Playing Card Detection on a Standard 52-Card Deck

Richard Hanulewicz
University at Buffalo
Buffalo, NY
rshanule@buffalo.edu

Abstract

In this paper I present an algorithm for detecting and identifying playing cards from a standard 52-card deck using the French playing card symbols: Diamond, Heart, Spade, and Club with values ranging from 2-10, J, K, Q, and A. I implement this algorithm in Python using OpenCV 2.

1. Introduction

The point of my project is to design a flexible algorithm that can detect a single playing card on a reasonably dark surface, re-orient it and extract the necessary information for identification against a Template Deck. By flexibility, I mean it does not require an extremely strict environment to function. One should be able to take a photo of a standard playing card, in a reasonable position (cannot correct perspective distortion, but can correct rotation), on a dark surface and feed it into the algorithm to get a correct response.

2. Related Works

2.1. Texas Hold em Hand Recognition and Analysis

This paper by Dan Brinks and Hugh White explores the same problem but with the larger scope of detecting an entire hand of cards. The general methods for single card detection used within the larger hand detection algorithm provided the base ideas for my own algorithm implementation. I did not follow it to its fullest due to differences in scope and time constraints.

2.2. Playing Card Recognition Using Rotational Invariant Template Matching

This paper by Chunhui (Brenda) Zheng and Dr Richard Green provide a smaller scope for playing card recognition that more closely matches the scope of my own project. It is primarily concerned with using affine transformations to efficiently correct the rotation of playing cards to detect and identify them in a more flexible environment. They also use template matching for their recognition system, a method also used in my own implementation.

2.3. Whos Counting?: Real-Time Blackjack Monitoring for Card Counting Detection

This paper by Krists Zutis and Jesse Hoey explores a different problem of detecting whether or not Blackjack players are counting cards. However, in the implementation of this much larger-in-scope system, they have come up with their own individual card detection system. It differs a bit from the last two papers as well as from my own implementation, but it offers interesting insights into how card detection could be done differently and other ways that information can be isolated and extracted from the card.

2.4. Introducing Computers to Blackjack: Implementation of a Card Recognition System using Computer Vision Techniques

This paper by Geoff Hollinger and Nick Ward is an earlier computer vision project with the same goal of detecting all the members of a standard deck of playing cards. Their procedure for card recognition is similar to my own, but their scope goes beyond this and introduces the ability for the computer to use its knowledge of cards to actually play Blackjack. It will look at a human's hand, for example, and be able to recommend which action should be taken by the human player. The computer can also play autonomously against the human.

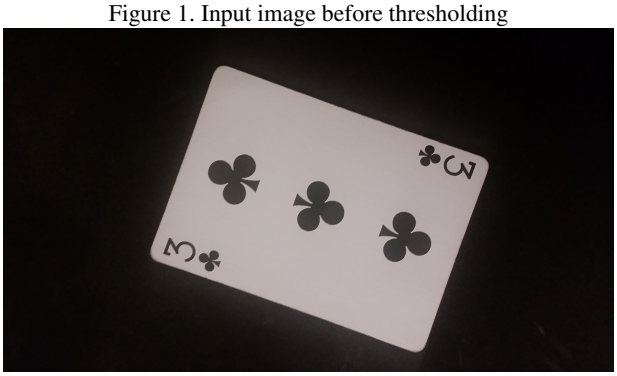
3. Algorithm

3.1. Outline

Threshold Input Image
Perform Opening
Find Rotated Bounding Box of Card
Perform Homography Transformation
Find Bounding Boxes of Rank and Suite
Isolate and Resize Rank and Suite Images
Template Matching

