SUDOKU SOLVER

Twisha Vyas | Shweta Oak

Motivation

Our aim was to design a robot that could solve a puzzle and make it write the solution to the puzzle. We chose sudoku as the puzzle, as it is fairly challenging to humans. We wanted to see if we could make a machine solve a problem, and present the result in a way that is close to humans (write the solution).

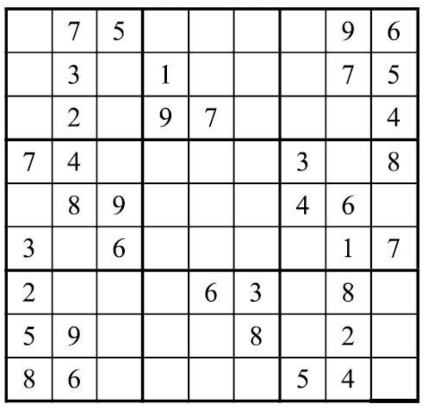
What we did:

To achieve this goal, we broke down this problem into various tasks:

A. Input of problem
B. Image processing to extract data
C. Design an algorithm to solve the puzzle
D. Design a robot to write the solution

A. Input the problem

The user can upload an image file containing the sudoku problem.



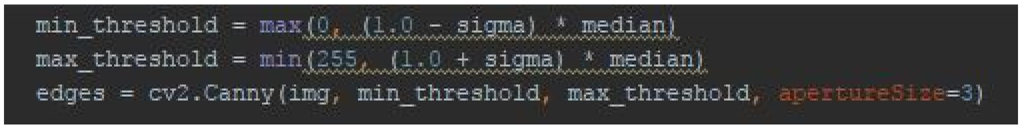
	7	5					9	6
	3		1				7	5
	2		9	7				4
7	4					3		8
	8	9				4	6	
3		6					1	7
2				6	3		8	
5	9				8		2	
8	6					5	4	

Fig 1. User uploaded image

B. Image processing

The entire process can be broken down into the following parts:

- We Identified the edges in the image using canny edge detection. The minimum and maximum threshold for edge detection is defined as a function of the median of the uploaded image.

- 

```
min_threshold = max(0, (1.0 - sigma) * median)
max_threshold = min(255, (1.0 + sigma) * median)
edges = cv2.Canny(img, min_threshold, max_threshold, apertureSize=3)
```

Fig. Calculating minimum and maximum threshold for Canny edge Detection

- After detecting the edges, this is how the input image looks.

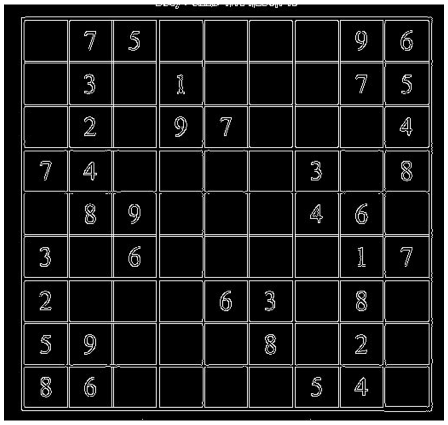


Fig. Image after edge detection

- Once we had the edges, we applied Hough Transform on the obtained image to identify all the lines present in the image and we divided the lines into two parts - (close to)vertical and (close to)horizontal and ignored the others. Only if the number of identified lines was greater than 20, we proceeded.

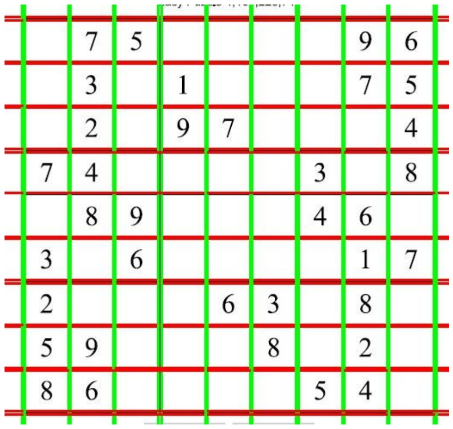
- 

Fig. Vertical and Horizontal lines identified from the image

- After identifying the lines, we found the intersection points and based on those points, we divided the entire grid into 81 boxes, each containing either a digit or blank space. We converted each box into a jpg image

