# An Optical Sudoku Solver

Martin Byröd

February 12, 2007

**Abstract**

In this report, a vision-based sudoku solver is described. The solver is capable of solving a sudoku directly from a photograph taken with a standard digital camera. Given a photograph of a sudoku, the software finds the sudoku, interprets it, solves it and then projects the solution onto the original photograph. The solver can handle images of sudokus of varying appearance such as different fonts and different lighting. The solver can also handle rotated sudokus and perspective distortion.

## 1 Introduction

The sudoku is a popular number game played on a 9 by 9 grid like this

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

The objective is to fill all the blank squares with the digits 1 to 9. The rules are simple; When the sudoku is completed, each row and

column has to contain all digits from 1 to 9 exactly one time, i.e. no row or column is allowed to contain for instace two fives. In addition, each subgrid of size 3 by 3 also has to contain all digits from 1 to 9 exactly once.

Sudokus can be found in almost all newspapers and magazines these days and they have been hugely popular for a couple of years now. Some sudokus are easy to solve by hand and some are furiously difficult.

The purpose of this report is to describe an automatical, vision-based sudoku solver. The solver is capable of solving a sudoku directly from a photograph taken with a standard digital camera. Given a photograph of a sudoku, the software finds the sudoku, interprets it, solves it and then projects the solution onto the original photograph.

# 2 An Overview of the System

The sudoku solver solves the problem of optical sudokuing in a number of steps. In this section, an overview of the complete chain from JPEG sudoku-image to JPEG solved-sudoku-image is given. To perform this, the necessary steps are: Finding the sudoku grid, interpreting the numbers, solving the sudoku and projecting the solution onto the photograph. In a little more detail, the sequence is as follows (also illustrated in Figure 1):

1. **Preprocessing:** To make the sudoku image more computer friendly, it is first down-sampled, then filtered and finally thresholded to produce a binary image (consisting of black and white pixels only). The filtering consists of blurring, which removes noise and highpass filtering which equalizes the intenisty throughout the image.
2. **Finding the squares:** The interpretation part of the solver is based on finding connected components in the image. There are well known and fast algorithms for doing this which makes it an appealing approach. The idea is that most of the connected components in the image will be sudoku squares. The output of this step is a list of component centers.
3. **Finding the grid:** Given the list of component centers, this step tries to find the grid and estimate the transformation (rotation, translation and perspective distortion) applied to it.
4. **Correcting perspective distortion:** Using the transformation estimated in the previous step, the image is inverse-transformed to form an upright frontview of the sudoku with known coordinates of each square.

5. **Interpreting the digits:** Now that the coordinates of all the grid points are known, the sudoku is cut into 81 squares, each of which is classified as a digit between one and nine or as an empty square. The classification is done using a nearest-neighbor method based on feature extraction.
6. **Solving the sudoku:** In comparison to the interpretation, the step of solving the sudoku is relatively straight forward and quite similar to how most humans solve sudokus. Finally, an image of the solution is created and projected onto the original image.

# 3 Finding and Rectifying the Grid

The interpretation part of the solver is based on finding connected components in the image. By connected components are meant regions of white pixels in the image where you can get from one part of the region to another by jumping from one white pixel to another. There are well known and fast algorithms for doing this, which makes it a fast and easy base for the grid finder. The idea is that most of the connected components in the image will be sudoku squares. Obviously, there might be lots of false components. To get as many correct and as few false components as possible, only components of plausible size are kept. The output of this step is a list of component centers.

Based on the list of component centers, a ransac-like method tries to fit lines to these points. Once a given number of mutually orthogonal lines have been found, an intersection point of two of these lines is selected. Around this point four more points are selected as illustrated in Figure 3. Based on these points, a homograph map in the plane is estimated which takes these five points to the points $(0, -2), (2, 0), (0, 2), (-2, 0)$ and $(0, 0)$. This hopefully results in a reasonably rectified grid with respect to scale and angles but with unknown translation. Now, a grid-structure of 81 points is fitted to the rectified grid to estimate also the translation of the grid. To measure the quality of the fit, the number of inliers is counted. If this number is above a pre-specified threshold, the fit is considered succesful.

