# Sudoku Puzzle Detection Report

**Onur YILMAZ S009604**

## 0.Introduction

Sudoku Puzzle Detection is an interesting project that I created an app that can recognize a Sudoku puzzle from a picture taken by a camera (maybe on a phone). I'll explain a lot of stuff: image processing, detecting lines using the HoughLines algorithm, etc. This report is a technical overview of how things will work for identifying Sudoku puzzles.

Note: This application was implemented using Python 3.7 by taking advantage of an open-source computer library(OpenCV 4.0).

## 1.Problem Statement

Our problem is that reading the input images and displaying the detected sudoku rectangles on the input image. However, on the other hand, camera based computer vision problems remain challenging for several reasons. For example;

• Cameras are of various qualities and different cameras may produce different pictures for the same scene.

• Focus is rarely perfect and zoom is often of poor quality.

• Light conditions may highly vary.

• The rotation of the image varies from one shot to another.

• Pictures taken from newspapers have other caveats.

• Newspaper pages are never completely flat, resulting in curved images.

• Each newspaper uses different font styles and sizes.

## 2.Proposed Solution

Every computer vision application starts with the conversion from colour (or grayscale) to monochrome image. For a good conversion to monochrome, I used adaptive thresholding. Adaptive thresholding doesn't use the fixed thresholding value 128. Instead, it calculates the threshold value separately for each pixel. It reads 15x15 surrounding pixels and sums their intensity. Then the average mean value of sums becomes the threshold value for that pixel. The formula for the pixel is threshold = sum/225, where 225=15x15 and sum is the surrounding intensity summed. If the central pixel intensity is greater than the calculated mean threshold, it will become white, otherwise black. In the next image, it's thresholding the pixel marked red. The same calculation is required for all other pixels. This is the reason why this is so slow. The program needs width*height*121 pixel reads.

**How do I detect Grid Lines ?**

In order to extract the numbers from the grid, we need to precisely locate where a Sudoku grid starts and where it ends because there is a lot of noise. In most cases, a Sudoku grid printed in a magazine or newspaper is never alone. There are other grids and lines around it that make the noise. To solve this challenge, I used the contour features of the OpenCV library. It tries to find the biggest rectangle in an image, after finding the biggest rectangle on image, it directly ignores pixels which are out of the rectangle.

**Why do I use the Hough Transform Algorithm ?**

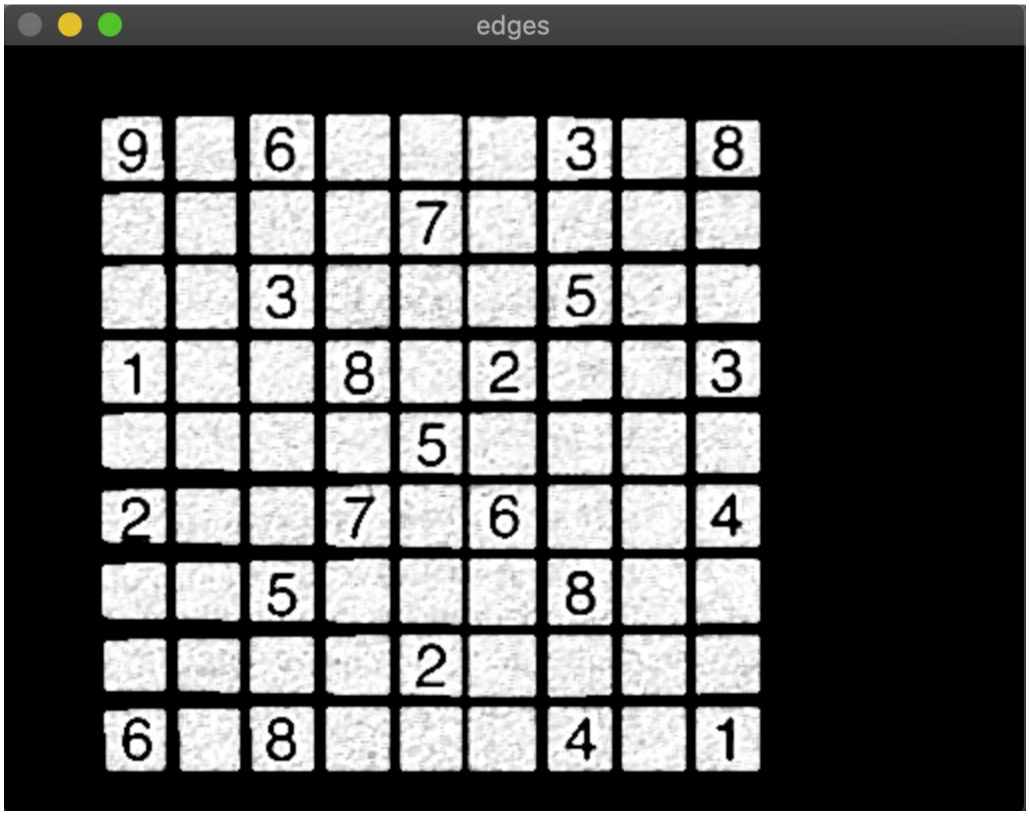
I used an algorithm to detect lines in a monochrome image called Hough transform. How it works: Formula of the line: y = (x * cos(theta) + rho) / sin(theta).Where theta is the angle of the line and rho is the distance of the line from the centre coordinate and every line expressed with only two variables as theta and rho. Algorithms traverse through every pixel in the grey image. It skips white pixels. When on a black pixel, it tries to draw all the possible lines passing through that pixel. To clarify with a little more technical detail, OpenCV function, cv2.HoughLines(). It simply returns an array of rho and theta values. The first parameter, input image which I gave the result of canny edge detection before applying the hough transform. Second and third parameters are rho and theta accuracies respectively. The fourth argument is the threshold, which means the minimum number of lines on the image.

