Özyeğin University

Engineering Faculty

Computer Vision (CS523)

Assignment No: I

Problem: Detecting Sudoku Puzzle Rectangles with Computer Vision Algorithms

Mustafa Ghanim Department of Electrical-Electronics Engineering

INTRODUCTION

In this course, computer vision; we try to learn the basics of how to process an image, interpreting it a proper shape that our programs and smart machines can understand. Thus, the importance of having a strong theoretical and practical background cannot be underestimated. Computer vision is a huge field that is related to obtaining useful information from images and videos in order to process that information and act accordingly to perform specific actions using Artificial Intelligence (AI) or classical mathematical models-based algorithms. This area of interest is used in many applications such as, healthcare, security, Internet of Things (IoT), automotive, industry, and banking. In short, it can be used for any application that is affected by "vision". Thus, being a researcher and developer at this subject has many motivations for the present and future. In this project we discuss an important basic problem in computer vision related to object and line detection. The problem is explained and analyzed in terms of theoretical and programming aspects. The proposed methods to solve the problem are analyzed as well. Moreover, the obtained results are shown with comments on them.

PROBLEM STATEMENT

The total project is about designing a program that can detect all digits in a Sudoku puzzle image given and solve the puzzle. The first part of the project (assignment I) deals with detecting the puzzle's enclosing shape (largest rectangle) and each box inside the puzzle using computer vision methods basically implemented on OpenCV library.

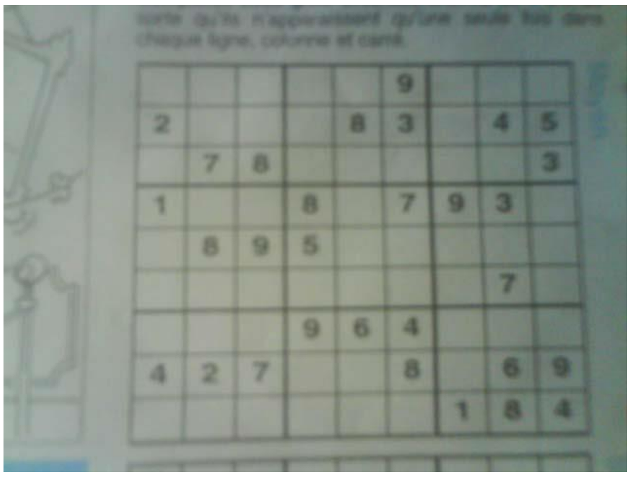
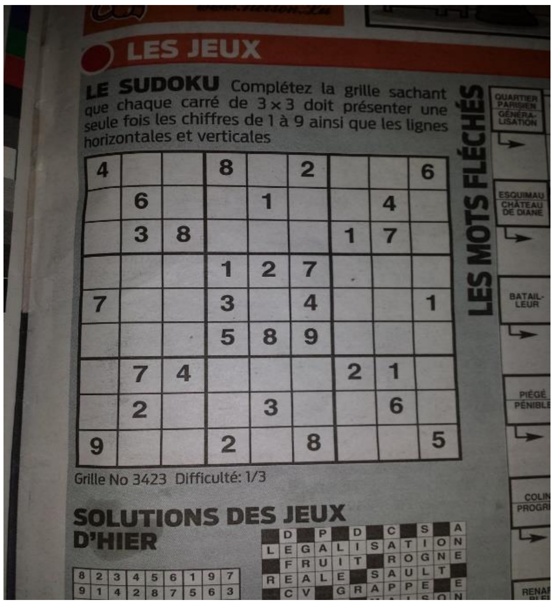


Figure 1: Sudoku Puzzle Example (i)


LES JEUX

LE SUDOKU Complétez la grille sachant que chaque carré de 3×3 doit présenter une seule fois les chiffres de 1 à 9 ainsi que les lignes horizontales et verticales

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

Grille No 3423 Difficulté: 1/3

SOLUTIONS DES JEUX D'HIER

LES MOTS FLÉCHÉS
QUARTIER PARISIEN
GÉNÉRA-LISATION
ESQUIMAU
CHÂTEAU DE DIANE
BATAIL-LEUR
PIÈGE PÉNIBLE

Figure 2: Sudoku Puzzle Example (ii)

To test any solution for the problem, the second version of dataset provided by GitHub Link given at the assignment PDF document is used. As it can be seen from figure 1 and figure 2, this dataset has many challenges such as low-quality blurred images taken by a cell phone camera and some images that do not contain only the Sudoku puzzle, but other objects as well. Thus, any proposed method should be powerful enough to overcome such problems.

