Automated Visual Analysis and Playing of a Memory Card Game

Cazzaniga David
Chalmers University of Technology
davidcaz@chalmers.se

***Abstract*—This project developed a system that can automatically see and understand the state of a physical Memory card game. I built a computer vision solution using OpenCV library that processes images through grayscale conversion, Gaussian blurring, morphological operations(erode and dilate), and Canny edge detection to effectively segment individual cards. I used SIFT (Scale-Invariant Feature Transform) as the main feature descriptor to recognize and match cards. Under controlled lighting conditions, the system achieved perfect performance: 100% accuracy in detecting cards, 100% accuracy in classifying face-down cards, and 100% accuracy in matching face-up pairs.**

***Index Terms*—Memory game, computer vision, image processing, SIFT, OpenCV, card detection, object recognition, feature matching.**

I. Introduction

The game of Memory is a card game in which all cards are laid face down on a surface, and two cards are flipped face up over each turn. The object of the game is to turn over pairs of matching cards, if the two cards match, the player wins the pair and plays again. If they do not match, they are turned back and the turn is passed to the next player. The game continues until all pairs have been found, and the player who has the most pairs wins.

Automating the play of such a game using computer vision addresses challenges in object detection (finding the cards), object segmentation (isolating individual cards), and object recognition (identifying face-up/face-down cards and matching face-up cards).

The primary objectives of this project were:

- To develop a system that can reliably detect and segment individual cards from a static image of a Memory game.
- To classify detected cards as either face-up or face-down using SIFT feature matching against a known card back template.
- To determine if two revealed face-up cards are a matching pair, also using SIFT feature matching.
- To simulate the actual game play and track player scores based on what the vision system detects.

This report details the methodology employed, the results obtained from testing the system, a discussion about the algorithm's limitations, and potential ideas for future work.

II. Methodology

The system was developed in a Jupyter notebook using the OpenCV, NumPy and Matplotlib libraries, along with two functions (`extract_sift_features` and `match_descriptors`) provided in the third laboratory. The choice of SIFT features was motivated by their proven invariance to scale, rotation, and illumination changes, making them suitable for card recognition tasks where viewing conditions may vary. The matching threshold of 20 good matches were determined empirically through preliminary testing on a subset of the dataset. To simplify the explanation, images of the processing steps are provided. The raw image used as an example is shown in Fig. 1

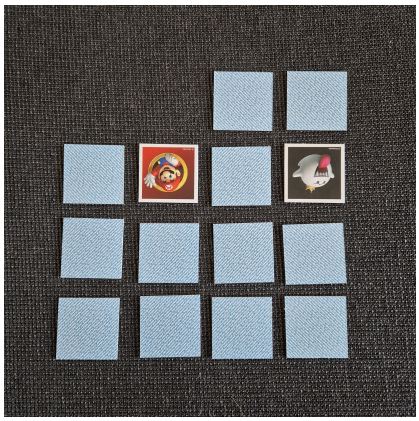

Fig. 1. Original image

The steps of the `card_matches` function are as follows:

A. Image Acquisition and Setup

Input images of the Memory game board were captured using a smartphone camera positioned overhead under consistent indoor lighting conditions. The camera was maintained at a fixed distance to ensure uniform scale across all captured images. A "Super Mario" themed Memory card deck was used. All input images were resized to a fixed resolution of 1024x1024 pixels to standardize the processing and make the computation faster. A dedicated reference image of a card back (specifically, `jpg/IMG_2131.jpg` from the dataset, Fig. 2) was pre-processed and used for face-down card identification.

B. Image Pre-processing

To enhance features relevant for card detection, a series of pre-processing steps were applied to the input 1024x1024 pixel image:

Fig. 2. `IMG_2131.jpg` used as a reference for the face-down identification

1) *Grayscale Conversion:* The color image was converted to a single-channel grayscale image with `cv2.cvtColor` function, as shown in the left image in Fig. 3.
2) *Gaussian Blur:* A Gaussian blur with a (5,5) kernel was applied to reduce high-frequency noise using the `cv2.GaussianBlur` function, as shown in the right image in Fig. 3.