In this case, all the inliers are used to re-estimate the homography map, yielding a better estimate. The mapping is constructed so that it takes the sudoku grid onto the square $[-4.5, 4.5]$. Using this mapping the whole sudoku image is rectified. If this was done correctly we now know exactly where the sudoku is, which means that we can cut out the 81 small sudoku squares containing the essential sudoku information we need for the next step.

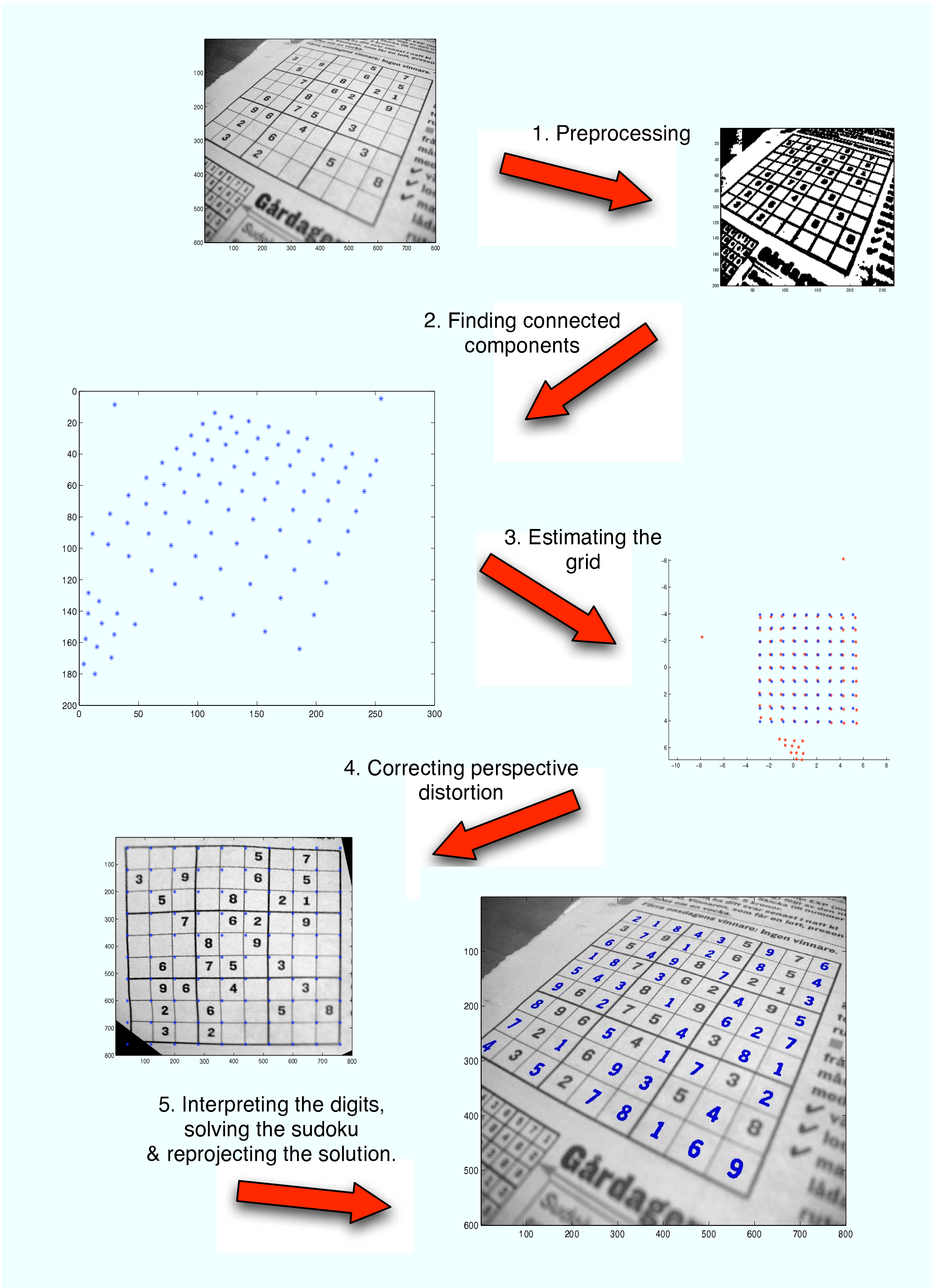


Figure 1: An overview of the different parts of the sudoku solver.

