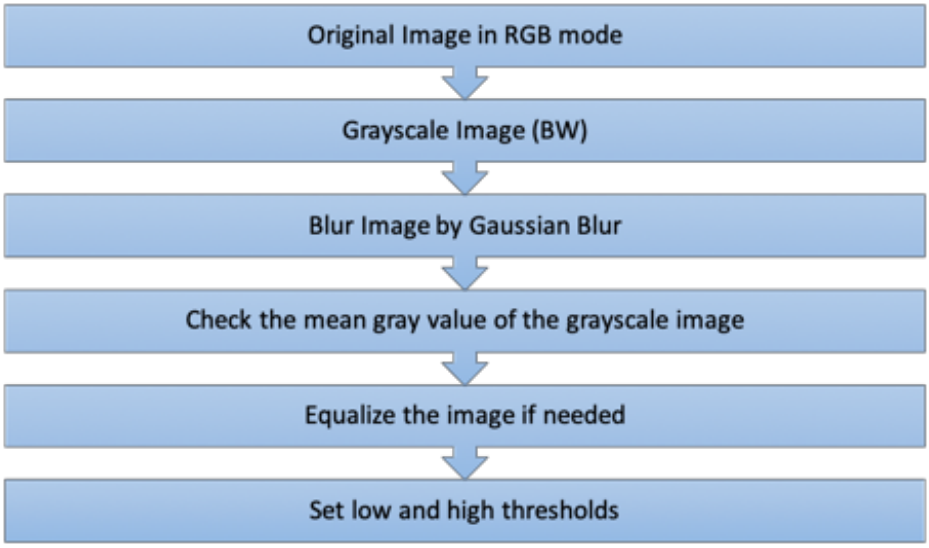


Figure 4.4: Image processing operations

The original image is in a 3-D space, which makes the image processing more complicated. We convert the image into grayscale, which is a 2-D space in order to apply further processing steps and to reduce the complexity of the problem.

As discussed in step 4 of Cannny's Algorithm, section 3.2, selecting low and high thresholds is crucial. A white background photo has a higher gray mean value than a dark background photo since the pixel value of light colors is higher than the pixel value of darker colors [10]. In Figure 4.5 and 4.6 below,

the histogram of each gray image is shown to see the distribution of dark and light pixels. In the range from 0 to 255, dark pixels are in between 0 - 50, and light pixels are in between 150 - 255 [4].

