Playing Card Scanner with Higher or Lower game

Gruppe 8

av

Lars, Giar og Oskar

i

IKT213

Machine Vision

Veiledet av Nadia Saad Noori

Fakultet for teknologi og realfag

Universitetet i Agder

Grimstad, November 2022

Contents

List of Figures

1 Project idea

When a computer are going to recognize a card it is often through long computations by AI, or other algorithmic equations. This is often confusing, takes space and requires decent amount of processing power. Therefore we wanted to try to make a more simpler version of a playing cards scanner that could recognize which card is shown. We created our product to offer the ability to scan a card, recognize which rank and suit it has and then use that information, to be used by other applications. We demonstrate this with the example game we created called higher or lower. Our product will satisfy the needs of people who want to create card based applications. We have estimated that the attributes that are critical for this products success is the camera quality, the deck of cards and the lighting. The product we have envisioned will also be able to scan the users own deck of card in to a template file, this makes it so that the users will not be dependent on a specific deck. The reason why users should use our product is because it is modifiable, cheap and low resource requirement.

2 Proposed solution

There are a number of ways in which this problem can be approached. One simple method would be to use feature matching to identify the card, however we opted out of this solution because there are many different variations in appearance of playing cards and it would be difficult to get enough good matches to get this to work reliably.

Another solution would be to use an AI model, with this we would need a large amount of data within a dataset, this is needed because there are many different variations in the appearance of a playing card deck. and another reason is that we want to use what we have learned in this course, we also want to understand a greater part of our curriculum by incorporating more image processing in our project.

Our proposed solution is to use template matching where you create templates of each rank and suit and then compare these with the card we want to identify. We isolate each card first, then isolate each rank and suit. This we can achieve by the following: Start by pre-processing the image by grayscaling and using GaussianBlur, then we threshold the image. After this, we can find the contours of this image. Then we assume the contour with the biggest area and if the contour have four edges. Then it is likely that the image contains a playing card. If we have identified a card in the image, then we can grab a corner of our card where the suit and rank of the card is located. Finally, we can compare the region of interest of our corner and region of interest in our database.

3 Why we changed project scope

3.1 Project idea of HSD

We have observed that within the surveillance aspect of video processing there is often a need for shortening the amount of time used on surveying a video for humans. That's why we want to make a system that creates videos containing only the video parts of humans being present.[4]

3.2 Proposed Solution

We want to use an AI trained image processing system that can detect humans within a video frame, and then apply a parameter around the human that is detected. Then a highlight video of the humans within the input video is created by the system. For us to do this we need to train an AI that can detect if there is a human present within the frame/image that it has been given. We also need to create a system that can draw a parameter around the human silhouette, and lastly we need to create a system that handles the overall creation of the new video.

3.3 Why we changed project

We decided to change the project three weeks after project start because of the simplicity of the project. We had managed to use the YOLOv7 AI model to detect humans within the first days of the project with little work. The project only covered a small portion of the curriculum and we were not satisfied with the project.

We changed to a card recognition project to further cover the curriculum and use what we have learned in class and the labs to create the card recognition project.

4 Theoretical background

4.1 Gaussian Blur

Gaussian Blur is a type of image filter that can be used to reduce noise in an image. The filter works by averaging the neighboring pixels, which gives the effect of blurring the image. The amount of blur can be controlled by adjusting the size of the kernel.[2]

4.2 Thresholding

It converts a grayscale image to a binary image, according to a threshold value. Pixels with values greater than the threshold are set to 1 (white), while pixels with values less than or equal to the threshold are set to 0 (black).[6]

4.3 Contours

In order to comprehend how contour detection works in OpenCV, let 's first check out how edges are detected using the Canny Edge Detector algorithm. This particular algorithm takes as input a grayscale Image and then you apply Gaussian blur to reduce noise. Then finds local maxima in the gradient magnitude using Sobel operatiors. These local maxima represent applicant edges that are then thresholded to create actual edges by reducing those whose pixels that fall below some user-specified threshold value T1 or above another user-specified tolerance value T2 (where T1 $<$ T2). Then any leftover edge pixels are then linked with each other into continuous figure called "contours".[7]