Even Though I utilized `Hough Transform Algorithm`, the algorithm could not find well for all types of images, when it could not detect lines, the algorithm was giving 'NoneTypeError and it directly exits the loop. To solve this problem, I gave different edge detection parameters before running the Hough Transform Algorithm. For example, Canny Edge detection function reads this line → edges = cv2.Canny(gray, 48, 280, apertureSize=3), then this edges is used parameter of HoughLines as → lines = cv2.HoughLines(edges, 1, np.pi / 180, 145). However, lines can not draw due to the Canny Edge detection parameter. This parameter uniquely adjusted for each different type of image, therefore, we need to give it the most suitable parameter for whole images. What did I do to solve this parameter problem? I used a try-catch statement. If gridlines detected easily, it automatically shows the image, else if it does not detect grid line, followingly, edges are recalculated strictly with increasing aperture size to 5.

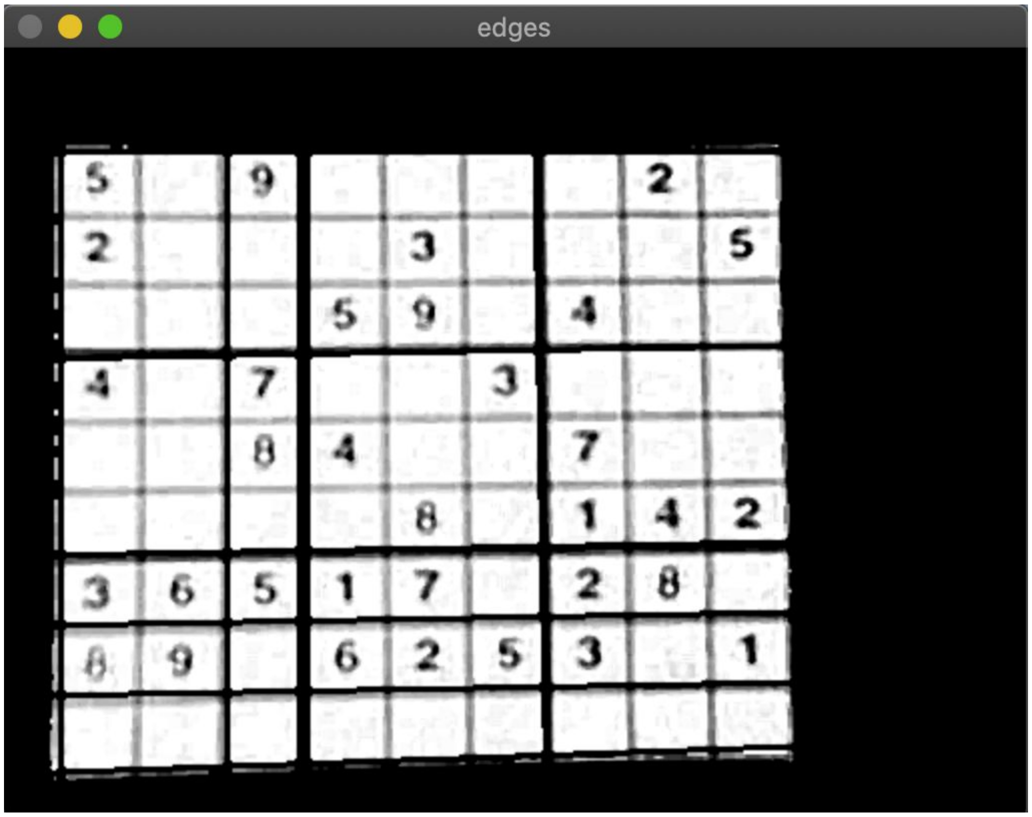
To sum up with a few words what I did, Basic image preprocessing (Thresholding, Resize, Contrast Adjustment etc.) → Crop out approx. Sudoku puzzle (Largest contour) → Get the largest contour of the image. → Get the largest bounding rectangle within the contour. → Compute the grid corners.

