CS216 Final Project - Poker Recognize

Luis Hsu, Jeng-Hsun Ho

Luis' version

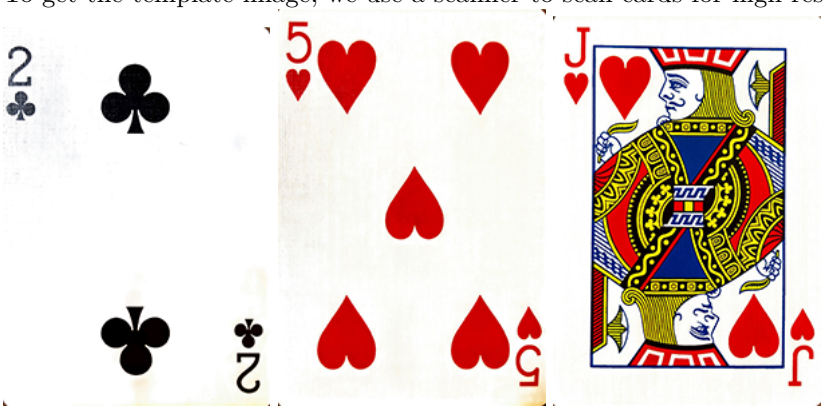
Overview

In this project, we're going to recognize poker cards from an image. Before testing, we'll prepare several images of poker cards as templates, and being used to train the matcher. Then, we'll take an image to test, and output the recognization result of matched card. The project seems like a simple application, however, it needs to combine several techniques we learned from this course. Combining these skills is the main value that make this project suitable as the final evaluation after taking this course.

Data Sets

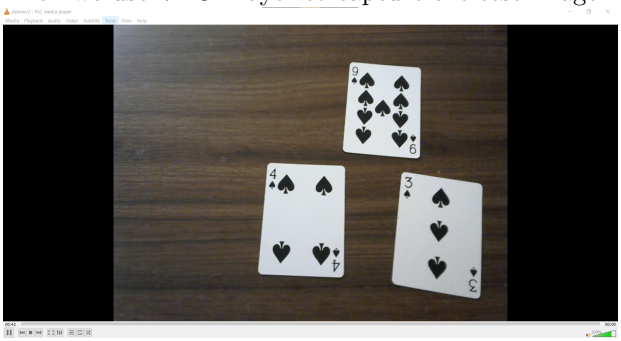
For templates, we use several poker cards from a standard Bicycle Playing Cards, which are the spades from 2 to 10, club 2, diamond K, heart 5, and heart J.
The set of spades is used to check if the application can distinguish the cards of different numbers from the same shape.
The cards of diamond, club and hearts are used for recognize different shapes.

To get the template image, we use a scanner to scan cards for high resolution images. Here comes some examples below.



The test images are taken with a webcam that set on a certain distance above a plain table, and the test items are putting on it, as the picture shown.

Then we use VLC Player to capture the test image from the webcam.



Algorithms

Overview of recognization process

- Get mask & contours
  - Blur
  - Canny edge detection
  - Find contours
  - Generate mask
  - Get contours from the mask
- Get SIFT features and descriptors of each RGB channel from test image
- Train FLANN matcher
  - Get SIFT features and descriptors of each RGB channel from template image

Add features and descriptors to FLANN matcher

Repeat for every template images

- Match & generate recognition matrix

  Get good matches from matchers

  Calculate recognition matrix by the frequency of good matches

  Wipe out lesser frequencies with in the matrix

- Output final results

Most of the detailed algorithms are leveraged with OpenCV library's implementation. We put them together for our overall process.

1. Get mask & contours

Most of time, the illumination is not perfect and may makes some shadows that interfere the result. Since the poker cards are in a close shape, we can assume that the items not in close shape are background or something not interested, and make a mask to filter them out.
To get the mask, these steps are applied:

1. Blur

   Blur image to filter out noises. Use OpenCV's implementation.

2. Canny edge detection

   Apply edge detection that the edges will be used by the next step. Use OpenCV's implementation.

