Assignment 1

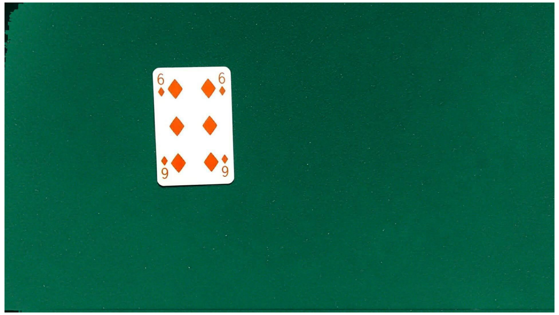
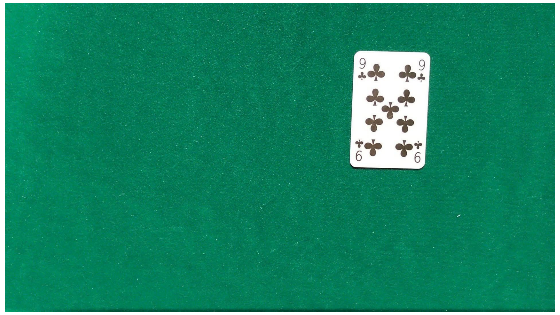
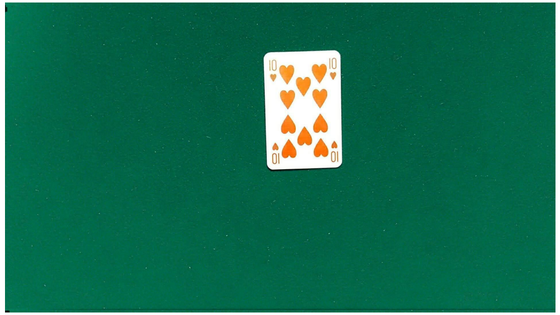
Basic Image Processing
Fall 2019

The problem

This problem is a real life scenario. People get paid for such programs! :)

Given a high resolution RGB camera watching a casino table. The program has to:

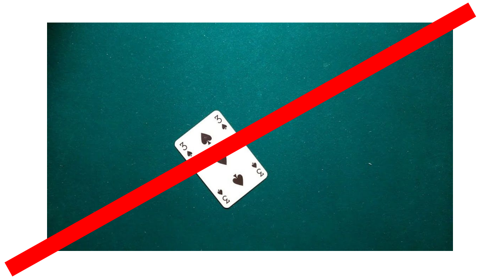
- detect a card on the table,
- crop the card,
- tell the color of the card (🟥 or ⬛), and
- tell the suit of the card (♣, ♦, ♥, ♠)

Specification

You can be sure that:

- The casino table is always green and the illumination is pretty much constant.
- The cards are the same size.
- The camera is fixed (i.e. viewport angle, distance is constant).
- There is only one card per image.
- The cards are from the standard 52-card deck.
- Only values 2–10 will be shown; there are no J, Q, K, A cards or Joker.
- The cards are always placed in a portrait position, face up; there are no rotated or flipped cards.

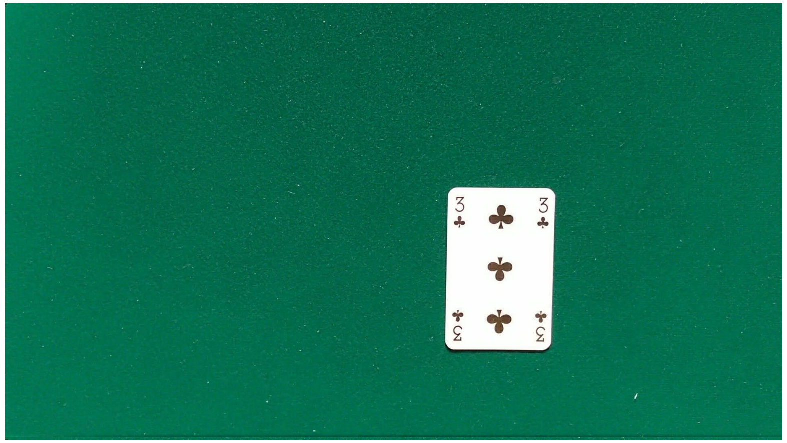


Theory – Finding cards on the table

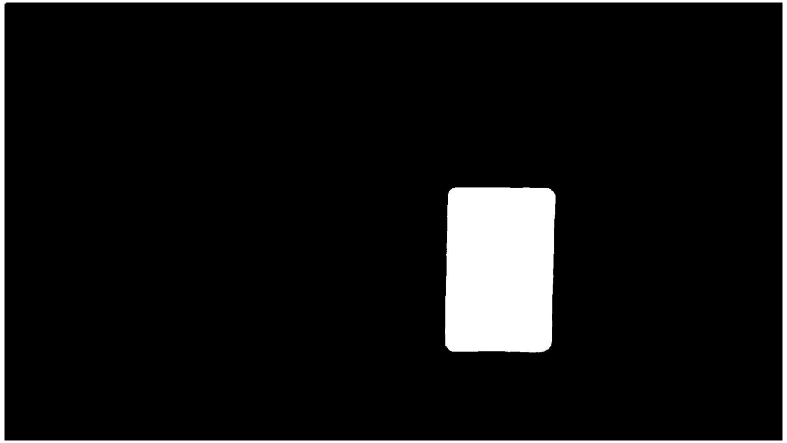
To find a card on the table you should use thresholding in the RGB space. Take into account that the table is green (and the cards are not green).

