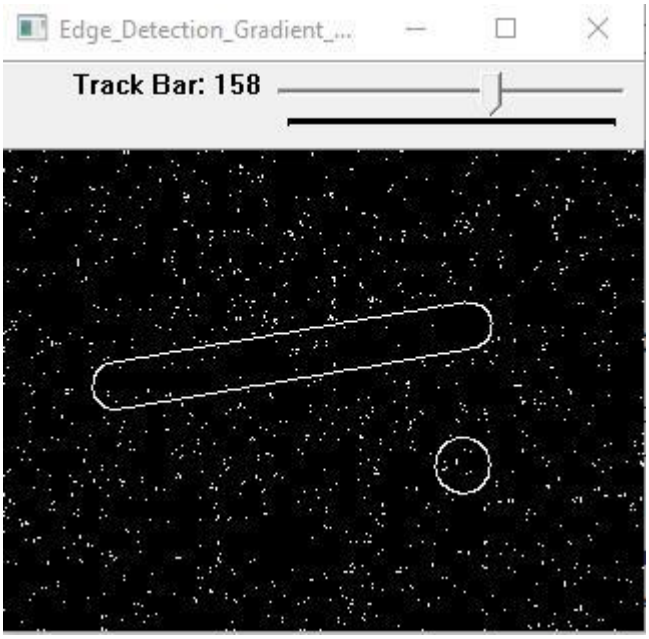
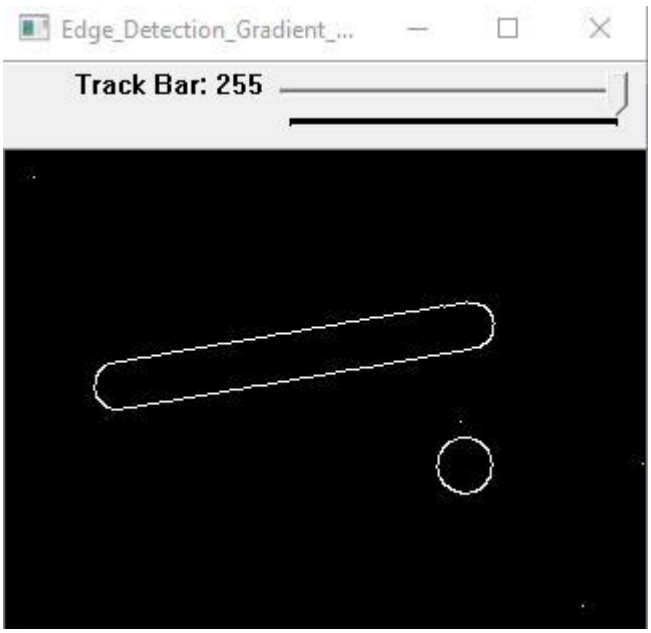
/*----------
Student's Name: Yixiang Fan
Student's number: 5083898
Student's email address: yf874@uowmail.edu.au
----------*/

1. Accuracy and Computational Complexity of 2 boundary detection

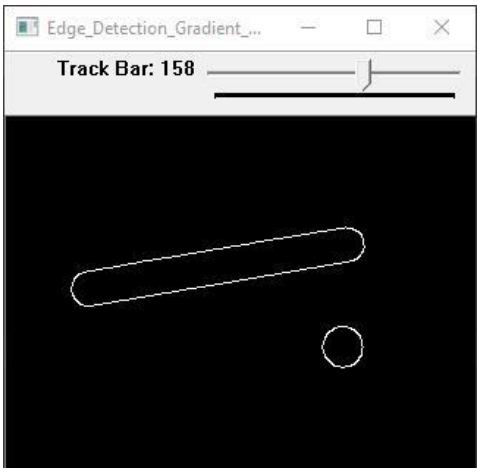
Generally, as to accuracy, binary morphology is a little better according to my test results. When coming to asm2n.bmp, the gradient operator method cannot remove all single point even setting the threshold value of cvCanny function to 255. The binary morphology method does not have this problem. For example, when the threshold of edge detection (gradient operator) is assigned to 158, there are many





noise points in the picture. As a result, the parameter I set for line detection(cvHoughLines2) is not suitable for this situation. In other words, line detection failed to deal with this picture. When I switch the track bar to the maximum 255, there still exists several noise points. All in all, as to accuracy , binary morphology is better.