3. Find contours

   Find contours from the edges. Use OpenCV's implementation.

4. Generate mask

   A black image will be created first. Then the contours will be filled with white color to generate a mask. This is also the solution that some contours may reside within another contour. Use the filling method will give us the outermost shape, and that's what we need. Use OpenCV and Numpy array operations.

5. Get contours from the mask

   After getting the mask, we are going to get the contours from the mask again. At this time, only the outermost shapes are remained, and the inner contours, like numbers, won't appear. That's how we can get the possible regions of cards from the mask now, and the contours will also be used in the later steps.

2. Get SIFT features & descriptors of each RGB channel from test image

After having the mask, it's time to apply SIFT algorithm to the image. Here we also use OpenCV's function to detect STFI features and calculate descriptors.
The SIFT functions in OpenCV only accept monochrome image, so we convert color image to grayscale at first. This method is runnable to the function, however, spade and heart cards are not distinguishable since the information of color is reduced, and the shape of spade and heart is similar.
To solve this issue, we separate the RGB channels of the image, and regard them as 3 independent monochrome images instead of convert to grayscale. With this change, spade and heart cards can finally be recognized correctly.

3. Train FLANN matchers

To match the test image with templates, a bunch of matches between the features are calculated by the FLANN matchers of OpenCV. In this step, 3 FLANN matchers are created for each RGB channel. Then, the SIFT features & descriptors of templates are calculated just as the same way of the test image. Finally, the SIFT descriptors of templates are added to FLANN matchers respectively. The matchers will be trained automatically right before finding matches.

4. Match & generate recognition matrix

Finally, we're going to do recognition with the preparations above. The matches are calculated by the FLANN matchers with the discriptor of test image as input.
Now we have the possible regions of cards and the matches. How do we map the card with these regions?
The idea is that if a region is a specific card, the matches of the card will located within such region.
With this idea, we can create a matrix that take region indices as row, and template indices as column.
The value is the amount of matches of each template that the region contains.
For example:

```
[[ 41   0   0   0   0  16   0   0   0   0   0   0   8]
 [  0   0   0 299   5   0   0   0   0   0   0   0   0]
 [  0 183   0   0   0   0   5   0   0   0   3   0   0]
 [  0   0  10   0   0   0   0   0   7   0   0   0   0]]
```

The "299" in [3, 1] means there are 299 matches of template 3 within region 1.
To calculate this matrix, we iterate through every matches and use OpenCV's "pointPolygonTest" function to check whether the match point is within each region. Besides, the value of the 3 channels will sum up together.
After that, since there will be only one specific card in a standard poker card set. We assume that one template only resides in one region, so we only leave the maximum value of the column, and zero out others. Moreover, we also zero out the column that total numbers are less then the threshold.

