Mohamed khaled 221003036

Project Report: Automatic Tic-Tac-Toe Board Analysis and Symbol Recognition

1. Introduction

This project aims to automate the digitization of hand-drawn Tic-Tac-Toe games using computer vision techniques. The pipeline involves loading raw images of game boards, preprocessing them to handle varying lighting conditions, detecting the grid structure using edge detection and Hough transforms, and classifying the contents of each cell ('X', 'O', or Empty) using geometric shape analysis.

2. Methodology and Source Code

a) Core Libraries and Setup

We utilize `cv2` (OpenCV) for image manipulation and `numpy` for matrix operations. The system is designed to process images sequentially from a directory.

Python

```python
import cv2
import numpy as np
import os
import pandas as pd
from matplotlib import pyplot as plt

# Path setup and image loading logic
folder_path = "/content/drive/MyDrive/Digital_Detection/data/train"
image_files = [f for f in os.listdir(folder_path) if f.lower().endswith(".jpg")]
```

b) Image Preprocessing (Segmentation)

The preprocessing stage converts the raw RGB image into a clean binary mask. This is critical for isolating the chalk/marker lines from the blackboard/paper background.

Python

```python
def preprocess_img(img):
    # 1. Convert to Grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 2. Reduce Noise (Median Blur)
    median = cv2.medianBlur(gray, 5)

    # 3. Adaptive Thresholding
    # Handles varying lighting conditions across the board
    thresh = cv2.adaptiveThreshold(
        median, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY_INV, blockSize=19, C=5
    )

    # 4. Morphological Closing
    # Reconnects broken lines caused by faint chalk or glare
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    processed = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=1)

    return processed
```

Detailed Step-by-Step Explanation:

- **Grayscale Conversion:** Reduces the image to intensity values, removing unnecessary color data.
- **Median Blur:** Smooths the image to reduce high-frequency noise while preserving edges.
- **Adaptive Thresholding:** Unlike global thresholding (Otsu), this calculates thresholds for small regions. This is essential for images with uneven lighting or shadows.
- **Morphological Closing:** A dilation followed by erosion. This step bridges small gaps in the drawn lines, ensuring the grid and symbols are connected shapes.

c) Grid Line Detection

To locate the game cells, we must identify the four main grid lines (two horizontal, two vertical). We use Canny edge detection followed by the Probabilistic Hough Transform.

```python
def detect_grid_lines_edges(img_mask, border_margin=100):
    edges = cv2.Canny(img_mask, 50, 150)
    lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=50, minLineLength=80, maxLineGap=10)

    # Logic to separate and cluster lines (simplified for report)
    # ... (See get_separated_lines implementation) ...

    final_h = get_separated_lines(h_coords, min_dist=100)
    final_v = get_separated_lines(v_coords, min_dist=100)

    return final_h, final_v
```

Key Logic:

- **HoughLinesP:** Detects line segments in the binary mask.
- **Clustering (`get_separated_lines`):** Raw Hough lines often result in multiple detection lines for a single drawn line. A custom clustering algorithm groups nearby lines and selects the strongest candidates that are at least `min_dist` apart to ensure valid grid structure.

d) Symbol Classification (Geometric Analysis)

Instead of training a neural network, we utilize geometric properties to classify the symbols. This approach is computationally efficient and requires no training data.

Python

```python
def classify_cells(mask, cells, edge_ratio=0.15, min_fill=0.03):
    symbols = []
    for (x1, y1, x2, y2) in cells:
        # Crop cell and remove margins
        cell_mask = mask[y_start:y_end, x_start:x_end]

        # 1. Empty Check (Pixel Density)
        white_pixels = cv2.countNonZero(cell_mask)
        if (white_pixels / total_pixels) < min_fill:
            symbols.append("Empty")
            continue

        # 2. Contour Analysis
        contours, _ = cv2.findContours(cell_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        all_points = np.vstack(large_contours)

        # 3. Geometric Classification (Hull vs Circle)
        hull = cv2.convexHull(all_points)
        hull_area = cv2.contourArea(hull)
        (_, _), radius = cv2.minEnclosingCircle(all_points)
        circle_area = np.pi * (radius ** 2)

        # Calculate Ratio
        ratio = hull_area / circle_area

        if ratio > 0.70:
            symbols.append("O") # Circular shapes fill their enclosing circle well
        else:
            symbols.append("X") # X shapes have low area compared to their enclosing circle

    return symbols
```

Classification Logic:

- **Empty Detection:** If the percentage of white pixels in a cell is below `min_fill` (3%), it is classified as Empty.
- **Convex Hull vs. Enclosing Circle:**
  - **'O':** A circle is convex. Its Convex Hull Area is very close to the area of its Minimum Enclosing Circle. (Ratio $\approx$ 1.0, threshold set > 0.70).
  - **'X':** An 'X' is highly non-convex. Its actual area is much smaller than the circle required to enclose it. (Ratio < 0.70).