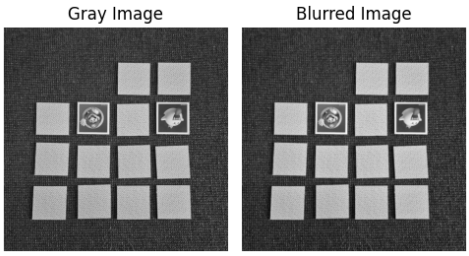

Fig. 3. The first two steps of preprocessing

3) *Morphological Operations (Set 1):*
   - Erosion: Applied using a 3x3 kernel for 3 iterations. The primary goal of this operation is to sharpen object edges and remove small noise elements and logos from the card backs, which could interfere with contour detection, as shown in the left image of Fig. 4. The erosion function used was `cv2.erode`.
   - Dilation: Applied using the same 3x3 kernel for 4 iterations. This step aims to restore the size of objects after erosion and can help connect broken parts of edges as shown in the right image of Fig. 4. The dilatation function used was `cv2.dilate`.

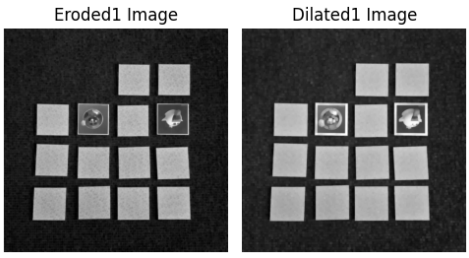

Fig. 4. The third and fourth step of preprocessing

4) *Canny Edge Detection:* `cv2.Canny` (with empirically determined thresholds of 40 for the lower bound and 255 for the upper bound) was used to identify strong edges as shown in Fig. 5.

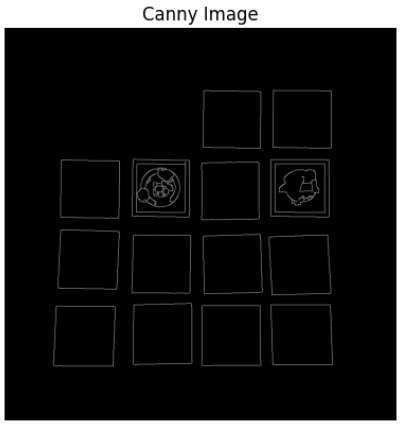

Fig. 5. Edge detection

5) *Morphological Operations (Set 2):*
   - `cv2.dilate`: Applied on the Canny edge output using a 3x3 kernel for 5 iterations to close gaps in the detected card contours as shown in the left image of Fig. 6.
   - `cv2.erode`: Applied using the same 3x3 kernel for 5 iterations to refine the contours back towards their original size, removing some of the excess thickening from dilation as shown in the right image of Fig. 6.

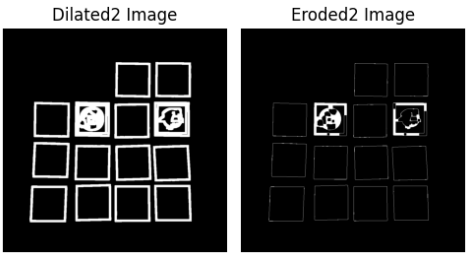

Fig. 6. The last two steps of preprocessing

C. Card Detection and Cropping

Following pre-processing, individual cards were detected and segmented:

1) *Contour Finding:* `cv2.findContours` was executed on the binary image resulting from the pre-processing steps. Only external contours (`cv2.RETR_EXTERNAL`) were retrieved, using the `cv2.CHAIN_APPROX_SIMPLE` method.
2) *Contour Filtering:* Each retrieved contour was subjected to filtering:
   - *Area Filtering:* Contours with an area less than `img.shape[0]*5` (equivalent to 5120 pixels for a 1024×1024 pixel image) were discarded to eliminate small noise-induced contours.
   - *Shape Approximation:* The perimeter of the contour was calculated using `cv2.arcLength`.

