Sudoku puzzle extraction

Final project by
Boris Spektor
borissp@post.bgu.ac.il

Introduction

Occasionally one finds himself having a newspaper at hand, and with nothing better to do, one opens the puzzle section and starts solving a Sudoku puzzle. Being a software engineer one must sit in front of a screen during most of his time, making him appreciate the good old way of solving puzzles using pen and paper. With all the technology at hand, and the countless Sudoku apps and games, one might be used to getting a hint (or the entire solution) by simply pressing a button, but unfortunately technology hasn't advanced yet to the point of allowing such perks to be available in simple newspapers.

But what if one had an app to do just that?

The goal of the project is to extract a Sudoku puzzle from a given Sudoku image, and optionally display a solution to that puzzle.

Approach and Method

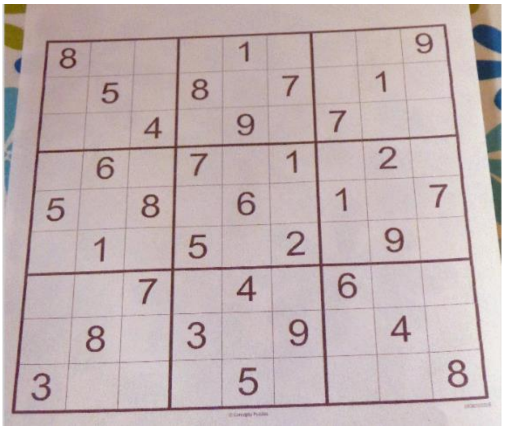
The approach taken in this project was to first align the puzzle with the screen, and then divide it into 9X9 squares and treat each square separately.

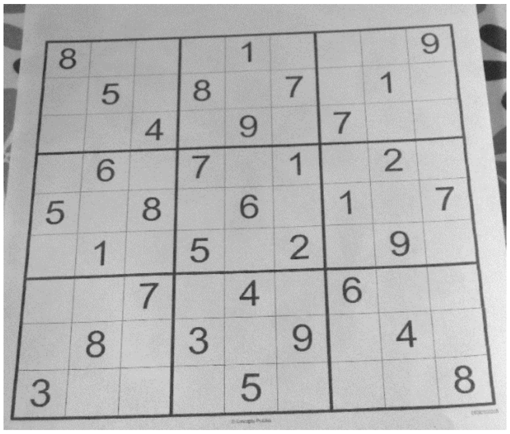
The method used consists of four main parts:

I. Preprocessing
II. Finding the corners of the puzzle
III. Calculating and applying an homography
IV. Extracting and identifying the numbers

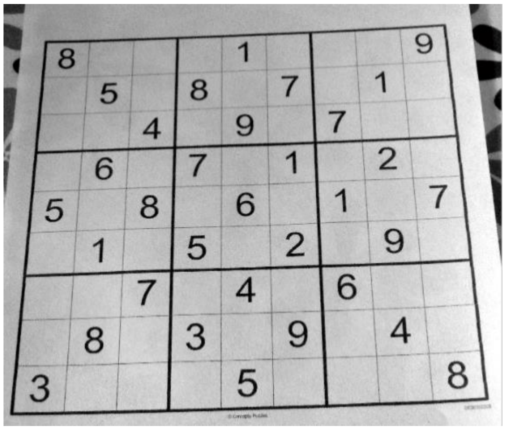
I. Preprocessing

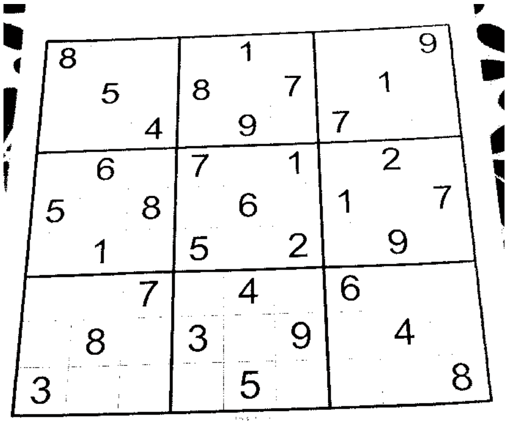
The preprocessing part starts with converting the given image to grayscale in order to simplify processing.