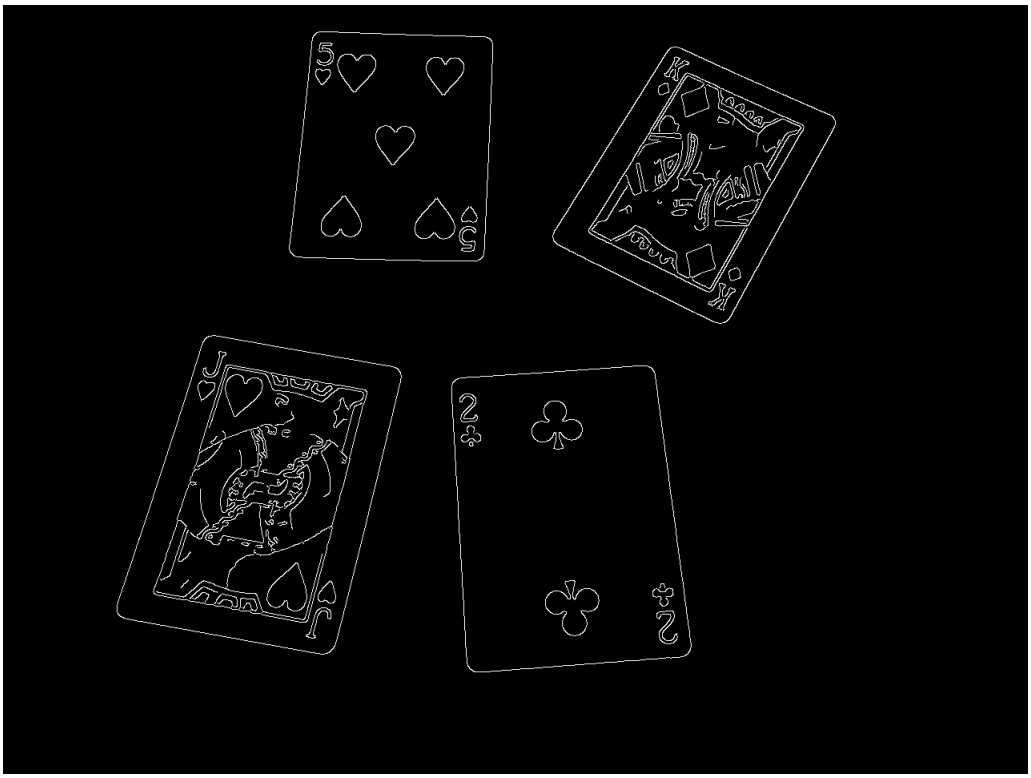
5. Output final results

The last step is to output and visualize the final results. We iterate through every template, draw the matches of the template, and highlight the region with the maximum value that not less then the threshold. There will be a set of images with the same amount of templates.