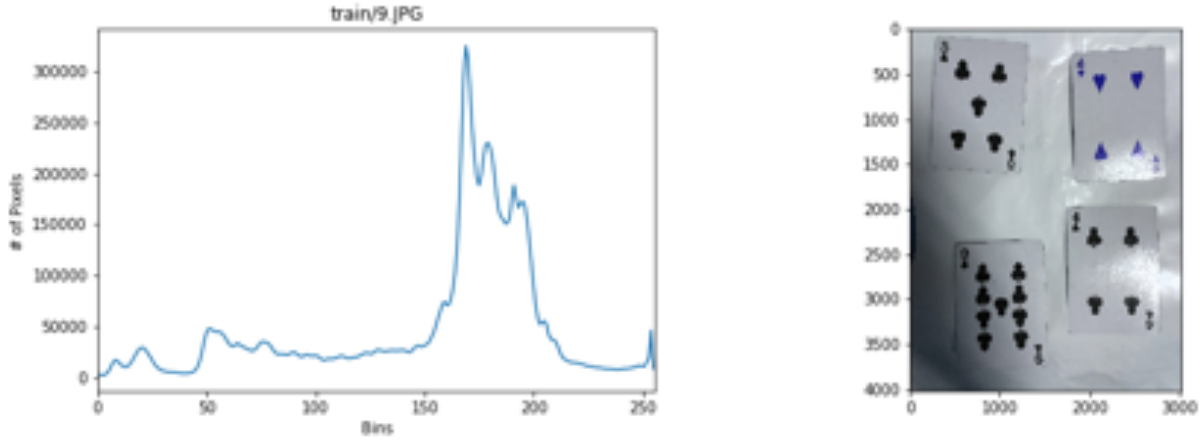


Figure 4.5: Histogram of a white background photo

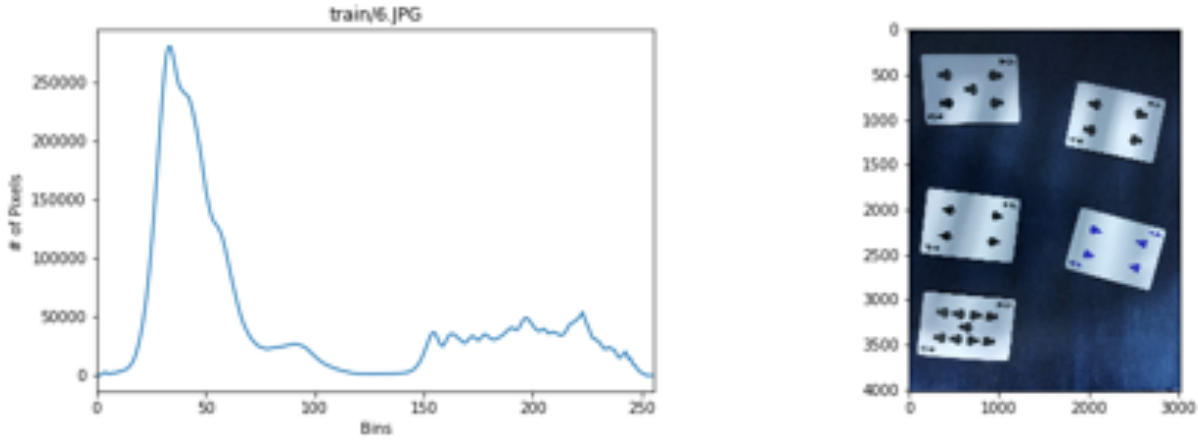


Figure 4.6: Histogram of a black background photo

After categorizing images by their background colors, the program sets low and high thresholds differently for each group. With the group of the

white/color background, the program only takes low threshold equals to approximately 35% of the mean value of the gray image [14]. Since the mean gray value of white/color background is extremely large, taking approximately 60% of the mean gray value greatly affect the result of Canny Edge Detector.

4.3 Card Detection with Canny Method

Use Canny's method to detect card in each image, then draw a square around object detected.

A visualization of card detection process:

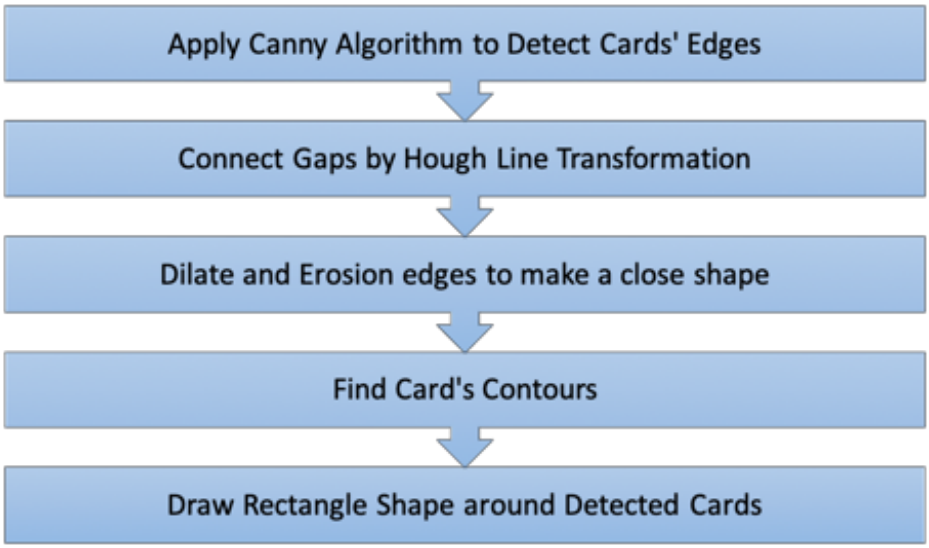


Figure 4.7: Card detection process with Canny method

Canny's is used to detect edges, as well as increase contrast and remove image noise. HoughLines uses the Hough Transformation to determine whether those edges are lines or not. Pixels that are close as a line after process with Canny Algorithm then determine and connect by Hough Line Transformation.

After applying Hough Line Transformation, the program dilates edges to connect the gap between lines by enlarging the size of lines across horizontal and vertical axes. Dilation connects and then possibly creates the intersection

between lines. A closed shape of the card is formed, which is helpful for finding card contour in the next step. The side benefit of dilation is the creation of false localization of the object. Hence, erosion is used to make the dilated edges thinner.

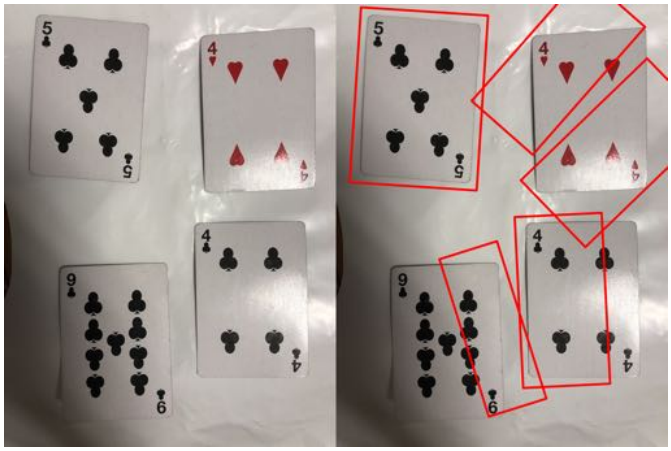
To detect cards on the photo, the program uses findcontour function in OpenCV to detect a curve joining all the continuous points (along the boundary), having same color or intensity [2]. The contours are a useful tool for shape analysis and object detection and recognition. drawContour function is used to draw a bounding rectangle around each card. Bounding rectangle is drawn with minimum area, so it considers the rotation also. The function used is cv2.minAreaRect(). It returns a Box2D structure which contains following details: top-left corner(x,y), (width, height), angle of rotation. The function cv2.boxPoints() then gets 4 corners of the rectangle [2].