4.4 Normalize

Normalizing is where you alter the intensity pixels of the image to give the image more contrast, this can be done with different algorithims like $NORM\_HAMMING2, NORM\_RELATIVE, NORM\_MINMAX$ these algorithms are different ways to alter the pixels. [3]

4.5 Warping

Perspective warping in image processing is a technique used to alter and change how you perceive a image, this can be because your camera lens is distorting your image or your camera positions on an angle, this can have big effect on the data you collect. Perspective warping is a powerful tool that makes it possible to flip, rotate or correct your image.[5]

4.6 Absolute Difference

Absolute difference is used to find areas of change between two images. Absolute difference can be used to estimate change in an image or how much it corresponds with another image, however the absolute difference function only accepts same size and type or array and an scalar. Absdiff is often used to find changes over time in an image.[8]

5 System design and Setup

5.1 System design

We have designed and created our product by using Python as our programming language. Python offers many useful libraries to do computer vision tasks, such as OpenCV2. With OpenCV2 we were able to create a system where the user can scan playing cards with the use of a web camera. The system can tell what playing cards are within the frame and will save the data of the suit and rank to be used later. In our product, we have a basic card game to demonstrate the card scanner that we have created. The game is simple, the user draws a card from their own deck and uses our program to scan their card and the computer will draw a card from their deck. The one with the highest rank will win the round and the winner will be decided when both decks are empty.

5.2 Setup

The product requires that the user has access to a computer with python installed and it is recommended to use an external web camera for scanning the playing cards. The camera should be aligned above the card for accurate data. See figure1 for how an example setup would look like. The user should have a table with black background or access to cloth to use as a background for the scanner, such as a mousepad. The scanner will be most accurate when scanning a card on a black background. The software has a setup function that is used to store the templates of the users playing cards. If the user's cards have different appearances from the classic layout, the user has to use this function to create a new template for their own playing cards. The user needs to scan the ranks (A, 2, 3, ... -> King) and the suits (Clubs, Diamonds, Hearts, Spades) before the user can play. This is so that if the user's deck is different from the classic appearance, they will still be able to play with our card scanner.

6 Use Case

In our project there are two main use case scenarios, the first one is the drawing of the card and is the primary function when the user interacts with the GUI. This function in a story involves the user wanting to draw/scan a card. This initialize the recording function, which sends the returned frames from the camera to the card detection function. This function will either discard the card as unrecognizable or it will sent further to the function that handles the deck. If the card is unrecognisable then it begins again to find a new frame. However if the card is recognized then it will take that card out of the deck and then show it to the user in GUI. This is done to play the card that is drawn.

The second main use case scenarios is the scanning of the cards and saving them. This function in a story involves the user wanting to scan and save a template-set of the ranks and suits. This is done to make the detection of cards suitable to the users deck of cards if the appearance of the user's playing cards differ from the classical appearance. The story is as follows the user calls on the scan cards function, this function initialize a recording function. In this function the user needs to choose which rank or suit to be changed, then it can choose to send the frame, returned from the camera, to a card detection function. The user can also choose to exit the function as well. If the user choose to send the frame further then the card detection function will try to recognize which rank or suit it is. If it is unrecognizable then it will go back to the recording process. If the card is recognized it will display the image of the rank or suit to the user. the user now have either a choice to discard the image and return to the recording process, or save the image as a new rank or suit.

The Use case scenarios is shown below in the figures 22 and 23.

7 Dataset

In our project we use some template images of ranks and suits that is created with our scanning deck function. These templates contains fixed sized images that is to be compared with input images, and these images is the size of 130 px wide and 100 px high. The reason why the size was decided to have these dimensions is because it was necessary in our matching function so each imaged has the same size. The reason why the dimension is wider then taller is because it enhances the unique features of the ranks and the suits, and makes it more accurate when comparing.

Widening a template image makes it more accurate because they contain more unique pixel features in comparison to the not widened image. We can see this more clearly when we compare a widened template image of a rank eight with a rank six, than a not widened image of them both. Summarized this means that if we compare the images when overlapped then there is a higher difference of none similar pixels.

