CSE 4127: Image Processing and Computer Vision Laboratory

Detecting and Identifying Playing Cards from an Image

By

Md. Sunzidul Islam

Roll: 1907079

Submitted To:

Dr. Sk. Md. Masudul Ahsan

Professor

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Dipannita Biswas

Lecturer

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

July, 2024

Acknowledgement

All the praise to the almighty Allah, whose blessing and mercy succeeded me to complete this course work fairly. I gratefully acknowledge the valuable suggestions, advice and sincere cooperation of my course teachers, Dr. Sk. Md. Masudul Ahsan & Dipannita Biswas.

Contents

CHAPTER I

Introduction

1.1 Introduction

The detection and recognition of objects in images is a crucial aspect of computer vision, with applications spanning from autonomous driving to medical imaging. One specific application is the detection and recognition of playing cards, which can be used in various domains such as automated card games, card sorting systems, and magic tricks. This report delves into the design and implementation of a card detection system that employs advanced image processing techniques to identify and classify playing cards in real-time.This report provides a comprehensive overview of the working procedure of the card detection system, detailing the steps involved in preprocessing the input images, detecting potential card contours, and matching these contours to known card templates. Additionally, it explores the mathematical equations and principles underpinning the various stages of the detection process, such as the Canny edge detection algorithm, contour filtering and sorting, perspective transformation, and template matching using the sum of squared differences.

1.2 Objectives

1. **Develop a Robust Card Detection System**:
    - Implement a reliable system capable of detecting and identifying playing cards in images using advanced computer vision techniques.
2. **Preprocess Input Images**:
    - Apply image preprocessing techniques, such as grayscale conversion and Gaussian blurring, to enhance the quality of the input images for better edge detection and contour identification.

3. **Implement Edge Detection**:
    - Utilize the Canny edge detection algorithm to identify the edges of playing cards within the input images.
4. **Detect and Filter Contours**:
    - Identify contours in the preprocessed images and filter them based on size and aspect ratio to isolate potential card contours.
5. **Perform Perspective Transformation**:
    - Apply perspective transformation to obtain a top-down view of detected cards, ensuring consistent card orientation for accurate identification.
6. **Isolate Card Features**:
    - Extract and isolate the rank and suit features of each detected card for further analysis and matching.
7. **Match Cards with Templates**:
    - Implement template matching techniques to compare detected card features with predefined card templates and determine the best match for rank and suit.
8. **Visualize Detection Results**:
    - Draw the detected card contours and identified rank and suit information on the original images for visualization and verification purposes.
9. **Optimize System Performance**:
    - Ensure the system operates efficiently in real-time, with considerations for frame rate calculation and optimization of processing speed.

CHAPTER II

Methodology

2.1 Overview

The methodology for the card detection system is structured into several key phases, each involving specific tasks and techniques to achieve accurate detection and recognition of playing cards. The phases include image acquisition, preprocessing, edge detection, contour detection, perspective transformation, feature extraction, template matching, and result visualization.

2.2 Image Acquisition

The system captures images from a camera or loads them from a file. The captured images serve as input for further processing.

2.3 Preprocessing

The acquired images are preprocessed to enhance the quality and prepare them for edge detection. This involves converting the images to grayscale and applying Gaussian blur to reduce noise. Applying a Gaussian filter to smooth the image and reduce noise:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.1}$$

where σ is the standard deviation of the Gaussian distribution.

2.4 Edge Detection

Using algorithms like Canny or simple thresholding to detect the boundaries of the chessboard. In this implementation, the thresholding method is applied to simplify the edge detection process.

2.4.1 Mathematical Explanation

Edge detection involves finding regions in the image where there is a significant change in intensity. For thresholding, this means setting a pixel to white if its intensity is above a certain value and black otherwise. Mathematically, for a pixel intensity $I(x, y)$:

$$I'(x, y) = \begin{cases} 255 & \text{if } I(x, y) \geq T \\ 0 & \text{if } I(x, y) < T \end{cases} \tag{2.2}$$

where T is the threshold value.

2.5 Contour Detection

Contours representing potential cards are detected in the edge-detected images. The contours are then filtered based on size and aspect ratio to isolate the ones that likely correspond to cards.

2.5.1 Contour Tracing

Once the image is binarized, contour tracing algorithms such as the Moore-Neighbor tracing or Suzuki's algorithm are used to identify the contours. A contour is defined as a curve joining all the continuous points along a boundary with the same intensity.
Mathematically, a contour can be represented as a sequence of points (x_i, y_i) where each point is a pixel on the boundary of an object in the binary image.