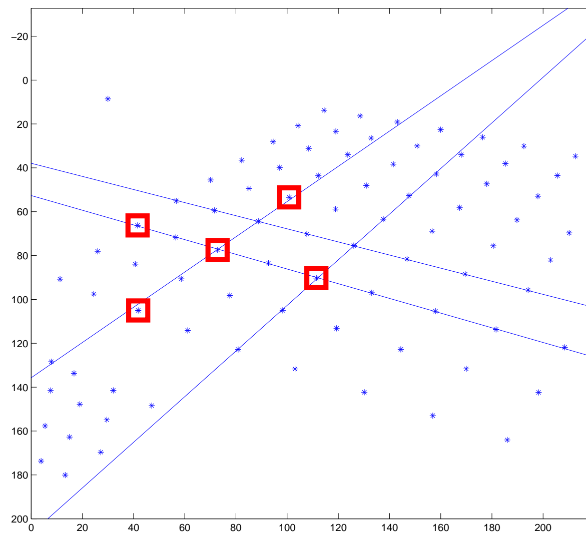


Figure 2: Points selected to estimate the homography mapping. First a few lines are fitted to the component centers. Then two of these lines which are orthogonal are chosen and the intersection and the neighbors of the intersection (two steps away) are selected.

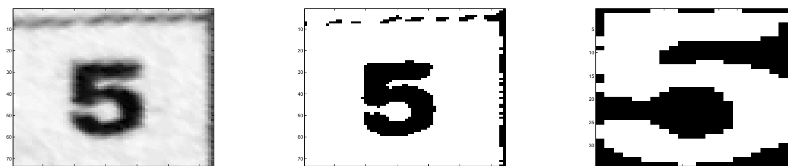

Figure 3: A cut out image of a digit is to be classified (left). First the image is normalized and thresholded (middle). Thereafter all connected components (of black) are found. The segment most likely to be a digit is selected (right).

# 4 Interpreting the Digits

The cut out squares (as described in the previous section) are classified as either empty or as one of the digits 1 to 9. This is done using a nearest neighbor method with some modifications to make it more stable. The process is roughly as follows: First the image is normalized and thresholded. Thereafter all connected components (of black) are found. The segment most likely to be a digit is selected and a number of features based on this segment are calculated. Examples of features are: proportion of black pixels, number of holes in the segment, a few statistical moments, symmetry features, etc.

To classify the digits, the solver has a relatively large set of example digits. The same features have been calculated for these example digits as well and the digit is classified by selecting the most similar example digit based on these features. The similarity is calculated by taking the weighted 2-norm of the difference between the unknown digit and the example digit. The weights are based on the variances of the features over the example digits.

The digit in Figure 4 has features as given in Table 4

## 4.1 Determining the Orientation

The grid finder described above will only be able to estimate the grid up to orientation, i.e. up to rotations of 90 degrees. This is because the sudoku grid is symmetric under such rotations. In order to estimate the orientation, the digits have to be used. Therefore, each example digit is copied into four instances, one of each orientation. This means that digit and orientation can be classified at the same time. The global orientation is then chosen which gets the most ”votes” from the individual digits. However, this means that the classification of the digits themselves is not optimal. To compensate for this, the classification is re-run once the orientation is determined.

| | |
|---|---|
| Proportion of black pixels | 0.537 |
| Maximal row sum | 0.8750 |
| Maximal column sum | 0.7778 |
| Number of alternating rows | 0.250 |
| Number of alternating columns | 0.8750 |
| Number of holes | 0.0 |
| $x^2$ moment | 0.2832 |
| $y^2$ moment | 0.3139 |
| $xy$ moment | 0.0334 |
| $x^3$ moment | 0.0095 |
| $y^3$ moment | 0.0099 |
| x symmetry | 0.7866 |
| y symmetry | 0.6875 |

Table 1: Features for the digit shown in Figure 4.