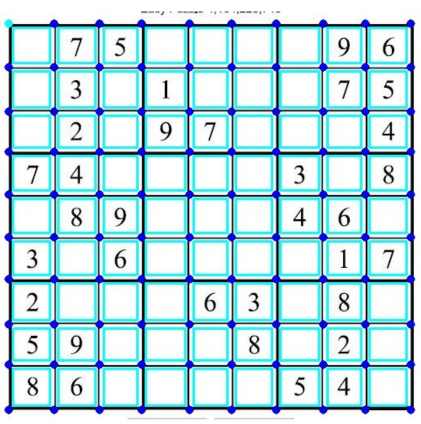


Fig. Boxes containing numbers

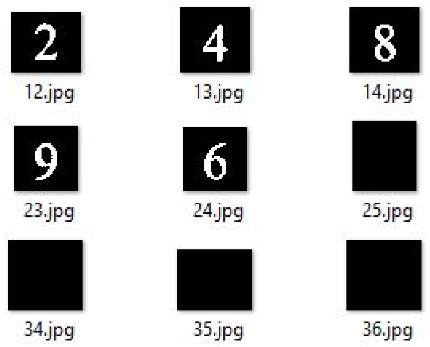


Fig. Images obtained from boxes

- Using Tesseract[3], we extracted digits from the images and converted it into a 9x9 matrix with 0s at position of the blanks.
- We used OpenCV[1] for image processing.

We tried to capture images from the webcam. After edge detection, we tried to smoothen and sharpen the image, however, we were unable to identify all the lines in the image.

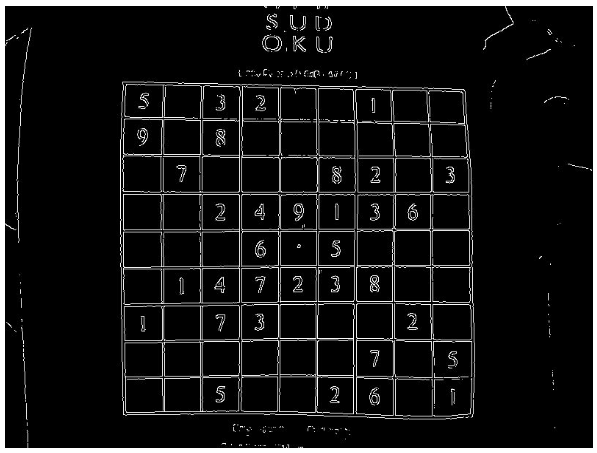


Fig. Image from webcam after edge detection

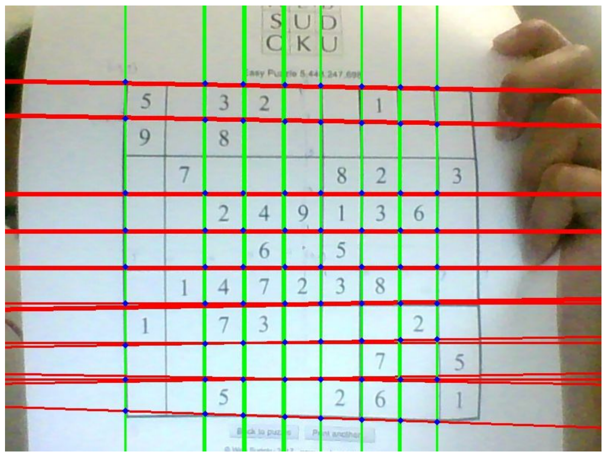


Fig. Identified Lines from the image from webcam

C. Solving the Sudoku Problem

The sudoku problem has a grid of 9x9 with 9 subgrids of 3x3 as shown in the images above. These grids have to be filled with digits from 1 to 9 such that they follow the following rules:

- All digits in a **row** must be unique, 1 to 9, no repetitions.
- All digits in a **column** must be unique, 1 to 9, no repetitions.
- All digits in a **subgrid** must be unique, 1 to 9, no repetitions.