4.4 Result and Analysis

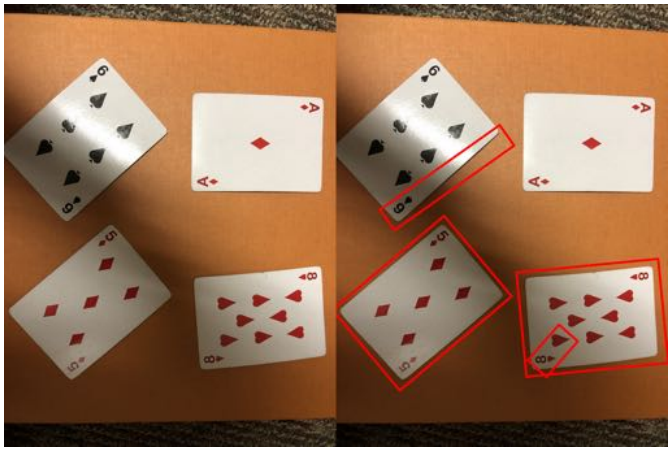
The performance of the program is based on three main criteria of a good optimal detection method, which is already mentioned in section 2.2: minimal probability of false positives (edges caused by noise) and false negatives (true edges), good localization, and single response.

4.4.1 Photos with White/Color Background

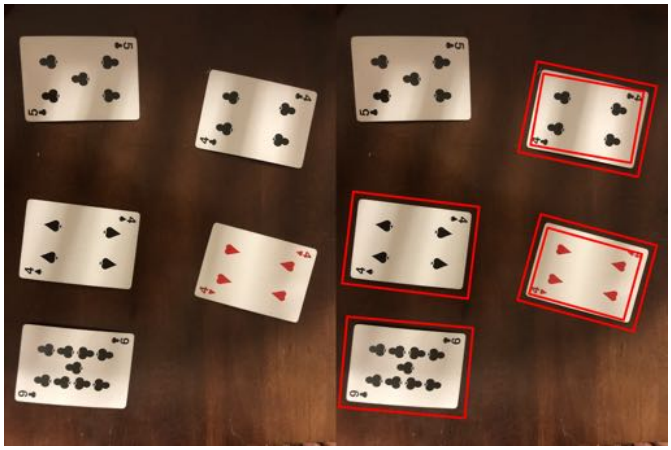
Figures below are the final result of the program. Predicted cards are predicted are marked by red line.



(a) Shiny white background with shadow light



(b) Color background with shadows



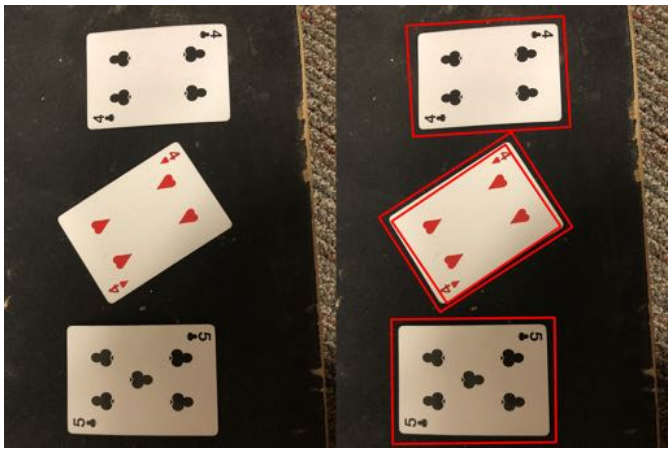
(c) Color background with low light

Figure 4.8: Program results on white/color background

In a white/color background, the program performance is not really good. It cannot predict all the cards in one photo. This can be explained by the use of Contours function in OpenCV. The finding contours are like finding white objects from black background [2]. Since the playing card is a mainly white object, it's hard to find the contour of white objects on a white background. As seen in these photos, there are uneven light and shadows on the background. For example, in the photo (b) of Figure 4.8, the program cannot fully predict the Six of Spades card since there is a light flare on top of the playing card. The program also cannot predict the Ace of Diamond. This can be explained by the uneven light in the image. The program sets low and high thresholds based on the mean gray value of a corresponding image. Hence, cards in the low light area should have different low and high thresholds compared to cards in a better lighting area. In the future, the program can divide the image into different lighting area and set low and high thresholds separately.

Additionally, the Contour function draws some small rectangle inside the card. This problem can be fixed by modifying the hierarchy known as the parent-child relationship in Contours function [3]. The program detector returns a clean rectangle around the detected card. However, there are still gaps between the card's edge and the bounding rectangle. This results from the dilation process. Even when the program erodes the dilated image, it still needs to keep the connected line. Hence, it's hard to make the dilated line as thin as the original edge.