3.2. Threshold Input Image

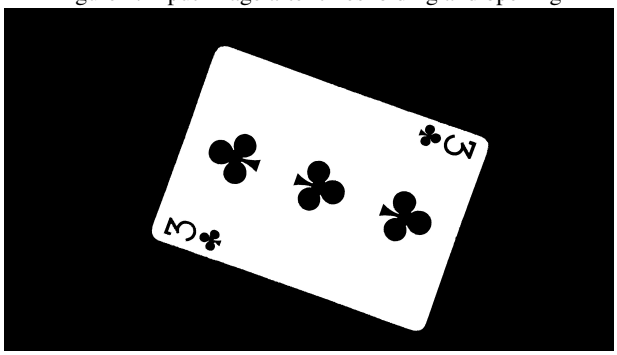
Figure 1. Input image before thresholding



First, I import a photo of a playing card placed alone on a dark surface as a grayscale image. Then, I threshold the image to turn it into a binary image where the intent is for the dark background to disappear and the remaining white pixels to represent the card. I choose a thresholding value of 128, which seems to work fine, but this can be tinkered with.

3.3. Perform Opening

Figure 2. Input image after thresholding and opening



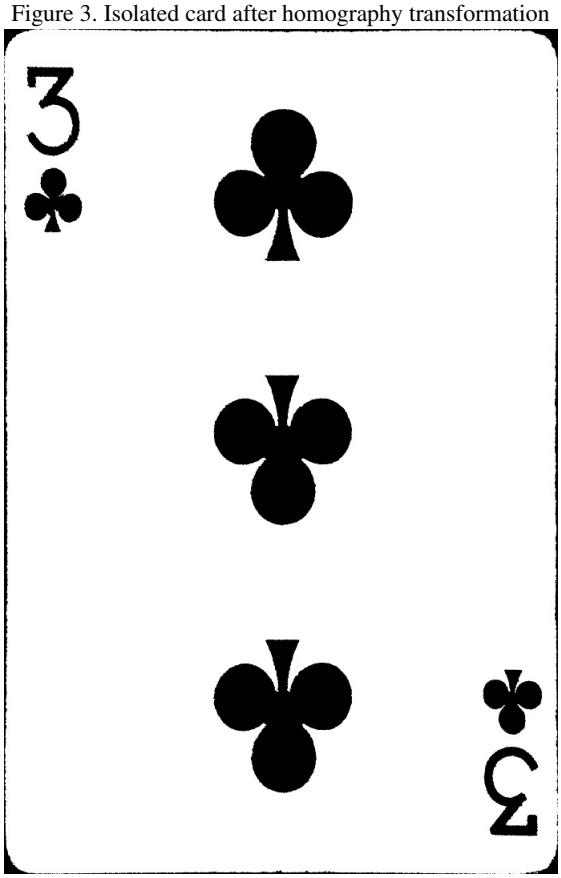
Photos tend to have some amount of noise in them, which can produce scattered extraneous white pixels outside the area of the card after thresholding. To remove this "noise", I perform morphological opening with a kernel size of 6 on the thresholded image.

3.4. Find Rotated Bounding Box of Card

To isolate the card from the rest of the image, I first find the contours using 'findContours'. I distinguish the contour for the whole card by picking out the largest contour. I then use 'minAreaRect' to find the rotated minimum-area rotated rectangle that contains the card. I can convert that rectangle to a set of four corner points by using 'boxPoints'. I now know the coordinates of the four corners of the card in my image.

3.5. Perform Homography Transformation

Figure 3. Isolated card after homography transformation



Since it's possible, and even very likely, that the original photo of the card is not perfectly straight, I apply a homography transformation on the four corner points using 'findHomography' and 'warpPerspective' to "straighten" out the card and orient it correctly. This also isolates the card into its own image of a set pre-determined size (691 x 1056).

3.6. Find Bounding Boxes of Rank and Suite

First, I crop the image of the card down to the top corner so that the only visible information is the Rank and Suite symbols. I perform the same thresholding and opening on this cropped image, as I did to the original image, to remove any artifacts introduced by the homography transformation in the previous step. I then invert the binary image such that the Rank and Suite are white pixels and the rest of the image is black pixels. I then use 'findContours' on the image and distinguish the contour for the Rank by choosing the first contour from the bottom, and the contour for the Suite by choosing the second contour from the bottom. I use 'boundingRect' on the contours to find the Bounding Boxes around the Rank and Suit symbols.

