Project Report

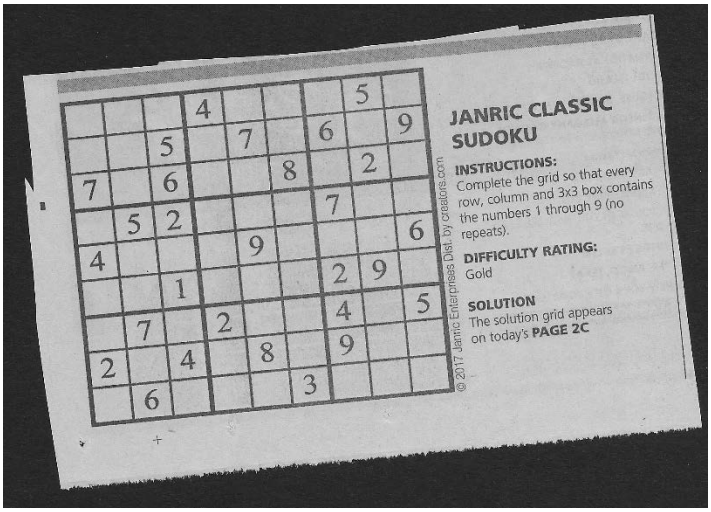
1. Abstract

For this project, we are provided with images that have a Sudoku puzzle and our task is to print out the numbers in the puzzle.

2. Overview

In order to first find the puzzle in the newspaper, I used the Hough transform for finding lines in the image and counted the highest number of lines that all have the same length, under the assumption that these lines belong to the Sudoku. It is possible that those lines could belong to the solution of the puzzle, so I chose the set of lines with the longer length as the actual puzzle. And then using the x- and y-co-ordinates of the lines, I found the four “borders” of the puzzle.

Once I had the four lines of the box surrounding the Sudoku puzzle, I rotate the image so that the puzzle is upright and crop out the background. On the cropped image, I used OCR to get the numbers in the puzzle. The example output looks as follows:

Puzzle	OCR Output
	, , , 4 , , , 5 , , , 5 , 7 , 6 , 9 7 , 6 , , 8 , 2 , , 5 2 , , , 7 , , 4 , , , 9 , , , 6 , , 1 , , , 2 9 , , 7 , 2 , , 4 , 5 2 , 4 , 8 , 9 , , , 6 , , , 3 , , ,

3. **Sub-Sections Per Part:**

a) Approach Used

i) Finding the puzzle

For this part, I used the function I wrote for Homework 10. Upon reading the image, I first convert the image to double and then into a binary image in order to perform morphological operations.

I check if the background is black, since it is hard to apply the Hough transform to images with a black background. If it is black, then I use the erosion operation on the image to get rid of the white specks on the black background and then the dilation operation to reduce the amount of black text on the white paper. I then use the bwlabel() function to get the label matrix of the white object (paper) against the black background. Then I find the angle of the white object with respect to the x-axis using the regionprops()

method and the ‘Orientation’ property. Using this angle, I rotate the image so that the white object is upright. Then, again using the regionprops() method and the ‘Boundingbox’ property, I calculate the smallest rectangle bounding the white object to crop out the black background.

Once the black background has been removed, I continue the program the same way as with the other images that didn’t have a black background. I take the complement of the binary image and perform the erosion operation on the image to reduce the amount of black text. I then use the dilation operation to make the Sudoku borders more prominent so that the Hough transform can detect them.

Using the hough(), houghpeaks(), and houghlines() functions, I get a maximum of 30 lines in the image. Once I have these lines, I count the highest number of lines that all have the same length (with some tolerance), under the assumption that these lines belong to the Sudoku. I then check if they are vertical or horizontal.

If they are horizontal, I first find the top and bottom lines of the Sudoku box using the y-coordinates of the lines. Then, for all the other lines that intersect with the top line, I check if they are at a 90-degree angle with the top line and if they are, I find the left and right lines of the Sudoku box using the x-coordinates.

Similarly, if the lines are vertical, I first find the left and right lines of the Sudoku box using the x-coordinates of the lines. Then, for all the other lines that intersect with the left line, I check if they are at a 90-degree angle with the left line and if they are, I find the top and bottom lines of the Sudoku box using the x-coordinates.