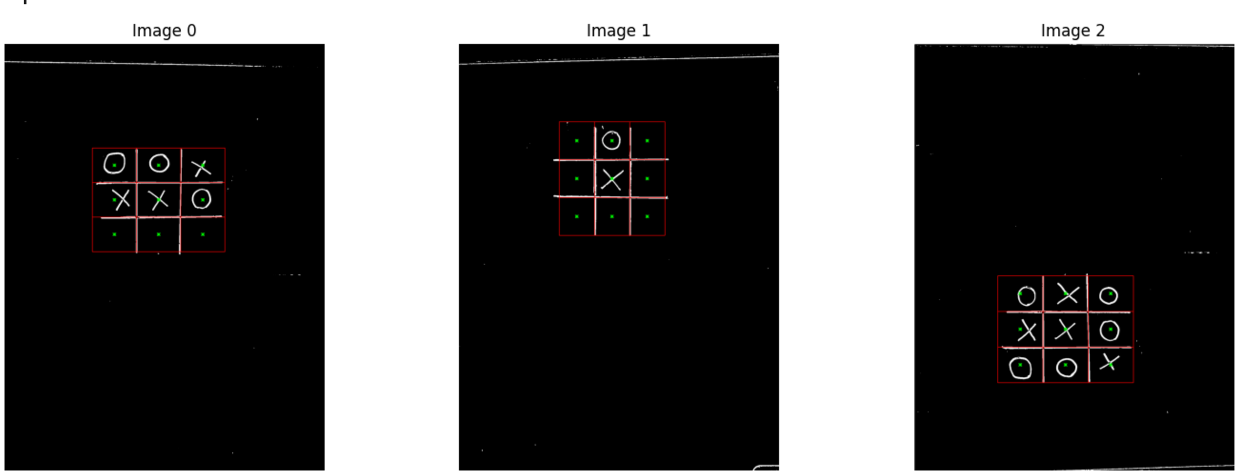
3. Visual Results

a) Preprocessing Pipeline

The images below demonstrate the extraction of the binary mask from the original image, highlighting the effectiveness of adaptive thresholding.

b) Grid Detection

The system successfully identifies the separating lines even when they are hand-drawn and impefect.



c) Final Output and Visualization

The following results show the identified cells and the classification overlay (Green circle for 'O', Red circle for 'X').

Image 38 symbols: ['Empty', 'Empty', 'Empty', 'Empty', 'Empty', 'X', 'X', 'O', 'O']