Similar to the duck and pine segmentation task (Lab 02) you can create a binary mask where *true* value indicate ‘card’ pixel.

Original input image



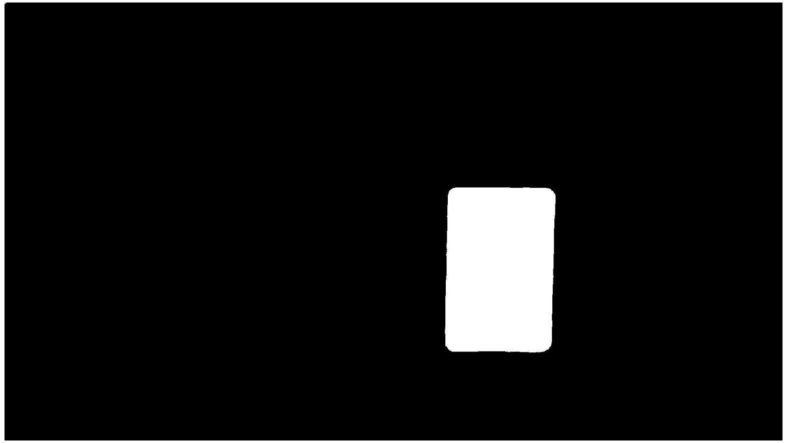
Binary mask



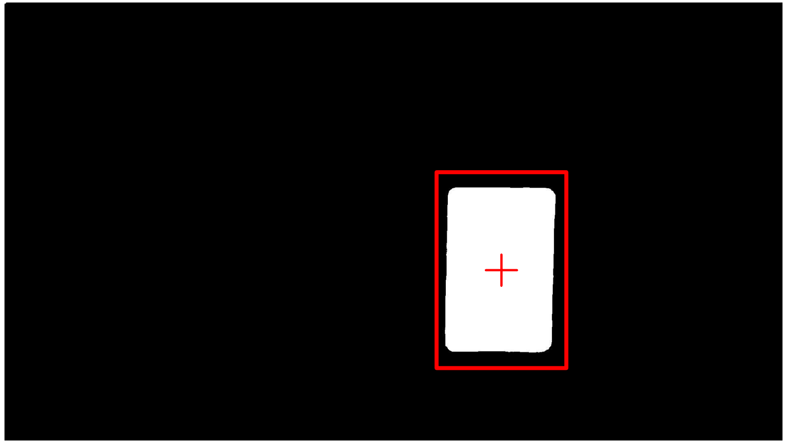
Theory – Cropping the cards

Use the fact that the cards are always the same size. In order to deal with the noise of the mask (white pixels outside the card) we compute the center of mass of the binary image. Then the card is cropped using this centerpoint and the known size of the card.

Binary mask



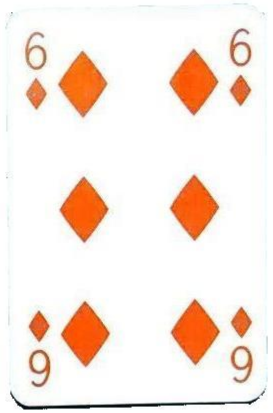
Center of mass and crop area



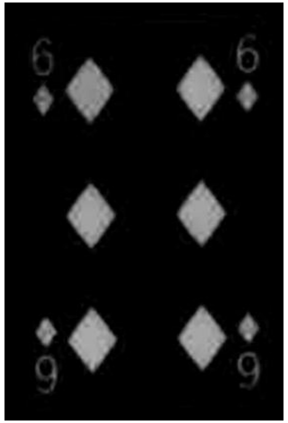
Theory – Color detector

A card contains red symbols if at certain areas the value of the red channel is very different from the value of the other channels (e.g. green). Based on the intensity values of the difference image the color of the objects can be easily determined.

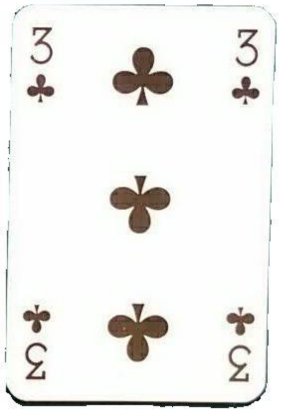
Cropped image



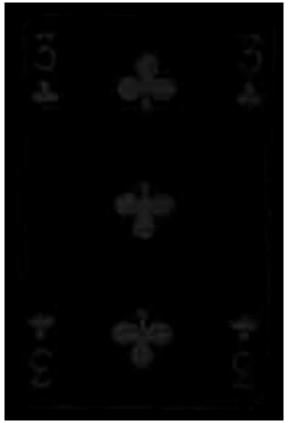
R-G channel difference



Cropped image



R-G channel difference

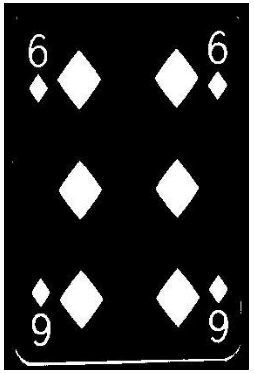


Theory – Suit detector

Suit detection is realized using a 2D finite impulse response filter where the filter’s kernel is the suit symbol itself. The filter is applied to the image using convolution.

