Sudoku Solver

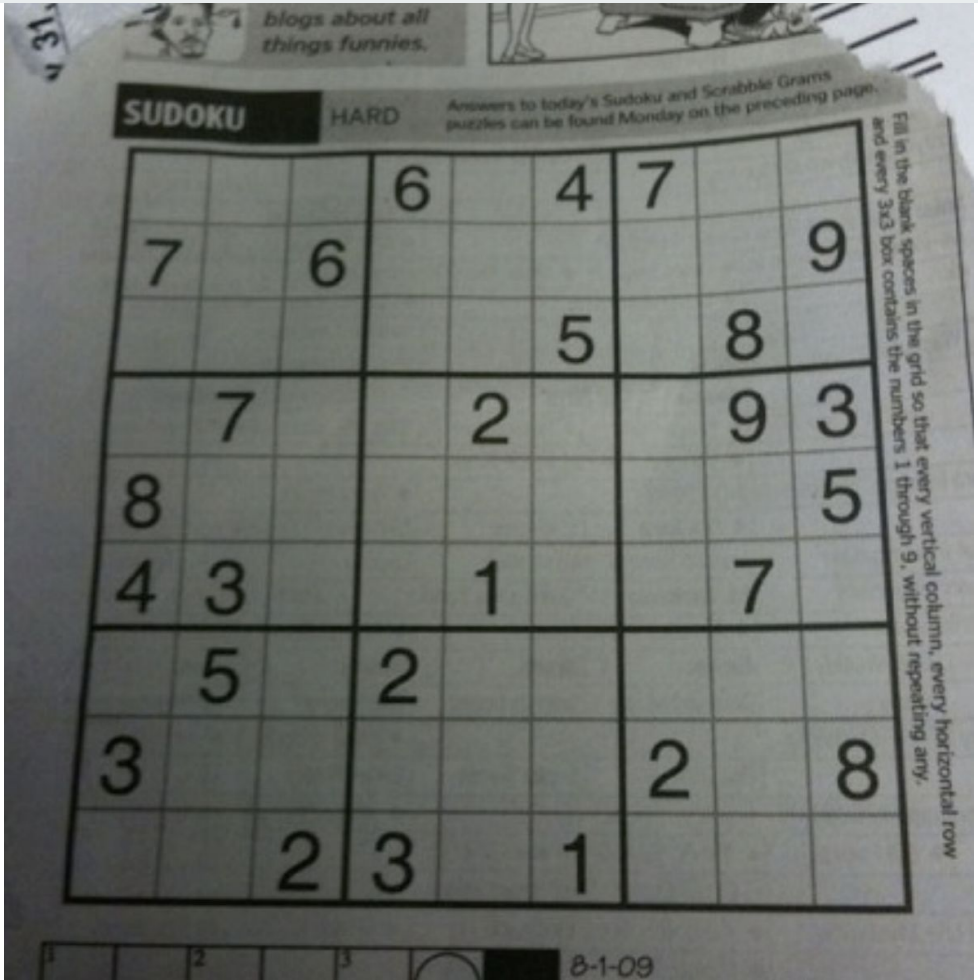
2D Computer Vision

Kevin Castorina
Vlad Bratulescu
Marco Mollo

Agenda

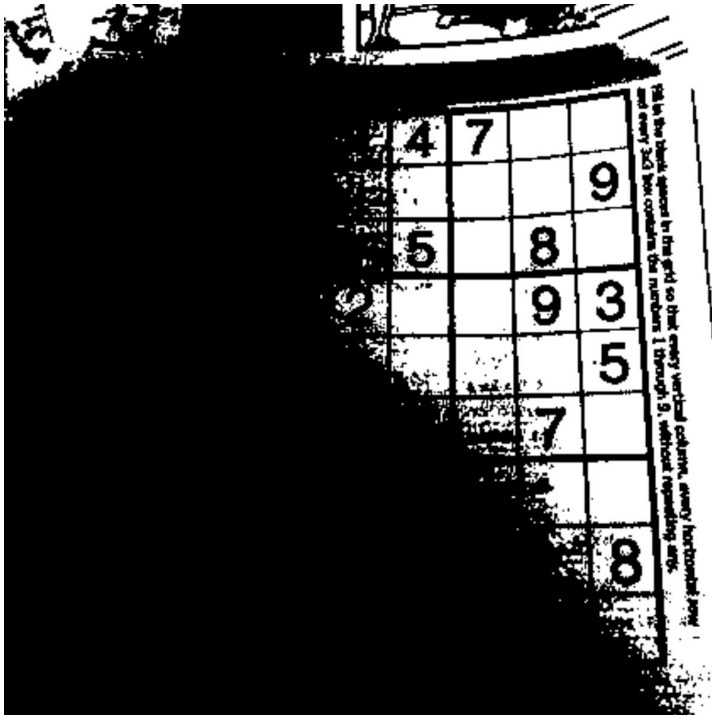
1. Zielsetzung
2. Vorgehensweise
3. Vorführung
4. Ausblick