2.5.2 Contour Approximation

To reduce the number of points representing a contour, contour approximation techniques such as the Ramer-Douglas-Peucker algorithm are used. This algorithm approximates a curve with fewer points while maintaining the shape of the contour.
The algorithm works as follows:

1. Start with the endpoints of the curve.

2. Find the point that is farthest from the line segment connecting the endpoints. If this point is farther than a predefined distance ϵ, it is kept.

3. Recursively apply the algorithm to the segments formed by this point and the endpoints.

4. Terminate when no points are farther than ϵ from the segment.

Mathematically, for a set of points P:

- Let d be the perpendicular distance from a point P_i to the line segment formed by points P_1 and P_n.
- If $d > \epsilon$, the point P_i is included in the simplified contour.

2.5.3 Contour Area

The area A of a contour is calculated using the Green's theorem, which relates a line integral around a simple closed curve to a double integral over the plane region bounded by the curve.

$$A = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)\right| \tag{2.3}$$

where (x_i, y_i) are the vertices of the contour.

2.5.4 Contour Perimeter

The perimeter P of a contour is calculated by summing the Euclidean distances between consecutive points on the contour:

$$P = \sum_{i=1}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{2.4}$$

where (x_i, y_i) are the vertices of the contour.

2.6 Perspective Transformation

A critical step in transforming the image to a top-down view. The transformation matrix M is derived from the source and destination points using the homography method. It reshapes the image to align with the board's perspective.

2.6.1 Mathematical Explanation

Given four points in the source image $p_i = (x_i, y_i)$ and their corresponding points in the destination image $p'_i = (x'_i, y'_i)$, the goal is to find a 3x3 matrix M such that:

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = M \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{2.5}$$

This results in a system of linear equations, which is solved to find M.
The matrix M can be represented as:

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{2.6}$$

For each point correspondence $(x_i, y_i) \rightarrow (x'_i, y'_i)$, we have the following equations:

$$\begin{aligned} x'_i &= \frac{a_{11}x_i + a_{12}y_i + a_{13}}{a_{31}x_i + a_{32}y_i + a_{33}} \\ y'_i &= \frac{a_{21}x_i + a_{22}y_i + a_{23}}{a_{31}x_i + a_{32}y_i + a_{33}} \end{aligned} \tag{2.7}$$

Rewriting these equations, we get:

$$\begin{aligned} x'_i(a_{31}x_i + a_{32}y_i + a_{33}) &= a_{11}x_i + a_{12}y_i + a_{13} \\ y'_i(a_{31}x_i + a_{32}y_i + a_{33}) &= a_{21}x_i + a_{22}y_i + a_{23} \end{aligned} \tag{2.8}$$

These equations form a system of linear equations which can be solved using linear algebra techniques to determine the elements of the transformation matrix M.

2.7 Feature Extraction

The system extracts the rank and suit features from the transformed card image. These features are used for matching the card against predefined templates.

2.8 Template Matching

The extracted features are compared with a set of predefined template images using a similarity measure such as the sum of squared differences (SSD). The best match indicates the identified rank and suit of the card.

2.8.1 Mathematical Explanation

The SSD is calculated as follows:

$$\text{SSD}(x, y) = \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} [I(x + i, y + j) - T(i, j)]^2 \tag{2.9}$$

where I is the input image, T is the template image, and (w, h) are the width and height of the template image.

The location with the minimum SSD value indicates the best match:

$$(x^*, y^*) = \arg\min_{(x,y)} \text{SSD}(x, y) \tag{2.10}$$

2.9 Result Visualization

The detected cards and their identified ranks and suits are drawn on the original images for visualization. This step provides a visual verification of the detection results.

2.10 System Optimization

To ensure real-time performance, the system calculates and optimizes the frame rate. This involves measuring the time taken for each iteration of the main loop and adjusting the processing speed accordingly.

CHAPTER III

Functionalities

3.1 Functionalities

The card detection system is designed to perform a variety of functions, enabling it to accurately detect and recognize playing cards in an image. These functionalities are implemented through a series of steps, each contributing to the overall process. Below is a detailed description of the main functionalities of the system.

3.1.1 Image Acquisition

- **Capture Image from Camera**: The system can capture live images from a connected camera.
- **Load Image from File**: The system can also load images from a file for processing.

3.1.2 Image Preprocessing

- **Grayscale Conversion**: Convert the captured or loaded image to grayscale to simplify processing.
- **Gaussian Blurring**: Apply Gaussian blur to the grayscale image to reduce noise and enhance edge detection.

3.1.3 Edge Detection

- **Canny Edge Detection**: Use the Canny edge detection algorithm to identify the edges of the cards in the image.
- **Thresholding**: Apply simple thresholding to convert the image to a binary format, making it easier to detect contours.