ii) Printing the numbers from the puzzle

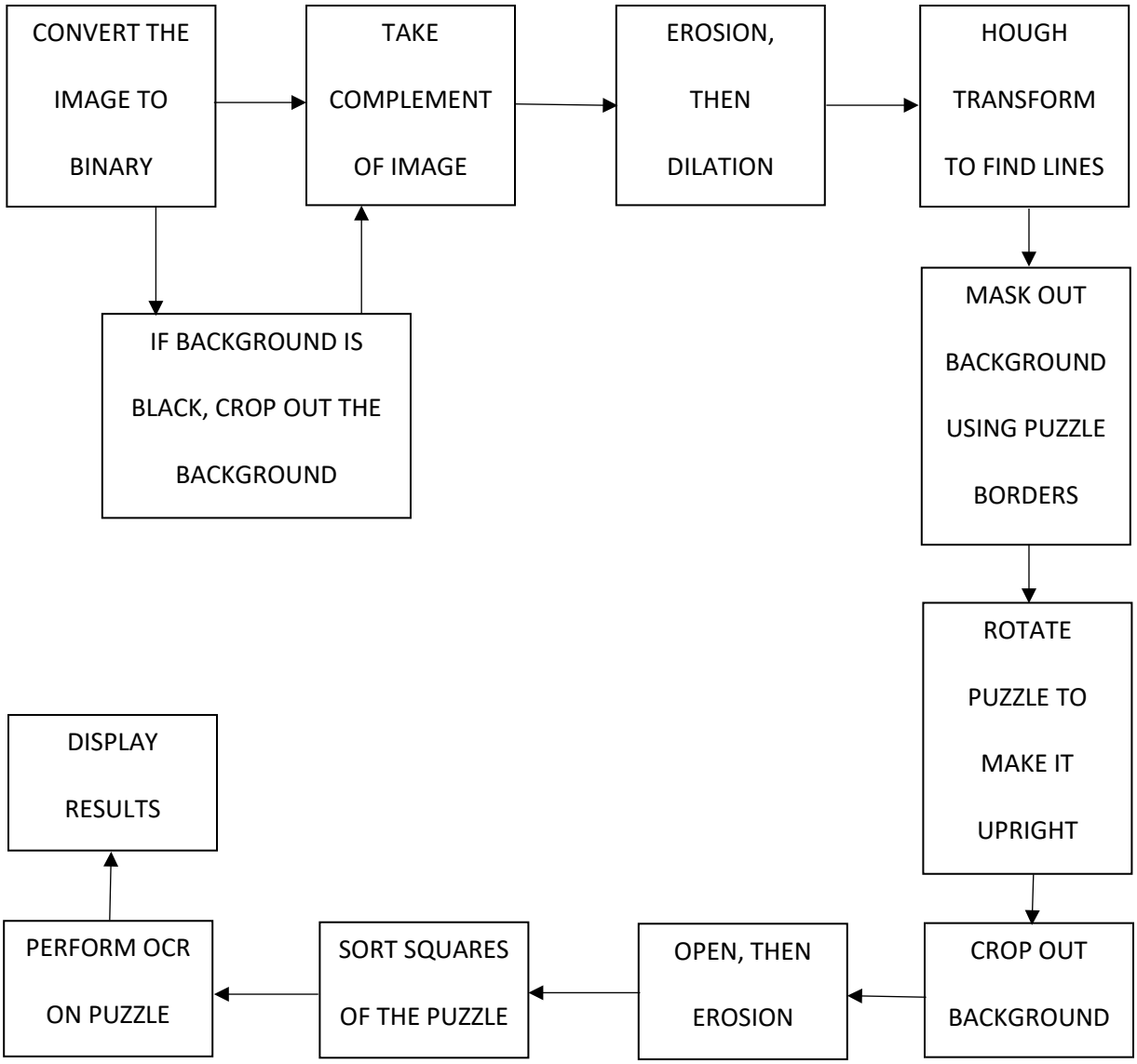
For this part, I used the four lines of the box return by the above function and created a binary mask using the coordinates of the lines. Using the mask, I turned the background pixels to black to get the region of interest, i.e., puzzle.

Once I had just the puzzle, I calculated the angle by which to rotate the image so that the puzzle is upright. After rotating the image (and the mask), I used the regionprops() method and the ‘Boundingbox’ property to calculate the smallest rectangle bounding the white object in the mask and crop out the black background. Then I used the open operation on the new cropped image to make the black dots on the white background go away while not losing the black text, and then the erosion operation to make the black text more prominent. I then used the bwlabel() function to get the label matrix of the image and the number of squares in the puzzle.