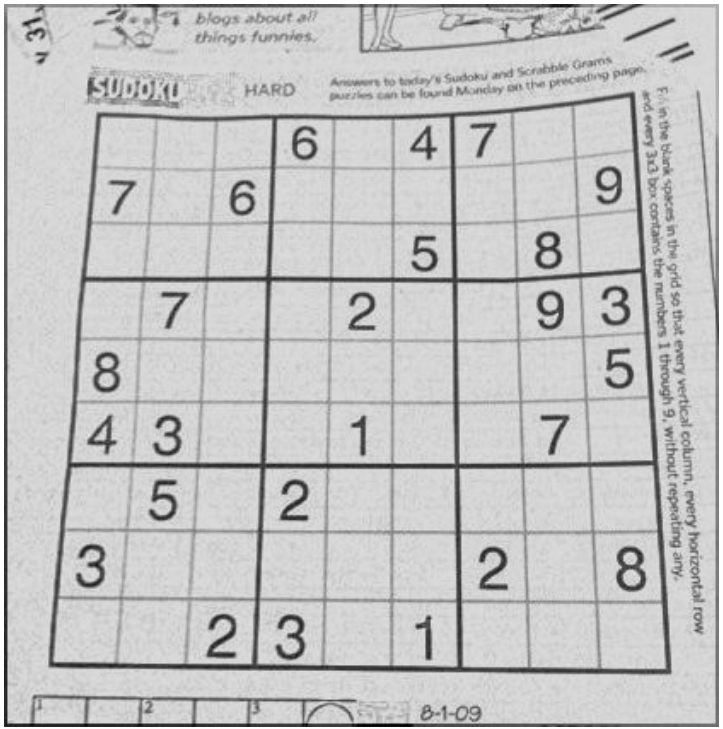
Zielsetzung


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.
6 4 7
7 6 9
5 8
7 2 9 3
8 5
4 3 1 7
5 2
3 2 8
2 3 1
8-1-09

Vorgehensweise


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





Adaptive thresholding

```
element = np.array([
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
])
morphologic_closing_img = closing(img, element)
adapted_threshold_img = img / morphologic_closing_img * 0.85
```

