**INFOIP Image Processing - Assignment 3**

## Part 1: Object to be detected: Playing Cards

Our object of choice is playing cards from a standard 52 card deck. The main goal will be to detect the cards themselves and therefore being able to count how many cards are in a given image. One of the possible refinements would be detecting the suit and rank of each card. We made this choice of object because playing cards appeared to be easy enough to detect in images with very little distractions and proportionally harder the more noise and distractions you add. Furthermore the challenge of detecting rank and suit seemed appropriate as an refinement option.

|  |  |
| --- | --- |
| **Criterium** | **Possible Values** |
| Minimum/Maximum size | All images are of size < 600x600, object size should be reasonably big (at least ~5% of image) |
| Lighting variations | Indoors, Artificial Light, No direct sunlight |
| Rotation variations | All possible rotations, front facing up |
| Occlusion | No occlusion, partial occlusion |
| Other | Only background noise, cards may be viewed from a slight angle, no white backgrounds |

## Part 2: Pipeline

Phase 1

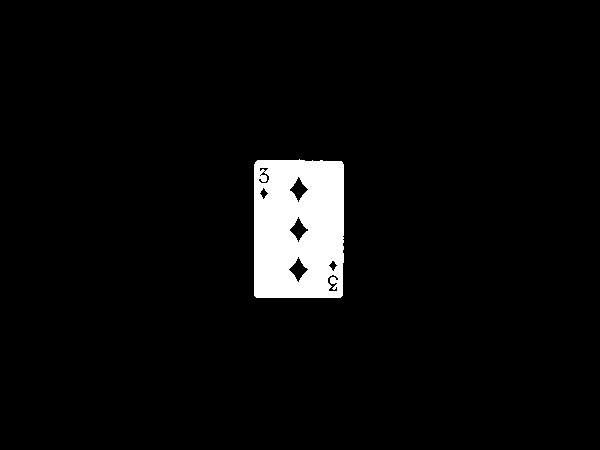
In the first phase of our pipeline we get rid of background noise and output a binary image where the white pixels indicate the white parts of a card. This is achieved by first converting the input image to grayscale and applying our custom threshold to it, which looks for the first pixel value that at least 10% of all pixels share, going from 255 to 0. After this value has been found we threshold at the pixel value 30 % below it. This has proven to give good results for playing cards, since we can expect to have large areas of the image be white. After thresholding the image we apply opening to a compare Image and geodesic dilation of the threshold image with the compare image to get rid of any background noise that might still be left. A simple example of the application of Phase 1 can be seen in Image 1 and Image 2.

Image 1: Input

Image 2: Output Phase 1

## Phase 2

## In phase two we continue on the images retrieved from phase one. We created subimages for every connected shape, and calculate the chaincodes. If there are more than 10 subimages, the subimages are sorted by the area calculated from the chaincodes, and the highest 10 are selected. The remaining images are filtered on the amount of ‘holes’ in the shape. The holes can be anything. The threshold is four, since playing cards always have four or more holes (Ace being the playing card with four), the maximum amount of inner shapes is set at 25 to filter out any objects which are definitely not playing cards. Afterwards the remaining images are printed on a black background with a different color for every shape (which is also randomly generated).

## Phase 3

## This phase has not been implemented. But the idea was to do template matching on the suit and rank of the card.

## Part 3: Parameters

An important part of every program are the defined parameters by which the measures are taken, since they directly influence the outcome of the pipeline and are crucial for getting the right result out of a multitude of input images.

The first 2 important parameters are used in our custom automatic threshold. First we look for the first pixel value that is shared by at least **10%** of all pixels, to determine the predominant white value, afterwards we threshold on that value times **0.7** to add a bit of leeway. This choice of values was empirically determined and appeared to produce great results, when looking for playing cards in our images. A known problem that results out of this method and choice of parameters is the restriction that we cannot use images with white backgrounds, since we assume that the white of the cards is the “whitest” color in the given image.

Another is important parameter is the sampling accuracy used for calculating the Hough graph, which would in our case be **600**. This means that we draw **600** lines through every edge pixel in order to determine the lines going through the image. A sample size this big allows us to inspect almost every possible rotation. Accompanying the accuracy is the threshold used to determine the strongest value; it has to be chosen to fit our accuracy, since drawing more lines means more possible matches. For this we chose the value **100** as we found that it produced the most desirable results.

## Part 4: Example Outputs and Discussion

Example 1: 1 card without occlusion, not angled (best case scenario)

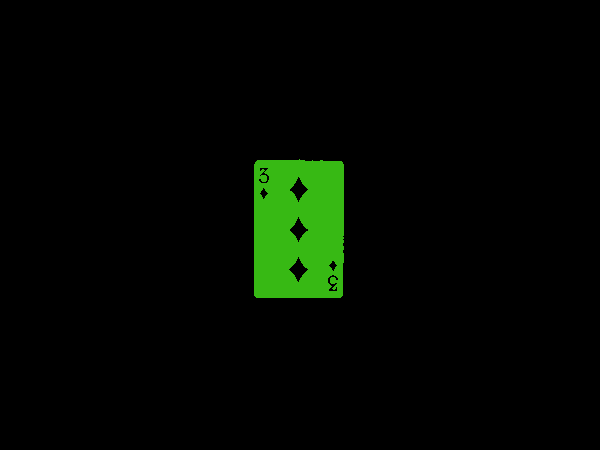


Image 4: Pipeline Output

Image 3: Input

This card has been perfectly detected, since it has very few distractive factors (or even none).