4.4.2 Photos with dark background



(a) Mixed dark background with normal light

(b) Dark background with low light

(c) Textured background with normal light and many cards

Figure 4.9: Program results on dark background

Figure 4.9 above shows the program results on dark background photos.

In a dark background, the program performs better than in the white/color background. It predicts most of the cards appearing in the photo, even in low light condition. The program detects really well in photos that have many playing cards. This can be explained by the quality of the image. The photos that are taken closer to the playing cards has better quality than the photos which are taken further for the playing card. The program recognizes the card's elements, which can be edge noises in the future step. Additionally, in a low-quality image, the double thresholding and edge linking performs better since the algorithm can easy distinguish the high and low value of pixels. Dark background also reduces the uneven lighting in photo.

Chapter 5

Conclusions and Future Work

In conclusion, image processing is an extremely important step in object detection. The sequence of image operations is varied from data sets to data sets, so the deep understanding of each filter and algorithm is necessary. This paper introduces basic concepts in image processing with multiple image features and Canny Edge Detector. A program is built to test the performance of Canny Edge Detector and image processing operations by detecting playing cards on different backgrounds and lighting settings.

The program is good at detecting playing card on dark and light texture background. In the future, it can apply to similar cases which want to detect rectangular card. Additionally, the program can crop detected card in the image to generate a dataset for machine learning model. The program performance on white background and uneven lighting condition should definitely be improved. Possible improvements include the following:

- Split image into different lighting area to select separate low and high thresholds for each region.
- Try new Edge Detector method, for example, Sobel Detector.
- Perform further investigation to analyze image properties and classify them by group.

From recognizing faces to processing the live action of a football game, computer vision rivals and surpasses human visual abilities in many areas.

This paper is a good source for any beginner in computer vision. It provides fundamental concepts of image operation for edge and object detection. The program is a starting point for any object detection project with a sequence of simple operations.

Appendix A

Program Results

The following pages show card detection results for 10 photos.