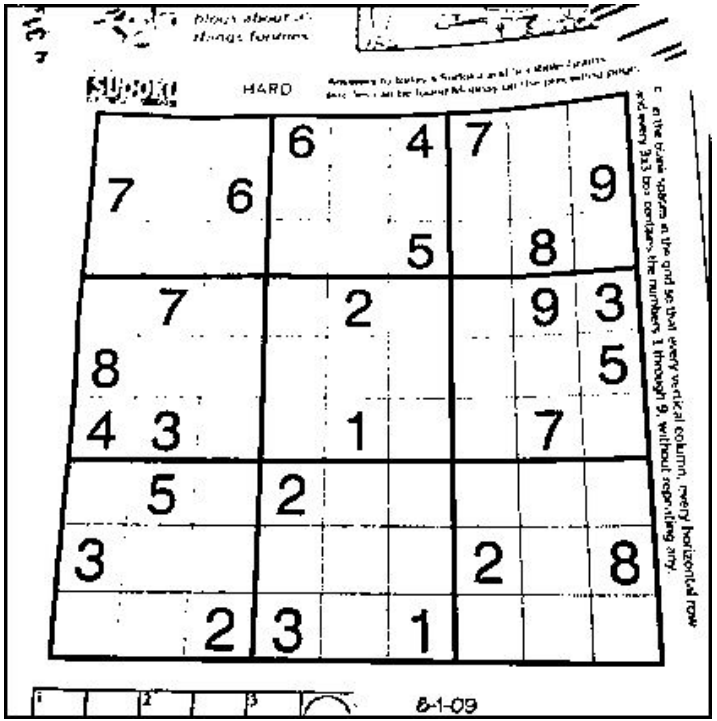


Bild binär machen

```
adapted_threshold_img_binary = make_img_binary(adapted_threshold_img, 0.6)

def make_img_binary(img, threshold):
    _, bw_img = cv2.threshold(img, threshold, 1, cv2.THRESH_BINARY)
    return bw_img
```

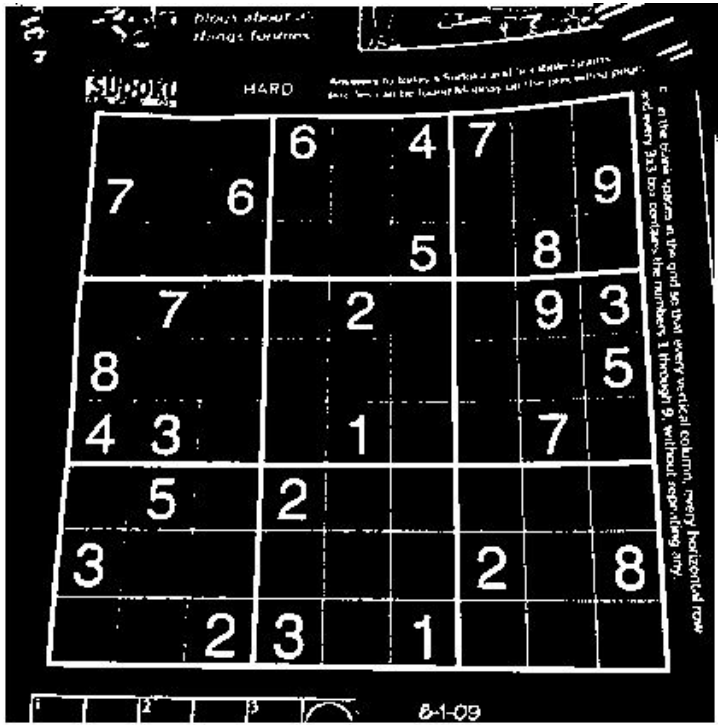
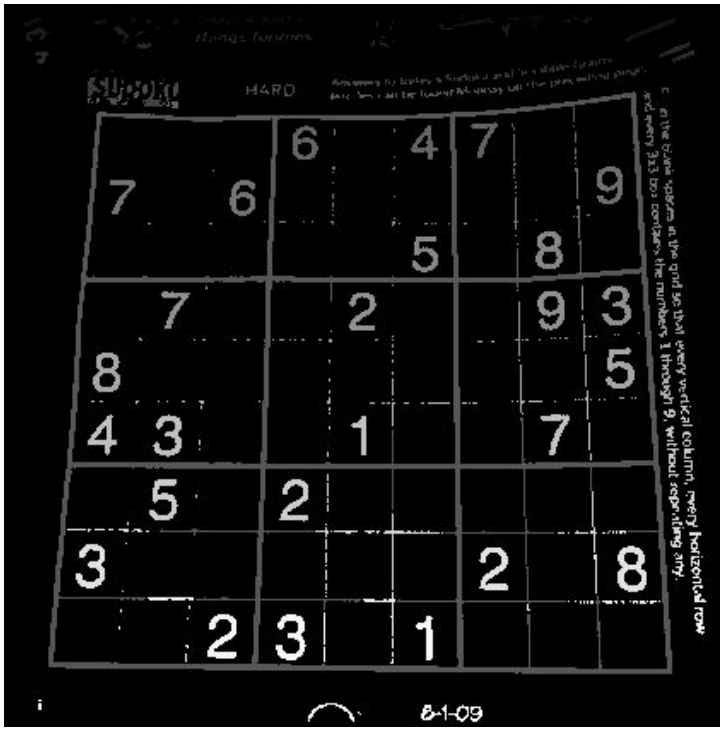