PROPOSED SOLUTIONS

As Sudoku puzzle consists of rectangles and squares which in turn consist of vertical and horizontal lines, a good idea is to perform Hough Transform to detect those lines. Hough Transform tries to find the best intersection match for line possibilities to detect pixels that lie on the same line of equation y = ax + b. Hough Transform is known for its robustness when noise exists since it does not need all the pixels when there are only short breaks. However, it can obtain unlikely bad results when noisy or unwanted pixels are aligned as lines. Thus, performing Hough Transform only will not be sufficient for our problem. Based on literature review and technical research, one method is presented for detecting the enclosing borders of Sudoku and two different methods (with their results) are presented to detect the smaller squares after that in this project.

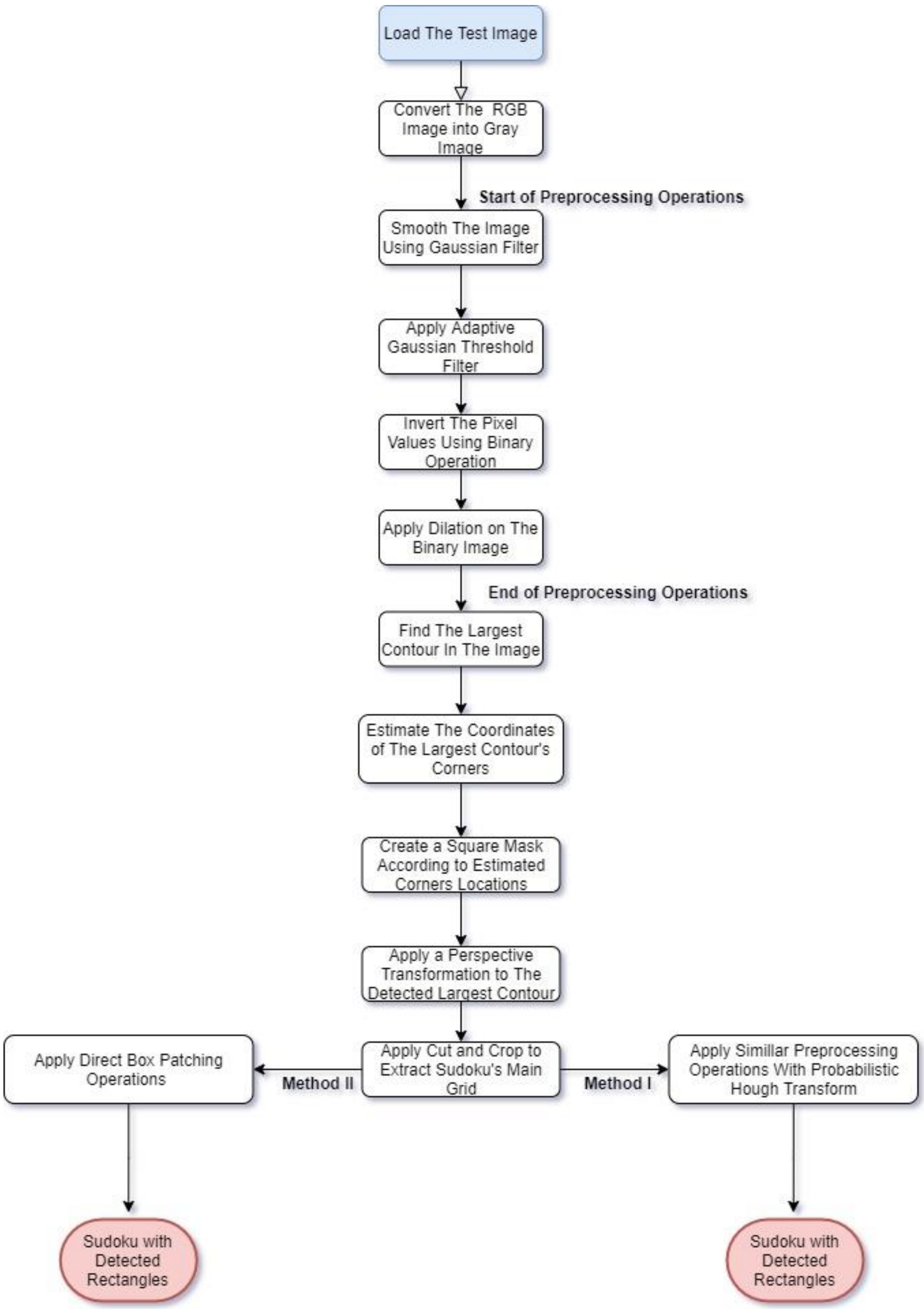

Load The Test Image
Convert The RGB Image into Gray Image
Start of Preprocessing Operations
Smooth The Image Using Gaussian Filter
Apply Adaptive Gaussian Threshold Filter
Invert The Pixel Values Using Binary Operation
Apply Dilation on The Binary Image
End of Preprocessing Operations
Find The Largest Contour In The Image
Estimate The Coordinates of The Largest Contour's Corners
Create a Square Mask According to Estimated Corners Locations
Apply a Perspective Transformation to The Detected Largest Contour
Apply Cut and Crop to Extract Sudoku's Main Grid
Method II
Apply Direct Box Patching Operations
Method I
Apply Simillar Preprocessing Operations With Probabilistic Hough Transform
