Texas Hold’em Hand Recognition and Analysis

Dan Brinks, Hugh White
Department of Electrical Engineering
Stanford University
Palo Alto, CA
{dbrinks, hwhite}@stanford.edu

Abstract— **In this paper we present an algorithm for playing card detection and identification. This algorithm is robust to card orientation and photograph angle, and is achieved through thresholding, corner detection, projective transformation, and template matching. We implement this algorithm in MATLAB, and demonstrate its capability with a mobile application that can capture photos of Texas Hold’em scenarios and give the user relevant statistics.**

Keywords-mobile image processing, playing card detection, template matching, corner detection, poker statistics

I. Introduction

Playing card recognition is an ongoing problem in image processing. Our brief survey of the literature revealed algorithms that depended upon top-down photos at a known distance, as in [1], [2] and [3]. To create an application that would be useful on a mobile platform, where angle and distance are variable, an algorithm must be robust to these factors. Our algorithm uses corner point detection to perform a projective transform. This allows template matching of cards at a variety of scales and angles. We developed a mobile application to capture photos and send them to a server running a MATLAB implementation of our algorithm. The server identifies the cards and returns relevant Texas Hold‘em statistics. More specifically, the mobile application shows the user the odds that a single player has a given hand. In our testing, we found that our application identifies cards with an accuracy of 88%.

II. Algorithm

Our algorithm implemented the following process:

1. Contrast enhancement
2. Binary image thresholding
3. Region detection and filtering
4. Image transformation
5. Orientation determination
6. Template matching
7. Statistic calculation

A. *Contrast Enhancement*

Early prototyping indicated that a boost in contrast improved several aspects of our algorithm including thresholding and edge detection. Values in the neighborhood of two yielded optimal results, so for computational simplicity we used 2.0 for our gamma value. Larger values tended to wash out the image.

B. *Binary Image Thresholding*

Binary thresholding was performed using Otsu’s method. Adaptive thresholding was considered but rejected as our ultimate goal was to determine the card outline; further detail provided by adaptive thresholding would have been unnecessary and cluttered the image.

C. *Region Detection and Filtering*

We perform region detection and then filter based upon several properties. Using the assumption that the cards were centrally located, we rejected those regions whose centroids were found near the edge of the image. Additionally, we reduced the areas considered by removing small regions. As our algorithm uses template matching, small cards would be unlikely to be correctly detected anyway. Next, we remove regions with shapes inconsistent with playing cards, although we allow for angle and perspective distortion.

D. *Image Transformation*

Using the corners on the captured image and the dimensions and shape of a playing card, we calculate a projective transform. This transform, when applied to the captured image, yields an upright playing card suitable for template matching.

E. *Corner Detection*

Due to the projective nature of a playing card captured at an angle, true edge length cannot be determined, resulting in orientation ambiguity. Playing cards contain activity in the upper left and lower right corners of the card. Conversely, the upper right and lower left corners are empty. Thus, measuring activity in a corner resolves orientation ambiguity to one-hundred-eighty degrees, which is sufficient for our template matching.

F. *Template Matching*

Card identification is achieved using template matching. Through experimentation, we discovered that the upper left and lower right corners of the card are sufficient for classification. Traditional template matching failed to produce a reliable result owing to the similarity between suit shapes. However, template matching of edges yielded more consistent results.

G. *Statistic Calculation*

The final step in our algorithm is the generation of statistics relevant to Texas Hold'em. The image passed to our algorithm is intended to be a picture of the community cards (known as the flop, turn and river). We calculate statistics for each of nine hand representations: Straight Flush, Four of a Kind, Full House, Flush, Straight, Three of a Kind, Two Pair, Pair, and High Card. For each hand, we determine the odds of a single player having said hand, given the known cards. These statistics are generated when we detect three, four, or five cards on the table.

III. Implementation

Our image processing used a client-server architecture.

A. *Client Side*

The client side application is responsible for image capture, transmission to the server, and displaying the results.