Results

Test image with 4 cards

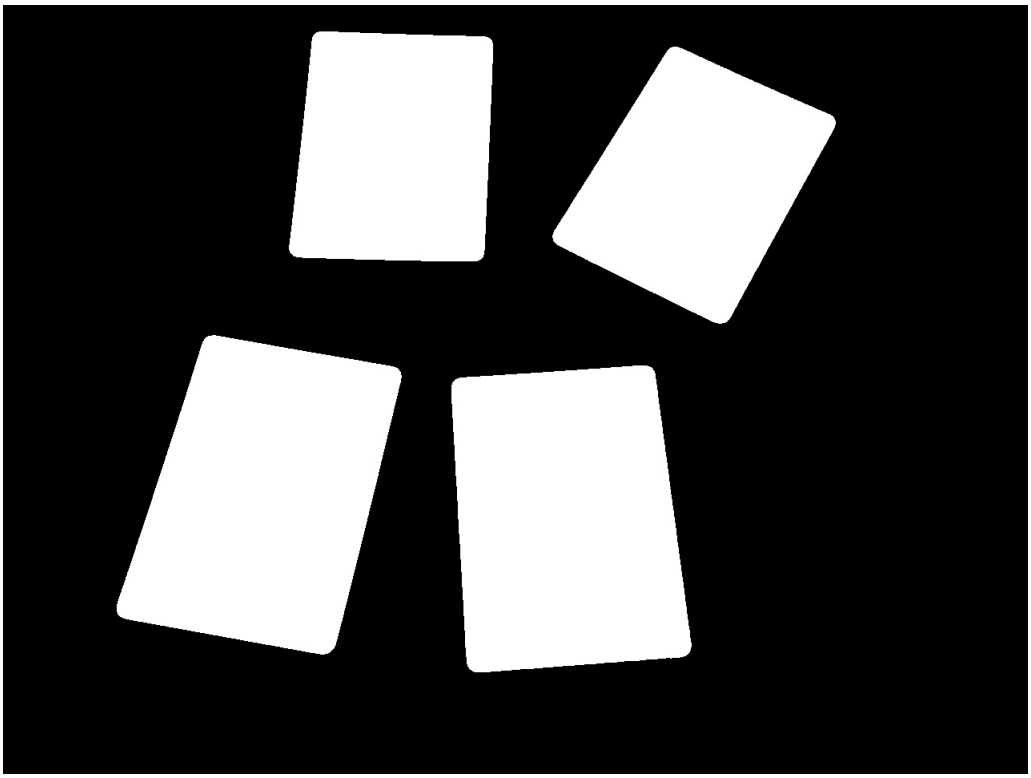
Canny edge detection



Contours detected from test image

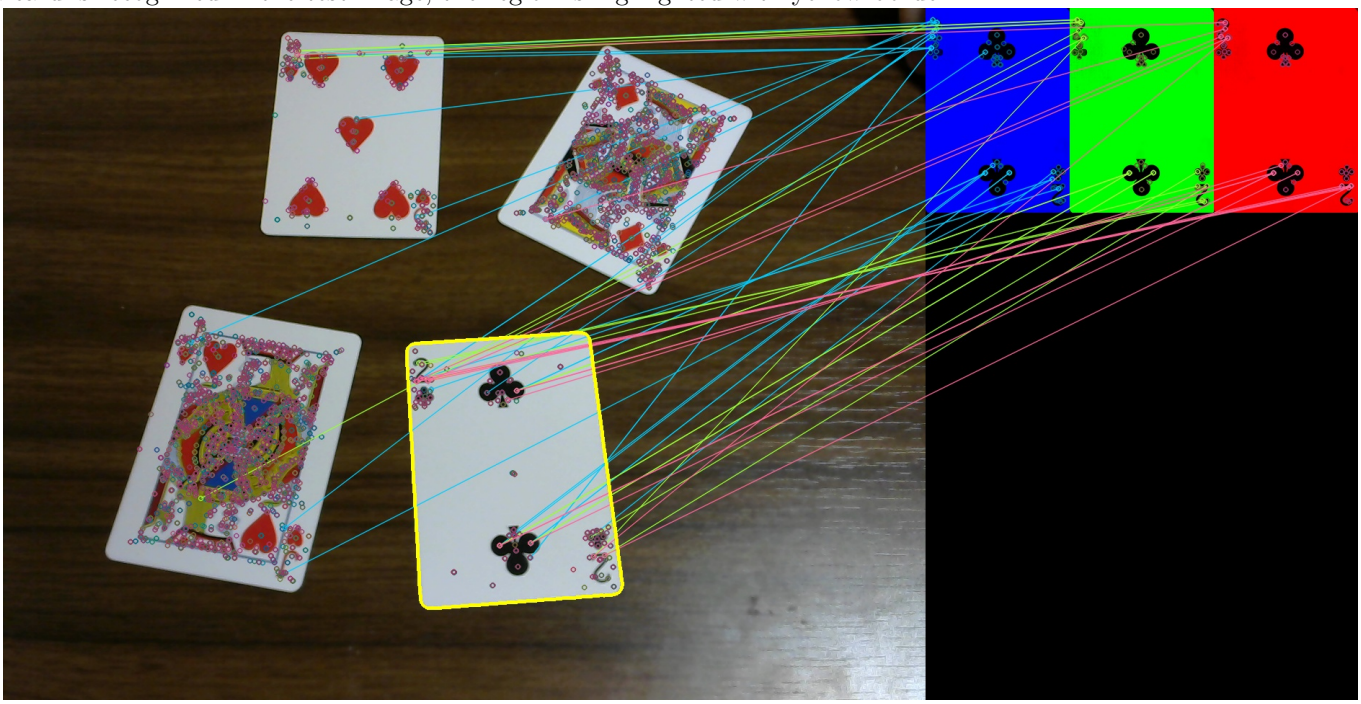
The borders of contours are drown with light green.

