Visual Sudoku Solver

Image Processing and Pattern Recognition

Christian Ertler (1030970)
Peter Prodinger (0430935)

Graz, February 1, 2014

1 Introduction

The goal of this project was to implement a visual sudoku solver which should be able to locate a sudoku field in a video taken from a webcam and present the solution of it directly on the empty cells in the original image. The application should run in realtime and be capable of solving arbitrary sudokus.

2 Methods

In this section we will describe the methods we used to reach our goals.

2.1 Finding and preparing the sudoku field

First of all, the sudoku field must be localized and prepared for further processing. By means of simplicity we assume that the sudoku field must be the dominant object in the image. Furthermore, it must be clearly outlined by an thick dark frame. Most of the sudokus in papers and magazines fulfill this criteria.
To find the sudoku a contour based segmentation of the image preprocessed by a Canny edge detector is done. A sample output of the Canny filter can be seen in figure **??**. Since a sudoku is always square or rectangular it will appear as a quadrilateral in the image. Therefore, we can just take the biggest convex contour as the contour of the sudoku field.
Most of the time the sudoku will not be presented parallel to image plane so it will be distorted perspectively. In order to simplify the further processing we need to remove the

perspective by finding a homography between the found contour and a proper square. Since the contour consists of many points we need to reduce it to the 4 corner points of the quadrilateral. For that purpose we implemented an algorithm from [?] which approximates the best fitting quadrilateral to a polygon.
We also tried to use a Hough-Transformation to find those points but the other approach appeared to be more robust.
Having the corner points the homography can easily be computed and applied to the image to perspectively unwarp the sudoku field. Additionally the field is scaled to a fixed size to avoid scale issues. Figure **??** shows the unwarping.

2.2 Digit Segmentation

Now that the image of the sudoku field is present without distortion at a fixed size we can apply a grid to it to roughly divide it into the single cells. In each cell we have to decide if it contains a digit and segment it, respectively.
The segmentation of the digits is illustrated in figure **??**. The coarse segmentation is done by applying a gaussian blur to remove some noise followed by an adaptive thresholding ([?]). In contrast to an ordinary threshold, the threshold value varies locally depending on the mean value of the pixel intensities in the neighborhood around the pixel.
Although this operation works pretty well there is often the problem that parts of the grid lines lie inside the image of the cell. Since these lines have approximately the same thickness as the digits they can not be removed by the thresholding. To overcome this issue we assume that a small region in the middle of the image does only contain pixels belonging to the digit. Starting from these pixels we start a a connected component search to find the rest of the digit. All other pixels will be discarded.

2.3 Digit Classification

In order to be able to find a solution of the sudoku we must classify the found digits. First, we were thinking of using an existing OCR framework for this task. Finally we decided to build our own classifier to have more influence on the classification.
We evaluated three different types of classifiers for this task: *K-Nearest Neighbor*, *Support Vector Machine* and a *Neural Network*. Each of them can be enabled during compilation of the application. We have chosen to use the Support Vector Machine since it yielded the best results. In section 4 we will discuss the performance of them in detail.
Although there would be the possibility to compute descriptors of the segmented digits (eg. Histogram of Oriented Gradients) we decided to use the plain binary image data as input to the classifier.
The only preprocessing steps for the training and classification are:

Deskewing: This operation makes the classification more robust for different fonts and for handwritten digits in future. This is done by calculating the central moments of the pixels. The skewness is given by the third central moments, which is used to construct a affine transformation matrix for deskewing.

Crop: Since there is a lot of unimportant background in the digit images, we crop the image to only include as much background as required. This is done by finding the bounding box of the digit contour.

Rescale: The support vector machine requires all training and test data to have the same dimensionality. After cropping the image, there is no guarantee, that all resulting images will have the same size. We overcome this issue by simply rescaling all input images for the Support Vector Machine to size 16×16. Therefore, every input vector has 256 dimensions.

Principal Component Analysis (PCA): All of the three classificators are able to perform a PCA prior to the training (and consequently, also prior to the classification). However, at this time there is no possibility to save and load the PCA, respectively. So this feature is disabled by default.

2.3.1 Training data

The training data for the classificator can be extracted directly from the application. It provides a gui, which provides the user with a possibility to label the training images by hand. The application is then able to save the extracted training data to the file system. The file system structure reflect the labeling.

2.4 Sudoku Solving

2.5 Presenting the Solution

The projection of the solution onto the original image is not a big deal, since we already found the homography as described in section 2.1. Therefore we can draw the solution on the rectified imaged and apply the homography inversely.

3 Implementation

The application was developed using OpenCV and Qt. No further libraries were used. The most important parameters of the implementation can be found in *include/settings.hpp*. Since these parameters are tweaked to be best for the test system, it may be necessary to adjust them if you are trying to run the application on other webcams.

4 Evaluation/Results

4.1 Field Extraction

In most cases, the field extraction works very well. The performance of it highly depends on the parameters *CANNY_LOW* and *CANNY_HIGH*. Currently, they are tweaked to yield the best results for the images delivered by the webcam of the test system.

However, the application is currently not able to distinguish between contours of sudoku fields and contours of other objects. The only assumptions that were made are:

- The sudoku field is the most prominent object in the image, therefore the biggest contour belongs to the sudoku.
- The contour needs to be convex, so all non-convex contours are discarded.

If those properties are given within the image, the segmentation of the sudoku field works very well.
If the motion of the sudoku field in the video is very high, there will be no closed contour of it, and therefore, the contour will not be recognized. However, the application does not reset everything because the sudoku was not found in one frame. Instead it waits for 10 frames to decide wether the sudoku is gone or not.

4.2 Digit Segmentation

good for tested fields could fail for thin fonts

4.3 Classification

hit rate of 0.99 using k-fold cross validation on training data for new fonts: a few pics improves significantly

4.4 Sudoku Solver

5 Discussion

Blablabla