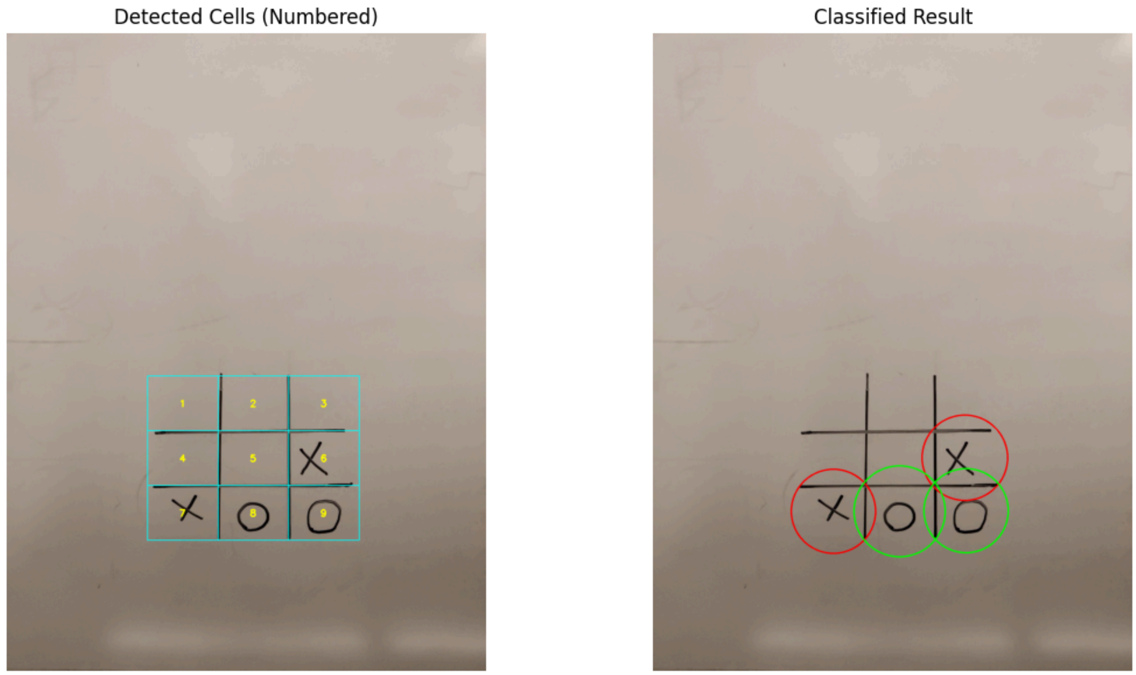


Image 39 symbols: ['O', 'Empty', 'O', 'Empty', 'Empty', 'X', 'Empty', 'Empty', 'X']

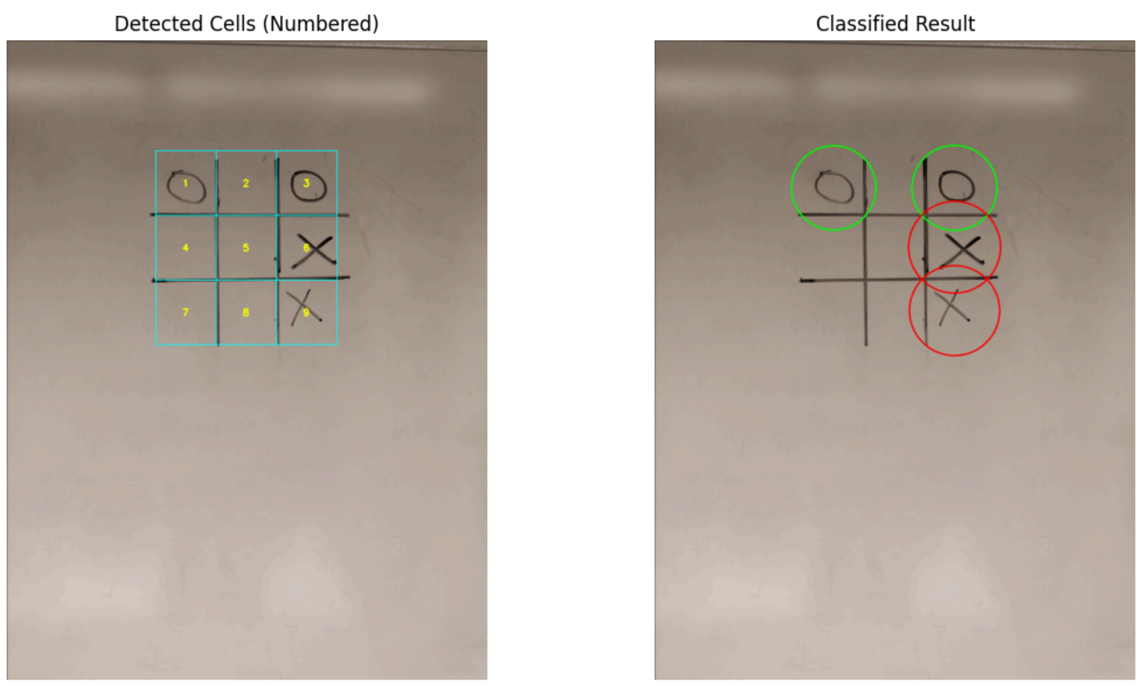


Image 40 symbols: ['Empty', 'Empty', 'Empty', 'Empty', 'Empty', 'Empty', 'Empty', 'Empty', 'Empty']

Detected Cells (Numbered)