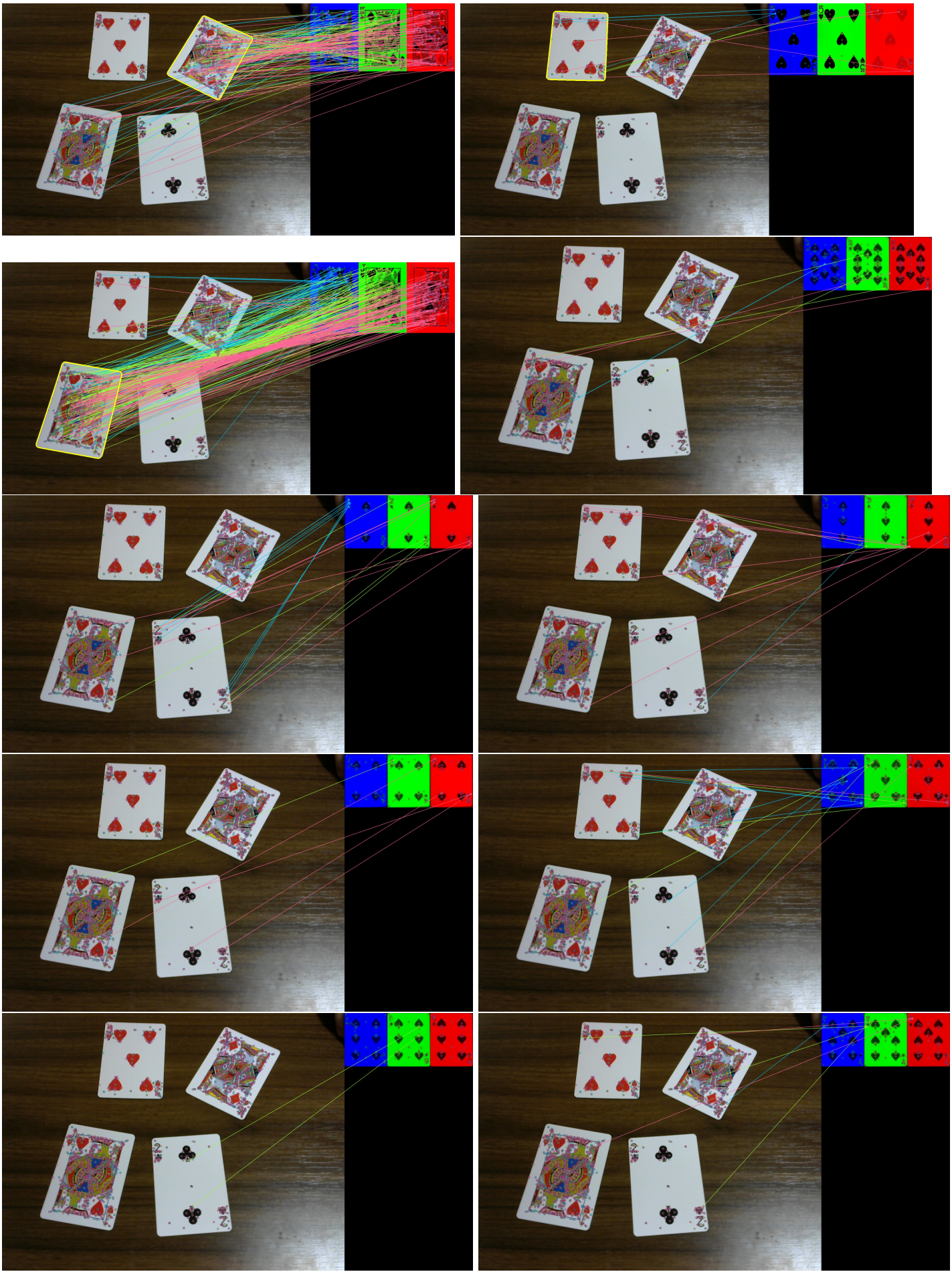
Generated mask

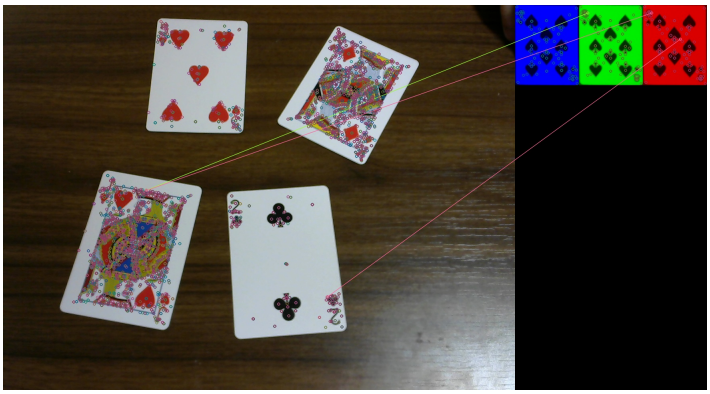


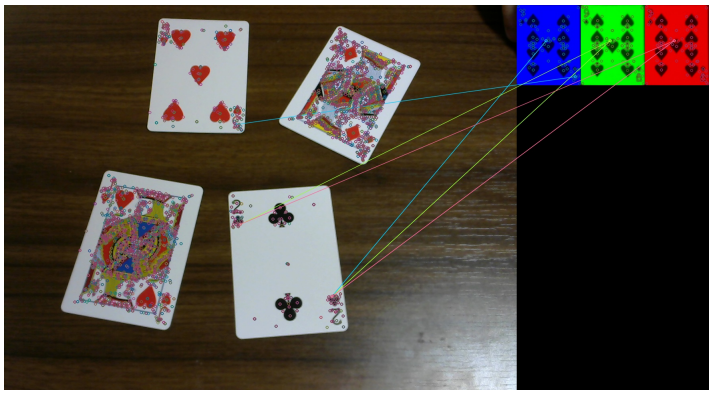
Final result

The left side is the test image, noted with dark red, green and blue circles indicate the SIFT features of RGB channels.
The right side is the images of Red, Green and Blue channels for a template, noted with SIFT features.
Matches of the RGB channels are drown with lines of light red, green and blue colors.
If a card is recognized in the test image, the region is highlighted with yellow border.