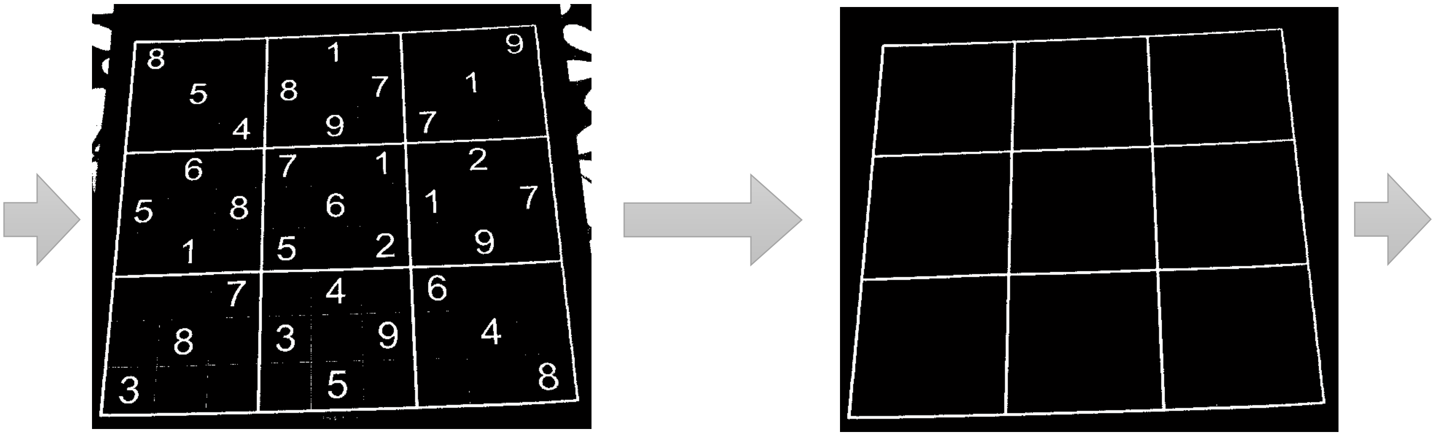


Next it is required to separate the puzzle itself from the paper – this is done by thresholding, but since each image would have a different threshold, the threshold used is the mean value of the image intensity values. In order to improve thresholding results the image's contrast is increased prior to thresholding.



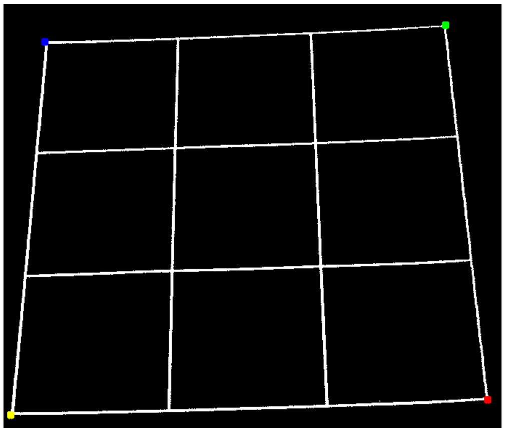


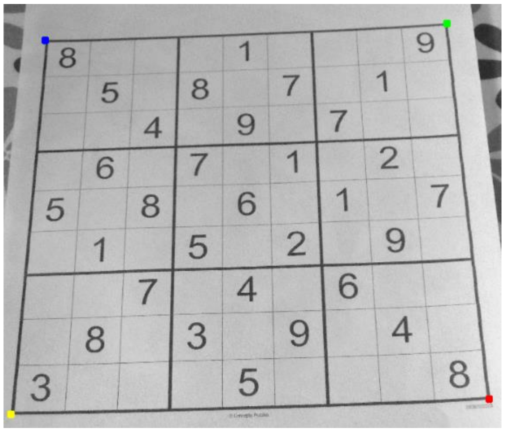
Now we have an inversed binary image since the background is where the 1's are and the puzzle itself is where the 0's are, so we need to flip their roles. Afterwards, we would like to remove small objects and keep only the frame of the puzzle. The small objects are saved in order to be used later for number recognition.