- We used a simple backtracking algorithm to find solution to the sudoku problem, that would check the possible digits at the position and places one, goes to the next position, places the next possible digit. If a digit is found to violate the rules of sudoku, it backtracks to the previous state. The complexity of this problem is O(n^m), where n is the number of blocks in a row i.e. 9 and m is the number of blanks that need to be filled in. This approach is similar to the N-Queens problem.

- We also tried out the approach by Peter Norvig [5] in which for every blank block, we find the set possible candidates, following rules of sudoku, for the blank and set the blank to one of the values in the set of candidates. Go on filling blanks this way, until there is only one possibility for a blank but it violates the sudoku rules. Under this case, backtrack to previous blank and set it to another candidate. This approach has a complexity may be similar to the previous approach, though lesser. However, I learned about constraint propagation and search methods.

D. Design a robot to write the solution

The aim in this module was to simulate a robot arm or end effector that could trace out the digits of the solution such that the digits in the given problem, and the solution could be differentiated.

To do this, we tried 5 approaches:

- Creating a 3D robot arm in matlab like the assignments and make it trace the numbers of the solution onto a plane. However, we found that MATLAB does not let us retain the trace of the end effectors' movement. Hence, this approach **failed.**

- Displaying the solution in empty grids using Augmented Reality on the image of the question itself. We tried using libraries like ARcore[6], to display the digits in the solution on the image. However, we found that detecting the grid from the ImageCapture of the webcam was difficult, and did not end up placing the digits of the solution in the correct position. Hence this method **failed.**

- Creating a 2D robot in matlab was another option, by drawing an orthogonal plot and making a pointer "end effector" move along the plot. This method **would have been successful** if we made 81 subplots for each grid, and plot numbers in each grid. However, defining plots for each grid for each problem was slightly tedious, and plotting it on a single plot was not possible, owing to how matlab is structured, we decided to find a simpler approach.

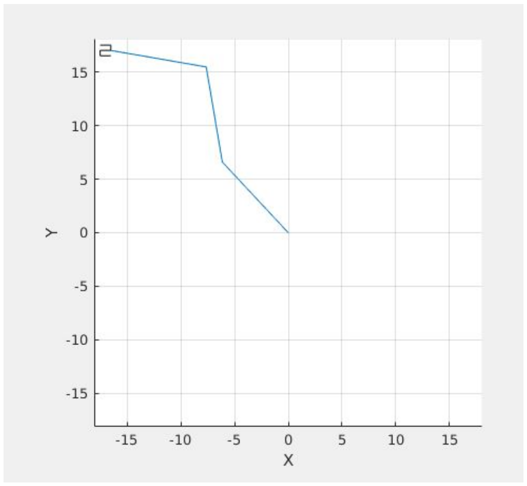
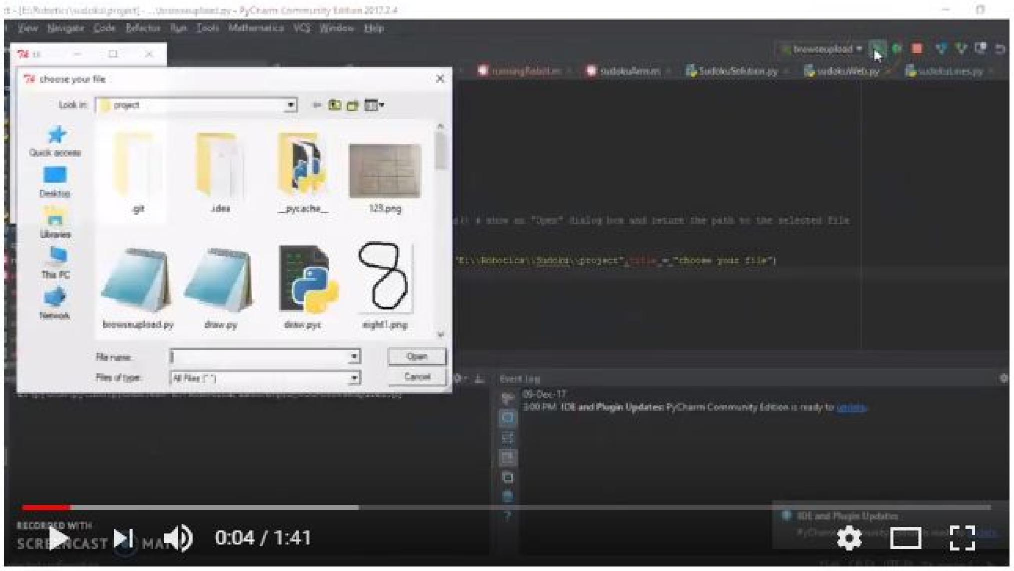


Fig. Plotting numbers using matlab

- A similar thing could be done using the matplotlib library [4] in python. We could create subplots and then plot for each digit. This solution was no better than the previous one. However, it gave better animation than the previous approach.