The template images is divided into two different datasets, one contains the templates of the ranks and the other contains the suits. Example in figure: 7 and 8

8 GUI

We opted for a GUI solution for our project to give the user a better experience when playing. The user interface is created with the pysimplegui library in python.

8.1 Higher or Lower Game

The main function of the GUI (figure 2) is to play a game of higher or lower. The game is very simple and consists of a player and a computer that will both draw a card from their own deck and play it. A round is played and the winner of the round will get one point towards their total score. The player will win the round if they play a higher ranked card than the opponent. The player has a virtual deck in the GUI to display the cards that are scanned with the scan card function. The cards are objects created from a class that contains the rank, suit, and an image of the card. Using the variables of the card, we can find out what card is being scanned and match it with a card in the player's virtual deck to display the correct card in the GUI.

8.2 Scan Your Card Button

The first button of the interface is the scan button. This button is going to take the card that is present within the camera view and save a frame that is going to get processed by our system. The system will pre-process (chapter 9.1) the frame and find the card within the frame. If there is a card present, information about the rank and suit of the card is sent to another function that will compare the information to a virtual deck and grab the card that matches the given information from that deck and display it in the GUI. After the card is fetched and display in the GUI, the computer will draw a random card from their virtual deck and the game starts. The system checks and finds a winner based on who has played the highest ranked card. The game ends when the user has played their whole deck.

8.3 Scan Deck Button

The scan deck button allows the user to scan their own deck of cards into the system (Figure 4). The option is available in case the deck of cards that the user has does not match the classical card layout. This makes our system modifiable and any deck of cards can be used in our system. When the button is pressed, another window in the GUI is opened and the user is presented with a camera view and options for scanning the cards. The window is to scan in the cards of the user into the system to change the data in the data set. The user needs to scan all the ranks from Ace to King and the four suits so that the data set is completely changed to the user's deck of cards. The user has the options to navigate the ranks and suits, take an image to scan the rank or suit, save the image to the data set or take a new image if the detection is not accurate. When the user takes an image to scan the card, the user will be shown the rank and suit before the user decides if they want to save the information or take a new image.

8.4 Restart Game Button

The Restart Game button will reset all the values to their default state so that the user can replay the game without having to close and open the GUI. The button is there if the user wants to start over.

9 Card detection

When thinking about the final product of our project we understood that we needed a few primary functions to make our product capable of producing the result we are looking for. The first functionality we set our mind on was to be able to scan a card and then recognize what card it is, based on predefined templates of both ranks and suit. By looking at similar concepts as our own on the Internet [1], we where able to conclude a step for step plan with functions to achieve our product goal, which was to recognize the card scanned. These functionalities are written in order below in the subsections.

9.1 Image pre-processing

The first thing we needed to do in our project is to process the images to suit our needs further in the pipeline. We did this by gray-scaling, gaussian blurring and then thresh-holding the image provided by the web-camera. We threshold the image with a binary threshold provided by the 'cv2' library. Example code in figure 5, example image in figure 6.

9.2 Contouring Image

In our project we decided to use contours as our way of identifying features/object within an image. In our code, after we processed the image, we needed to find the contours within it. We found the contours with the 'cv2' related 'findcontour' methode, this gave us the contours and its related hierarchies. As we discovered that the background of an image often gave a lot of smaller contours, we decided to sort based on the size of the contours and then filter out the ones smaller than a fixed size that was preceded. To do this we used the 'cv2''s 'contourArea' to sort the contours while we used the length of 'cv2''s 'arcLength' to filter out the smaller contours. This enabled us to deal with a smaller area of interest, that shortened our computing time and preventing possible future conflicts of interest. Example code in figure 9, example image in figure 10.

9.3 Obtaining Cards

One of the main functions needed to be performed before we are able to recognize the card, is to identify the possible cards within a image. We obtained this function in our product by filtering out the contours that does not meet the requirements of a card. These requirements includes having four corners, not being lower than another contour in the hierarchy and lastly being in between a minimum and a maximum size. We first filtered the cards that had four corners by using the 'cv2' function called 'approxPolyDP'. This function takes the desired input value of a curve and uses it to approximate the contour features whose shape is the same as the input value. This essentially means that it takes the desired sharpness of an angle and finds them within a contour. We used a low input value to find the sharp edges within an contour. We filter out the ones that did not have four sharp corners, and was of a lower plane within the hierarchy.