Sudoku with Detected Rectangles
Sudoku with Detected Rectangles

Figure 3: Flow Chart of The Implemented Algorithms

Figure 3 explains the direct flow of what is used to get the correct results for most of the input data set. To start with, color images in RGB space should be converted to grayscale images to simplify preprocessing and main algorithm operations since working in one channel of intensity is much easier. We use **cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)** function in Python OpenCV where img is the target image to be converted. Since **cv2.imread('image_name.jpg')** reads the input image as BGR instead of RGB, when converting to grayscale BGR2GRAY should be considered. The following equations expresses this conversion operation:

Gray(x,y) = 0.2989 * Color(x,y,Red) + 0.5870 * Color(x,y,Green) + 0.110 * Color(x,y,Blue)

After that we use Gaussian blur (smooth) filter of an odd kernel like 9,11,17 to filter the high frequency noises in the image using **cv2.GaussianBlur(gray_image, (odd_number, number), sigma)**. Odd number is generally chosen in filters in order to keep the symmetricity around the processed center pixel. Sigma represents the standard deviation which is taken generally as zero in many applications. Increasing sigma will generally increase the blur in image. The following equations expresses Gaussian filter mathematically in two-dimensions:

$$\boldsymbol{G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(\frac{x^2+y^2}{2\sigma^2})}}$$

Where x,y are the target image coordinates.