This perimeter was then used with `cv2.approxPolyDP` to approximate the contour shape with fewer vertices. An epsilon value of `0.04 * perimeter` was used for this approximation, where epsilon represents the maximum percentage of the perimeter length that can be removed from the original contour.
- *Vertex Count:* Only approximated polygonal contours with four or more vertices were considered as potential cards, since cards in the "Super Mario" memory game are all squares.

3) *Card Extraction:* For each contour passing the filters:
- A bounding rectangle was computed using `cv2.boundingRect`.
- A binary mask of the same dimensions as the input image was created. The approximated contour was drawn filled onto this mask with `cv2.drawContours`.
- This mask was used with `cv2.bitwise_and` to extract the corresponding region from the original input image. This extracted `cropped_card` was then used for subsequent classification and identification steps. Through these steps, the cropped card has the background removed. The cropped cards can be seen in Fig. 7.

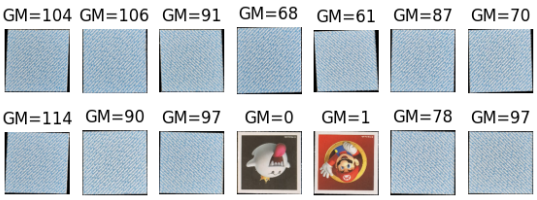

Fig. 7. Cropped cards from the original image, GM indicates the number of good matches between the face-down card and eaach cropped card

D. Face-Up/Face-Down Classification

To determine if a detected card was face-up or face-down, SIFT features were used:

1) *Reference Card Back Processing:*
- The designated card back image (`jpg/IMG_2131.jpg`, Fig 2) was loaded, resized to 1024x1024 pixels, and converted to grayscale.
- Contrast Limited Adaptive Histogram Equalization (CLAHE) with `clipLimit=2.0` and `tileGridSize=(8,8)` was applied to enhance local contrast and improve feature extraction reliability.
- A Gaussian blur ((5,5) kernel) was applied.
- SIFT keypoints and descriptors (`keypoints_face_down`, `descriptors_face_down`) were extracted from this processed reference card back image using the `extract_sift_features` function.

2) *Classification of Detected Cards:* For each `cropped_card` in Fig. 7 obtained from the card extraction section:
- The cropped card was converted to grayscale.
- CLAHE was applied, mirroring the processing of the reference card back.
- SIFT keypoints and descriptors (`keypoints_target`, `descriptors_target`) were extracted.
- The `descriptors_target` from the cropped image were matched against the `descriptors_face_down` of the reference card back using the `match_descriptors` function.
- If the number of good matches found was greater than or equal to an empirically set threshold of 20, the card was classified as face-down. Otherwise, it was classified as potentially face-up.
- Visual feedback was provided by drawing blue rectangles around detected face-down cards based on this classification.

E. Matching Potentially Face-Up Cards

This stage determines if two cards, classified as "not face-down" (i.e., potentially face-up) in the previous step, form a matching pair:

1) *Condition for Matching:* The following steps are invoked only if the number of potentially face-up cards is exactly two.
2) *SIFT Feature Matching:* The SIFT descriptors previously extracted from these two potentially face-up cards were matched against each other using the `match_descriptors` function.
3) *Decision Rule:* If the number of good matches between the SIFT descriptors of these two cards was greater than or equal to an empirically set threshold of 20, they were classified as a *matching pair* (indicated by `win = True`). Fig. 8 shows the two cards identified as potentially face-up and the number of good matches between them.
4) Visual feedback was provided by drawing green rectangles around matching pairs and red rectangles around non-matching pairs (or single face-up cards if the two-card condition wasn't met) as shown in Fig. 9