II. Finding the corners of the puzzle

This part can be done in two ways – automatic and manual. It is of critical importance since wrong detection of the corners would result in a wrong homography and the assumption of having an aligned square puzzle doesn't hold. The automatic corner detection works well mostly with images which contain only the puzzles or small objects around the puzzles, but fails with images which contain big objects comparing to the puzzle itself. For complex images like that there is an option for manual corner selection. In this example the automatic corner detection works well.

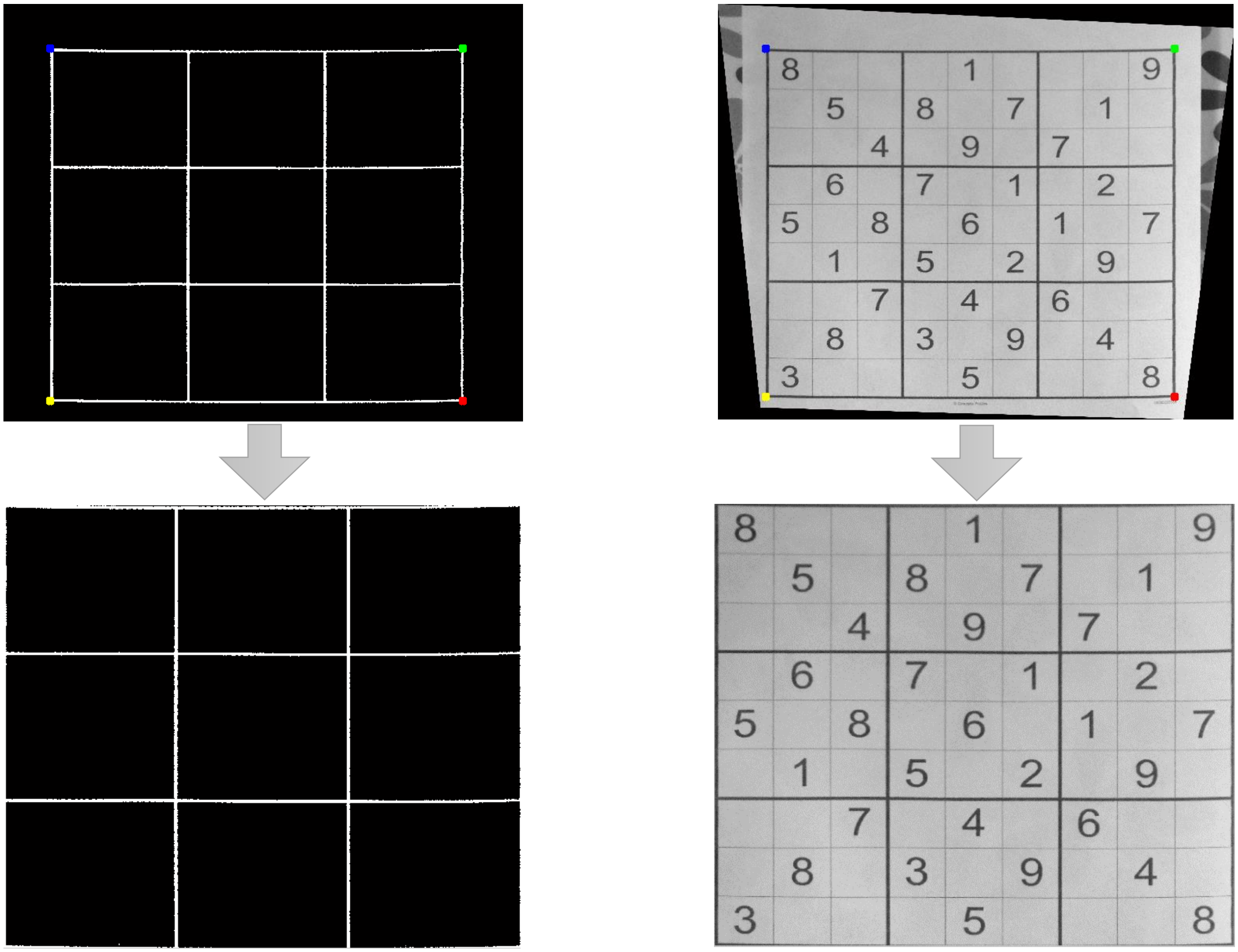




The algorithm used here finds the closest pixel to each corner in terms of vertical distance + horizontal distance.

III. Calculating and applying an homography

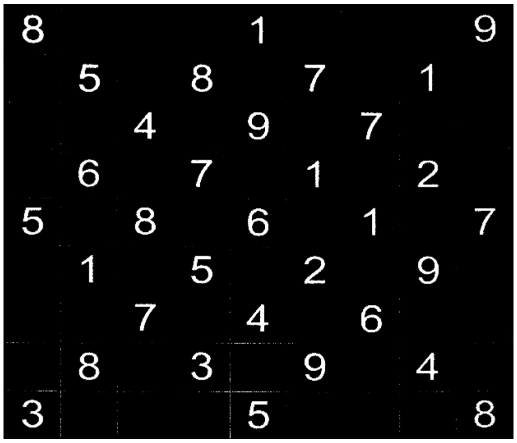
In this part we're calculating the homography matrix which will be used for transforming the puzzle into the screen coordinates, and then cropping the image to contain only the puzzle.