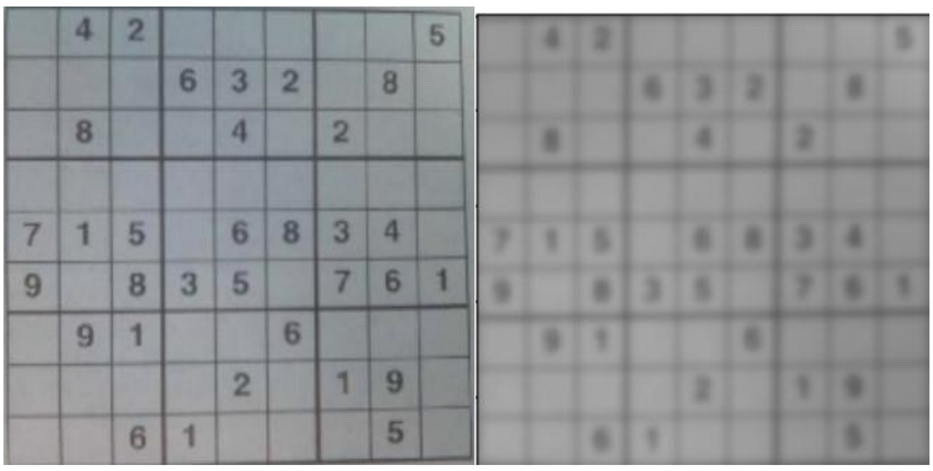


Figure 4: Example of Gaussian Smoothing

After that we apply adaptive threshold filter to the image using **cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, kernel_size, constant subtracted from the mean)**. The threshold adaptive filter's equation can be shown in figure 5. Where dst(x,y) represents the output image and src(x,y) represents the filter.

- **THRESH_BINARY**

$$\mathrm{dst}(x,y) = \begin{cases} \mathtt{maxValue} & \text{if } \mathrm{src}(x,y) > T(x,y) \\ 0 & \text{otherwise} \end{cases}$$

Figure 5: Adaptive Threshold Filter Equation

The adaptive threshold filter aims to convert an image to a binary image by processing each pixel’s intensity and depending on the threshold value as well as small regions around it (blocks) to decide the output intensity value of the pixel whether as 0 or 255 (maximum value). Since generally lines are darker than other objects in the Sudoku image, meaning that their intensity values are closer to zero. Adaptive filter outputs a binary image that is most likely to have black lines (0 intensity) and most of the other objects will have 255 value of intensity. Hence, it is necessary to apply now NOT binary operation to image in order to make lines have 255 intensity and other objects as 0 intensity. See Figure6.

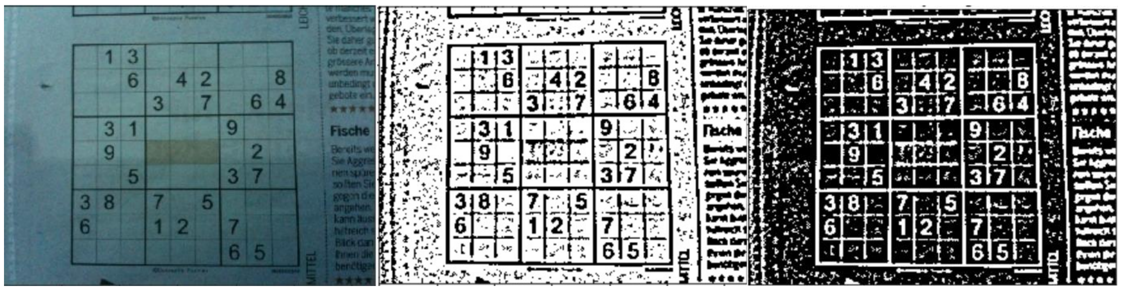


Figure 6: Input Image on the Right Side, Thresholded Image on the Center, and Binary Inverted Image on the Left Side

Moreover, image dilation which is expressed mathematically by $\oplus$ and equivalent to Makowski addition can be applied over the binary image in order to make the objects more visible and fill small holes in the objects. A kernel of the following coefficients can be defined for dilation as:

$$kernal = \begin{matrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$$

By passing such a kernel over the target image, dilated image is obtained by the rule that if the processed pixel (generally center pixel) becomes 1 (in binary form) if any of its neighbors within the kernel boundaries is 1, otherwise it is set to zero. Moreover, the opposite of dilation operation erosion can be also used for preprocessing operations which has the rule of setting the processed pixel into the minimum value of its neighbors. The decision whether to use dilation or erosion has a strong dependency on the image itself.

