Songzhe Zhu(sz2682)
Xuelin Yu(xy1184)
Yiming Zhang(yz5293)

CS-GY-6643
Final Project Report

Abstract
In this project, we are required to propose a computer vision problem then develop a program to solve it. Our program is a mix of several applications of different methodologies mainly based on edge-detection techniques. Using our program, people can rotate images (especially images of documents) to the correct orientation automatically without losing any information. The two principal algorithms we adopted and implemented were Hough transform and Canny edge detection. We also provide a user interaction function that users can adjust parameters until obtaining satisfied result with no need to re-run the program over and over again.

Why correction of image orientation?
The first idea that came to my mind was image orientation correction. It was derived from my own experience. My parents were going to apply visas to the US, but they don't understand English, so they needed me to translate some documents for them. Basically, they took photos of documents and send them to me. I opened those photos on my laptop and found that unlike on mobile phones, it was slightly difficult to rotate my monitor to get the correct orientation of an image. So, I decided to design a program for correction of image orientation as the topic of our final project and my teammates accepted the idea readily. The implementation can be widely used to correct scanned pages, photos of documents, providing easier reading. It will bring great user experience if combined with some picture preview or display software.

Why Hough Transformation and Canny Edge Detection?
Since we focus on image of documents, the most obvious feature of documents is that characters are arranged line by line. Characters in the same row can be considered as disperse points on the same line, and those lines can be detected by using Hough transform.
To apply Hough transform, we need to distinguish characters as feature points from background. In this semester, on the area of edge detection, we implemented Gaussian Laplacian operator and zero-crossings of LoG for edge detection, leaving canny edge detection on only theoretical level. So, this time we decide to take this chance and implement canny edge detection.

How to execute the program:
- Step 1: Type the file name you want use. (Directly type the name on prompt)
- Step 2: Pick clockwise rotation or counter-clockwise rotation.
- Step 3: Choose the threshold you want use for Hough Transform.
    - High threshold makes harder to detect lines.

    - Low threshold slows the program down.
    - Ideal threshold gives ideal result.
    - Ideal range: 180 -220.
- Step 4: Program will show the result out. Base on your satisfaction, choose next step:
    - If you are satisfied with the result, go to step 6.
    - Otherwise, go to step 5.
- Step 5: You can choose another threshold to generate a new result here.
- Step 6: You can choose another image here if you want. Otherwise, type 'q' to quit.

Important functions:

- **Canny(image):**
    - Parameter:
        - *image:* the source image from your choice.
    - This function is used to implement the canny edge detector. For more detail, please look at **Canny edge detector** part.
- **rotateImage(src, degree):**
    - Parameter:
        - *src:* the source image from your choice.
        - *degree:* the rotation degree.
    - This function is used to rotate the image based on the degree we calculated from **CalcDegree()**.
- **houghLines(img, threshold):**
    - Parameter:
        - *img:* the image after applying canny edge detector.
        - *threshold:* the threshold from your choice.
    - This function is used to implement the Hough Transform. For more detail, please look at **houghTransform** part.
- **CalcDegree(srcImage, th, clock):**
    - Parameter:
        - src*Image:* the source image from your choice.
        - *th:* the threshold read in from your choice.
        - *clock:* clockwise or counter-clockwise rotation based on your choice.
    - This function is used to calculate the rotation degree based on the result from Hough Transform.

Canny edge detector

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. And the process of Canny edge detection algorithm is shown as below:

1. Apply Gaussian filter to smooth the image in order to remove the noise

- Since all edge detection results are easily affected by image noise, it is essential to filter out the noise to prevent false detection caused by noise. Therefore, Gaussian filter is applied to convolve with the image for smoothness.