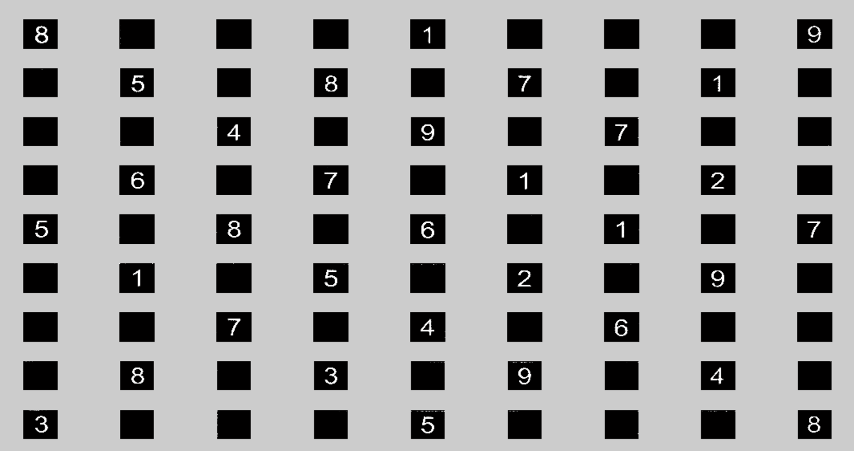
IV. Extracting and identifying the numbers

In this part we're dividing the image into 9x9 squares in order to process each square separately. The image used is the one containing the small objects we removed in part I. For each square do the following: find and keep the biggest connected component, apply an averaging filter on the square and threshold, remove small leftovers, check if square contains anything, and if it does extract the digit and identify it using correlation.

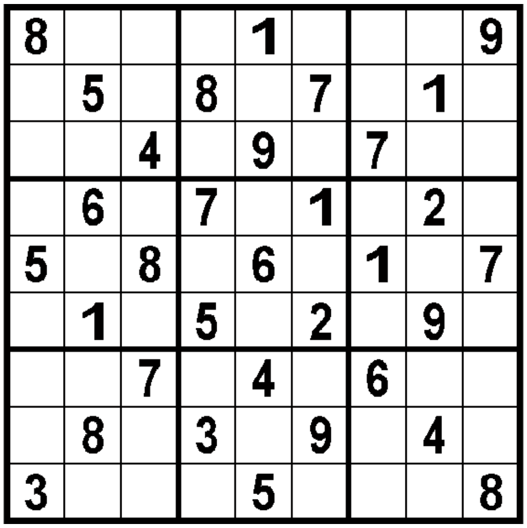
The averaging filter is used because sometimes what is left in a cell is a part of the thin line of the square, or some other noise, and we would like to remove it. This way, if the cell contains a digit it will be a little blurry, but the noise will blend in with the background and filtered out when thresholding.