3.7. Isolate and Resize Rank and Suite Images

Figure 4. Isolated and resized Rank and Suite images

I then use the dimensions of their respective Bounding Boxes to isolate the Rank and Suite symbols into their own separate images. I then resize those images to a set predetermined size (100x100).

3.8. Template Matching

Figure 5. The Template Card for Three of Clubs

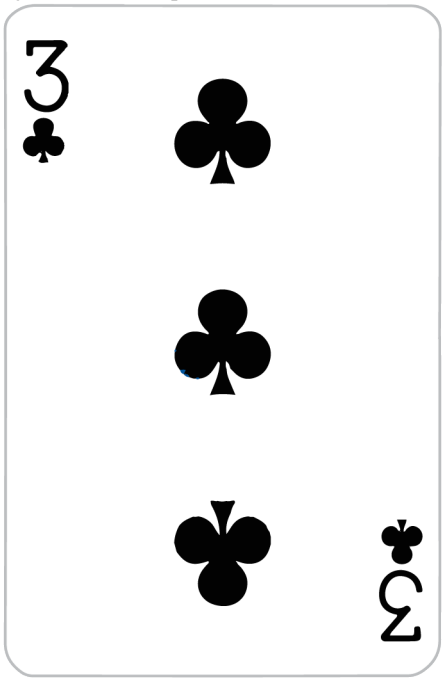
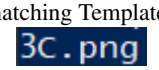


Figure 6. Visualized differences between Candidate Card and Template Card Ranks and Suites

For the purposes of actually identifying the cards, a Template Deck is maintained by the program for the Candidate Card (the input) to be compared against. There is a Template Card for each of the 52 cards. The algorithm loops through every card in the Template Deck and performs all the same operations on each card to isolate the Rank and Suite. The difference between the binary images of the Rank and Suite from the Candidate Card and those from each Template Card is calculated. The Template Card that produces the lowest overall difference from the Candidate Card is reported as the answer to the identification problem.

Figure 7. Console output of the program. "3C.png", the Three of Clubs, was the closest matching Template Card.



4. Experimental Analysis/Results

4.1. Accuracy

I tested my program on a full Bicycle brand deck of 52 playing cards. The design of the cards was fairly standard, but different enough from the Template Deck maintained by the program to offer a small challenge in identification. Nonetheless, the identification was very accurate, but not completely consistent across photographs given different lighting and positioning. In a single pass through the whole deck, only 3 cards were incorrectly identified:

8 of Hearts was identified as a 6 of Hearts
6 of Hearts was identified as a 5 of Hearts
3 of Hearts was identified as a 5 of Hearts

As you can see, the suite was consistently identified correctly, but the rank had some trouble. The numbers 8, 6, 5, and 3 are fairly similar in form, so it's understandable how Template Matching could result in such misidentification on a somewhat unclear picture. These cards were later correctly identified by simply taking better photos. There appears to be no algorithmic significance to the fact that all three misidentified cards were Hearts. I chalk it up to coincidence.

4.2. Optimizing Parameters and Environment

For the identification to work reliably, the lighting in the environment for the photo should be plentiful and there should be a stark contrast between the card and the background. The photo should also be a straight-on shot, not at an angle. I found that the lighting in my room where I tested this was not entirely sufficient and resulted in more incorrect identifications. However, by modifying the threshold value for the initial thresholding operation, one can optimize the program for one's own environment to produce better results in differently lit environments. I also found that the lighting on the card should be evenly distributed across the card. That is, if the card is brighter in one corner

than the other, you risk the algorithm extracting information from the darker corner, which will lower the accuracy of identification.