3.1.4 Contour Detection

- **Find Contours**: Detect contours in the binary image to identify potential card boundaries.
- **Filter Contours**: Filter out contours that do not resemble card shapes based on size and aspect ratio criteria.
- **Sort Contours**: Sort the remaining contours by area to prioritize the largest (most likely) card contours.

3.1.5 Perspective Transformation

- **Calculate Transformation Matrix**: Compute the transformation matrix using homography to align the detected card with a top-down view.
- **Warp Perspective**: Apply the perspective transformation to obtain a rectified image of the detected card.

3.1.6 Feature Extraction

- **Isolate Card Features**: Extract the rank and suit features from the transformed card image for further analysis.
- **Preprocess Card**: Preprocess the card image to isolate and enhance the features of interest.

3.2 Template Matching

- **Load Template Images**: Load pre-defined template images for each rank and suit.
- **Compare with Templates**: Use template matching techniques, such as the sum of squared differences (SSD), to compare the extracted features with the templates.
- **Determine Best Match**: Identify the best match for the card's rank and suit based on the minimum SSD value.

3.3 Result Visualization

- **Draw Contours**: Draw the detected card contours on the original image for visualization.
- **Annotate Image**: Annotate the image with the identified rank and suit of each detected card.
- **Display Results**: Display the final annotated image with all detected and recognized cards.

3.4 System Optimization

- **Frame Rate Calculation**: Calculate and display the frame rate to monitor the performance of the system.
- **Real-time Processing**: Ensure the system operates efficiently in real-time by optimizing the processing speed and minimizing latency.

CHAPTER IV

Implementation and Results

4.1 Pseudocode of the card detection system

Algorithm 1 Card Detection System

```
function CARD_DETECTION_SYSTEM(source, templates)
    initialize_system()
    while is_running() do
                                                    ▷ Step 1: Image Acquisition
        image = acquire_image(source)
                                                    ▷ Step 2: Image Preprocessing
        preprocessed_image = preprocess_image(image)
                                                    ▷ Step 3: Edge Detection
        edges = detect_edges(preprocessed_image)
                                                    ▷ Step 4: Contour Detection
        contours = detect_contours(edges)
        matches = []
        for contour in contours do
                                                    ▷ Step 5: Perspective Transformation
            warped_image = perform_perspective_transformation(contour, image)
                                                    ▷ Step 6: Feature Extraction
            features = extract_features(warped_image)
                                                    ▷ Step 7: Template Matching
            match = match_with_templates(features, templates)
            matches.append(match)
        end for
                                                    ▷ Step 8: Result Visualization
        visualize_results(image, contours, matches)
                                                    ▷ Step 9: System Optimization
        optimize_system()
        if user_requests_exit() then
            break
        end if
    end while
    finalize_system()
end function
```

4.2 Output and Intermediate Images

The best threshold level depends on the ambient lighting conditions.For bright lighting, a high threshold must be used to isolate the cards from the background. For dim lighting, a low threshold must be used. To make the card detector independent of lighting conditions, the following adaptive threshold method is used. A background pixel in the center top of the image is sampled to determine its intensity. The adaptive threshold is set at 50 higher than that. This allows the threshold to adapt to the lighting conditions.

Figure 4.1: Input Image

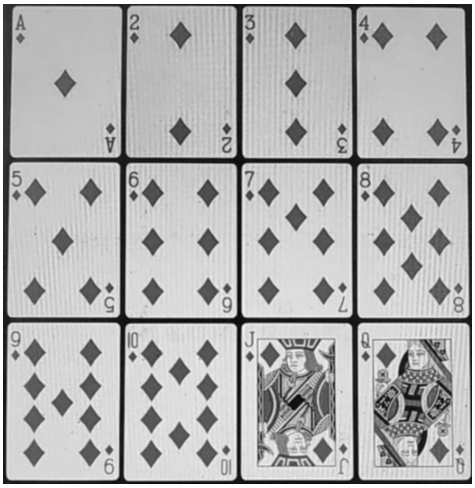

Figure 4.2: Gaussian Blur Image

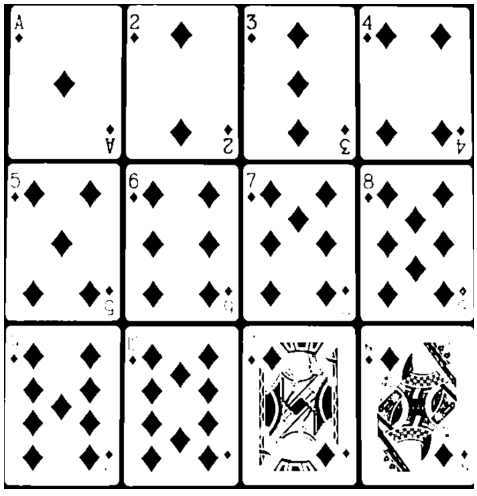

Figure 4.3: After Thresholding

Finds all card-sized contours in a thresholded camera image. Returns the number of cards, and a list of card contours sorted from largest to smallest.Split in to top and bottom half (top shows rank, bottom shows suit. Finds best rank and suit matches for the query card. Differences the query card rank and suit images with the train rank and suit images. The best match is the rank or suit image that has the least difference.