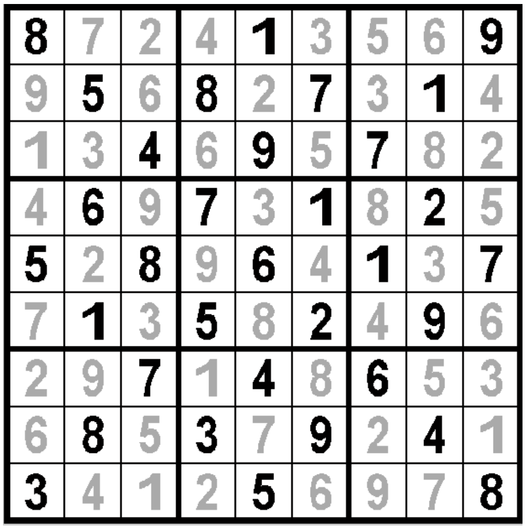
8				1				9
	5		8		7		1	
		4		9		7		
	6		7		1		2	
5		8		6		1		7
	1		5		2		9	
		7		4		6		
	8		3		9		4	
3				5				8



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



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

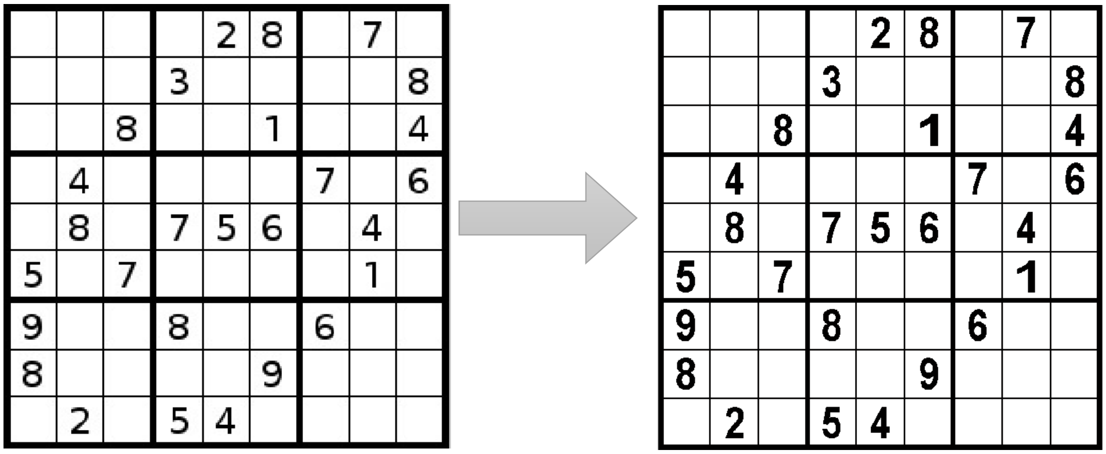


8	7	2	4	1	3	5	6	9
9	5	6	8	2	7	3	1	4
1	3	4	6	9	5	7	8	2
4	6	9	7	3	1	8	2	5
5	2	8	9	6	4	1	3	7
7	1	3	5	8	2	4	9	6
2	9	7	1	4	8	6	5	3
6	8	5	3	7	9	2	4	1
3	4	1	2	5	6	9	7	8