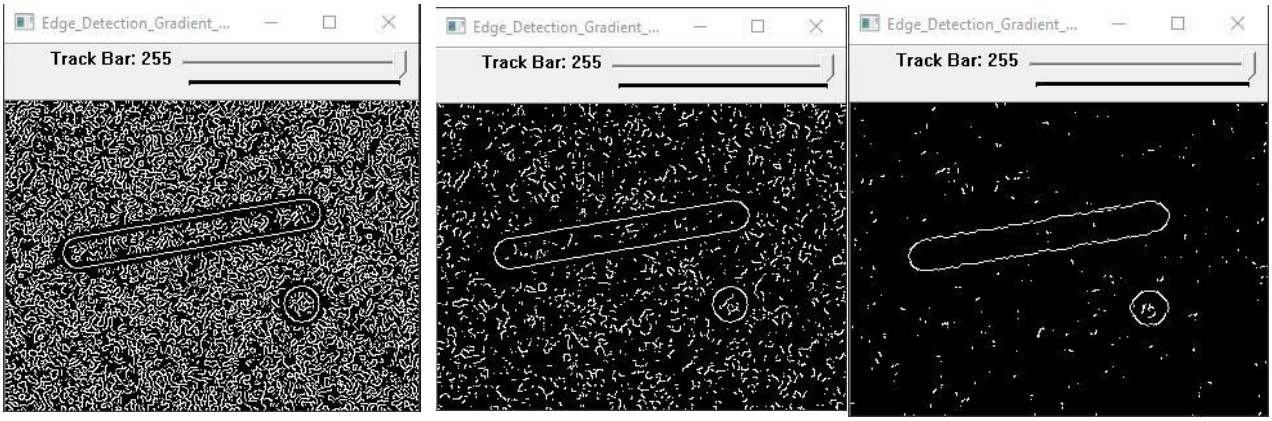
As to computational complexity, theoretically, the complexity of gradient operator is O(n) and the one of binary morphology is O(n) as well. In detail, the average complexity of gradient operator depends on the size of its Prewitt kernel. The average complexity of binary morphology is fixed. The procedure of binary morphology is binary image processing, erosion and boundary extraction successively.



2. Efficiency of edge detection with LPF and without it

As to binary morphology, with LPF or without

does not make any difference. Binary image processing can handle the noise in asm2n.bmp pretty well.



As to gradient operator, LPF is quite important. As I mentioned above, noise in asm2n.bmp makes edge detection difficult. However, after applying LPF, when the threshold is set to 158, all noise points are removed as following. LPF can smooth the edge of picture and blur the picture. As a result, noise will be blended into the background, So they are not obvious in contrast with the background. Then edge detection will not regard them as edges. However, I adjusted the size of Prewitt kernel to 5 and finded that noise cannot be removed at all even setting the threshold of cvCanny to 255(top left). Then I set the size of blur to (5,5) , result become a little better(top middle). Finally, I set the size of blur to (7,7), although noise become less, but the edge we want become irregular as well. So, LPF can help to reduce noise, but it also reduce the resolution of image and make the post-processing difficult.

3. Computational Complexity and execution time of Hough Transform

The computational complexity of Hough Transform is still O(n) although its coefficient is big. For each point detected in the picture, system should calculate its curved line in accumulator array and then each peak which beyond the threshold value represents different line in the picture. Then check the line by standards designated by programmer. The standards include the distance resolution, the angle resolution, the minimum length of lines and the minimum distance between lines etc.