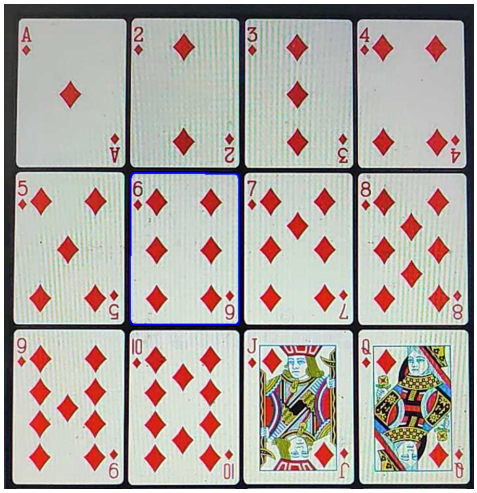

Figure 4.4: Find Card and Border

Figure 4.5: Rank Card

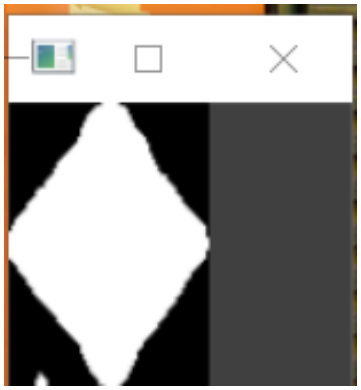

Figure 4.6: Suit Card

Figure 4.7: Final Image

CHAPTER V
Discussion and Conclusion

5.1 Discussion

In this project, I developed a system for detecting and identifying playing cards from an image. The process involved several key steps: detecting the card contours, performing perspective transformation to obtain a top-down view, and then identifying the cards using template matching.

Edge detection through thresholding proved effective in identifying clear and distinct edges necessary for the sum of squared differences (SSD) calculation. Adding a border around the image was essential to handle edge pixels and avoid potential issues during edge detection. This step ensured that the edges of the cards were well-defined, which is critical for accurate contour detection.

The perspective transformation was another crucial step in this project. By mapping the detected contour points to a square grid, I was able to compute the transformation matrix $\mathbf{M}$. This matrix allowed the conversion of the image's perspective to a top-down view, making it easier to analyze and classify the playing cards. The mathematical foundation of this transformation, involving the solution of a system of linear equations, provided a robust method to achieve accurate perspective correction. Ensuring that the cards were correctly aligned was vital for reliable feature extraction and matching.

For card identification, the system divided the detected card into two parts: the rank and the suit. Template matching was used to compare these parts with predefined templates. The method relied on the similarity between the template images and the patches in the transformed card image. By using the sum of squared differences (SSD) as the similarity measure, the system effectively identified the rank and suit of each card. The accuracy of this method depended on the quality and representativeness of the template images.

5.2 Conclusion

The methodology described above outlines a systematic approach to detecting and recognizing playing cards using computer vision techniques. By following these steps, the system effectively preprocesses images, detects card contours, applies perspective transformation, extracts features, performs template matching, and visualizes the results. The use of OpenCV facilitates the implementation of these techniques, providing a robust solution for card detection tasks.

5.3 Future Works

Future improvements to the system could include the integration of more sophisticated noise reduction and adaptive thresholding techniques to enhance edge detection. Additionally, incorporating machine learning algorithms for card identification could provide better accuracy and robustness, especially under varying lighting conditions and with different card designs. Exploring the use of deep learning models for object detection and classification could further enhance the system's performance.

CHAPTER VI

References

References

[1] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[2] Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), 679-698.

[3] Hartley, R., & Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

[4] Lewis, J. P. (1995). Fast Template Matching. *Vision Interface*, 120-123.

[5] Bradley, D., & Roth, G. (2007). Adaptive Thresholding Using the Integral Image. *Journal of Graphics Tools*, 12(2), 13-21.

[6] Gonzalez, R. C., & Woods, R. E. (2002). *Digital Image Processing*. Prentice Hall.

[7] Goshtasby, A. (1985). Template Matching in Rotated Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(3), 338-344.