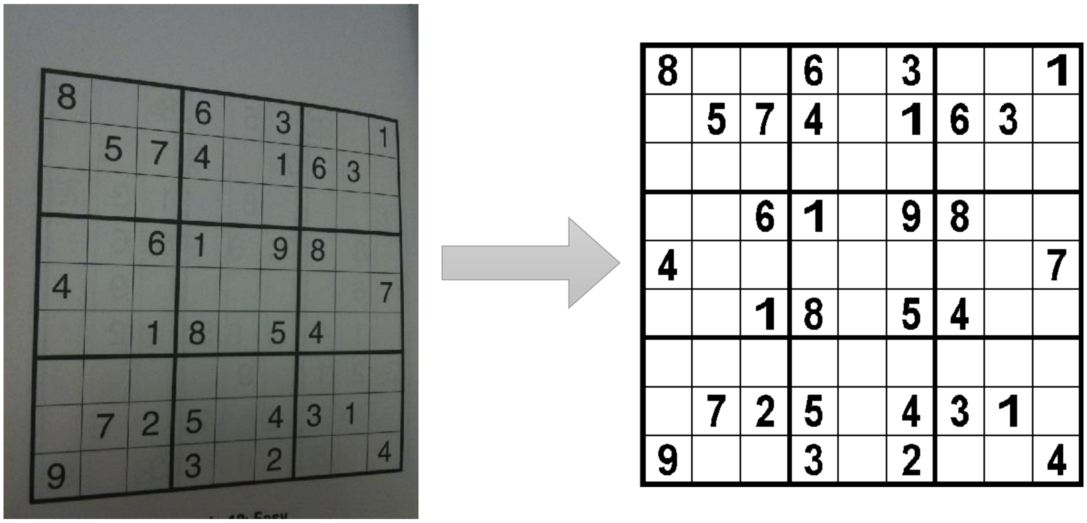
After the numbers have been identified they are displayed in a new Sudoku puzzle, and there is an option to show the solution to the puzzle.

Results

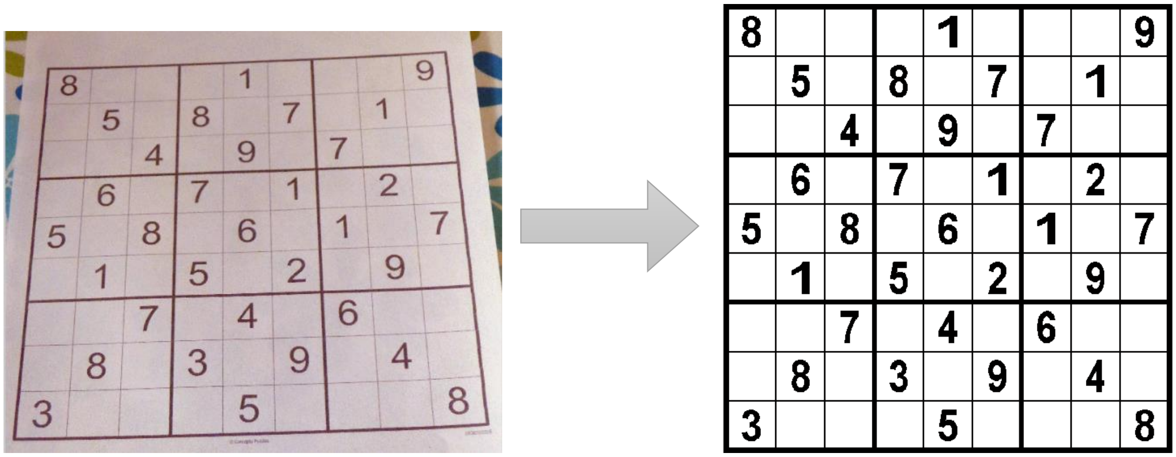
Running the program with various puzzles yielded overall good results, however, each of the parts I through IV can be further investigated and improved allowing better recognition of a larger variety of puzzles.