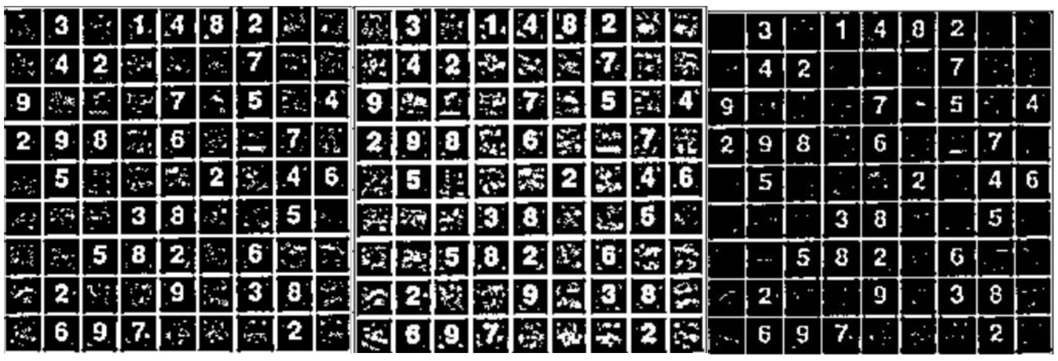


Figure 7: Examples of The Binary (Left), Dilated (Center), and Eroded (Right) Images

After these steps it becomes easier to detect the Sudoku Board corners. **cv2.findContours()** is used to find the contours of the image. For this project, cv2.CHAIN_APPROX_SIMPLE method is used for **findContours** approach which compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points. This function implements the mathematical background presented in Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985). “Border following is one of the fundamental techniques in the processing of digitized binary images. It derives a sequence of the coordinates or the chain codes from the border between a connected component of l-pixels (l-component)’ and a connected component of O-pixels (background or hole)”. Thus, can be used for many shape and object detection applications such as this project.

After finding each contour in the binary image and storing them in array or list, a sorting function is used to detect the largest contour. To detect the corners of this contour which is the Sudoku board in our case; we search inside the contour borders with **i** index for columns and **j** index for rows by defining:

Top Right Corner: The point where i - j is maximum

Top Left Corner: The point where i + j is minimum

Bottom Right Corner: The point where i + j is maximum

Bottom Left Corner: The Point where i – j is minimum

Finding these points using similar sorting, MIN, and MAX operations is possible.

Since typical Sudoku Puzzles have a shape of square meaning same length at each side. It is better to convert our semi-rectangle board into square shape as the optimal case. To do this the maximum distance between each found corner point is calculated to determine the length of the required square in the worst case. i.e.:

$$Square\ Side\ Length = MAX[DIST(BR,TR), DIST(TL,BL), DIST(BR,BL), (TL,TR)]$$

After that, **cv2.getPerspectiveTransform(input_img,required_square_mask)** function is used to find the best transformation matrix that would fit the input image into the required shape. Then **cv2.warpPerspective(input_img, Prespective Matrix, (maxWidth, maxHeight))** is used to fit our image into the required square-shaped image as a final step for extracting the Sudoku puzzle. At this stage, we can say that the enclosing largest rectangle of Sudoku is found.

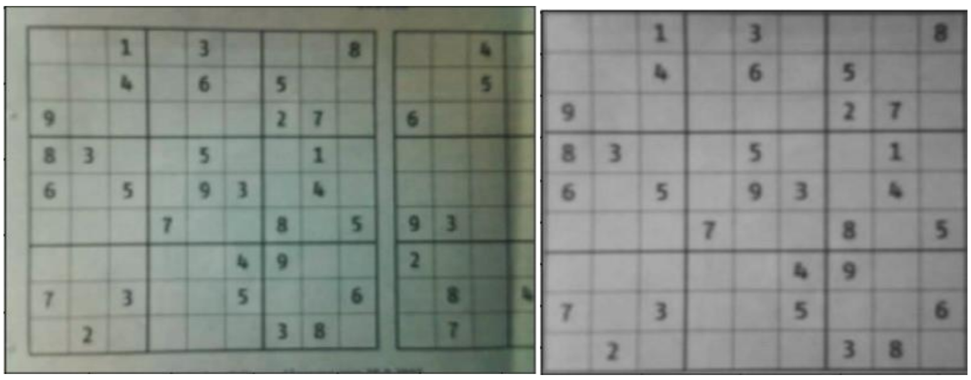
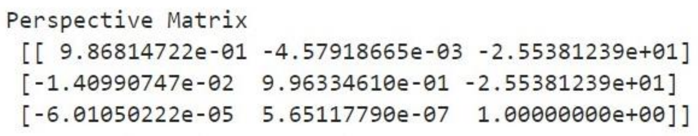