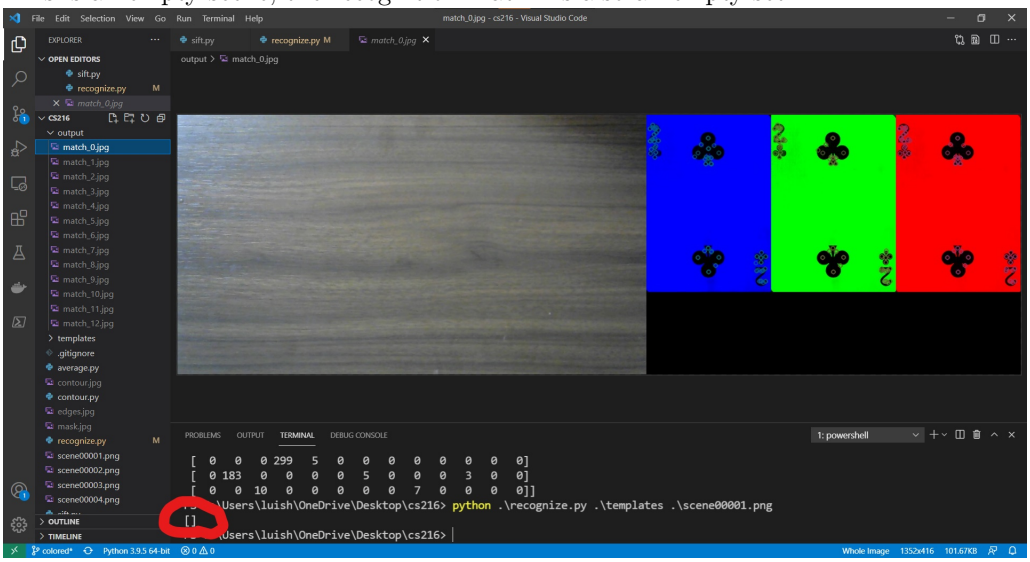




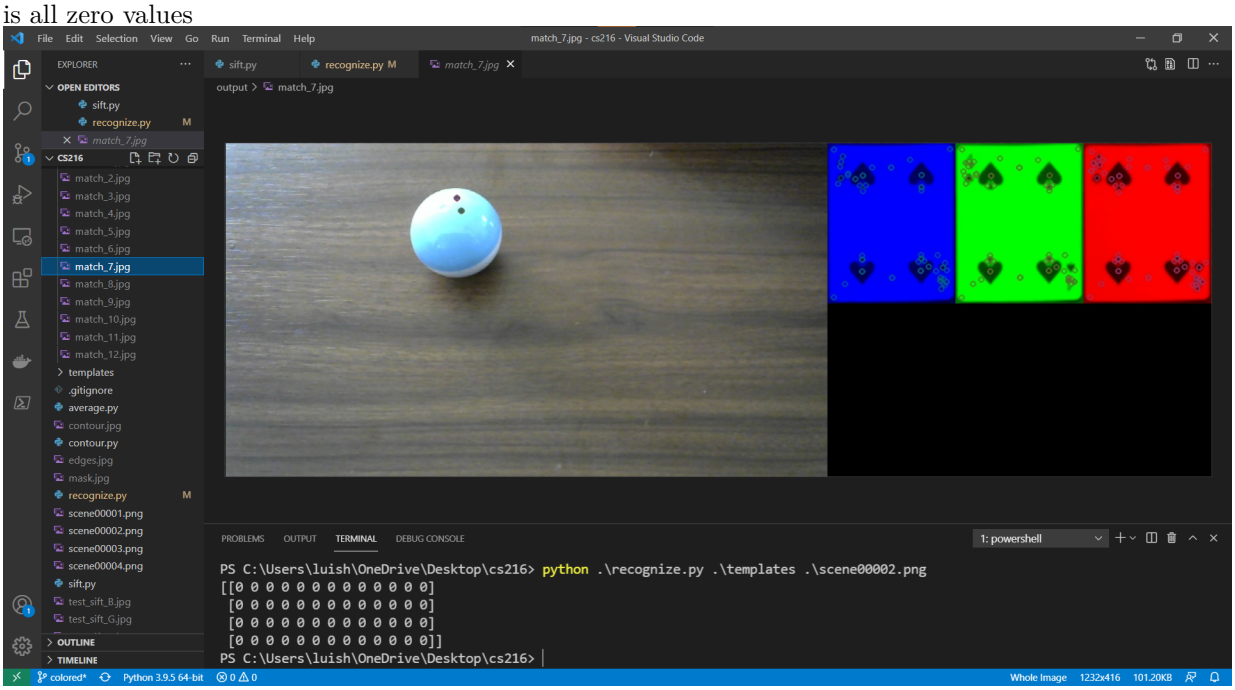
Additional result

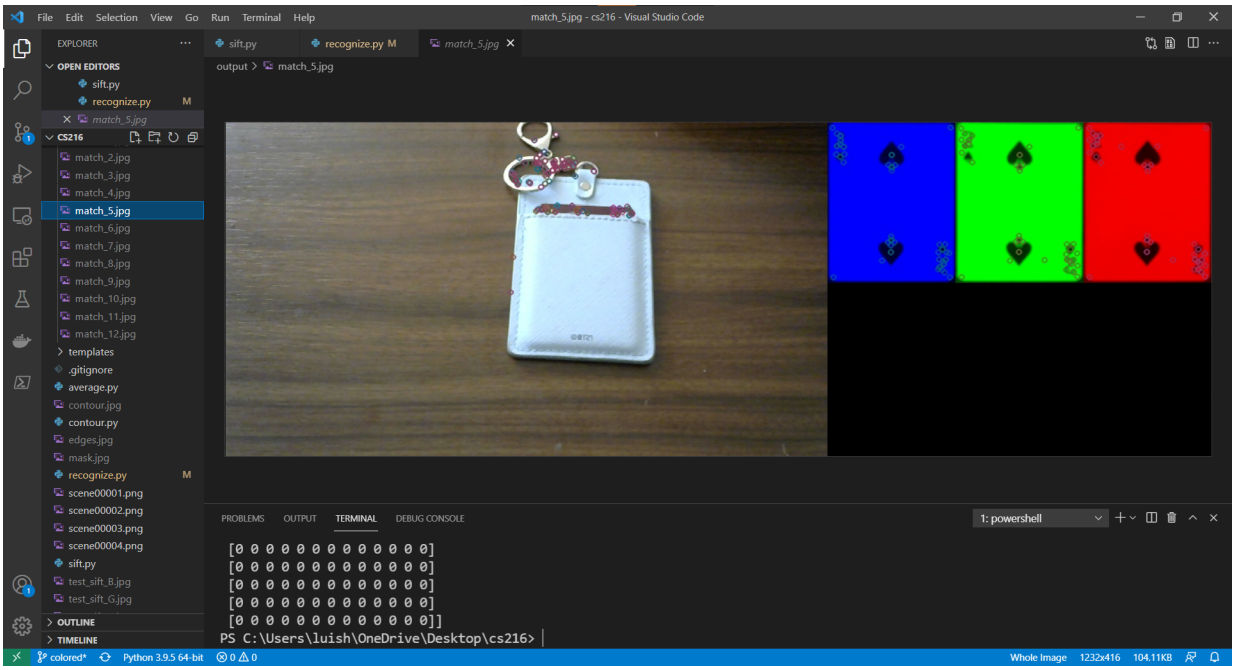
Here are 3 other test images that contains no cards.

This is an empty scene, the recognition matrix is also an empty set



These scenes contains random items. There are some closed regions within the image, however, the recognition matrix is all zero values





Assessment & Evaluation

Spotlights

The application of OpenCV tutorial can only one template within a process. Though the output results seem like the templates are recognized separately, they are actually matched at the same time. Moreover, this project can also detect multiple templates, not only one template as the tutorial did.
There are some other projects we've surveyed on the GitHub that also recognize poker cards. One of them using neural network to perform recognition, however, neural network is more complex then the computer vision way, and it also cost lots of computation power. Besides, another project implements in computer vision way, but it only take numbers and shapes as template instead of the whole cards, and that may the usage only limit to poker cards. In our approach, we can easily replace poker cards with some other template images, and they may also work. This is more flexible in usage than the other projects. To summerize, this project not only able to recognize poker cards but also achieve this goal in a simpler and more scalable method.

Limits & Assumptions

There are still some limitaion and assumptions with this project.

1. A specific card is assumed to appear only once.
2. The whole card should be shown in the image without cut off.