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



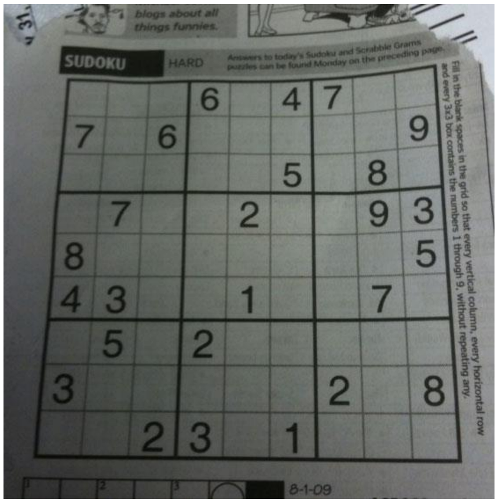
8			6		3			1
	5	7	4		1	6	3	
		6	1		9	8		
4								7
		1	8		5	4		
	7	2	5		4	3	1	
9			3		2			4

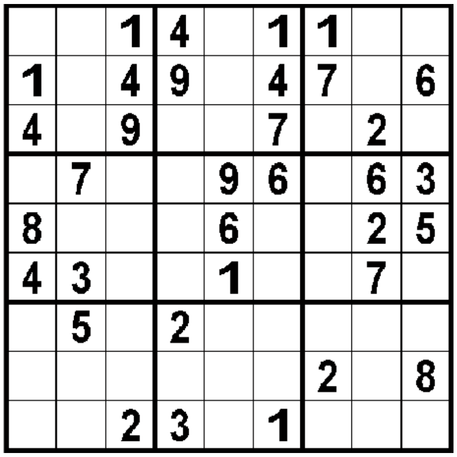


8				1				9
	5		8		7		1	
		4		9		7		
	6		7		1		2	
5		8		6		1		7
	1		5		2		9	
		7		4		6		
	8		3		9		4	
3				5				8

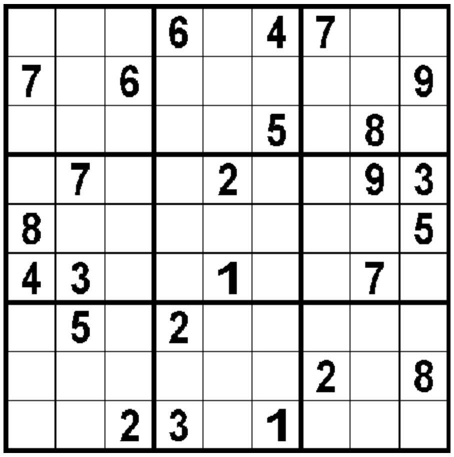
Problems

When the lighting in the picture is not uniform there might be problems extracting the corners correctly, making the rest of the process incorrect. The corner detection part of this algorithm in general is the most problematic part, since in order for it to work correctly a very certain type of image is assumed. For example, the following are examples of a puzzle which is non-uniformly lit, and the results we got were wrong, but when using the manual corner extraction we got almost the correct result.


blogs about all things funnies.
SUDOKU
HARD
Answers to today's Sudoku and Scrabble Grams puzzles can be found Monday on the preceding page.
Fill in the blank spaces in the grid so that every vertical column, every horizontal row and every 3x3 box contains the numbers 1 through 9, without repeating any.
8-1-09



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



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

Another problem is the digit recognition. We used a certain type of digits, and a different font would produce wrong results by mixing 1 and 7, 5 and 6 etc.

Conclusions

In this program, like in many other computer vision related programs, there is no perfect solution. Every solution fits best for a certain type of images, and for a program to fit a different type of images a different solution or tuning of the program is required.

Overall the results achieved are pretty good, and most well lit and distinct puzzles were recognized correctly. This should be enough for the programmer who would like to get some help from the digital world.

There are a few more features I would like to have added to this program if I was to continue improving it:

- Add some way of settling conflicts between similar digits (for instance 5 and 6)
- Find a more general preprocessing method which would fit more types of images and/or
- Find a better way of corner detection
- Improve empty cell detection
- Add an option of overlaying the solution on the original image
- Improve the Sudoku algorithm itself (little thought was actually put into this)

References

[1] **Introduction to Computational and Biological Vision** – Prof. Ohad Ben-Shahar.

[2] **Image rectifiction** - Link