- The approach that satisfied all the criteria and the aim for this module was to use the Turtle graphics library [2] created an instance of turtle called ninja, which will be our end effector that will draw/write the numbers. This method was a **success** as we could animate the trajectory traced by it. We could change the color to differentiate the digits in the problem from the digits in the solution obtained from the sudoku solver algorithm.

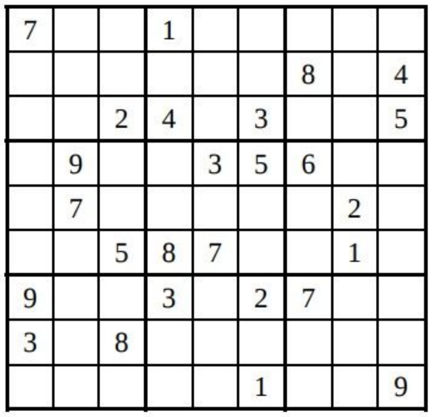
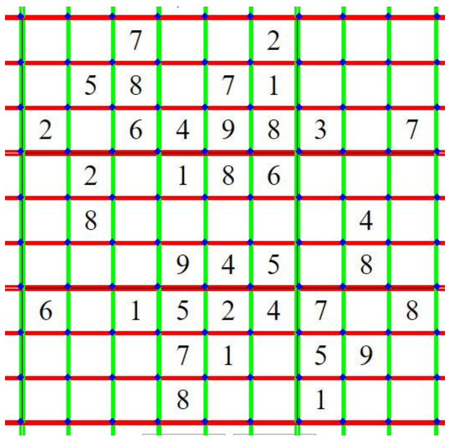