As our application client is hosted on an Android mobile platform, we initially attempted to use the Android API to configure and control the camera. Unfortunately, prior to Android API level 14, continuous camera autofocus is unavailable. Additionally, one of the development phones refused to autofocus through any camera API functions. Our algorithm is heavily dependent on in-focus photos to ensure proper edge detection. We overcame this obstacle by using the Media Store Action Image Capture Intent. This calls the OEM camera application, which takes the picture and returns the image. This native application provided us high-quality continuous focus unavailable through the camera API.

After capturing the image, we transmit it to the server through a Common Gateway Interface (CGI). The server responds with an ASCII string representing the detected cards and the computed statistics. With this information, the client displays an image of each identified card, along with the card title. Finally, the statistics are presented to the user.

B. *Server Side*

Upon receiving the captured image, MATLAB begins processing. The general processing flow is described in the Algorithms section above; implementation details are explained here. A reference image is provided as an example in Fig. 1.

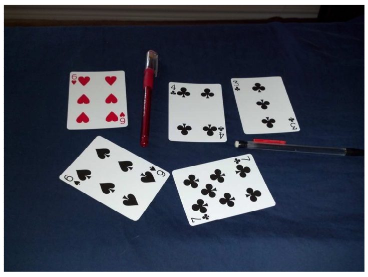


Fig. 1 Original Image

The image following contrast enhancement and binary thresholding is shown in Fig. 2.

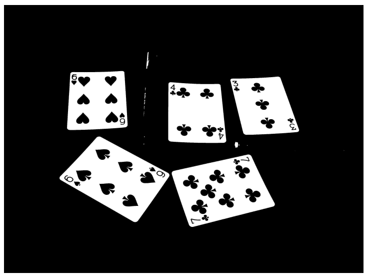


Fig. 2 Image After Thresholding

1) *Region Detection and Filtering*

Small regions are removed using MATLAB's `bwareaopen` function, applied to both the image and the image complement. Black regions less than a million pixels are removed, as are white regions smaller than ten thousand pixels. After removing small regions, the binary image is labeled using `bwlabel`. Next, all regions whose centroids are within 10% of any edge are removed. At this point, the image is shown in Fig. 3.

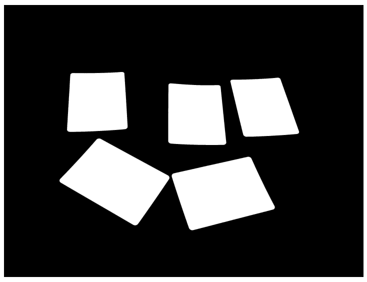


Fig. 3 Image After Region Filtering

Subsequent processing occurs individually on a per-region basis. First, the corners of the region are determined. An outline of the region mask is found using an XOR of the region with a slightly dilated version of the region. The angle and distance to each outline pixel from the centroid is calculated, and the pixels are sorted by angle. This creates local maxima in the distance vector. These maxima are computed by finding any pixel which is larger than all pixels within forty to either side (as sorted by angle). Up to four corners are discovered, and any regions with fewer than four are rejected.

For regions with four corners, shape filtering is performed. Despite projective angles causing distortion, at any reasonable angle a card can be approximated by a parallelogram. Therefore, we measure the angle through the centroid between opposite corners. Any regions whose corner angles are more than 0.3 radians away from pi are rejected.

2) *Image Transformation*

Image transformation is performed by creating a projective transform from the four detected corners to the expected aspect ratio of a card. Playing cards are 5"x7", so we transformed our region to a 500x700 pixel image. An example of a transformed region is displayed in Fig. 4.

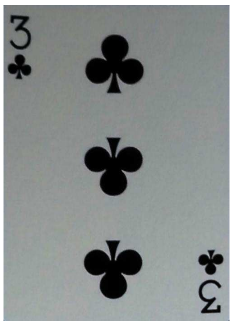


Fig. 4 Region After Projective Transformation

Since only the upper-left and bottom-right corners are used for template matching, we crop out the rest of the card, leaving only two 70x180 pixel regions. Next, we rotate the bottom-right corner by one-hundred-eighty degrees and place it next to the upper-left corner. This is the representation that will be used for template matching and is shown in Fig. 5.