Fig. 8. cropped cards not identified as back cards, GM indicates the number of good matches between the two cropped cards

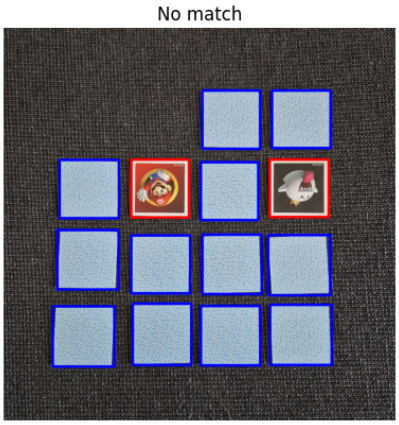

Fig. 9. Final image with all the bounding boxes drawn

F. Game Logic Simulation (`memory_match` function)

The `memory_match` function simulate a two-player Memory game. This function iterate through a predefined sequence of image paths, each representing a game state after a player has revealed cards:

1) For each image, the `card_matches` function was called to process the image and determine if a match (`win`) was made.
2) If `win == True`, the current player's score was incremented, and they retained the turn.
3) If `win == False`, the turn switched to the other player.
4) If the `card_matches` function returned `win is None` (e.g., due to an image processing error or if the conditions for pair matching were not met, such as not having exactly two face-up cards), the turn was logged as "not valid" and the turn is retained by the player that was already playing.

III. Experimental evaluation

Methodology: The experimental evaluation was conducted on a dataset of 32 images representing different game states, containing a total of 450 individual card instances. Each image was manually annotated to establish ground truth for card locations, orientations (face-up/face-down), and matching pairs.

A. Performance of Card Detection

- *Results:*
  - Total number of cards in the test: 450
  - Number of correctly detected cards (TP): 450
  - Number of missed cards (FN): 0
  - Number of false detections (FP): 0
  - *Precision:* 100%
  - *Recall:* 100%

B. Performance of Face-Down Classification

- *Results:*
  - Total number of face-down cards in the test: 393
  - Total number of face-up cards in the test: 57
  - Number of correctly detected face-down cards (TP): 393
  - Number of missed face-down cards (FN): 0
  - Number of false detections (FP): 0
  - Number of correctly detected non face-down cards (TN): 57
  - *Precision:* 100%
  - *Recall:* 100%

C. Performance of Pair Matching

- *Results:*
  - Total number of pairs: 13
  - Total number of non pairs (non matching pairs and images with just one face-up card): 17
  - Number of correctly detected pairs (TP): 13
  - Number of missed pairs (FN): 0
  - Number of false detections (FP): 0
  - Number of correctly detected non pairs (TN): 17
  - *Precision:* 100%
  - *Recall:* 100%

IV. Discussions about algorithm limitation

To evaluate the robustness limits of the proposed algorithm, additional test cases were conducted using images containing arbitrarily rotated cards, different camera perspectives and varying background conditions. For images with arbitrary rotated cards (Fig. 10) card detection performed well, however, SIFT feature matching failed. All the cropped cards were labeled as face-up because the number of good matches was below the threshold.

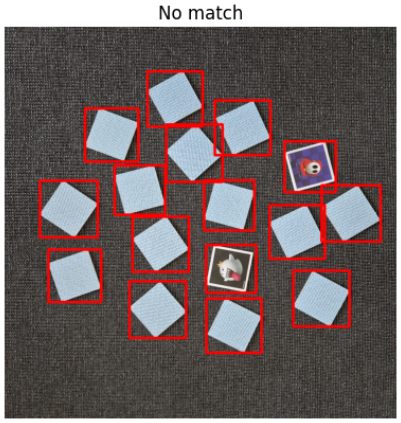

Fig. 10. Card detection with arbitrary rotated cards

Images captured with different perspectives (Fig. 11) revealed that while most cards were successfully detected, closely positioned cards were occasionally merged into single contours. The Fig. 12 shows the last step before the `cv2.findContours` function, where adjacent card boundaries became connected. SIFT feature detection failed as in the first test and all cards were identified as face-up cards.