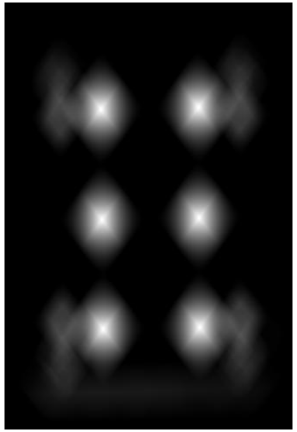
The result of the convolution (the response of the system) can be treated as a probability matrix: those parts of the image where the pattern of the image was similar to the convolution kernel will ‘glow’ in high values, others will remain dark.

`conv2(`



`,`

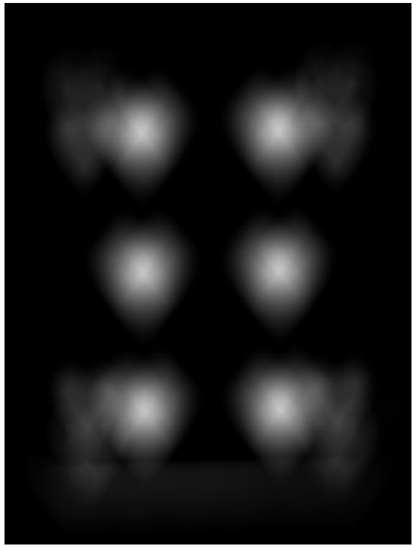
`) =`



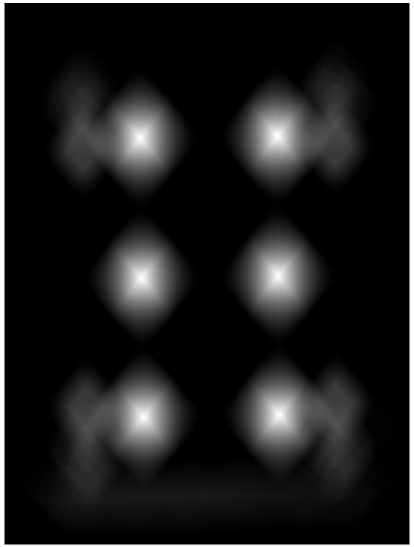
Theory – Suit detector

Convolve the same image with all the four kernels (corresponding to the four suits). Check the maximum value of each result. The likelihood of a suit is proportional to the maximum value in the image.

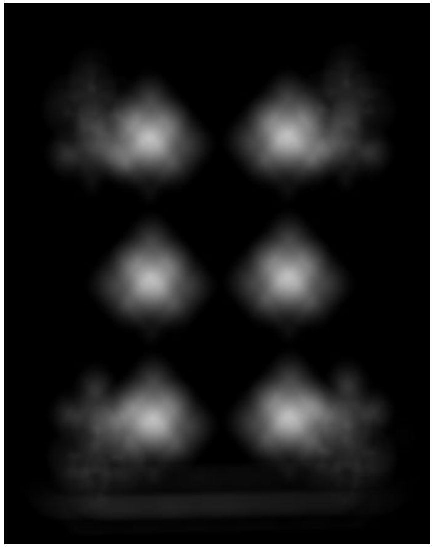
Conv. with Hearts
max = 0.2377



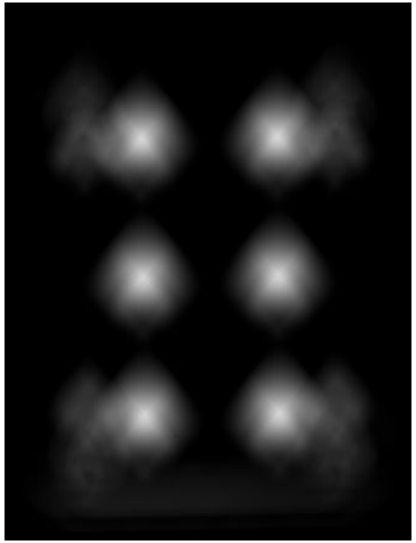
Conv. with Diamonds
max = 0.3041



Conv. with Clubs
max = 0.2291



Conv. with Spades
max = 0.2560



Please

download the ‘Assignment 1’ code package

from the

submission system

The maximum score of this assignment is

4 points

The points will be given in 0.25 point units.
(Meaning that you can get 0, 0.25, 0.5, 0.75, 1, 1.25 etc. points).

Exercise 1

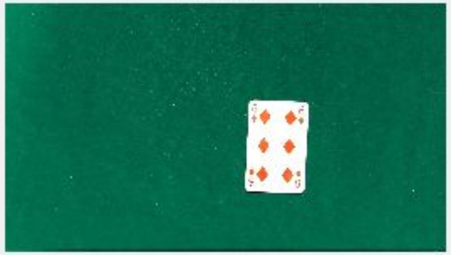
Implement the function `create_mask` in which:

- Based on an arbitrarily chosen threshold (it is your task to find a suitable threshold value) create a binary mask of the input image.
  Suggestion: use the difference of the green and red layers
- The mask should contain 1 if the pixel belongs to the card and 0 otherwise.

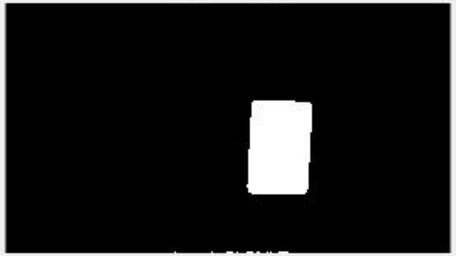
You can assume that the input of the function is a uint8 type RGB image with values in the [0, 255] range. You should return a logical array (only zeros and ones). The returned matrix must be the same size (width, height) as the input.