The execution time of Hough Transform is about 0.003s for 2 pictures and 2 methods. This result is too small to manifest anything, so I add a function. When the program comes to line detection part, my system will ask user “How many times would you

```
for (int i = 0; i < 5000; i++) {
    if (flag == 1)
        pcvSeqLines = cvHoughLines2(dstImage2, pcvMStorage3, CV_HOUGH_PROBABILISTIC, fRho, fTheta, nMaxLineNumber, 106, 20);
    else
        pcvSeqLines = cvHoughLines2(dstImage2, pcvMStorage3, CV_HOUGH_PROBABILISTIC, fRho, fTheta, nMaxLineNumber, fMinLineLen, fMinLineLen);
}
```

like to execute Hough Transform?”. User can designate the execution times by themselves. I have tried to time 2000 times Hough Transform. The execution times are all around 2.8s.

4. Sensitivity of my system to noise with asm2n.bmp

As I mentioned in section 1, my system is quite resistant to noise. Applying LPF will help to remove all noise for gradient operator method. As to binary morphology, binary image processing will remove all noise.

5. Introduction of my code

At first, I thought I should implement the program all by my own code. So I implemented scaling down and edge detection. Then I found I am allowed to used functions included in OpenCV, so I rewrote my program and put the previous program in the end in annotation. A difficult problem I found in this assignment is to find the appropriate parameter, the minimum length of lines and the minimum gap between lines. So I wrote a short code segment to find them. After finding the parameters, I put the searching segment in annotation. The following is the searching segment.

```
int c = 0;
for (int i = 0; i < 320; i++) {         //time hough transform
    for (int j = 0; j < 320; j++) {
        pcvSeqLines = cvHoughLines2(dstImage2, pcvMStorage3, CV_HOUGH_PROBABILISTIC, fRho, fTheta, nMaxLineNumber, i, j);
        c++;
        if (pcvSeqLines->total == 2) {
            CvPoint* line1 = (CvPoint*)cvGetSeqElem(pcvSeqLines, 0);
            int l1 = cvRound(((double)(line1[1].y - line1[0].y) / (double)(line1[1].x - line1[0].x)) * 100);

            CvPoint* line2 = (CvPoint*)cvGetSeqElem(pcvSeqLines, 1);
            int l2 = cvRound(((double)(line2[1].y - line2[0].y) / (double)(line2[1].x - line2[0].x)) * 100);

            if (l1 == -16 && l2 == -16) {
                cout << i << " : " << j << endl;
            }
        }
        if (c % 10000 == 0)
            cout << "c : " << c << endl;
    }
}
```

The operation steps of my system is little tricky. Firstly, run the code in VS2015. A cmd window will come out and ask user to input file name. The image file must in the same project file. After inputting file name, the original picture and scaling down picture will be displayed in 2 windows until the end of the system. Select either of them and input any key keyboard. The cmd window will ask user whether or not apply LPF. If user input ‘y’ or ‘Y’, the system will apply LPF and display the image after LPF in a window. The post-processing will base on this image. Otherwise, the post-processing will base on the scaling down image. Select any window and input any key, system will go to next stage - edge detection by gradient operator method. There is a track bar in the top of the edge detection window. User can slide the bar to change the threshold of cvCanny function. After adjusting the bar, input any key and the image will be saved as gradient.bmp. Then the cmd window will ask the execution times of Hough Transform. Input a number and the system will display execution time of hough transform base gradient operator. Then the system display

the lines detected in image gradient.bmp. Besides, a binary image will be displayed in a window. Select it and input any key. The system will require user input the execution times of hough transform based on binary morphology. Input a number and the system will display the execution time and the lines detected in image morphology.bmp which is the ouline of binary image and is saved immediately. Input any key twice then all windows will close. My system ends. Enjoy~

PS :
The edge detection is quite simple and is easy to implement. But the line detection function cvHoughLines is difficult to handle. cvHoughLines is quite sensitive to its parameters. If the minimum length of lines is not large enough, a line may be regarded as 2 lines.