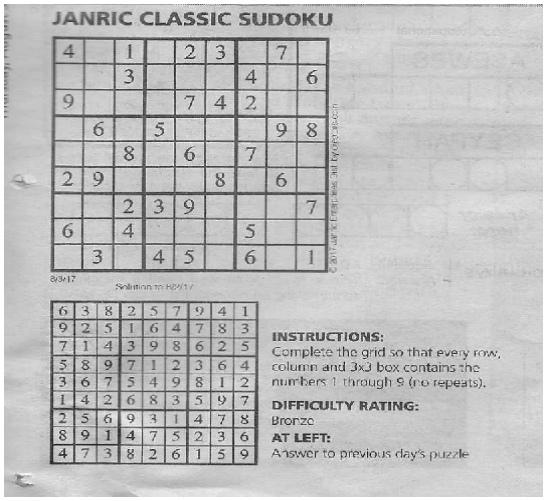
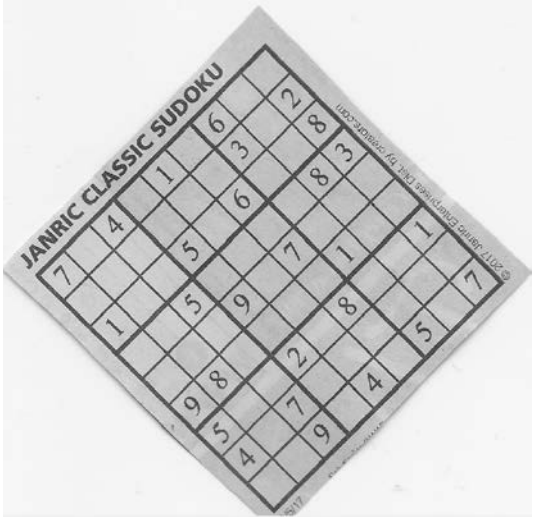
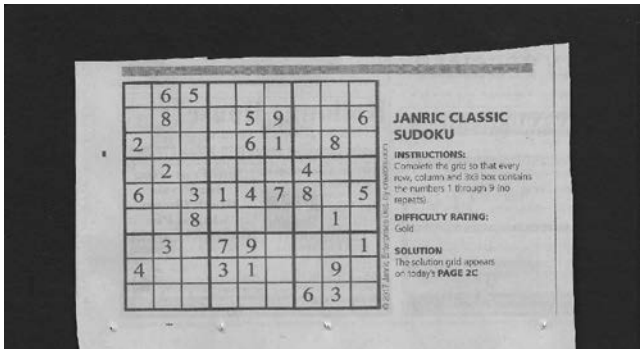
Using the "Extrema" property on the label matrix, I sorted the squares of the puzzle using the left-most-bottom coordinate of each square so that the squares of the puzzle can be traversed in the right order, i.e, from the top left square to the bottom right square [1]. And then using the “PixelIdList” property, I relabeled the matrix [2].

For each individual square in the puzzle, I calculated the smallest rectangle bounding the square and cropped out the rest of the image. Then I shaved off a few pixels from all four edges of each of the squares as a precautionary measure against any remnant noise so that the OCR function is able to detect the right character on the square. Finally, I saved the results of the OCR of each square in a matrix one-by-one and displayed the contents of the matrix at the end.

b) Imaging Chain



c) Results

Puzzle	OCR Output
	4 , 1 , 2 3 , 7 , , , 3 , , , 4 , 6 9 , , , 7 4 2 , , , 6 , 5 , , , 9 , , , 8 , 6 , 7 , , 2 9 , , , 8 , 6 , , , 2 3 9 , , , 7 6 , 4 , , , 5 , , , 3 , 4 5 , 6 , 1
	7 , 4 , 1 , 6 , , , , , , , , 3 , 2 1 , , 5 , 6 , , 8 , , 5 , , , , 8 3 , , , 9 , 7 , , , 9 8 , , , , 1 , , 5 , , 2 , 8 , , 1 4 , 7 , , , , , , , , 9 , 4 , 5 , 7
	, 6 5 , , , , , , , 8 , , 5 9 , , 2 6 , , , 6 1 , 8 , , 2 , , , , 4 , 6 , , 3 1 4 7 8 , , 5 , 8 , , , , 1 , , 3 , 7 9 , , , 4 1 , , 3 1 , , 9 , , , , , , , 6 3 ,

d) Discussion

The most challenging part of this project was tweaking the parameters in such a way that it works for all the images, not just one, such as different morphological operations, as well as the amount of tolerance when finding lines of the puzzle.

While this code does work for most images, in the event that the image isn't fully upright even after rotating, the code might output its numbers in the wrong order.

The code also takes care of dupes such as when the OCR function mistakes a "1" for an "l" or an "I", or a "5" for an "S". This works since once a square in the puzzle is isolated, it is either empty or has a character in it and the character must a number.

4. Conclusion

In this project, I've learned how to use the Hough transform in Matlab using functions like hough(), houghpeaks(), houghlines(), etc. to identify lines in the image and how different parameters in these functions can affect the output. Different parameters for the morphological operations affect the images differently if the size of the Sudoku puzzle differs between images.

I've also learned how to change the search order of the bwlabel() function for consistency, since the function doesn't always return its components in the same order.

5. Credits

The articles that I read to understand about sorting and relabeling a label matrix:

[1] https://blogs.mathworks.com/steve/2008/03/25/bwlabel-search-order/

[2] https://blogs.mathworks.com/steve/2008/04/14/relabeling-a-label-matrix/