Classified Result

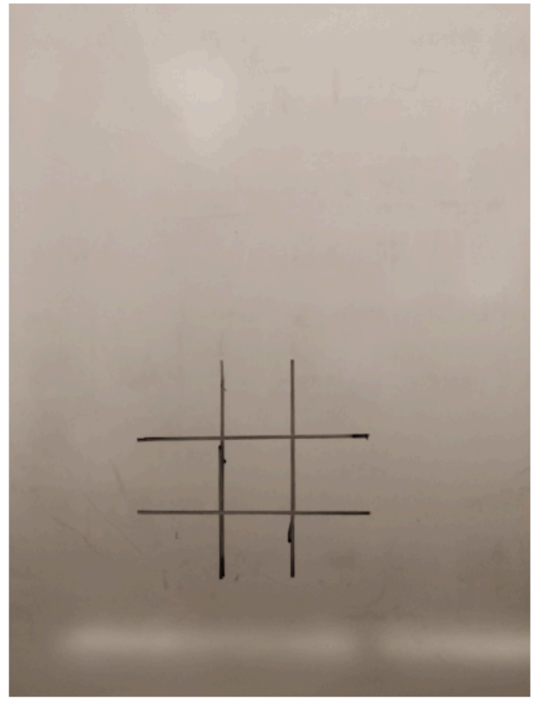


Image 41 symbols: ['O', 'X', 'Empty', 'X', 'X', 'O', 'O', 'O', 'X']

Detected Cells (Numbered)

Classified Result

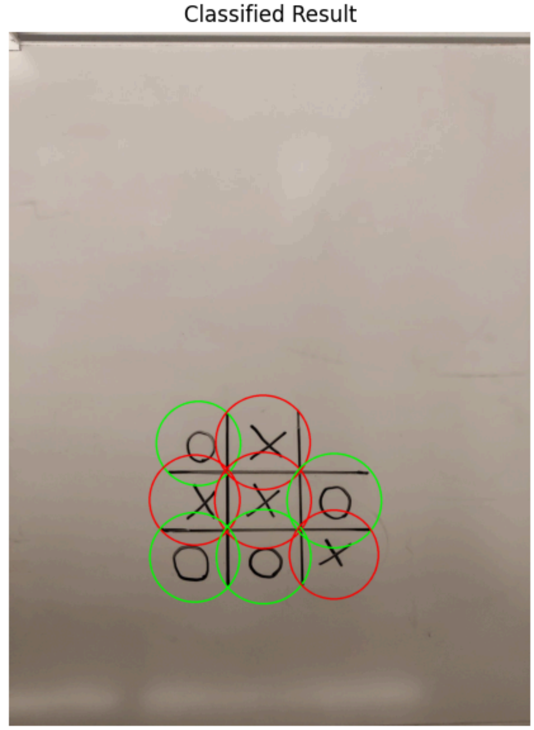
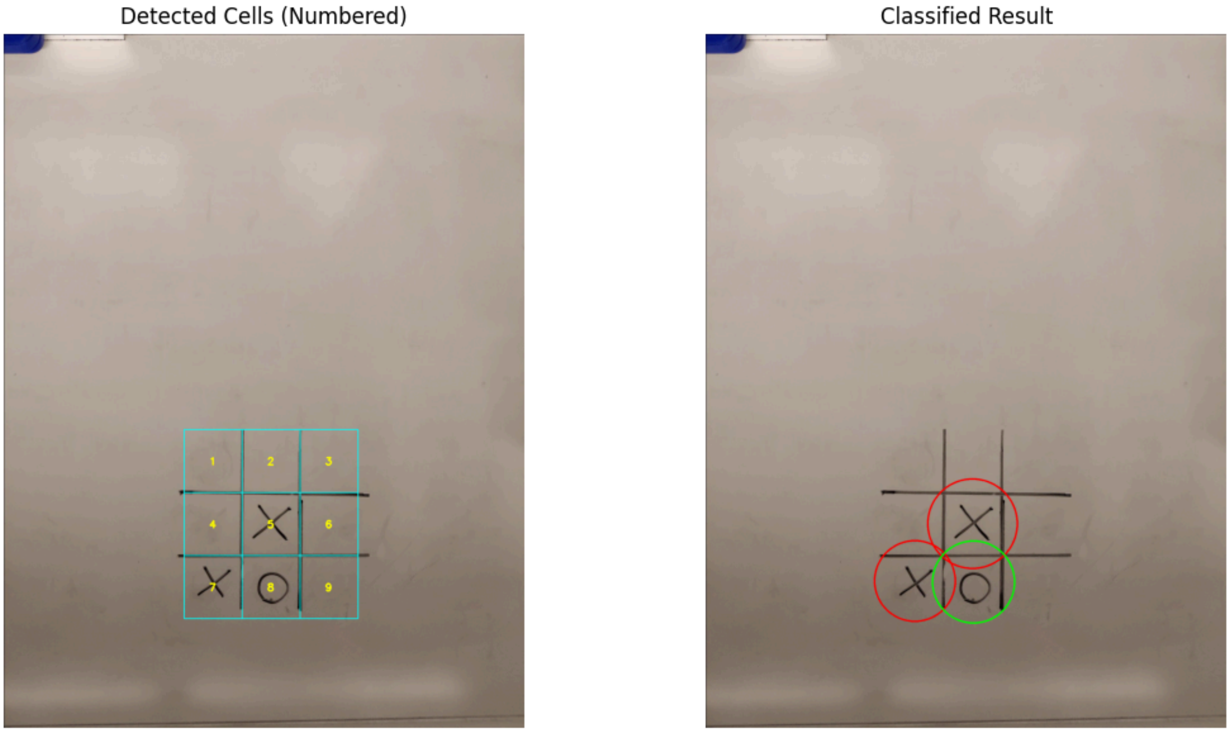