The color chosen to mark the detected cards is randomly generated, so the result may vary if repeated. All the background noise has successfully been filtered out in Phase 1 and Phase 2 could easily be applied to that result.

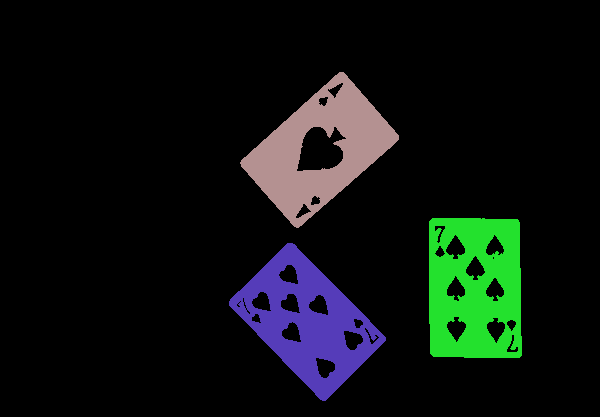
Example2: 3 cards, lighting changes, angled

Image 6 : Pipeline Output

Image 5: Input

All 3 of the cards have been correctly detected and they have been assigned individual color values. The small lighting variations, that are especially noticeable at the top right corner of the 7 Card, do not impair the card detection since the contour of the inner shapes, does not play a role in our detection process (yet).

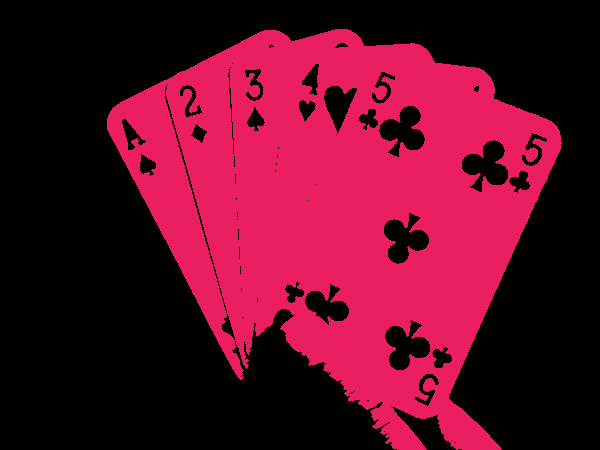
Example3: 5 Cards, highly occluded, not angled

Image 8: Pipeline Output

Image 7: Input

In this example we can see one of the main problems we encountered, that being occluded or rather overlapping cards. Since the borders between the cards are only very marginal it is very difficult to choose a threshold that properly preserves these edges and allows for the cards to be detected individually. Our pipeline still manages to flag the cards (alongside the finger) as playing cards, but it does think that it’s looking at one single card. The removal of the thumb in this image proved difficult, because it is very well lit and therefore does not stand out enough from the cards, to be properly removed from the image. This may have been solvable by including color detection into our pipeline.

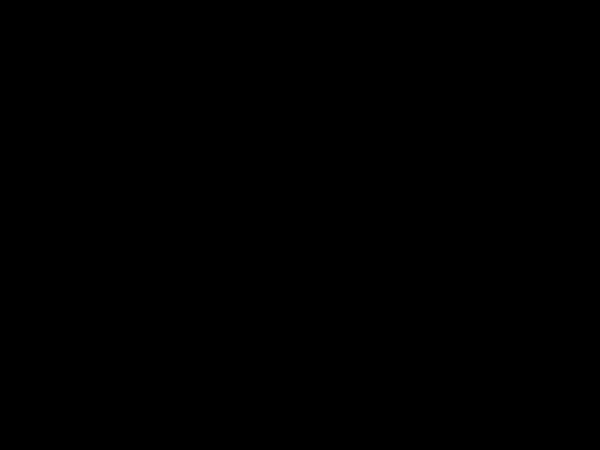
Example4: Distractor image, card-like shapes on table

Image10: Pipeline Output

Image 9: Input

This is an example of our program correctly recognizing that no playing cards are contained within the input image. The deciding factor here would be the amount of inner shapes being too high for a playing card (way more than 50 inner shapes for each object). Most of the noise objects are being filtered by the area check, since only the biggest 10 out of the 74 shapes detected in this image are being looked at.

# Remarks:

Some of the functionalities we implemented did not make it as far as to be used in the card detection, since i.e. the Hough Transform, towards the end, when the actual line drawing was working perfectly, we did not manage to properly implement the intersection detection and therefore were not able to use the information provided by the Hough Lines to tell us something about the image.

Our shape-labeling function takes a very long time to process, which we could not get rid of by adjusting and optimizing the code.

Our Program currently only detects the 10 “biggest” shapes in the given image. This restriction has been added due to time concerns.

# Reflection:

We noticed that we spent a lot of time on implementing things we did not end up using in our final program and in general spent a lot of time just thinking about what functionalities could yield good results and would help us in our detection process. Making a small design document in the beginning and asking for feedback about the chosen path would have helped us avoid that.