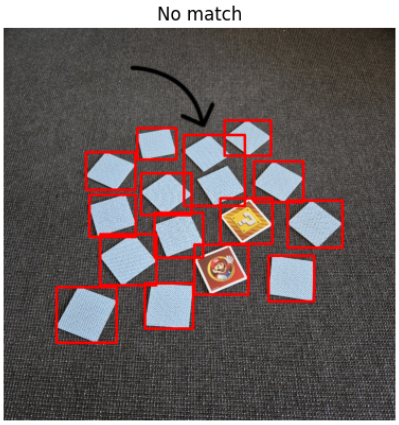

Fig. 11. Card detection with a different perspective

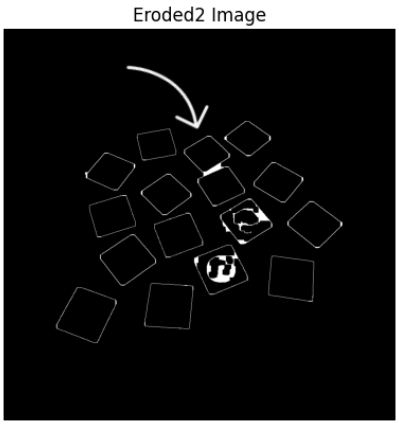

Fig. 12. Final preprocessing layer before the `cv2.findContours` function for images with different perspectives

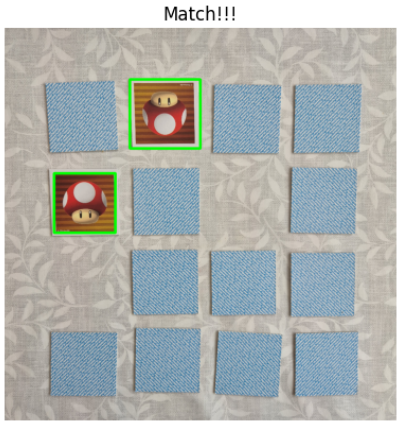

Fig. 13. Card detection with lighter background

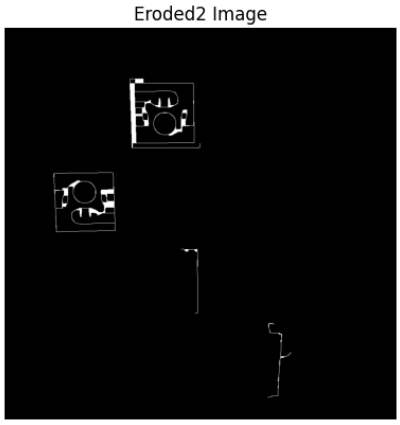

Fig. 14. Final preprocessing layer before the `cv2.findContours` function for images with lighter backgrounds

Images with lighter background provided the most challenging conditions for the algorithm. As demonstrated in Fig. 13, only face-up cards were successfully detected. Fig. 14 reveals that the Canny edge detector fails to distinguish between the low-contrast card edges and background texture patterns. However, the successfully identified face-up cards were correctly identified as a match from the SIFT feature matching.

V. Conclusions and Future Work

The system developed demonstrates its effectiveness under controlled conditions, while the identified limitations provided clear directions for future system development.

Several ideas for future work are:

- *Adaptive Thresholding:* Implement methods for adaptive thresholding of the SIFT matches, rather than a fixed value.
- *Machine Learning models:* Train machine learning models, such as Convolutional Neural Networks, for a more consistent face-up/down classification than a simple threshold.
- *Improved Card Segmentation:* Explore other segmentation techniques to better handle challenging conditions such as touching or overlapping cards.

References

[1] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 2008.

[2] OpenCV Documentation. [Online]. Available: https://docs.opencv.org/4.x/index.html