Image 42 symbols: ['Empty', 'Empty', 'Empty', 'Empty', 'X', 'Empty', 'X', 'O', 'Empty']



4. Model Evaluation

a) Methodology

The system was evaluated against a Ground Truth (GT) JSON dataset. We compared the number of 'X's and 'O's detected in each image against the actual counts. The primary metric used is **Root Mean Square Error (RMSE)**.

b) Performance Metrics

The system was tested on a dataset of validation images.

Metric	Value
RMSE (X Detection)	**0.125**
RMSE (O Detection)	**0.66**

c) Failure Analysis

Despite high accuracy, the system faces challenges in specific scenarios:

- **Broken Lines:** If a grid line is too faint or broken, the cell extraction fails, leading to misaligned coordinates.
- **Extreme Glare:** Strong reflections on the board can be interpreted as "white pixels," potentially causing empty cells to be classified as occupied.

.

5. Conclusion

The implemented pipeline successfully digitizes Tic-Tac-Toe boards with high precision. By combining robust preprocessing with geometric heuristics, the system effectively distinguishes between 'X', 'O', and Empty cells without the need for complex machine learning models. Future work involves improving the "Broken Line" logic to handle faint chalk lines more effectively.

6. Appendices

Appendix A: Project Repository

Repository Link: [Insert Link Here]

Date: December 2025

Appendix B: Classification Thresholds

The following thresholds were determined empirically:

- **Min Line Length:** 80 pixels (for grid detection).
- **Min Fill Ratio:** 0.03 (3% of cell must be white to be considered non-empty).
- **Shape Ratio:** 0.70 (Cutoff between 'X' and 'O').

Evaluating 64 images...

	Image Name	Xs Positions (True)	Xs Positions (Output)	Os Positions (True)	Os Positions (Output)
0	IMG_20220614_014432_jpg.rf.155ec4625b1f3d7c9d8...	[5, 7]	[5, 7]	[8]	[8]
1	IMG_20220614_014451_jpg.rf.6af0a1bcb1685e6b511...	[3, 5, 9]	[3, 5, 9]	[2, 7]	[2, 7]
2	IMG_20220613_231107_jpg.rf.a0928dbd4333d7f4d28...	[2, 4, 5, 9]	[2, 4, 5, 9]	[1, 3, 6, 7, 8]	[1, 3, 6, 7, 8]
3	IMG_20220614_014451_jpg.rf.48b0c662ed44ef43ac7...	[1, 5, 7]	[1, 5, 7]	[3, 8]	[3, 8]
4	IMG_20220614_014451_jpg.rf.ee5953dd4e6d9bb3b3b...	[3, 5, 9]	[3, 5, 9]	[2, 7]	[2, 7]
...	...	...	...	...	...
60	IMG_20220614_014228_jpg.rf.0e39a34a1abe2eb9a53...	[2, 4, 6, 7, 8]	[2, 4, 6, 7, 8]	[1, 3, 5, 9]	[1, 3, 5, 9]
61	IMG_20220613_231218_jpg.rf.f0c9cabc1b05c7ca5d7...	[]	[]	[1, 2, 3, 4, 5, 6, 7, 8, 9]	[1, 2, 3, 4, 5, 6, 7, 8, 9]
62	IMG_20220614_014150_jpg.rf.bd9b662d99b9c498729...	[1]	[1]	[9]	[9]
63	IMG_20220613_231005_jpg.rf.34311712876f4c5ea85...	[]	[]	[1]	[1]
64	RMSE		0.1250		0.6614

65 rows × 5 columns