You can test your function by running `test1.m`. If the difference is not significant (i.e. err < 0.01) then you are really good :)

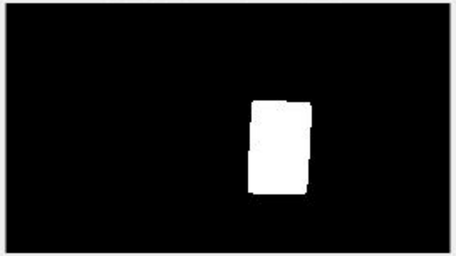
Input



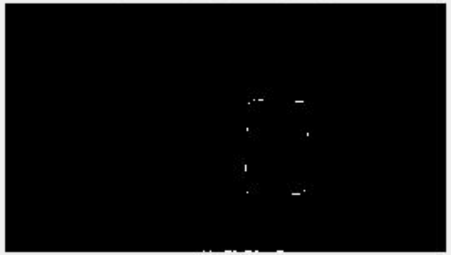
Your mask



Ground truth mask



Difference between masks
err = 0.0012804 %



Exercise 2

Implement the function `find_center` in which:

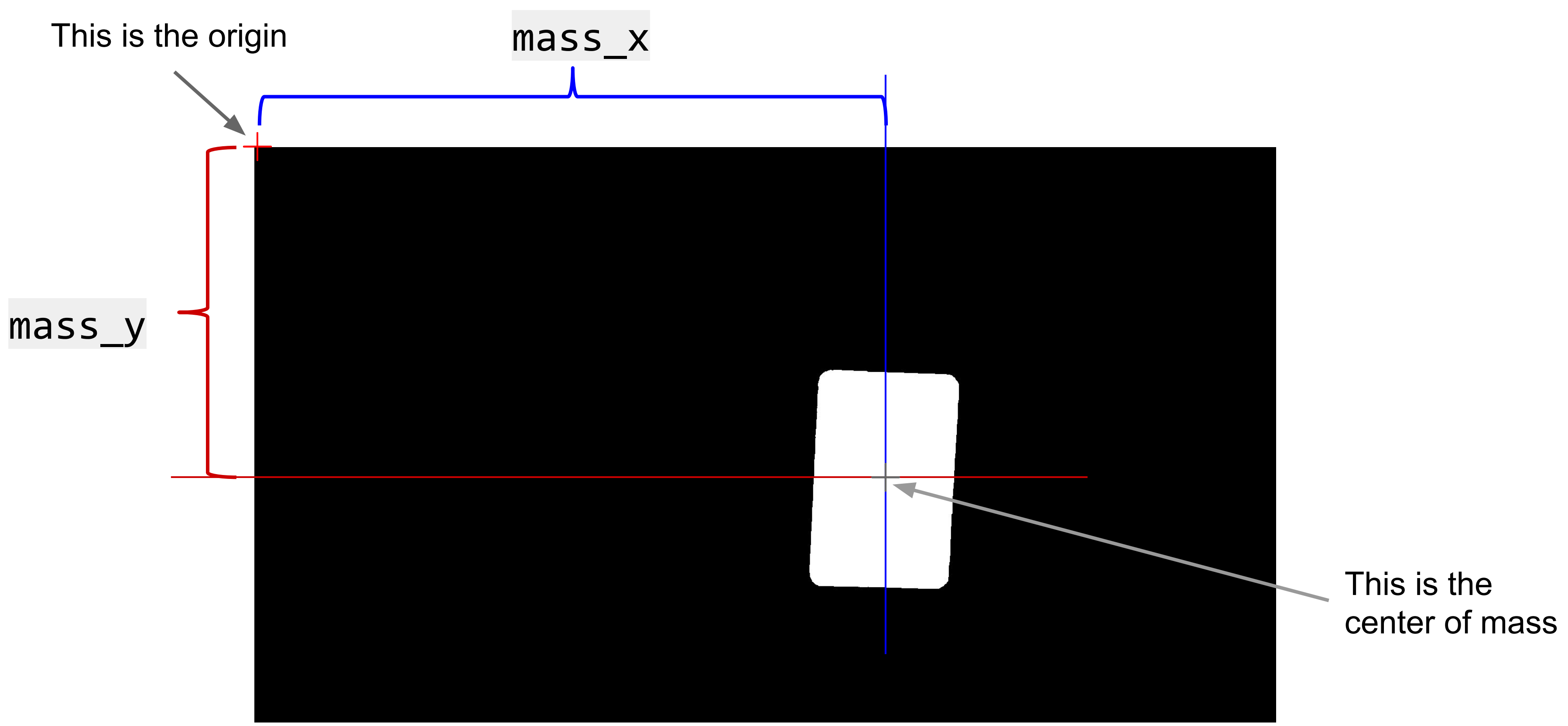
- Calculate the center of mass of the white area.

You can assume that the input of the function is a logical type mask with 0 and 1 values in it.

You should return two integers, `mass_x` is the horizontal, `mass_y` is the vertical coordinate of the center of mass.

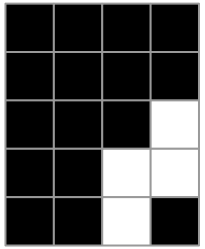
You can test your function by running `test2.m`.

See the upcoming slides for tips & tricks.