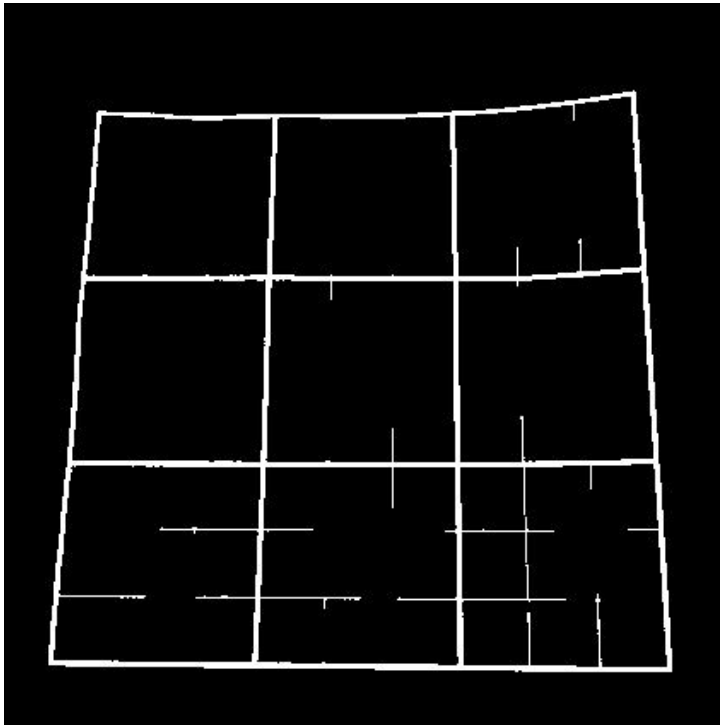
Bild invertieren

```
inverted_img = invert(adapted_threshold_img_binary)

def invert(img):
    height, width = np.shape(img)
    inverted_img = np.zeros((math.ceil(height), math.ceil(width)))
    for x in range(0, height):
        for y in range(0, width):
            inverted_img[x, y] = 255 - img[x, y]
    return inverted_img
```



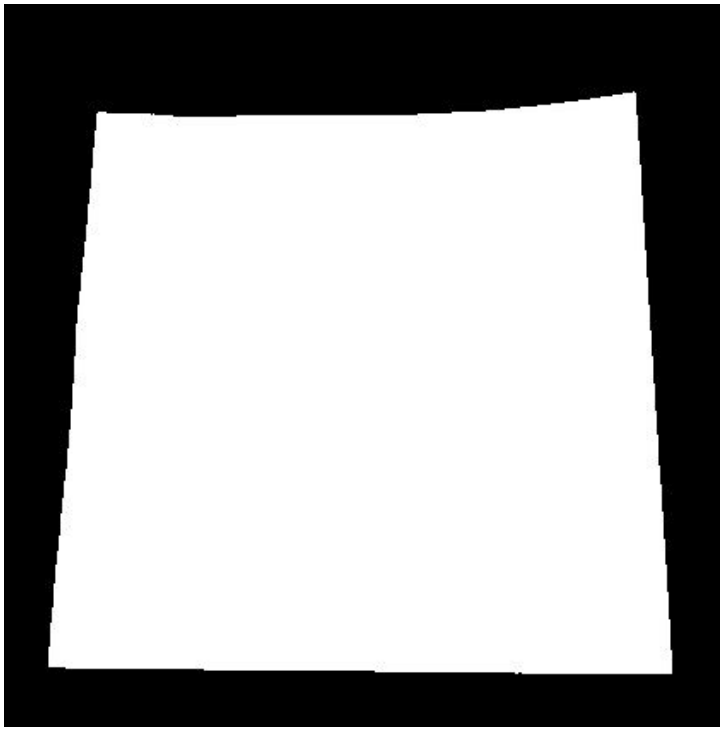
Regionen kennzeichnen

```
flood_filled_img = region_labeling(inverted_img, 1)

def region_labeling(img, region_value):
    m = 2
    copy = np.copy(img)
    for x, row in enumerate(copy):
        for y, value in enumerate(row):
            if (value == region_value):
                flood_fill(copy, x, y, m, region_value)
                m += 1
    return copy
```