With the filtering applied, we have only contours that have four corners and with this, we further filtered out the contour that did not fit our minimum and maximum size, and what we were left with were the card contours and corners of these card contained within the image. Example code in figure 11

9.4 Crop and warp of the cards

Before we go any further we create a bounding rectangle around the cards with the 'cv2' function 'boundingRect' with this we get four returning values which is the x and y coordinates, as well as the width and height of the rectangle.

After the bounding box we needed to standardizing the images such that it could be relatable to the template images, despite its orientation. We choose to crop the image and warp it based on the four corner points obtain previously in the pipeline. The warping code we used was inspired by another persons code [1], and works as a warping mechanism to correct the orientation of a card with four corners. It does this by using the cards percentage dimensions, and finding the labels of the corners such as top left corner, and finding the orientation the card has. Based on the current orientation such as horizontal it will correct it to be vertical oriented. To find the top left corner and the bottom right corner, it uses the 'argmin' and 'argmax' of the sum of the corner points. This is because the top left corner will always have the smallest sum of x and y, and bottom right will be the opposite. To find the top right and the bottom left corners it uses the 'argmin' and 'argmax' of the difference between the corner points. Example code in figure 12 and figure 13, example image in figure 14

After the warping of the card we crop it accordingly as shown in figure 15:

9.5 Corner Zoom

After the warping and cropping of the cards, we needed to increase the size of the image. We choose to resize the image, and increase its size by four times. we did this with the 'cv2' 'resize' function shown in figure 16, example image in figure 17:

9.6 Finding contour of zoom

Now that the images are zoomed into the top left corner of the cards, we again wanted to find the contours within the image, but we found it difficult to contour it after resizing. And therefor we choose to process the image, with normalizing and thresholding before we contoured it. We sorted it by size and filtered the smaller ones out ones more. Example code in figure 18, example image in figure 19

9.7 Finding rank and suit contours

With the zoomed in image and its contours we could now find the rank and suit contours of each card. We did this by finding the center of the contours, and calculating its distance from the top left and bottom left corners. The one closest to the top left is always the rank and the closest to the bottom left is always the suit of the card. The reason for this comes with the cropping of the image previously described.

9.8 Cropping out rank and suit contour

We crop the rank and suit contours out from the normalized image in to individual images that can be matched with our template images. We cropped the contours with the usage of 'cv2' 'boundingbox' as shown in figure 20. An example image of the final cropped out rank or suit is shown in figure 21.

9.9 Match template to rank and suit

With the contour images of the rank and suit, we can now match them with the template images we have in our data sets. We choose a type of template matching that was inspired by a similar project as ours [1]. The principal of the template matching is to overlay the template image and the image to be matched. This overlay of images is create by the absolute difference between the template image and the matching image. The result of the absolute difference is how many pixel there are in difference between them. By doing this with all the template images for rank and suit, we decided that the card would be the one with the minimum amount of pixel difference between them. This was done separately for rank and suit, and then put together to decide the card. The amount of pixel difference was used further to decide the reliability of the card. The returned values was therefore rank, suit, rank difference and the suit difference.

Example of the template matching is shown in the figure 24

9.10 Reliable or not

The last thing that was done in the card detection function is to check the reliability of the recognized card. This was done by checking if the rank or suit had a higher difference number then a pre-determined value. If it was not then our card was reliable enough to put in to production. Example code of this is shown in figure 25

10 Challenges and Obstacles

We needed to decide what kind of recognition technique that we wanted to use for this product. We were unsure of what technique that would satisfy our requirements we set for ourselves, as we could use feature matching, template matching(1. with the whole card image as our template, 2. rank and suit as our template or rank separately and suit separately as our template), or find an AI model or train an AI model ourselves. We resolved this by trying all of these and noted down what worked best. After lots of attempts we found that if we isolate each card detected, we could use template matching in one corner of each card and then template match against each rank and suit separately, yielded the best result as further explained in section 9(Card Detection).