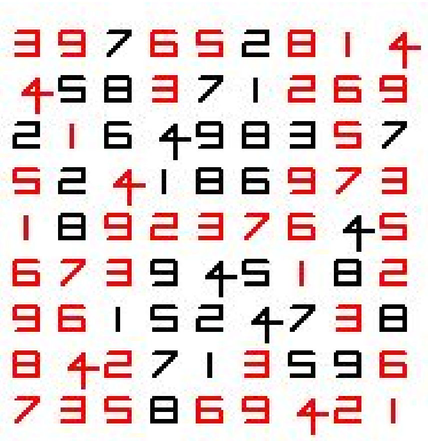
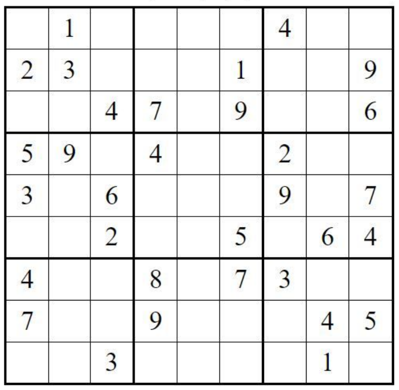
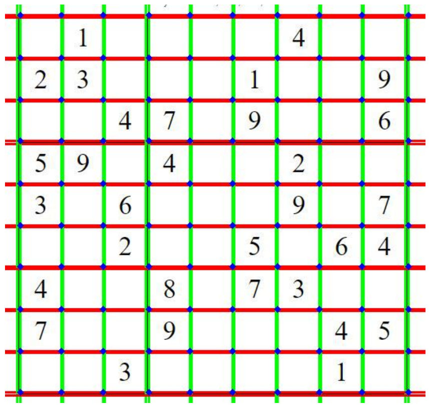
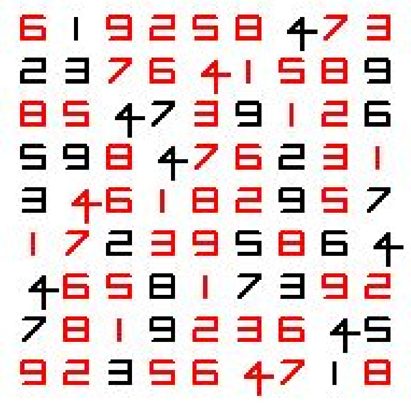
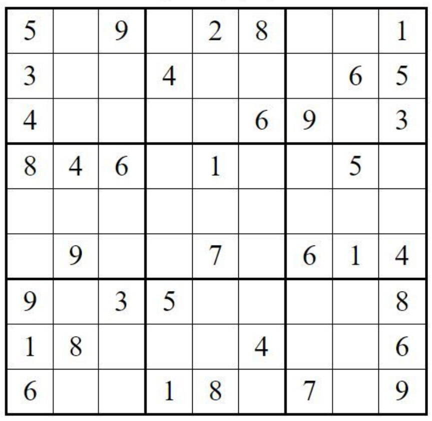
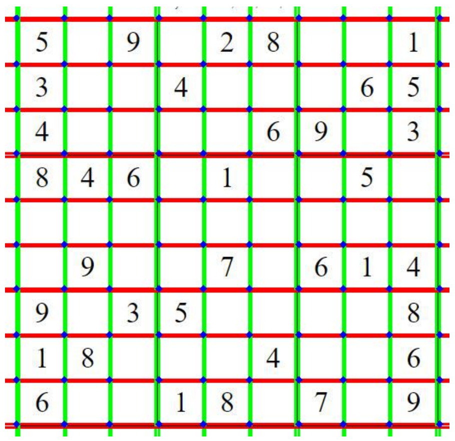
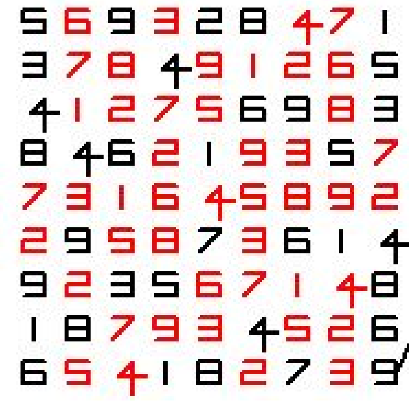


```
# show an "Open" dialog box and return the path to the selected file
"E:\\Robotics\\Sudoku\\project",title = "choose your file")
```

```
05-Dec-17
3:00 PM IDE and Plugin Updates: PyCharm Community Edition is ready to update.
```