2. Find the intensity gradient of the image

- An edge in an image may point in a variety of directions, so the it is essential to use filters to detect horizontal, vertical and diagonal edges in the blurred image. Edge detection operators like Roberts, Prewitt, or Sobel will return a value for the first derivative in the horizontal direction (Gx) and the vertical direction (Gy). From this the edge gradient and direction can be determined with the formula shown as below:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$
$$\theta = tan^{-1}(G_y/G_x)$$

(Please zoom up to see details)

3. Apply non-maximum suppression to get rid of spurious response to edge detection

- Non-maximum suppression is an edge thinning technique. It is applied to find "the largest/strongest" edge. After applying gradient calculation, the edge extracted from the gradient value is still quite blurred. But there should only be one accurate response to the edge. Thus, using non-maximum suppression to help to suppress all the gradient values except the local maxima will meet the requirements. The algorithm for each pixel in the gradient image is to decide the angle in which angle region (0, 45, 90 and 135) and then compare if its magnitude is not greater than the magnitude of the two neighbors in the gradient direction.

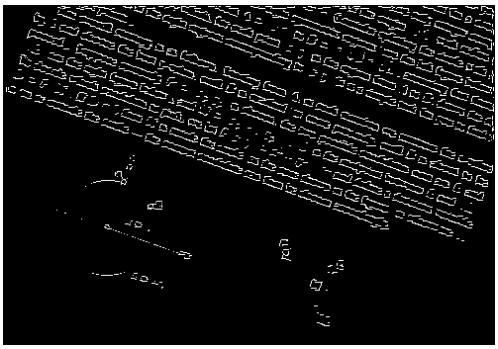
(Also please zoom up to see the details, the lines are thinner than previous image)

4. Apply double threshold to determine potential edges

- After application of non-maximum suppression, remaining edge pixels provide a more accurate representation of real edges in an image. However, some edge pixels remain that are caused by noise and color variation. Therefore, it is essential to filter out edge pixels with a weak gradient value and preserve edge pixels with a high gradient value. And we use high and low threshold values to filter out unnecessary pixels. The algorithm is that if an edge pixel's gradient value is higher than the high threshold value, it is marked as a strong edge pixel and be kept; if an edge pixel's gradient value is smaller than the high threshold value and larger than the low threshold value, it is marked as a weak edge pixel; if an edge pixel's value is smaller than the low threshold value, it will be suppressed. The two threshold values are determined in our as 0.3*max value and 0.1*max value.

5. Track edge by Hysteresis thresholding

- So far, the strong edge pixels should certainly be involved in the final edge image. But for weak edge pixels there are still some, as these pixels can either be extracted from the true edge, or the noise/color variations. If use a high threshold to start edge curves and a low threshold to continue them will produce a good result and keep the true edges.



houghTransform

Hough transform is a very powerful method for feature extraction. In our program, only the part of line detection is implemented.

In the rectangular coordinate system, a line can be represented by a constant parameter pair (θ, ρ) that $\rho = x\cos(\theta) + y\sin(\theta)$, where θ and ρ are the angle and distance of the normal from the original point to the line respectively.

In the Hough space, we consider the relationship between θ and ρ, thus each point in rectangular coordinate system projected to a cosine curve in the Hough space. The coordinate (θ_0, ρ_0) of an intersection of two curves gives equation of the line $\rho_0 = x\cos(\theta_0) + y\sin(\theta_0)$ in the rectangular coordinate system. Points on the same line will create the same intersection (θ_i, ρ_i). So we can get lines in image by looking for the intersections in Hough Space that most curves pass through.

In the program, we structured a 2-dimonsion array $H(\theta, \rho)$ as a counter to count the appearance of each (θ, ρ) pairs. For each feature point in the image, we computed ρ_i for each θ_i from 0 to 180, then counted $H(\theta_i, \rho_i) = H(\theta_i, \rho_i) + 1$.

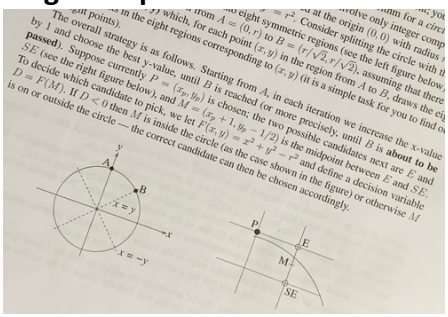
We extracted all (θ_i, ρ_i) pairs if $H(\theta_i, \rho_i)$ was no less than the given threshold. Then transferred each (θ_i, ρ_i) back to the rectangular coordinate system to show the lines on the original image.