Figure 8: Input Image Side and its Specific Complete Sudoku Board Extracted



```
Perspective Matrix
 [[ 9.86814722e-01 -4.57918665e-03 -2.55381239e+01]
 [-1.40990747e-02  9.96334610e-01 -2.55381239e+01]
 [-6.01050222e-05  5.65117790e-07  1.00000000e+00]]
```

Figure 9: Matrix Required for Perspective Transformation for Images at Figure 8

After obtaining figure7, it becomes easier to detect any rectangle or box of our board. The cheapest and successful method that I used as a main method is to divide the board into size-equal rectangles and draw their sides with different color on the board as well as drawing lines on the extracted puzzle’s borders. Furthermore, since it was asked in the assignment PDF and taught in the course, a Probabilistic Hough Transform based method is also tested and obtaining good results after extracting the main rectangle, but still, the first method is simpler and more logical to go with. Obtained results are shown in the “Results” section. As it is mentioned before; after extracting the main board of Sudoku the simplest way to draw detection colors on the main rectangle and sub-rectangles is to divide the image into equal sized patches with rectangle aspect ratio according to image's width and height as follows:

$$Rate\ of\ Horizantal\ Lines\ to\ be\ Drawn = \frac{Image\ Height}{9}$$

$$Rate\ of\ Vertical\ Lines\ to\ be\ Drawn = \frac{Image\ Width}{9}$$

It is important to be careful about division results with reminders which leads to error in Python. Thus, rounding up the divisions or increasing the rates constants by one will be sufficient in the worst cases.

The other approach to continue drawing the lines, is to preprocess the board-only image with similar operations explained before and detect the lines using Probabilistic Hough Transform (better Hough Transform version with control on line intersections, minimum line length, and maximum gap length between lines). PHT uses random sampling of the edge points and has less voting stage candidates since it uses less and random subset of the edge pixels. However, in a general manner PHT has the same idea of one-to-many mapping as Standard Hough Transform.

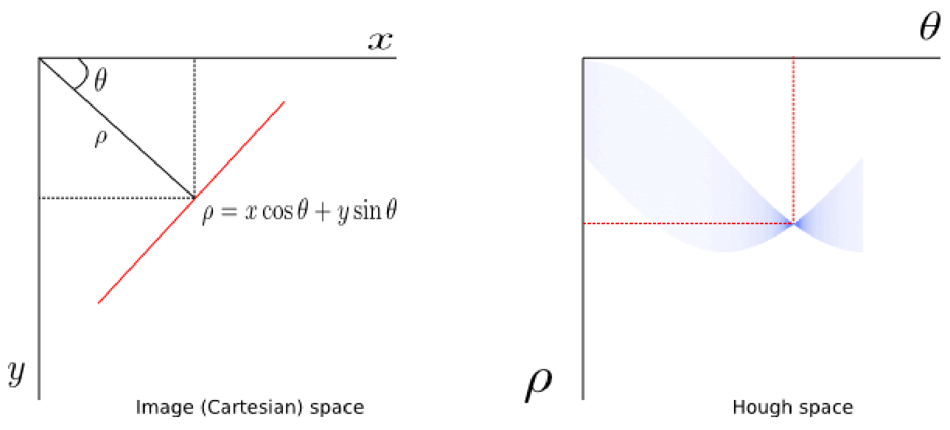


Figure 10: Hough Transform Mapping

RESULTS

This section shows results of different randomly selected input images. The detection results with green lines are obtained by equally patching method, and results with blue lines are obtained with preprocessing on the extracted Sudoku board images with Probabilistic Hough Transform.