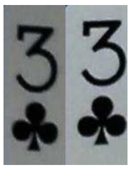


Fig. 5 Cropped and Concatenated Region

3) *Orientation Determination*

The corner detection algorithm used provides no guarantees about corner ordering. To resolve the orientation ambiguity, we apply the above image transformation once using the original corners and once using a circularly-shifted version of the original corners. Then, using a Prewitt edge detector, we compared the activity level to both orientations. The orientation with the most edge pixels is selected as the correct orientation while the other orientation is discarded.

4) *Template Matching*

Prior to template matching, we run a Canny edge detector against the corner image, displayed in Fig. 6. We found that straight template matching resulted in poor performance, especially for suits. We achieved much higher accuracy by matching only the edges.

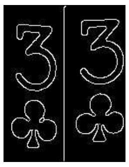


Fig. 6 Cropped and Concatenated Region after Edge Detection

We created the template images by taking sample pictures of cards and running them through the algorithms previously listed (neglecting the double corner). From the corner, we extracted two templates: one for suit and one for rank. When our MATLAB script begins, we read each template and dilate it slightly. The dilation helps to ensure that we are robust to slight edge shape discrepancies caused by minor imperfections in corner detection.

We perform template matching once for each suit template and once for each rank template. Template matching is implemented as a phase correlator. For computational efficiency, we pre-compute the Discrete Fourier Transform (DFT) of each template on startup. The correct rank and suit are chosen by selecting the template with the highest peak correlation.

5) *Statistic Calculation*

We generate statistics using a Monte Carlo simulation. First, we generate ten thousand full hands, starting each one with the known cards. A full hand is considered to be seven cards, five community and two pocket. From the full hand, we determine the classification and increment our histogram. The final results are reported to the client, along with the cards identified.

C. *Challenges*

Many algorithms for rapid fast image recognition rely upon feature point calculation and comparison. Examples include Speeded-Up Robust Features (SURF) and Scale Invariant Feature Transform (SIFT). Such methods are advantageous because they are robust to rotation, scale, and partial occlusion. However, when we implemented SURF with Random Sample Consensus (RANSAC), the results shown in Fig. 7 were downright poor. This poor performance is explained by the highly repetitive nature of features on playing cards combined with identical overall outline. Because the suit identifier can be found up two twelve times on a single card, a feature point from one card may match many locations on another card. Theoretically, RANSAC should mitigate this issue, but in practice, weakened by the need to calculate a projective transform, the matching failed.

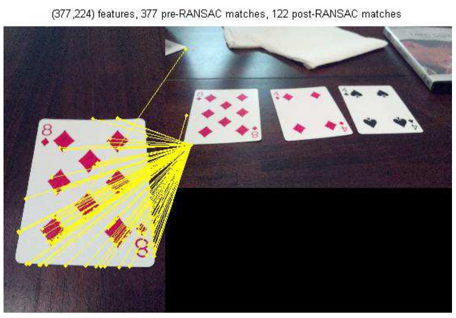
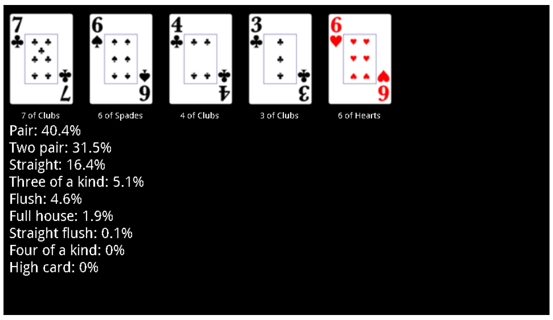

(377,224) features, 377 pre-RANSAC matches, 122 post-RANSAC matches

Fig. 7 RANSAC Matching Failure

IV. Results

An example photo taken by our application is shown in Fig. 1. The processed result is displayed in Fig. 8. The statistics show the likelihood that an opponent has each of the possible poker hands. As the reader can see, there is a 40.4% chance an opponent has nothing better than a pair. This is the most likely outcome, and makes sense as there is a pair of sixes on the table. There is a 31.5% chance of an opponent having two pairs, a 5.1% chance of three of a kind, and a 4.6% chance of a flush. Note that our algorithm correctly identifies that an opponent will never have a hand classified as "High Card", since the community cads dictate that he must have at least a pair.