### 4.2 Classifying Sixes and Nines

One thing which turned out to be tricky was to tell the difference between sixes and nines. The reason for this is that they are identical (in most fonts) up to a rotation of 180 degrees. The $x^3$ and $y^3$ moments measure assymetry and are the only features which can differ between sixes and nines. Therefore, whenever a digit is classified as a six or a nine, a separate test is performed looking only at these two features. Using this trick, a reasonably accurate classification of sixes and nines was obtained.

## 5 Solving the Sudoku

Once the sudoku has been accurately interpreted from the image, solving it is what remains. This turns out to be a pretty straight forward problem algorithmically. One approach is as follows (the pencil method full out): Start with a 9 by 9 grid where each square contains all the digits from 1 to 9. In the squares where there are given digits in the initial sudoku, cancel out all digits except the given ones. Now iteratively cancel out digits in rows, columns and subgrids until there is a single digit in all squares and the sudoku is solved. If the difficulty of the sudoku is in the higher range the described method will not always yield a complete solution and there might be uncertainties left. In this case, pick a square containing more than one digit and start a new instance of the algorithm for each possible guess. This can be done recursively many times if needed. Finally, in one branch of the

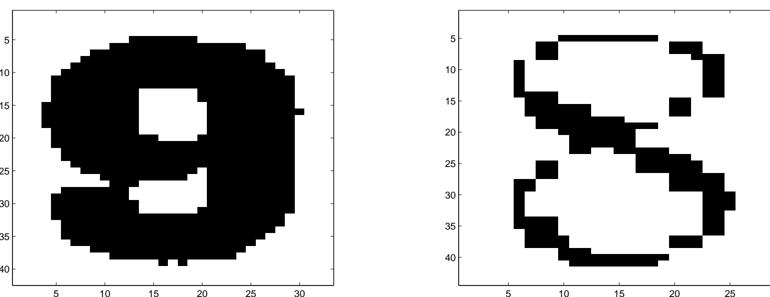

Figure 4: Two problematic digits. The nine is a bit to thick which makes the lines join in one place and form a second hole (left). The eight is to thin and breaks apart as a result of the thresholding (right).

recursion tree, a unique and correct solution will be found if the initial sudoku and the optical interpretation was correct.

# 6 Discussion

The sudoku interpreter and solver described in this report works reasonably well. The most time consuming part of the interpretation turns out to be finding the grid, but this also seems to be the easiest part. In general, the ocr part, i.e. interpreting the digits, is more sensitive. It is hard to get the interpretation stable under varying conditions such as different light, blurring, low resolution, shadows, different fonts, etc. With the features used here, especially the one which counts holes in the digits has turned out to be very important and many false classifications can be ascribed to e.g. thin eights breaking up or the lines of thick sixes or nines joining, producing two holes (see Figure 6). This indicates that the latter part is where most work should be put in to enhance the interpretation accuracy.

However, the grid finder is not perfect either and fails in some cases. This is almost always due to problems in the preprocessing. With some more effort and perhaps some more computing time, it should be possible to make this part of the algorithm very stable.

# 7 Conclusion

A functioning sudoku interpreter and solver has been designed, implemented and tested. The system works reasonably well and although it has some problems it works on a quite large number of sudokus

under varying conditions, such as different fonts, different light, different angles and different orientations. The whole process of finding the grid, interpreting the digits and solving the sudoku takes a couple of seconds, which seems acceptable. The implementation was made in matlab and it should be possible to increase the speed by a good amount by porting it to a lower level language lika java or c++.