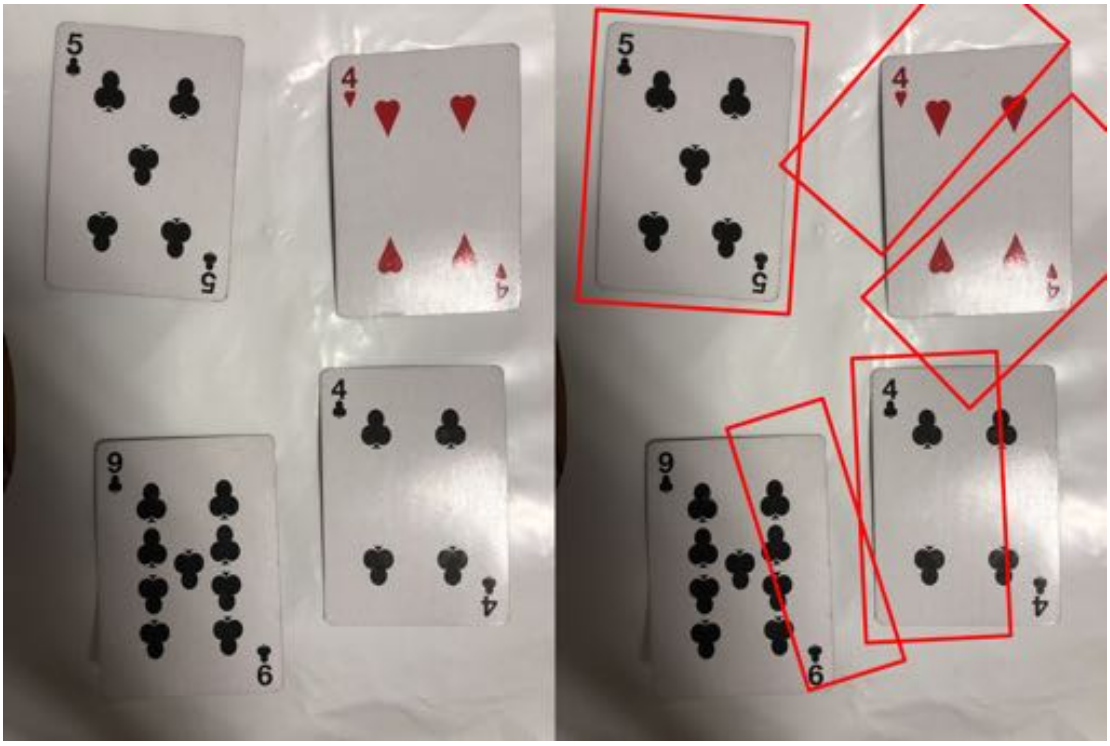


Figure A.1: Photo with white and glossy background

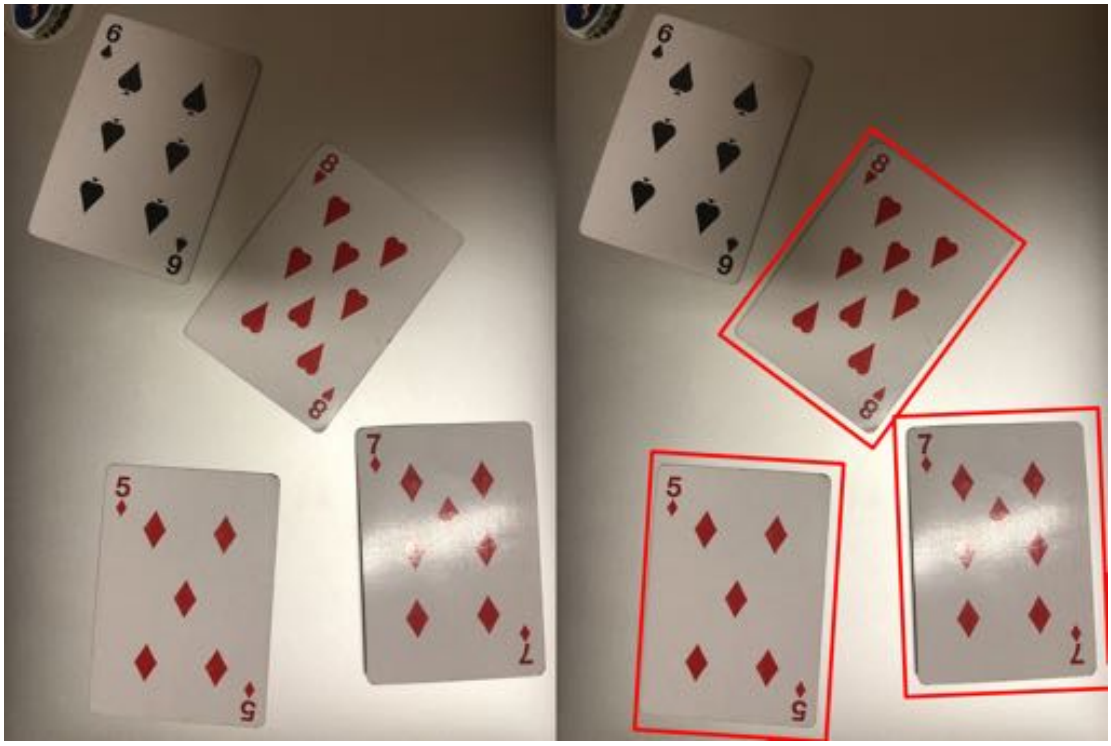


Figure A.2: Photo with grey and shadow background

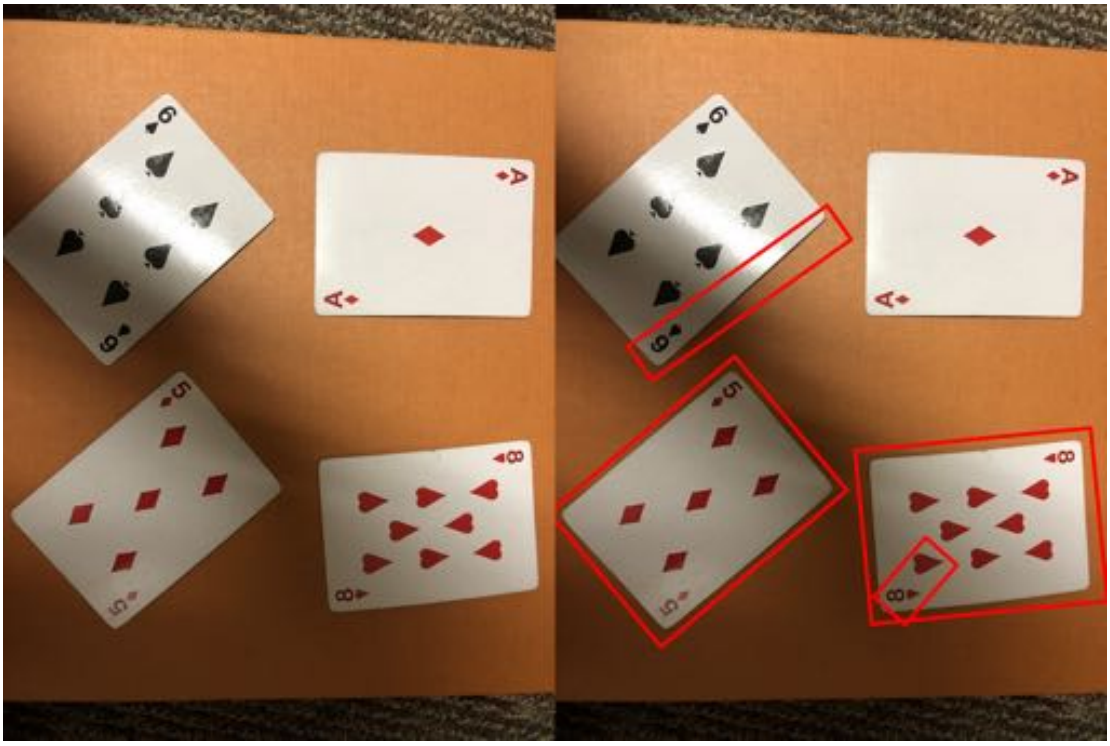


Figure A.3: Photo with color and shadow background

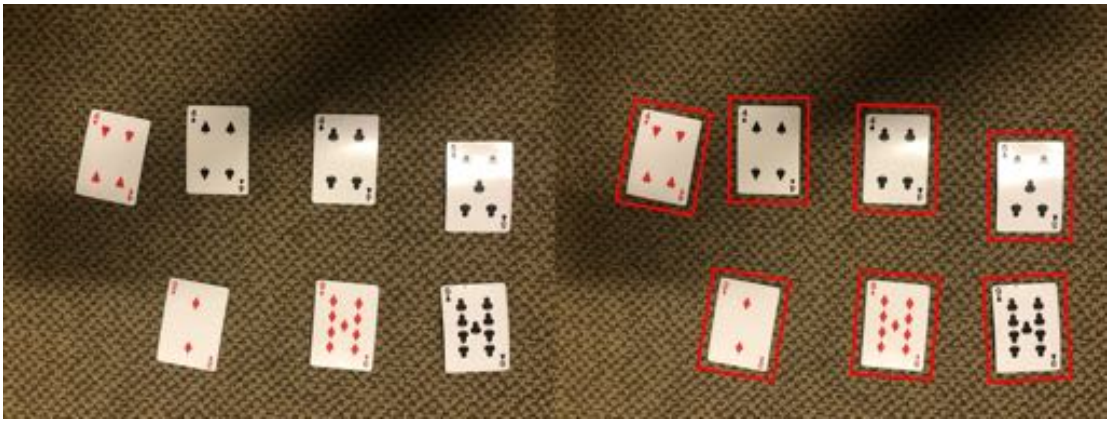


Figure A.4: Photo with textured and shadow background

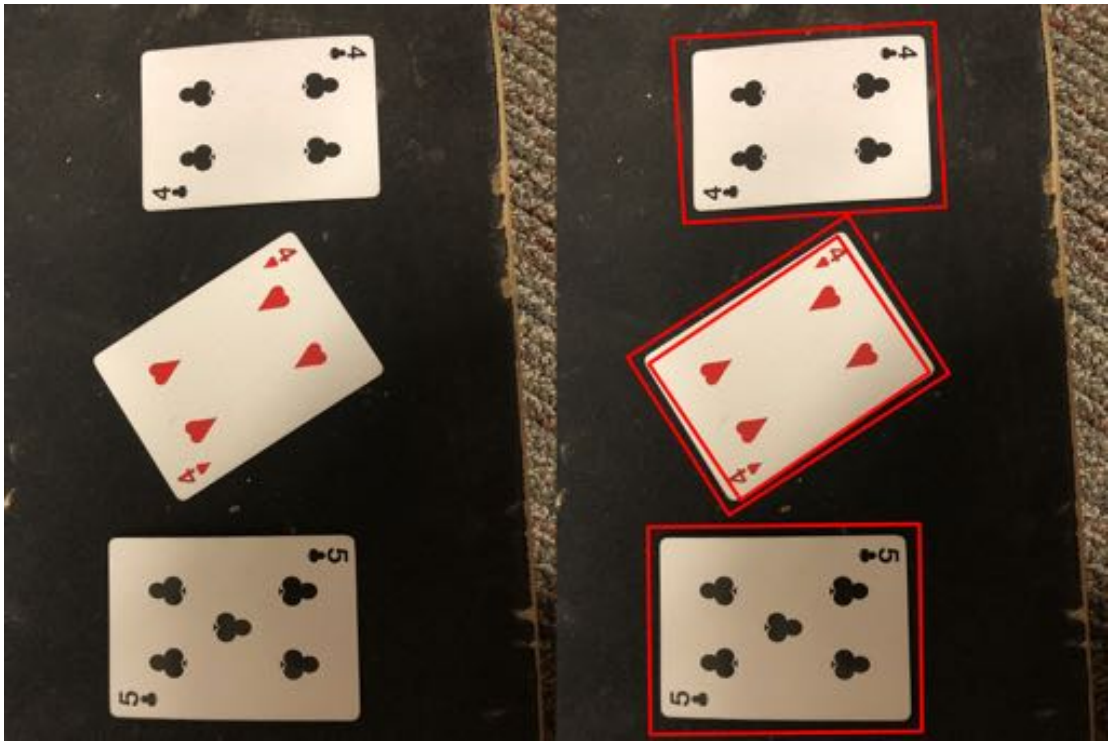


Figure A.5: Photo with mixed black background and taken in a close angle

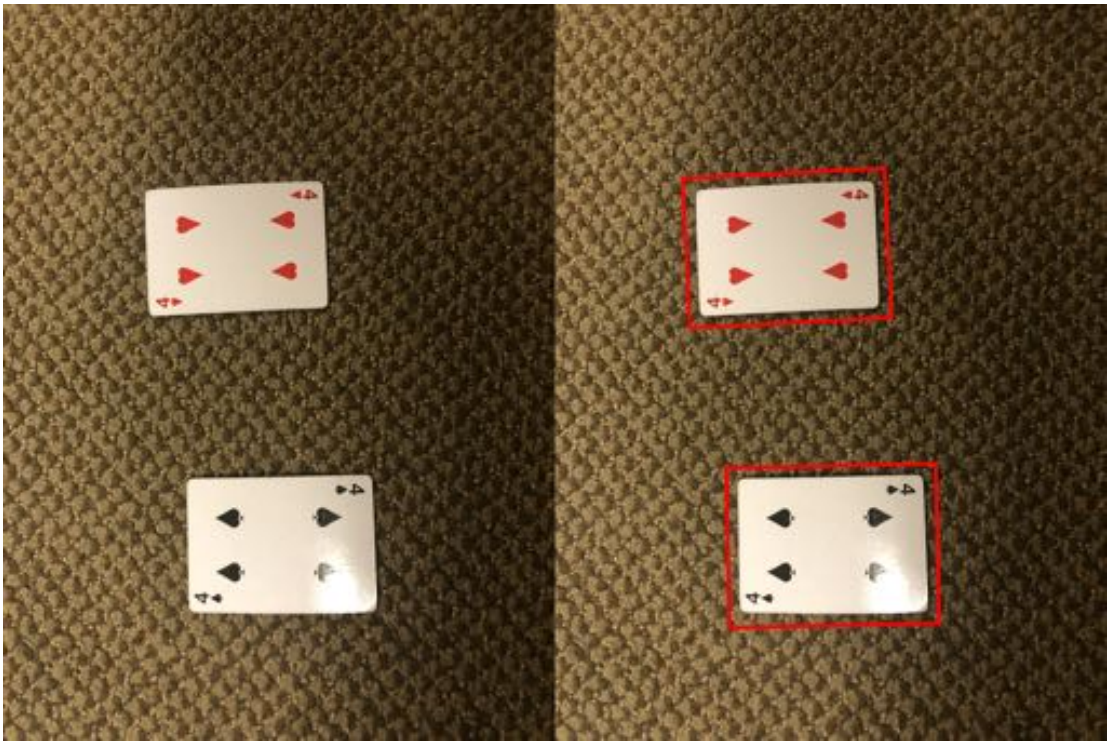