## 3.Result

### → Best Result

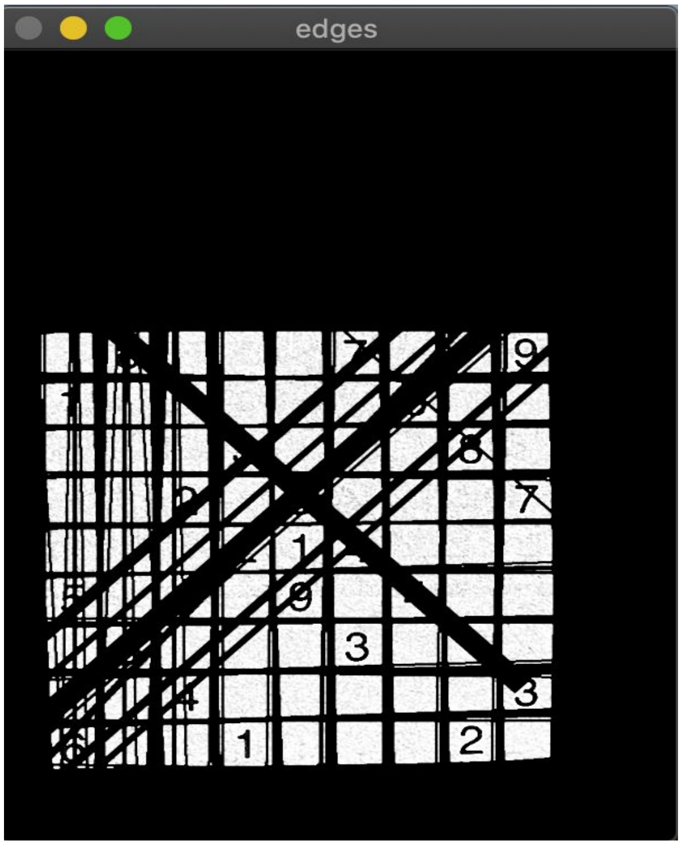


## → Average Result



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | | 9 | | | | | 2 | |
| 2 | | | | 3 | | | | 5 |
| | | | 5 | 9 | | 4 | | |
| 4 | | 7 | | | 3 | | | |
| | | 8 | 4 | | | 7 | | |
| | | | | 8 | | 1 | 4 | 2 |
| 3 | 6 | 5 | 1 | 7 | | 2 | 8 | |
| 8 | 9 | | 6 | 2 | 5 | 3 | | 1 |
| | | | | | | | | |

### → The Worst Result



## 4.Conclusion

To sum up, although sudoku grid line detection is a lot of fun, it took serious time to solve this problem. When we looked at the results, we were able to detect all the sudoku images included in the sudoku_dataset. However, when we looked at each box detection result, I had a success of 35/40, that is close to 87 per cent. As the parameters affecting these results are improved, the success of detecting this grid line will become higher.