7 of Clubs 6 of Spades 4 of Clubs 3 of Clubs 6 of Hearts

```
Pair: 40.4%
Two pair: 31.5%
Straight: 16.4%
Three of a kind: 5.1%
Flush: 4.6%
Full house: 1.9%
Straight flush: 0.1%
Four of a kind: 0%
High card: 0%
```

Fig. 8 Results of Image Detection Displayed on Android Phone

Our algorithm is generally consistent and accurate. For photos taken at a 45 degree angle at a distance of 24 inches, our overall accuracy in identifying both suit and rank is 88%, based upon a test of 156 cards. Rank is correctly determined 96%, and suit is identified correctly 93%, as displayed in Fig. 9.

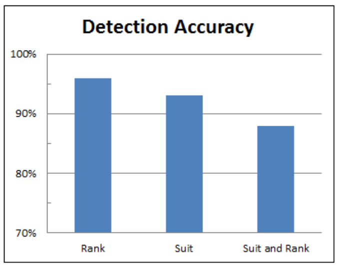


Fig. 9 Overall Detection Accuracy

Digging further into the results, our algorithm has issues detecting 3s (Fig. 10) and diamonds (Fig. 11)

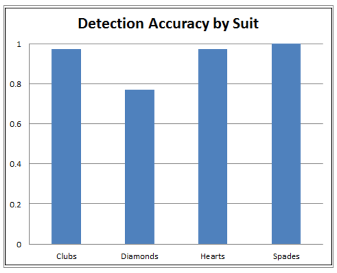


Fig. 10 Detection Accuracy by Suit

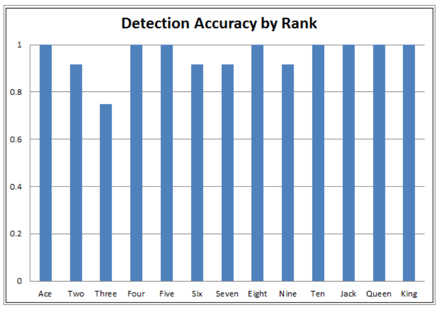


Fig. 11 Detection Accuracy by Rank

We are unsure as to exactly why our algorithm had such difficulty with said suits and ranks. However, we conjecture that these shapes display the most similarity to other ranks or suits, and are therefore most frequently mistakenly detected.

V. Further Work

While our algorithm performance is acceptable and our application is usable, there is room for improvement. As we rely upon separate rectangular shapes our algorithm fails if a card is even slightly occluded. Furthermore, it is very sensitive to reflections, so our application does not perform well on shiny surfaces. The mobile application response time is around fifteen seconds, with nine seconds attributed to upload time, and six seconds due to processing. This could be reduced by transmitting a grayscale image, or doing the processing locally.

VI. Conclusion

While playing card detection is not a new problem, we believe we have implemented a novel approach to detecting cards subjected to scale and projective distortion. This algorithm is demonstrated through a mobile application which detects cards with reasonable accuracy and returns relevant statistics.

References

[1] C. Zheng, R. Green, 'Playing Card Recognition Using Rotational Invariant Template Matching' Proceeding Image of Vision Computing New Zealand 2007, pp. 276-281, Hamilton, New Zealand, December 2007.

[2] K. Zutis, J. Hoey, 'Who's Counting?: Real-time Blackjack Monitoring for Card Counting Detection'.

http://www.computing.dundee.ac.uk/staff/jessehoey/papers/zutis_hoey_blackjack09.pdf.

[3] G. Hollinger and N. Ward, “Introducing computers to blackjack: Implementation of a card recognition system using computer vision”. http://www.andrew.cmu.edu/user/gholling/home/IEEEHollinger.pdf.

[4] Open Source Computer Vision. http://opencv.willowgarage.com.

[5] Poker Probabilities. http://wizardofodds.com/games/poker/.

VII. Appendix

Division of labor:

Java App Development:	Hugh
Server CGI Port:	Dan
Testing:	Dan
SURF RANSAC Investigation:	Hugh
Presentation Editing:	Hugh
Report Editing:	Dan
Report Development:	Both
OpenCV Programming:	Both
Matlab General Algorithm:	Both
Corner Detection:	Dan
Orientation Determination:	Dan
Projective Transformation:	Hugh
Template Matching:	Both
Statistic Generation:	Dan