Rahmen heraus filtern

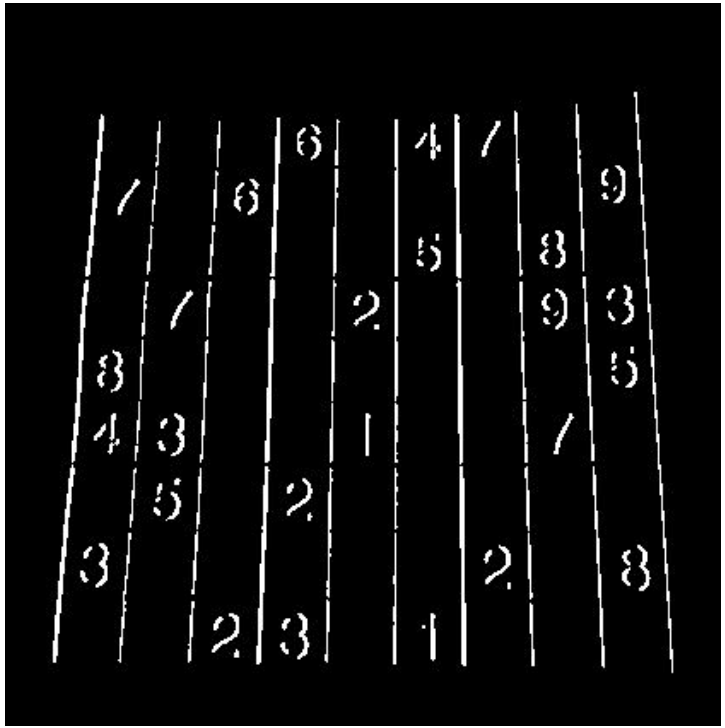
```
sudoku_board_img = find_biggest_blob(flood_filled_img)

def find_biggest_blob(img):
    biggest_blob = np.copy(img)
    unique, counts = np.unique(biggest_blob, return_counts=True)
    dict = np.asarray((unique, counts)).T
    # Get rid of label 0, which is the background
    dict = dict[1:]
    # Sort the label by occurences
    most_frequent_label = sorted(dict, key=lambda t: t[1])[-1][0]
    biggest_blob[biggest_blob != most_frequent_label] = 0
    return biggest_blob
```

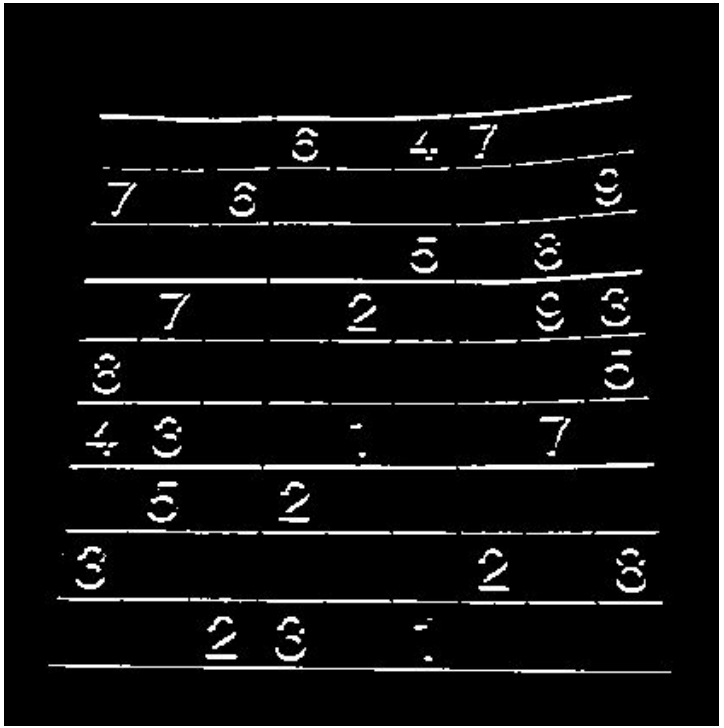


Maske bekommen

```
mask = get_mask(sudoku_board_img)

def get_mask(img):
    mask = region_labeling(img, 0)
    unique, _ = np.unique(mask, return_counts=True)
    first_label = unique[0]
    mask[mask != first_label] = 255
    return mask
```

