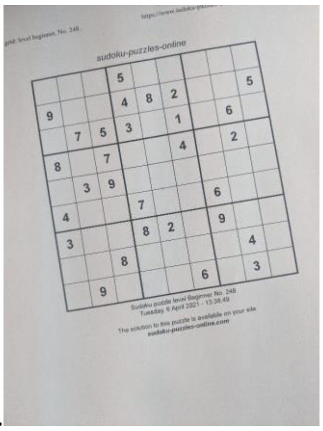
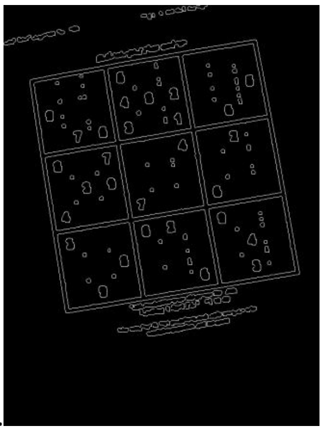
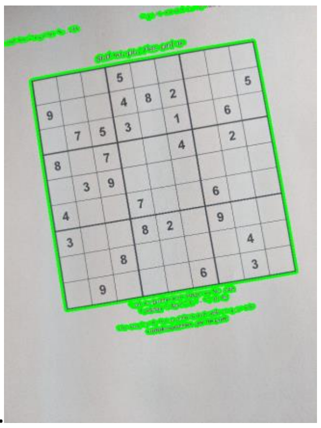
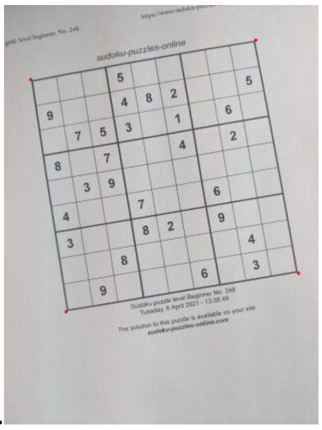
Extract information from sudoku grids project

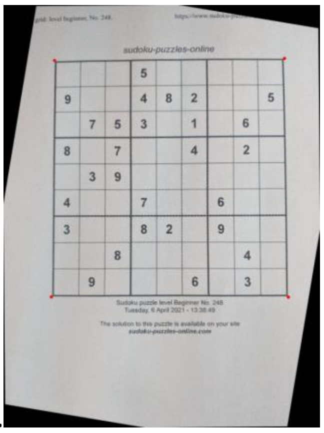
Step 1: Image preprocessing

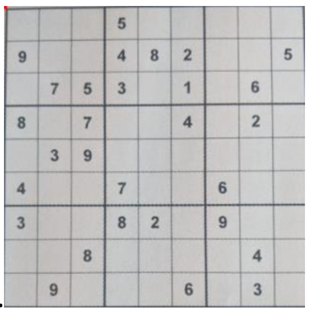
1.


2.


3.


4.


5.


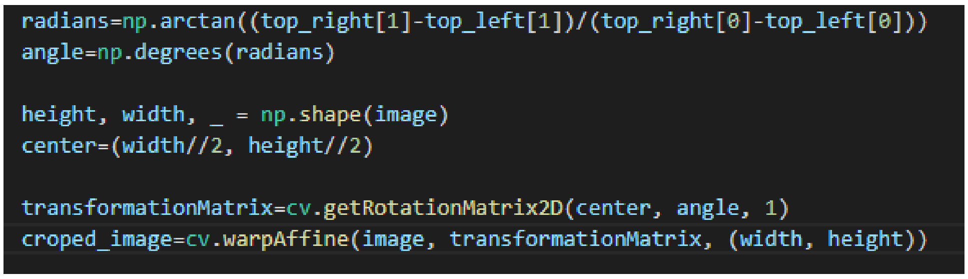
6.


First apply Canny filter to detect edgels (figure 2) and then use **cv.findContours** (figure 3) to find the contours in the image. Search for the biggest contour in your image and that will be your sudoku grid. After you find the corners of the sudoku grid, use them to rotate the image with the help of 2D affine transformations already implemented in openCV (figure 5). The following code uses openCV to get the affine rotation matrix (figure 7) with the center of the image as center and angle determined by the direction of the top line, then applies it to the image.

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot \texttt{center.x} - \beta \cdot \texttt{center.y} \\ -\beta & \alpha & \beta \cdot \texttt{center.x} + (1-\alpha)\cdot \texttt{center.y} \end{bmatrix}$$

7.

$$\begin{array}{l} \alpha = \texttt{scale} \cdot \cos \texttt{angle}, \\ \beta = \texttt{scale} \cdot \sin \texttt{angle} \end{array}$$



```
radians=np.arctan((top_right[1]-top_left[1])/(top_right[0]-top_left[0]))
angle=np.degrees(radians)

height, width, _ = np.shape(image)
center=(width//2, height//2)

transformationMatrix=cv.getRotationMatrix2D(center, angle, 1)
croped_image=cv.warpAffine(image, transformationMatrix, (width, height))
```

<u>Step 2: Get patches and results:</u>

- Write get_patches function

Write a function to decompose the sudoku grid into 81 smaller squares.

- Write get_results function

Use **cv.getContours** to get contours of the patch. Iterate through contours and get the min bounding rectangle of each, if the rectangle has the width in the range of (5,20) and height in the range of (5,30), then there it must be a digit, otherwise the patch is empty. Cut the bounding rect and use the masks given and **cv.MatchTemplate** to predict the digit. The mask were extracted from the training images.

<u>Step 3: Get zones of the jigsaw grids</u>

- Jigsaw color

Implement class **patchColorJ** where every object stores the color of the patch and its neighbors. Then check recursively if its neighbors have the same color and attribute the same zone to patches with the same color in the same neighborhood.

- Jigsaw black and white

Implement class **patchJ** where every object stores the walls of the patch, in respect to the bold lines in jigsaw grid, and its neighbors. Then, similarly to **patchColorJ**, get the zones recursively using the walls. This method works for the colored jigsaw grids as well but with lower accuaracy.