Figure A.6: Photo with textured background and taken in a further angle

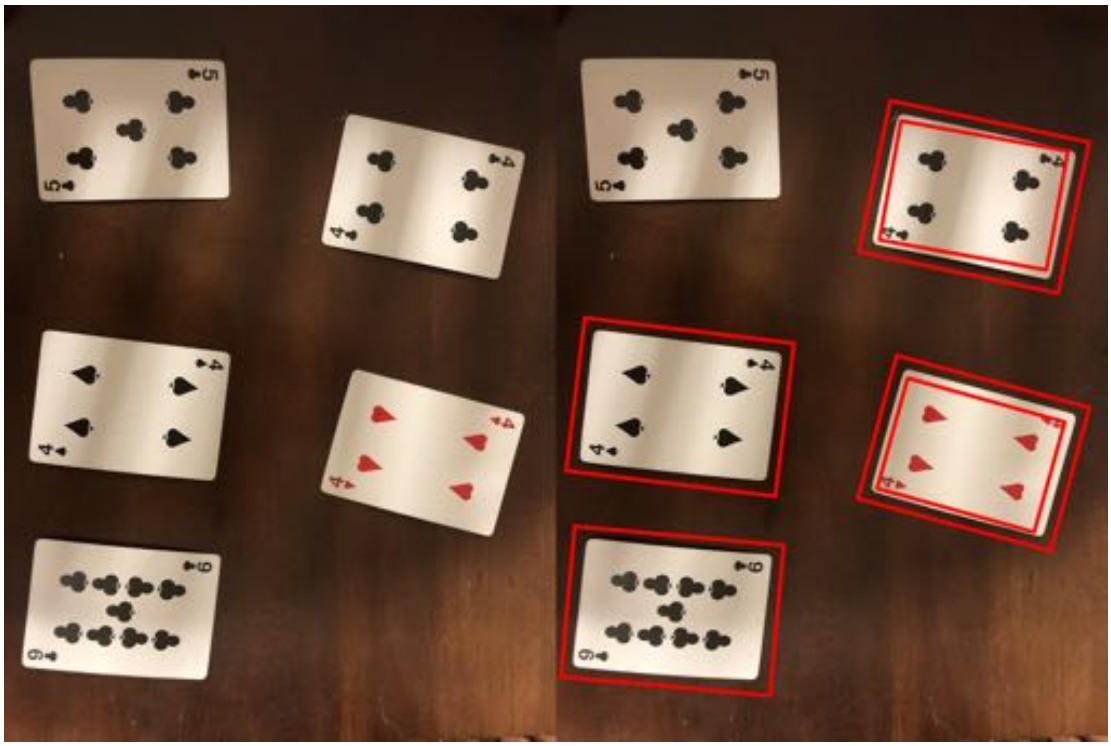


Figure A.7: Photo with color background and taken in a close angle

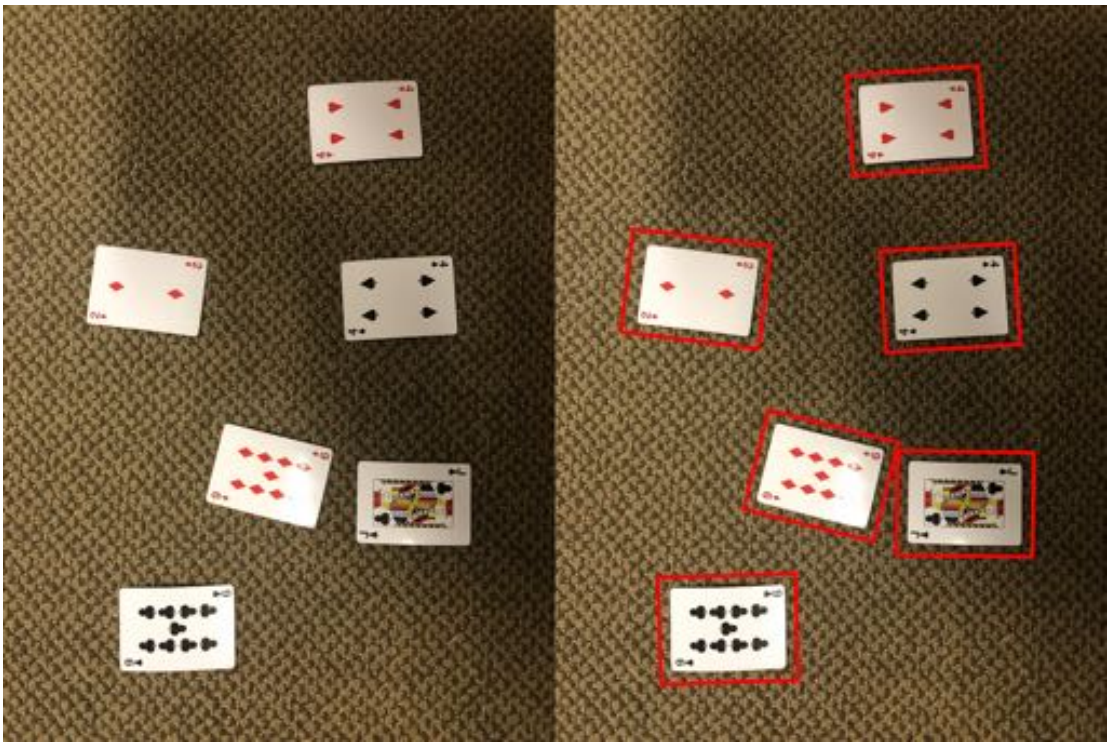


Figure A.8: Photo with textured background has many playing cards and is taken in a further angle

Figure A.9: Photo with textured and shadow background has many playing cards and is taken in a further angle

Bibliography

[1] Canny Edge Detection OpenCV-Python Tutorials 1 documentation.

[2] Contour Features OpenCV-Python Tutorials 1 documentation.