This is the origin
mass_x
mass_y
This is the center of mass

You can easily calculate the center of mass by masking and averaging the indices of the pixels.

How does it work?
Create two matrices containing the row and column indices. These matrices should have the same size as your original image.

$$I = $$  $$= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 \end{bmatrix}$$

You can create such matrices using MATLAB's built in `meshgrid()` function.

Once you have these masked matrices, please mask `X` and `Y` with the `I`.

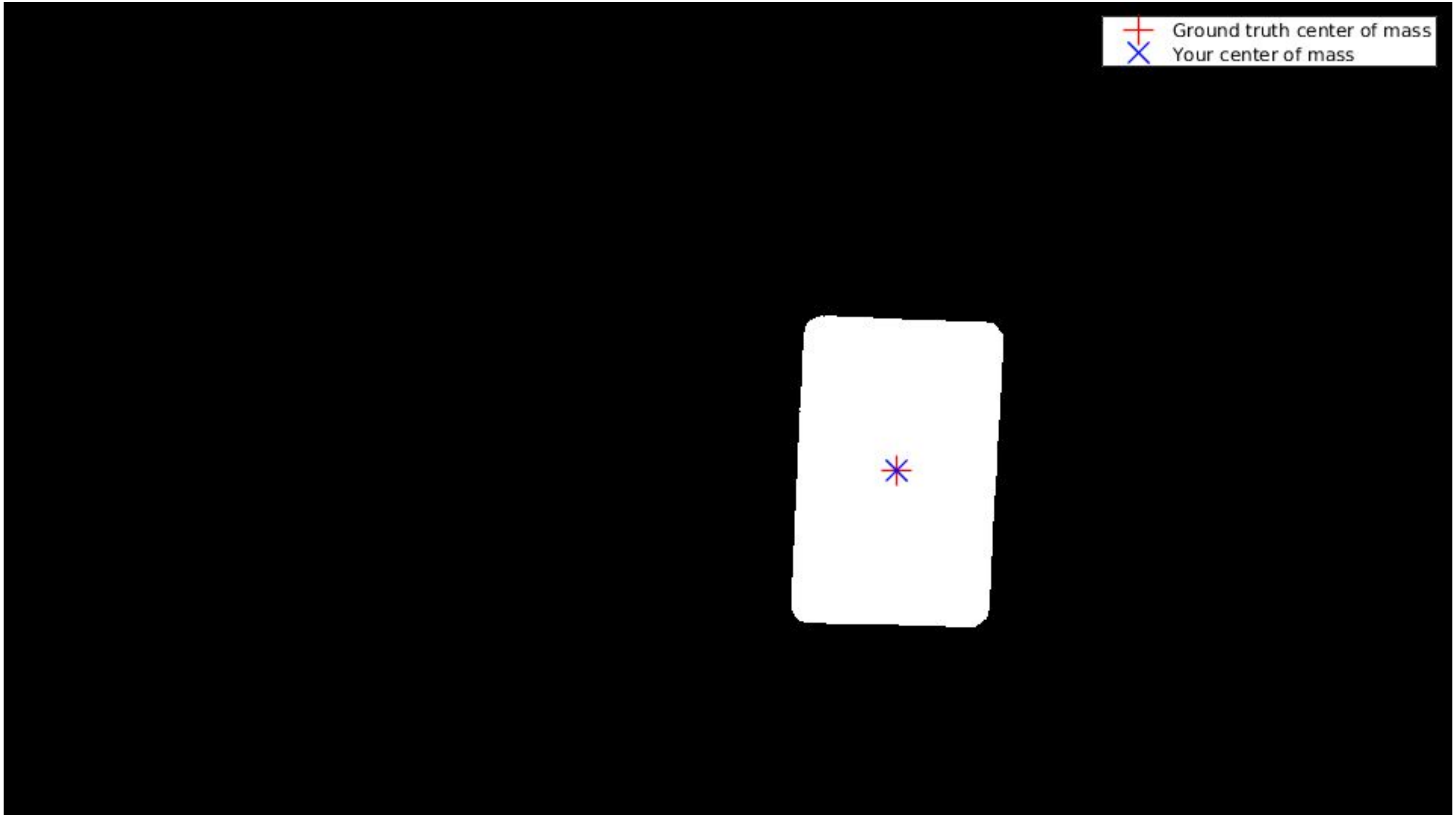
$$\texttt{X\_masked} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 3 & 0 \end{bmatrix} \qquad \texttt{Y\_masked} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 4 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

Then the center of mass can be calculated as the mean of the non-zero elements:

```
X_center = sum(X_masked(:)) / sum(I(:)) = (3+3+4+4) / 4 = 3.5
Y_center = sum(Y_masked(:)) / sum(I(:)) = (4+5+3+4) / 4 = 4
```

Don't forget to `round()` these values!

(The function should return two integers.)



Ground truth center of mass
Your center of mass

Exercise 3

Implement the function `apply_mask_and_crop` in which

- Using the known card size (420px × 280px) and the center of mass define a crop window.
- Mask the RGB image: put the card on a white background!
- Crop the masked card image using the crop window.

The function should return an RGB image in uint8 format where the values are in the [0, 255] range.

You can test your function by running `test3.m`.

See the upcoming slides for tips & tricks.