3. The cards can't overlap with each other.

These limitations exist since we use the whole contours without more detailed analysis, and the recognition matrix only use the maximum value instead of dedicated statistical analysis. They may be solved if we have more time to improve the algorithm in such points.

Future works

There are some future works we can do to improve this application.

1. Extract the training process that can be explicitly trained before testing. The trained parameters can be stored somewhere and then loaded during the testing process.
2. Find solutions for the limitations
3. Change the input source to real-time webcam capture instead of static image.

Resources

- OpenCV Python Tutorials Feature Matching

  `https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html`

- OpenCV Python Tutorials Feature Matching + Homography to find Objects

  `https://docs.opencv.org/master/d1/de0/tutorial_py_feature_homography.html`

- OpenCV Python Tutorials Contours in OpenCV

  `https://docs.opencv.org/master/d3/d05/tutorial_py_table_of_contents_contours.html`

- *Majumder, A., & Gopi, M. (2018). Introduction to Visual Computing: Core Concepts in Computer Vision, Graphics, and Image Processing. Natick: CRC Press.*

Appendix

These are main works in this cooperated project.

1. Create mask and detect contours (Line 13-22)

   The idea of create mask from contours was coming up by myself, and the implementation is composing with functions provided by OpenCV library.

2. Generate recognition matrix (Line 70-93)

   This idea is inspired with the Evaluation Metrics in the Assignment 1. Nevertheless, the algorithm and implementation was built on my own except for OpenCV's "pointPolygonTest" function.