Click to view Video! Sudoku Solver

ATTEMPTS:

QUESTION	LINES DETECTED	SOLUTION
		
		
		

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

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

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

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

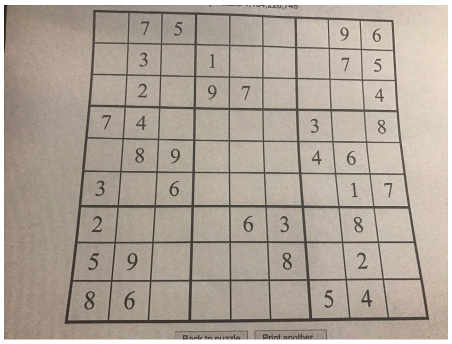
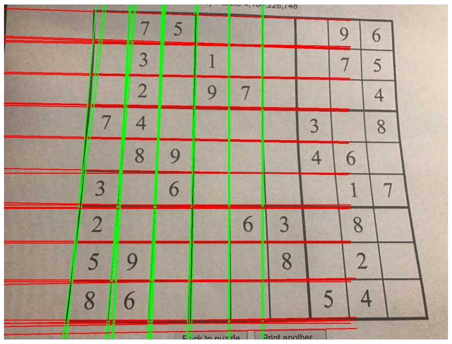
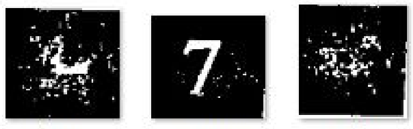
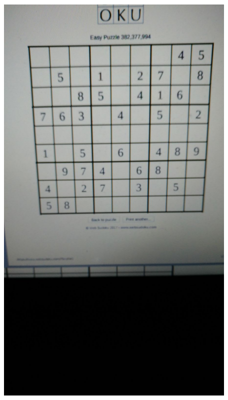
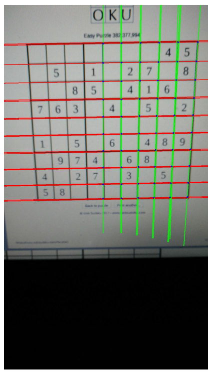
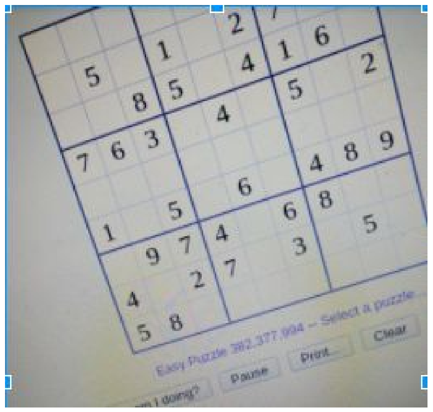
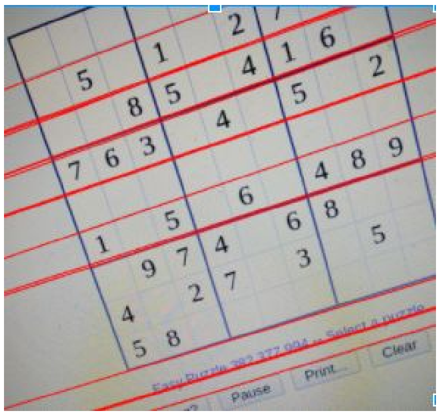
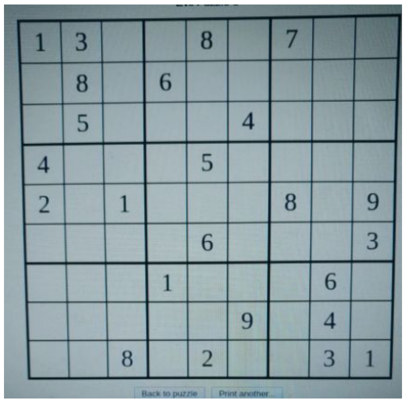
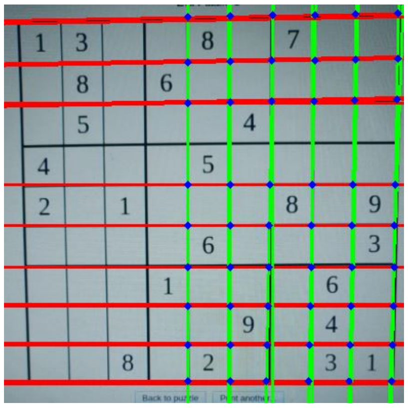
	1					4		
2	3				1			9
		4	7		9			6
5	9		4			2		
3		6				9		7
		2			5		6	4
4			8		7	3		
7			9				4	5
		3					1	

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

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

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

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

		**FAILED**  7.jpg 8.jpg 9.jpg **Distorted Images**
		**FAILED** 38.jpg 39.jpg 40.jpg 41.jpg **Couldn't identify sufficient lines**
		**FAILED** **No horizontal or vertical lines detected. Lines detected are at a slant.**
		**FAILED** **All lines not detected.** 2.jpg 3.jpg 4.jpg 5.jpg

Conclusion and Future Scope:

As demonstrated above, we could successfully make a machine solve sudoku problems from images and get a working simulation of a robot writing the solution.

In the future, we could extend this idea to solving more complex problems such as crosswords and other logic puzzles. We could also use neural networks to solve the sudoku problem.

Another step in the future would be to create a working model of a robot arm that would write the solution on paper. We could also refine the preprocessing pipeline such that it could easily recognize the grid and numbers from a noisy image.

References:

[1] OpenCV: https://opencv.org/
[2] Turtle Library: https://docs.python.org/2/library/turtle.html
[3] Tesseract : https://pypi.python.org/pypi/pytesseract
[4] Matplotlib: https://matplotlib.org/
[5] Peter Norvig's blog on solving sudoku: http://norvig.com/sudoku.html
[6] ARCore: https://developers.google.com/ar/