You can mask the image and place it onto a white background by doing this:

Create a white background that has the same size as the input image:

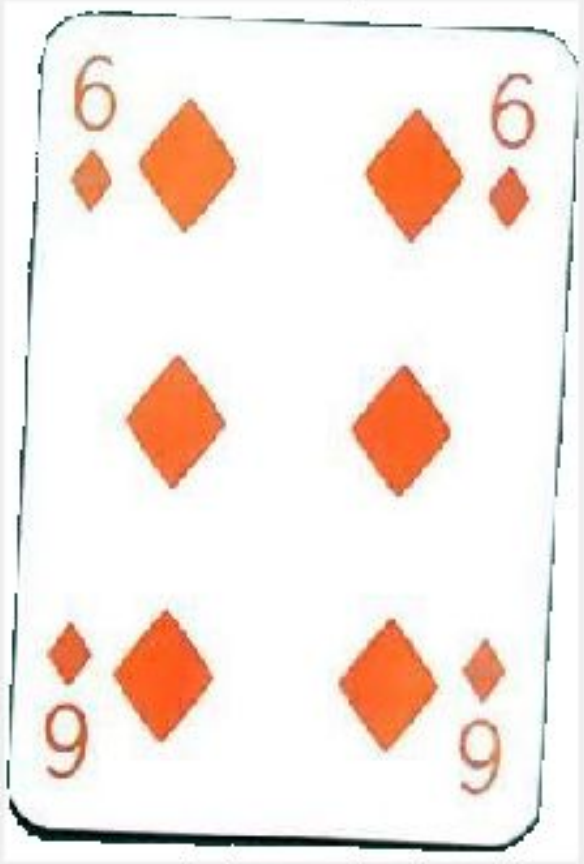
```
    WHT = uint8(ones(size(IMG)) * 255);
```

Create a new matrix for the white background masked image. Mask the original image with the mask, and mask the white background with the inverted mask. Copy both into the newly created matrix:

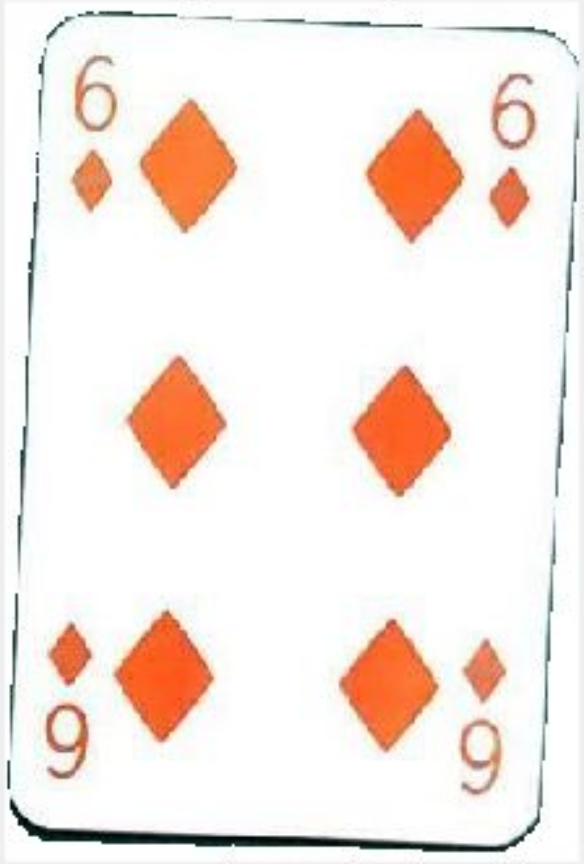
```
    WHT_masked = WHT .* uint8(~MASK);
    IMG_masked = IMG .* uint8(MASK);

    Masked = WHT_masked + IMG_masked;
```

Your output



Required output



Exercise 4

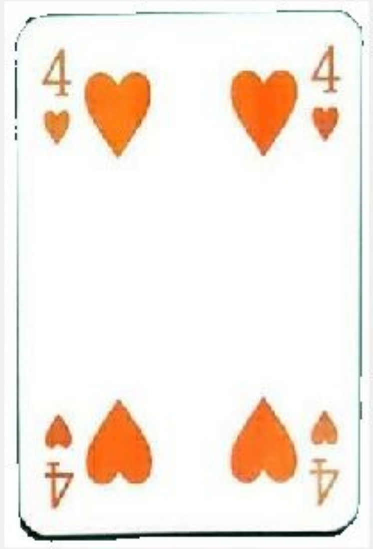
Implement the function `detect_color` in which:

- The only input is a uint8 RGB image.
- The function should return ‘R’ or ‘B’ depending on the color of the symbols.
  - ‘R’ for red
  - ‘B’ for black
- Based on an arbitrarily chosen threshold (it is your task to find a suitable threshold value) detect whether the card is red or black.
  Suggestion: use the difference of the green and red layers.

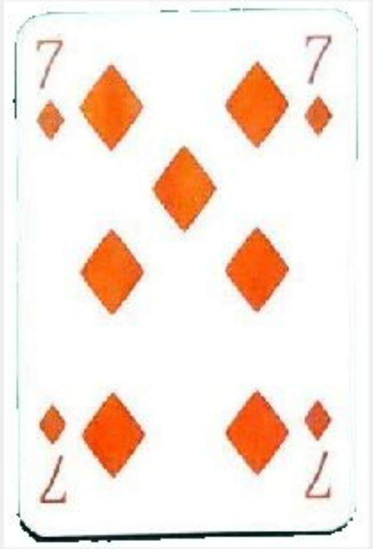
You can assume that the input of the function is a uint8 RGB image, [0, 255] range.