Anwendung der Sobel-Operatoren

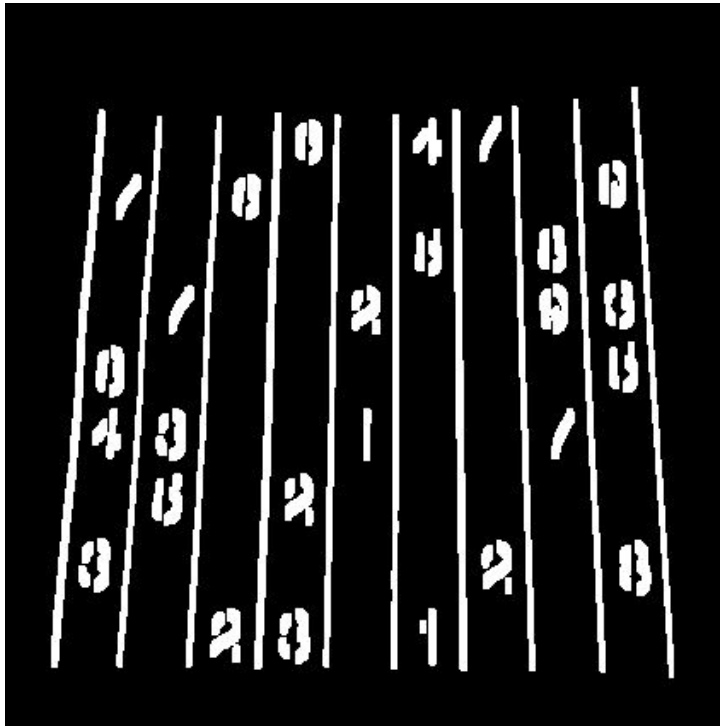


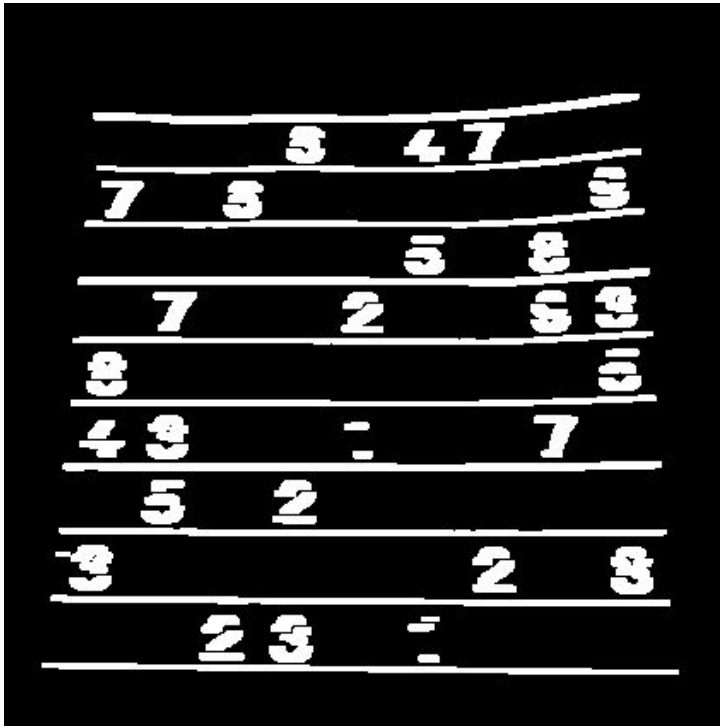


Anwendung der Sobel-Operatoren

```
s_vert, s_hor = sobel_operator()
sobel_vertical_img = apply_sobel_filter(adapted_threshold_img, s_vert, mask)
sobel_horizontal_img = apply_sobel_filter(adapted_threshold_img, s_hor, mask)
```