Parameter θ is important in our program. After finding the character lines on paper, we also got the rotation angle which is exactly the parameter θ. In some cases more than 1 lines were found, then average of angle would be given to the later rotation step.

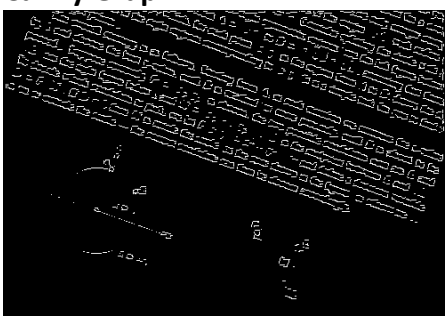
Demo Results:

Demo 1:

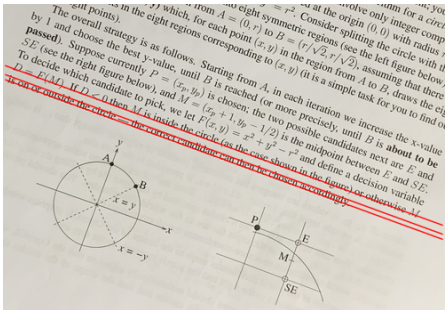
- **File:** resize2.png.
- **Clockwise vs counter-clockwise:** counter-clockwise.
- **Threshold:** 200.
- **Rotate angle calculation:** 20.000000000000014
- **Origin Graph:**



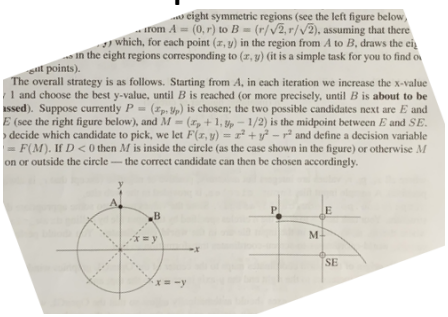
- **Canny Graph:**



- **Edge Graph:**

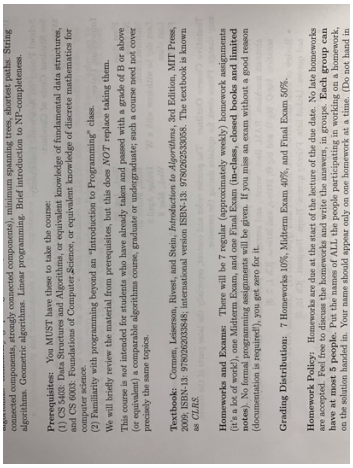


- **Rotation Graph:**

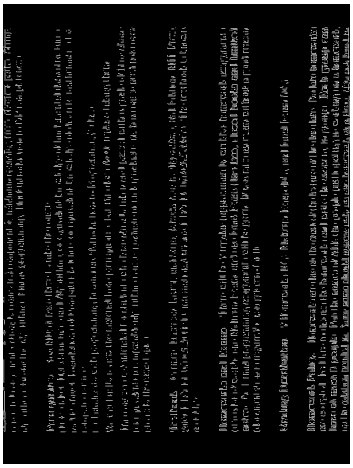


Demo 2:

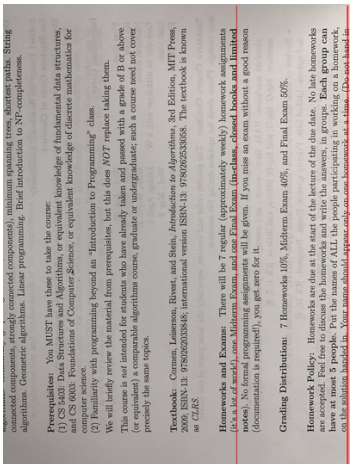
- **File:** resize4.png.
- **Clockwise vs counter-clockwise:** clockwise.
- **Threshold:** 220.
- **Rotate angle calculation:** 270.0
- **Origin Graph:**