[3] Contours Hierarchy OpenCV-Python Tutorials 1 documentation.

[4] Histograms - 2: Histogram Equalization OpenCV 3.0.0-dev documentation.

[5] Hough Line Transform OpenCV 2.4.13.7 documentation.

[6] Kerry D. Wong Blog Archive Canny Edge Detection Auto Thresholding.

[7] Spatial Filters - Gaussian Smoothing.

[8] Gaussian Blur - Noise Reduction Filter, August 2017.

[9] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8:679–698, 1986.

[10] zong fan. Detect Face in Bad Lighting Condition Using Color Histograms, March 2018.

[11] Saurabh Kapur. *Computer Vision with Python 3*. Packt Publishing Ltd, August 2017.

[12] Justin Lang. Canny Edge Detector.

[13] Phillip Glombik. How to Process Photos in Photoshop - Example: The Canals of Venice I | Photoshop Tutorial.

[14] Muhammad Rizwan, Md Kafiul Islam, and Hafiz Adnan Habib. Local Enhancement for Robust Face Detection in Poor SNR Images. 2009.

[15] Sofiane Sahir. Canny Edge Detection Step by Step in Python Computer Vision, January 2019.

[16] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Cengage Learning, January 2014. Google-Books-ID: DcETCgAAQBAJ.

[17] Drexel University. Canny Tutorial.