You can test your function by running `test4.m`.

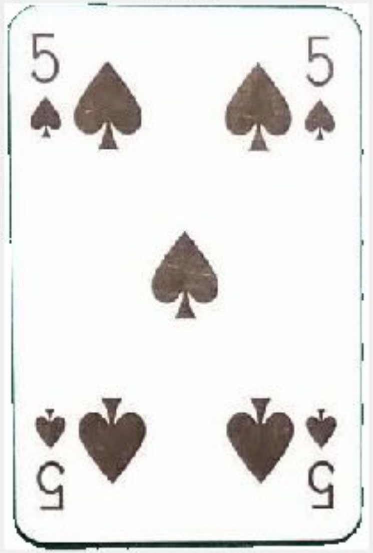
red



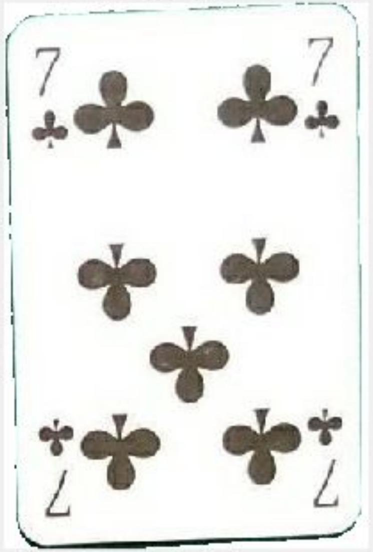
red



black



black



Exercise 5

Implement the function `detect_suit` in which:

- The only input is a uint8 RGB image.
- The function should return ‘C’, ‘D’, ‘H’ or ‘S’ depending on the suit of the card.
    - ‘C’ for clubs, ‘D’ for diamonds, ‘H’ for hearts, ‘S’ for spades
- The decision should be based on the response of the filters (you should use the 2D convolution which is `conv2()` in MATLAB).
- Additional smart decision making (e.g. using the `detect_color` function inside this function) is allowed.

You can assume that the input of the function is a uint8 RGB image, [0, 255] range.

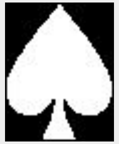
You can test your function by running `test5.m`.

See the upcoming slides for tips & tricks.

There is a MAT-file containing the normalized kernels for each suit symbol.

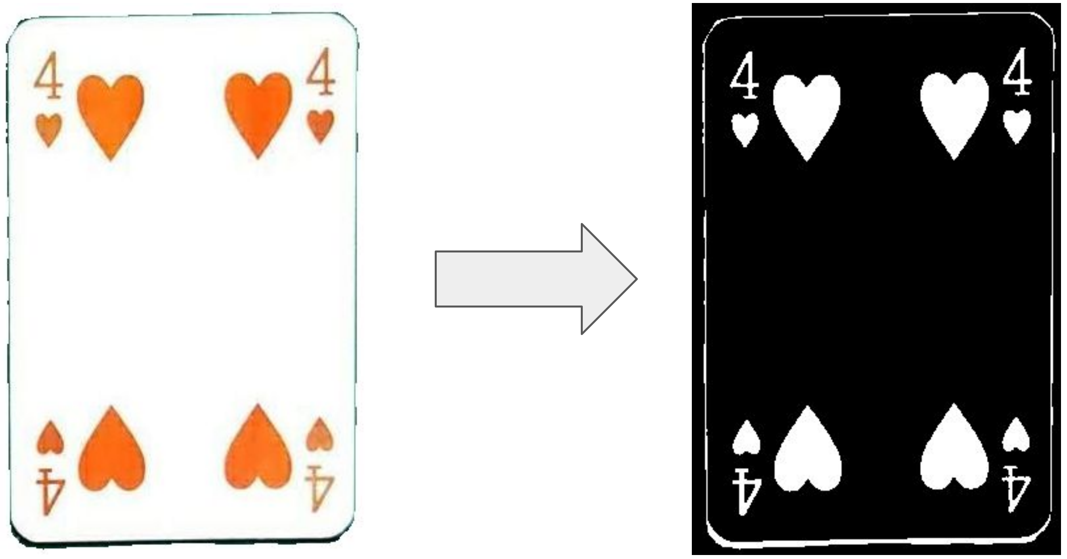
You can load this file by calling `suits = load('input/suits.mat');`

Then you can access the symbols using the following names:

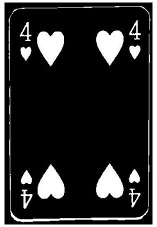
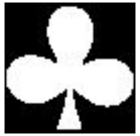
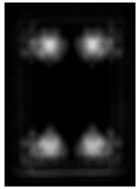
`suits.clubs`	`suits.diamonds`	`suits.hearts`	`suits.spades`
			

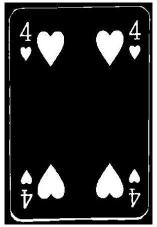
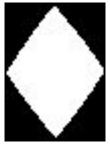
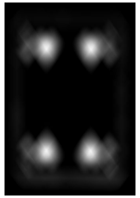
These matrices are normalized kernels meaning that you can use them in `conv2()` without any modification.

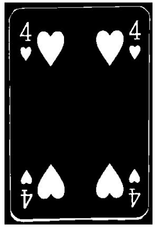
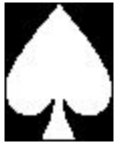
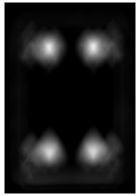
Besides the kernels you will need a thresholded, inverted image of the card. This should be a binary matrix; showing all the symbols in white, everything else in black.