We needed to create a pipeline which was reliable and could accurately recognize which card it is, and efficient in the time it used to recognize the card. This was challenging because as said earlier there are many different techniques that could be used and all have various trade offs.

We also needed to find a reliable way of creating good data quality of the data sets of ranks and suits, we needed for our matching. This were solved by our card scanner, where you could scan your own cards in to create your own template images so your card detection could be more accurate.

Light and the background of our playing cards made it so the object recognition was inconsistent and unreliable at times. Like for example if your light source is directly upon your card, this can cause light glare to occur which results in the card not detected. We still have problems with this and we are unsure what to resolve this issue other than just making sure the card is not directly upon any light source. For our background issue we had some minor problems when you have an uneven surface this can interfere and corrupt the card contour. We could maybe solve this by optimizing our blur in the prepossessing step or in one of the threshold steps, what we could have done is an narrower threshold and one wider threshold and then approximate the best threshold via these two, or maybe take a sample from our background and then determine our threshold.

11 Results and Discussion

In the end, the project was successful and we managed to complete our challenges and problems that we phased in the early stages of our project. The idea of the project was to make a program that can recognize playing cards and use it in a virtual environment, such as in a game of cards. We restricted ourselves to not use any AI models such as YOLO to do the recognition as that would be too simple and not insightful to what we have learned in class. We managed to create the product we envisioned by using pre-processing section 9.1, contours and template matching to solve the problems and challenges that occurred during the project. In our pre-processing pipeline we choose to utilize thresholding because cards notoriously are white so this means we can create an accurate contrast contra using Canny edge detection to locate the card. Later we used contours to map out where the card is in the image, this way we can retrieve the bounding box of the card's location. We used this later to warp the image and then crop out an corner we implemented this functionality because we want to eliminate unnecessary computations and eliminate points of failure in our program. We acknowledge the possible way to use contour hierarchy to find rank and suit, although we are not sure if this would be more reliable than what we are currently using. We decided to use the absolute difference function from 'cv2' to find our proposed rank and suit, by doing this we reduce the computational operations opposed to template matching the whole card as it would require 52 card-templates in the dataset instead of our current 17 templates that only contains the ranks and suits. We fulfilled the non-functional requirements of making the system accurate, reliable and fast when recognizing the playing cards. The main purpose of the project was not the game itself but the card recognition feature that can be used in other applications and games such as Poker or Blackjack. This solution is a cheap card scanner that can be used like casinos have their own scanners.

Even though we managed to complete the challenges, there are many improvements that can be made. The GUI is very simple and could have been designed better. The game is restricted to be played until the computer's deck is empty and could have been a best of 10 rounds match instead for better experience. The scanner itself requires the background to be dark for the card to be accurately scanned by the program and could use some tweaking. Assuming this project is a 1.0 version, if we were to improve our project for a 2.0 release, we would spend more time optimizing the scanner and improve or even rewrite the GUI design and structure for a better user experience.

12 Figures

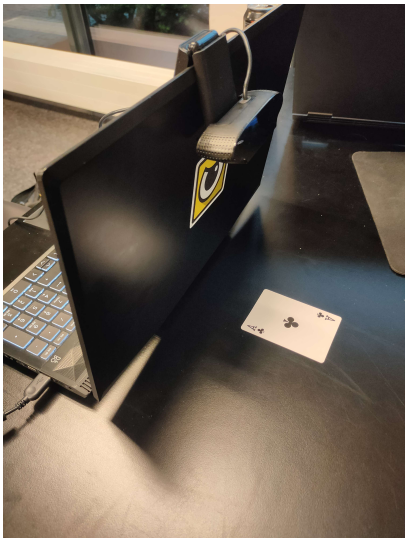

Figure 1: Example setup of our project with hardware and physical components

```python
def image_preprocess(img):
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_blur = cv2.GaussianBlur(img_gray, (5, 5), 0)
    ret, thresh = cv2.threshold(img_blur, 127, 255, 0, cv2.THRESH_BINARY)
    return thresh
```

