Automated Sudoku Solver

Marty Otzenberger
EGGN-510
December 4, 2012
Outline

• Goal
• Problem Elements
• Initial Testing
• Test Images
• Approaches
• Final Algorithm
• Results/Statistics
• Conclusions
• Problems/Limits
• Future Work/Improvements
• References
• Questions
Goal

From an image of a Sudoku puzzle, extract and solve the puzzle, and display the solution over the top of the puzzle.
Problem Elements

• Extract the puzzle from the image.
• Identify the numbers in each cell.
  – Correlate the numbers to their respective cell position.
• Solve the puzzle.
• Display the solution over the image.
Initial Testing

• Began by using an ideal digitally fabricated image to reduce complexity.
  – Chose the Wikipedia image for Sudoku.

• Used a Hough Transform to extract the lines in the puzzle, and segmented the image based on the rho values.
  – Parameters very sensitive to capture only one Hough line per puzzle line.
  – Used averages if multiple existed close together
Initial Testing cont.

• Cropped image between each set of rho values, and performed Normalized-Cross-Correlation on each sub-image.

• I saved the max score for a cross-correlation of each sub-image with all 9 template digits, and then used the max of those peak scores to identify the digit.
  – Used the total number of black pixels in a square as a threshold for determining if it was empty.
  – Cropped the template images out of the puzzle.
  – Processed sub-images in order to preserve location information.
Initial Testing cont.

• Solved the puzzle using a MATLAB script I found which recursively solves for possible values for every blank cell in the puzzle[1].
• Used Hough line rho values to re-project the solution onto the image using MATLAB text.

Test Images

• Want to expand code to handle real images of puzzles taken from a variety of angles.
• Took a series of 25 test images of 25 different puzzles to test with.
  – All images taken from the same Sudoku book to keep digits in a common font.
• Intentionally took some extreme images to test the robustness of my algorithm.
Test Images cont.
Approaches

• Initially tried to again use Hough Transforms, but had difficulty.
  – Curvature in the lines caused finding a single line difficult without human intervention on every image.
  – Even extraction of the puzzle boundary was difficult using Hough.

• Found connected components helpful to extract puzzle region by looking for objects of the right size.
  – Still had difficulty finding lines.
  – Could have caused problems if the images were at different zoom levels.
Final Algorithm

• Use MATLAB `cpselect` tool to manually identify the corners of the puzzle, and project them onto a template image of a square.
Final Algorithm cont.

• Used a projective transform which preserves quadrilaterals through scaling, rotation, and translation.
  – This made the puzzle the same size and shape in every image.
  – Also reduced the size of the images to speed up processing.
• Next used `regionprops` to find connected components and extract the puzzle by looking at the component’s width, and height.
  – Absolute pixel changes every time because MATLAB does not crop the image when transforming.
• Used the bounding box of the puzzle to crop the image.
Final Algorithm cont.

• After cropping the image to extract the puzzle, I again looked at the connected components to extract the numbers.

  – I then cropped out the region around the centroid of each number to ensure it was larger than my template image and used the same normalized-cross-correlation strategy as before.
Final Algorithm cont.

- Used the absolute pixel value of the digit centroid to identify its row and column within the puzzle.
  - Possible because the projective transform made the puzzle the same size in every image.
- From here I used the same solver to solve the puzzle, and used absolute pixel locations to display the solution on top of the puzzle.
Results/Statistics

• 18 of my 25 test images processed successful.
  – 3 failed due to shadows on the puzzle.
  – 2 failed due to discontinuous borders on the puzzle preventing `regionprops` from finding it.
  – 1 failed due to noise connected to the border of the puzzle.
  – 1 failed due to distortion leading to incorrect numeric classification.

• This is a 72% success rate.
Results/Statistics cont.
Results/Statistics cont.
Conclusions

• Pleasantly surprised by the performance of the algorithm.
• Handled off angle images of the puzzles very well.
• Most issues due noise/thresholding.
Problems/Limits

• Current algorithm requires user interaction.
• Difficulty handling shadows on the puzzle.
• Can only process puzzles using the same font set.
• Cannot handle distortion in the puzzle.
  – This will probably be difficult to fix, particularly without changing away from cross-correlation.
Future Work/Improvements

• Automate corner detection to find projection.
  – Should be able to make process fully automated.
• Look at better thresholding algorithm.
  – Try to eliminate some of the shadow and noise issues.
• Use mean centroid value of digits to align solutions.
  – Sometimes the solutions don’t line up well with the grid because of distortion in the projected image.
• Project the solution onto original image.
  – Re-project the solutions back to the orientation of the original image and show them on the original.
• Implement camera calibration.
  – Could help improve cross-correlation reliability.
Questions?