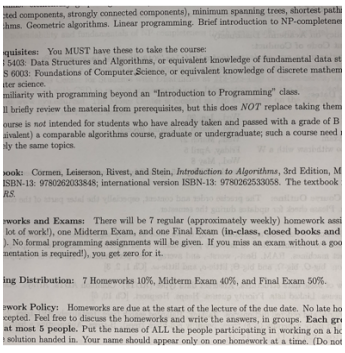
- **Canny Graph:**



- **Edge Graph:**

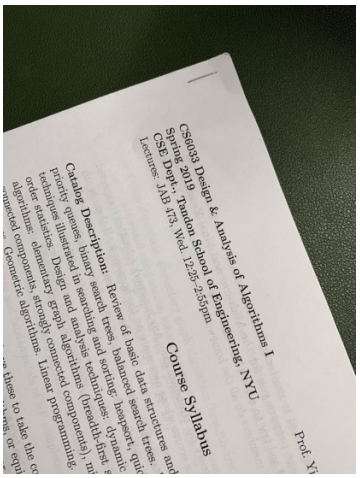


- **Rotation Graph:**

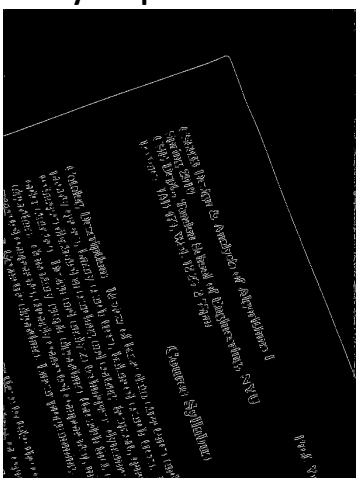


Demo 3: (Fail case)

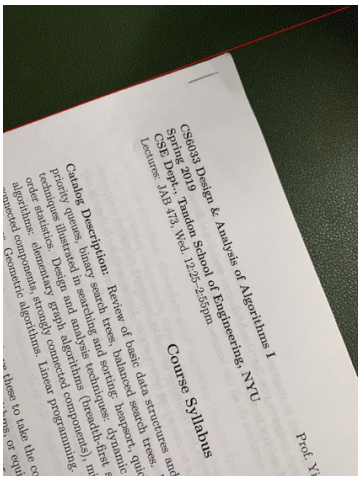
- **File:** resize3.png.
- **Clockwise vs counter-clockwise:** counter-clockwise.
- **Threshold:** 195.
- **Rotate angle calculation:** -20.0
- **Origin Graph:**



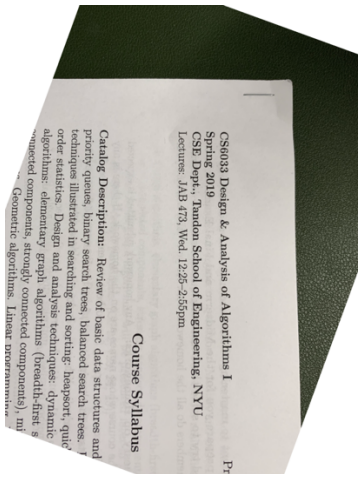
- **Canny Graph:**



- **Edge Graph:**



- **Rotation Graph:**



Demo result analysis:

- Demo 1 is a general case, as the graph is rotated by 20 degrees. As shown by the rotate graph of demo 1, our program works well in this case.
- Demo 2 is a special case, as the graph is rotated by 90 degrees. In this case, we picked clockwise rotation in order to make it rotate back to general style. As shown by the rotate graph of demo 2, our program works well in this case.
- Demo 3 is a special case, as the page edge are perpendicular to each other. In this case, our program fails because it cannot make the correct solution between two ways. This is a refinement case in the future.

Team work

Songzhe Zhu: System design and implementation, UI design and implementation.
Xuelin Yu: Hough Transformation implementation.
Yiming Zhang: Canny edge detector implementation.

Future Implementation

Although our program is limited to series of useful function (such as dealing with noise, optimization of running time, extracting lines in same direction, multiply parameter control and user-friendly IU etc.), the implementation of automatically correction of image orientation has extensive range of usage. Image display and preview software can adopt this function to provide better reading experience when user read scanned paper or photo of documents. Social media application can also use it to help people automatically correct their photo before post.