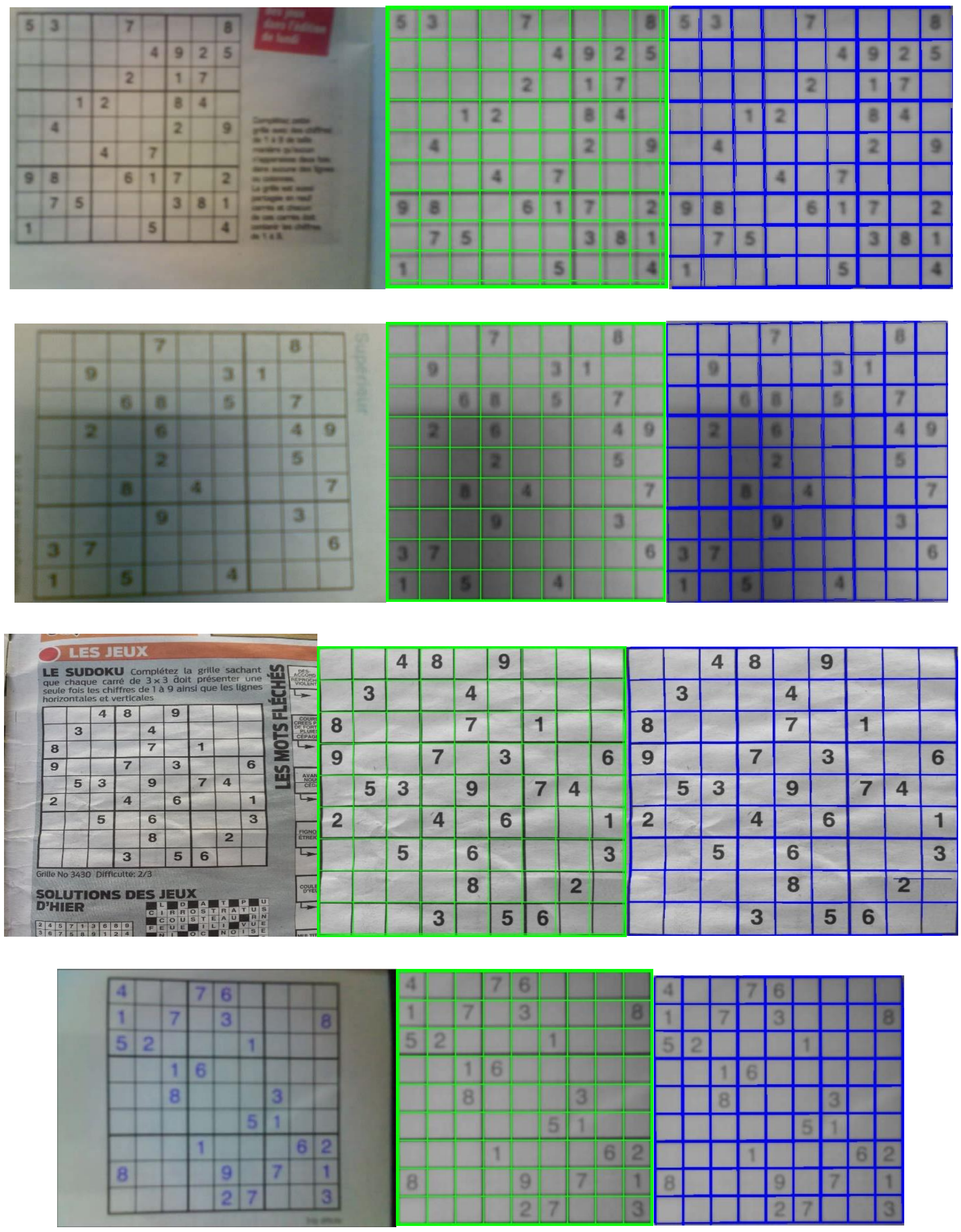


Figure 11: Results of The Implemented Detection Solutions

As it can be seen from the results, both of Probabilistic Hough Transform and Direct Patching methods are very successful at detecting each box thanks to preprocessing operations and extracted Sudoku board using contour detection and perspective transformations.

We can say that now these results are satisfying enough to move forward towards the next stage of the project.

REFERENCES

- https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html
- https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html
- Suzuki, S. and Abe, K., *Topological Structural Analysis of Digitized Binary Images by Border Following*. CVGIP 30 1, pp 32-46 (1985)
- https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html
- https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html?highlight=adaptivethreshold
- https://en.wikipedia.org/wiki/Gaussian_blur
- https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html