4.3. Trouble Detecting 10's

It appears I got lucky in my first test pass through the deck. A quirk that was discovered through further testing is that my algorithm is actually not adequately suited to detect 10's as consistently as other cards. The reason for this is that the number 10 on the card is made up of two disjoint "contours", one for the '1' and one for the '0'. My algorithm only grabs one of them. The majority of the time, the '0' will be detected instead of the '1', so the '0' is used for identification purposes. However, it is not uncommon that the '1' will be detected instead of the '0'. This causes confusion when the program goes to compare the '1' to the '0's in the Template Deck. The identification step will fail if this occurs. This happens if the '1' is positioned even a pixel below the '0', as the contour detector just grabs the first two contours from bottom to top corresponding to Suite and Rank. Through further testing I have found that this error occurs approximately 40 percent of the time, making 10's the most difficult card for my algorithm to detect.

5. Discussion and Conclusions

5.1. Improving Card Detection

Currently, the algorithm can only properly detect a card in the space of a photograph if it is not taken at an angle. It can correct for rotation, but if there is any perspective distortion, this is not corrected. The flexibility of this algorithm could be improved if I found the four corners of the best fitting parallelogram rather than the four corners of the best fitting rectangle. I did not implement this because it requires an intimate knowledge of how to work with convex hulls in OpenCV, which I did not have time to figure out. In other words, this was a feature cut out of my implementation due to time constraints. This is a fairly basic feature in modern card detection algorithms, as well as some of the ones mentioned in my "Related Works" section.

5.2. Improving Identification Robustness

Overall, my program performed well and produced fairly accurate results. However, the misidentification of some similar numbers as well as the problem detecting 10's gets in the way of what could be a very robust system. At the root of this problem is using Template Matching for identification. Due to the fact that this system is not scale-invariant, the symbols need to be scaled and stretched to approximately the same size. This works in general, but if there is any significant offset between the input symbols and the template symbols, the Template Matching may fail. By replacing this system with a scale-invariant feature detection system, both the inaccuracy of detection and the trouble detecting 10's could be resolved (I would no longer have to isolate simply one contour from the Rank for the purposes of "fitting" it to set size). Dan Brinks and Hugh White in [1] fix a similar problem by performing Template Matching on edges instead of whole symbols.

5.3. Final Thoughts

Through the course of constructing this program, I have learned a great deal about what is necessary in a card detection and identification algorithm. The importance of thresholding, opening, contours, convex hull, homography transformations, and scale-invariant feature detection was made apparent by the results I received in my experimental analysis. The use of a simple bounding rectangle in place of convex hull and overlay Template Matching in place of more flexible feature detection proved to still produce a competent algorithm for card detection and identification, but it is not as robust as it could have been otherwise. The benefit of my implementation is that it is simpler and easier to implement given a limited amount of development time. If given more time, it is wiser to implement the fixes proposed in sections 5.1 and 5.2.

References

[1] D. Brinks, H. White, 'Texas Hold em Hand Recognition Analysis', Standford University, Palo Alto, CA

[2] G. Hollinger, N. Ward, 'Introducing Computers to Blackjack: Implementation of a Card Recognition System using Computer Vision Techniques', IEEE Paper Contest, 2004

[3] K. Zutis, J. Hoey, 'Whos Counting?: Real-Time Blackjack Monitoring for Card Counting Detection', School of Computing, University of Dundee

[4] C. Zheng, R. Green, Playing Card Recognition Using Rotational Invariant Template Matching, Proceedings of Image and Vision Computing New Zealand 2007, pp. 276281, Hamilton, New Zealand, December 2007.

[5] Stack Overflow. https://stackoverflow.com

[6] Open Source Computer Vision. https://docs.opencv.org

[7] Edje Electronics, YouTube. https://www.youtube.com/watch?v=m-QPjO-2IkA

[8] sentdex, YouTube. https://www.youtube.com/watch?v=Z78zbnLlPUA