Sudoku Image Processor

Academic Project - C++ Image Processing

Arraj Kamel

April 21, 2025

Abstract

This document describes the implementation of a Sudoku puzzle solver that processes images of Sudoku games. The system corrects image rotation, extracts numbers from the grid, and either solves the puzzle completely or suggests the next move. The implementation uses C++ with OpenCV for image processing and Tesseract OCR for number recognition.

Contents

1 Project Overview

1.1 Project Statement

You are given an image of an uncompleted Sudoku game. The image can be rotated or the illumination conditions might not be ideal. Detect the correct rotation of the game and rotate the image. Extract the numbers from the grid and store the state of the game in an object. Solve the game or show a hint for the next move.

1.2 Objective

Develop a system that:

- Accepts an image of a Sudoku puzzle (possibly rotated or with poor lighting)
- Corrects image orientation
- Extracts the puzzle state
- Solves the puzzle or suggests next moves

1.3 Key Components

1. Image preprocessing
2. Grid detection and alignment
3. Number recognition
4. Puzzle solving logic

2 System Architecture

2.1 Module Breakdown

Image Processing Module Handles rotation correction and illumination normalization

Grid Recognition Module Identifies Sudoku grid boundaries

OCR Module Extracts numbers from cells

Solver Module Implements game logic

3 Detailed Implementation Steps

3.1 Step 1: Image Preprocessing

- Convert input image to grayscale using OpenCV's `cvtColor()`
- Apply Gaussian blur with 5x5 kernel to reduce noise (`GaussianBlur()`)
- Use adaptive thresholding (`adaptiveThreshold()`) to handle lighting variations
- **Output**: Cleaned binary image ready for edge detection

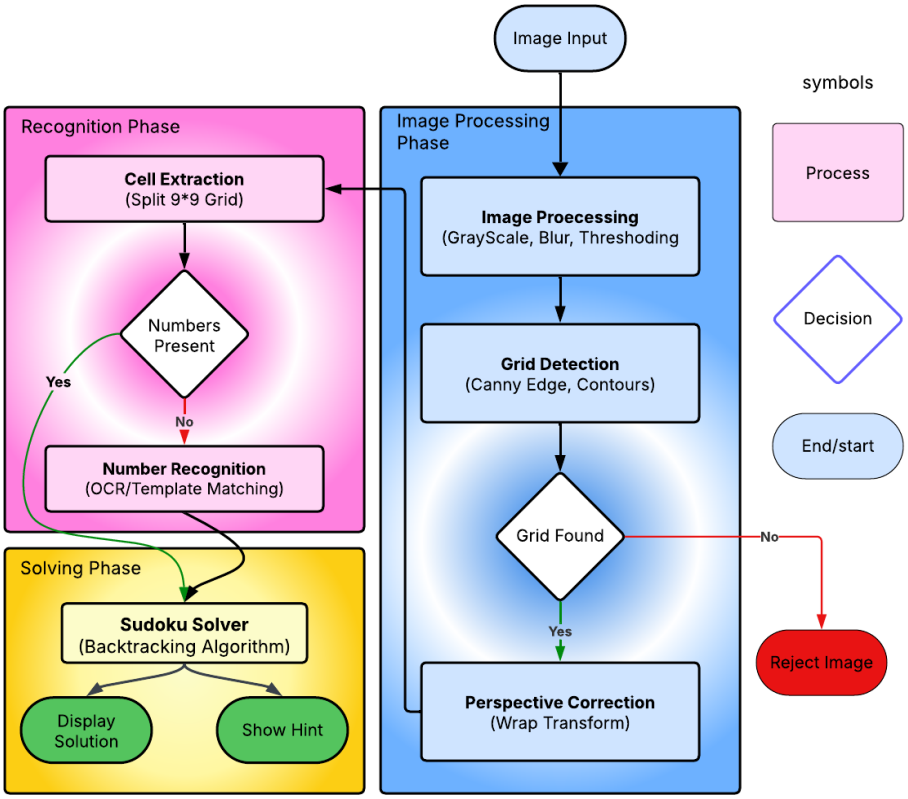

Figure 1: System workflow diagram from image input to solution output

3.2 Step 2: Sudoku Grid Detection

- Detect edges using Canny algorithm (`Canny()`)
- Find contours with `findContours()` (RETR_EXTERNAL mode)
- Approximate largest quadrilateral contour (Sudoku grid)
- Calculate perspective transform matrix (`getPerspectiveTransform()`)
- Apply transform to get aligned top-down view (`warpPerspective()`)
- **Output**: Perfectly aligned Sudoku grid image

3.3 Step 3: Grid Line Detection

- Detect lines using probabilistic Hough transform (`HoughLinesP()`)
- Filter and cluster horizontal/vertical lines
- Calculate all intersection points
- **Output**: 10x10 grid of intersection coordinates (9x9 cells)

3.4 Step 4: Cell Processing

- For each of 81 cells:
  - Extract cell image using intersection coordinates
  - Check if cell contains number (pixel density threshold)
  - If number present:
    * Apply preprocessing (binarization, centering)
    * Recognize digit using Tesseract OCR or template matching
    * Validate digit (1-9)
- **Output**: 9x9 matrix representing Sudoku puzzle

3.5 Step 5: Puzzle Solving

- Implement backtracking algorithm:
  - Find next empty cell (value = 0)
  - Try numbers 1-9 checking validity
  - Recursively attempt solution
  - Backtrack when stuck
- For hint generation:
  - Identify cells with only one valid possibility
  - Return first such cell found
- **Output**: Solved puzzle matrix or hint coordinates

References

[1] OpenCV Documentation. *https://docs.opencv.org*

[2] Sudoku Solving Algorithms. *https://en.wikipedia.org/wiki/Sudoku_solving_algorithms*