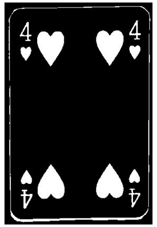
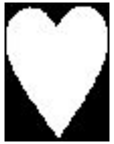
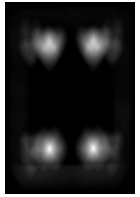


After you got this inverted thresholded card image, convolve it with the kernels and store the responses.

conv2( , ) = 

conv2( , ) = 

conv2( , ) = 

conv2( , ) = 

Please note that the card image is the same in all 4 cases, only the kernel changes.

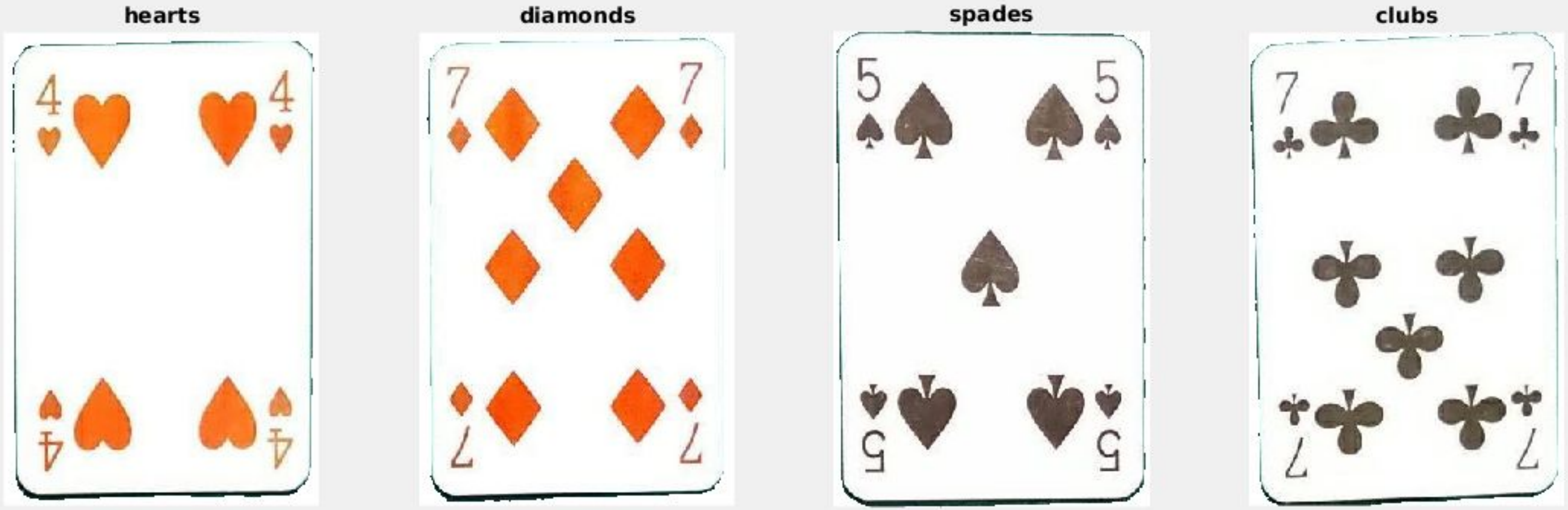
On each resulted image, search for the maximum valued pixel. You can find that using the `max()` function and the colon operator (`:`).

Compare the maxima values of the 4 images. The highest ‘wins’; in other words if the max intensities were

0.9251 for spades, 0.9563 for diamonds, 0.9991 for hearts and 0.8371 for clubs

then you should return the most likely label which is the hearts (‘H’) in this case.

If you want to improve the detection you can take into account the color of the card as well (for this, please use your color detection function).


hearts
diamonds
spades
clubs

Exercise 6

After all functions were implemented, run the script `run_for_few`.

This script calls your functions on 10 different cards and displays the accuracy of your detectors. Check whether the script runs without errors.

If everything was correct, run the script `run_for_all`.

This script processes the whole set of images (100 of them) and prints the accuracy of the color and suit detection. Aim for 100%!

These results are <u>for your eyes only</u>, don’t save them, don’t send them.

Finishing touches

Please check your code for unused or silly variables, meaningless comments, awful hacks etc. The quality of your code is important!

Please clear your workspace and try to run every test again. There should be no errors.

If you think that you are ready with all the exercises, please upload and submit the **functions only**.

If you have any question regarding the assignment, please send an email to your Practice leader (Miklós or Márton).

Do you want some more?

!! THESE ARE OPTIONAL TASKS !!

DO NOT UPLOAD THEM

You can try to find solutions for the following problems:

- Rotated cards I. (a card is rotated by exactly 90 degrees)
- Rotated cards II. (a card can be rotated by any angle from 0 to 360 degrees)
- Figure cards (try to detect the color and suit of J, Q, K, A cards)
- Joker detection (write a detector to tell if a card is a Joker or not)
- Value detection in 2-10 range (guess the value of the card by counting the symbols in the center)
- Full-range value detection (you can try the convolution-filter method with the numbers as kernels)
- Anything else that comes into your mind…

YOU CAN DOWNLOAD AN EXTENDED SET OF CARD IMAGES FROM HERE

THE END