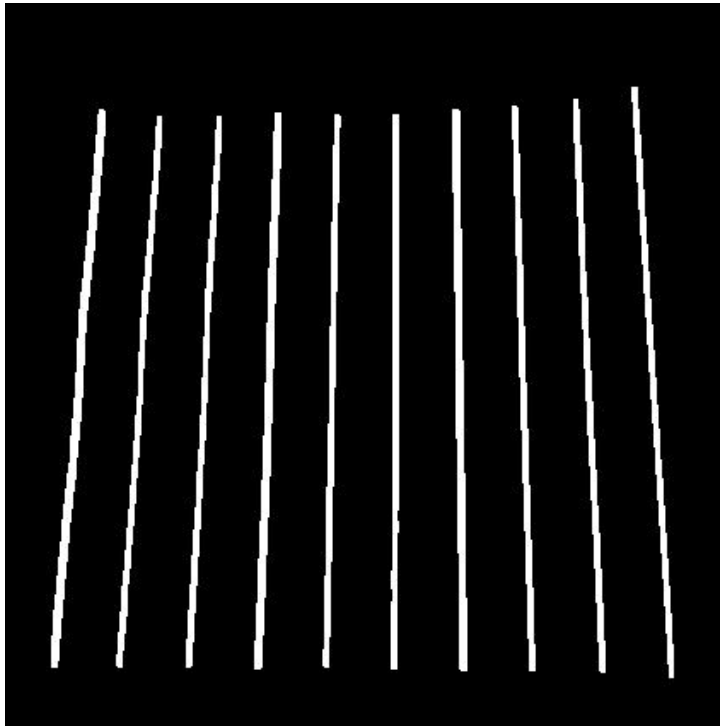
Vertikale und Horizontale Dilation

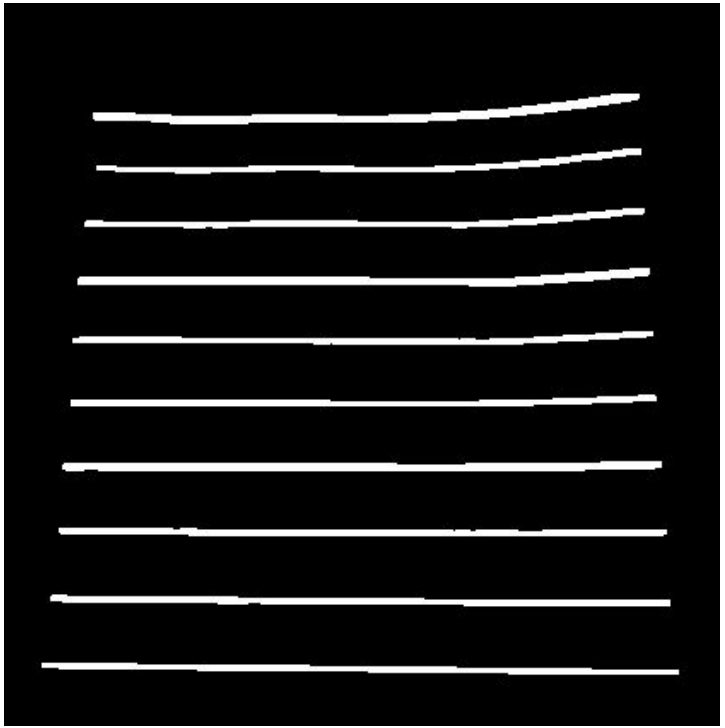


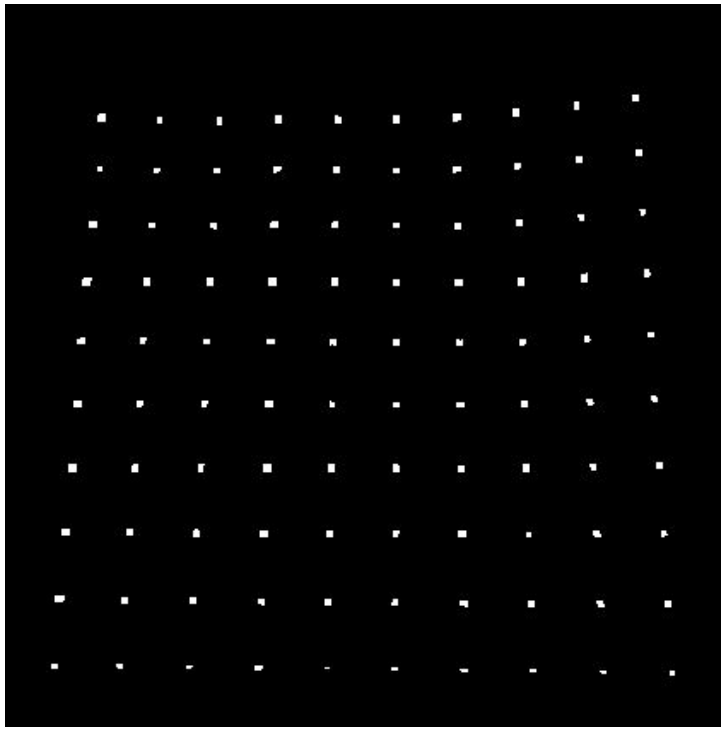


```
structure_element_horizontal = np.array([
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0]])
dilated_horizontal_img = dilate(sobel_horizontal_img, structure_element_horizontal, 1)

structure_element_vertical = structure_element_horizontal.transpose()
dilated_vertical_img = dilate(sobel_vertical_img, structure_element_vertical, 1)
```

Entfernung der Zahlen



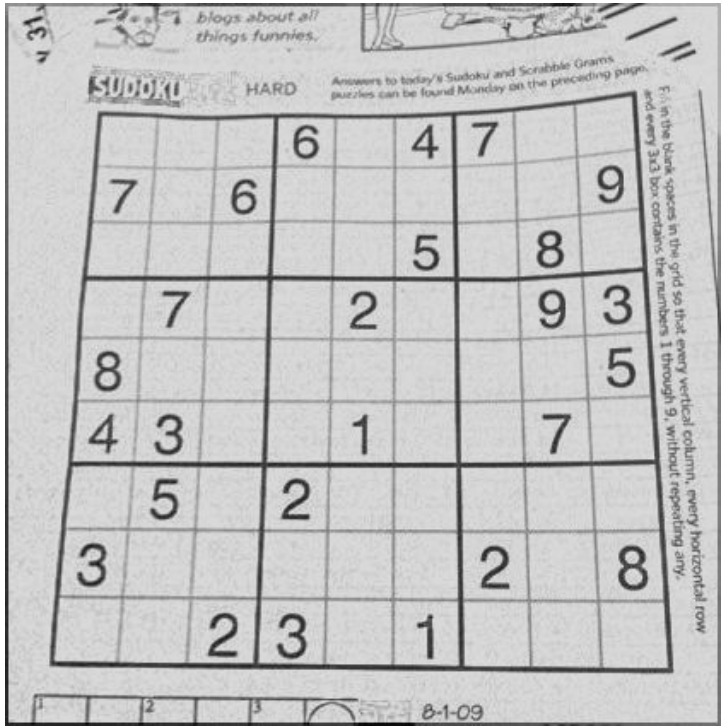




Kreuzungsbereiche

```
intersected_regions_img = cv2.bitwise_and(vertical_img, horizontal_img)
```

Transformierung der Schnittpunkten



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

```
for src_points, dst_points in zip(src, dst):
    matrix = cv2.getPerspectiveTransform(src_points, dst_points)
    warp = cv2.warpPerspective(img, matrix, (450, 450))

    i = 0
    for x in range(int(dst_points[0][0]), int(dst_points[3][0])):
        j = 0
        for y in range(int(dst_points[0][1]), int(dst_points[3][1])):
            board_croped_img[y, x] = warp[y, x]
            j += 1
        i += 1
```

Vorführung

Ausblick

- Alle 100 Schnittpunkte müssen erkannt werden!
- Hough-Transformation oder Harris-Detektor als Alternative
- App zur Bildaufnahme
- Lösung direkt auf Bild zeichnen

Danke für die Aufmerksamkeit!