Figure 5: Example code of the image pre-processing done

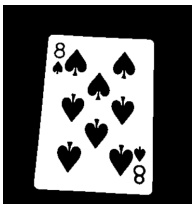

Figure 6: Example of pre-process outcome

Figure 7: Example of rank template in dataset

Figure 8: Example of suit template in dataset

```python
# Find the contour of the image
contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# If the image have no contours then return empty array
if len(contours) == 0:
    return [], []

# We Sort the contours and removes the too small contours
index_sort = sorted(range(len(contours)), key=lambda i: cv2.contourArea(contours[i]), reverse=True)
for i in index_sort:
    perimeter = cv2.arcLength(contours[i], True)
    if perimeter > NOICE_DIMENSION:
        contour_sort.append(contours[i])
        hierarchy_sort.append(hierarchy[0][i])
    else:
        continue

# We return the sorted contours and the hierarchy
return contour_sort, hierarchy_sort
```

Figure 9: Example code of the contouring done

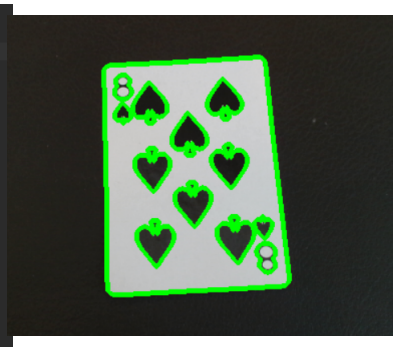

Figure 10: Example of contour outcome

```python
# Removes some small contours
perimeter = cv2.arcLength(contour[i], True)
if perimeter > NOICE_DIMENSION:
    corners = cv2.approxPolyDP(contour[i], 0.04 * perimeter, True)
else:
    continue

# checks if four corners and is within the card sizes, and saves the corner coordinates
if len(corners) == 4 and hierarchy[i][3] == -1 and \
cv2.contourArea(contour[i]) > CARD_MIN_AREA and \
cv2.contourArea(contour[i]) < CARD_MAX_AREA:
    contour_place.append(i)
    contour_outline.append(contour[i])
    for c in range(len(corners)):
        index = []
        for s in range(len(corners[c-1])):
            x = str(corners[c-1][s-1][0])
            y = str(corners[c-1][s-1][1])

        index.append(x), index.append(y)
        contour_card.append([index])
```

Figure 11: Example code finding the cards and their four corners

```python
# finds the card using corner coordinates, then creates a bounding rectangle around
# the card and then warps, and cuts it. This gives the left corner image of a card.
for i in range(len(corners_to_cards)):
    x, y, w, h = bounding_rect(contour[card_contour_place[i]])
    pts = np.float32(corners_to_cards[i])
    image_card = warp_contour(img, pts, w, h)
    images_cards.append(image_card[0:90, 0:37])
```

Figure 12: Example code of the warp pipeline

```python
def warp_contour(image, pts, w, h):
    temp_card = np.zeros((4, 2), dtype="float32")
    temp_sum = np.sum(pts, axis=2)

    top_left = pts[np.argmin(temp_sum)]
    bottom_right = pts[np.argmax(temp_sum)]

    difference = np.diff(pts, axis=-1)
    top_right = pts[np.argmin(difference)]
    bottom_left = pts[np.argmax(difference)]

    if w <= 0.8 * h:  # If vertically oriented
        temp_card[0] = top_left
        temp_card[1] = top_right
        temp_card[2] = bottom_right
        temp_card[3] = bottom_left

    if w >= 1.2 * h:  # If horizontally oriented
        temp_card[0] = bottom_left
```

Figure 13: Example of warp function

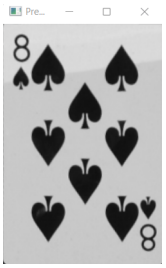

Figure 14: Example of the warp outcome

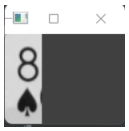

Figure 15: Example of crop outcome

```
# top most cropped out image of a card.
for r in range(len(images_cards)):
    z = cv2.resize(images_cards[r], (0, 0), fx=4, fy=4)
    images_zoom.append(z)
```

Figure 16: Example code of increasing image by four

Figure 17: Example picture of increasing image by four

```
for zoom in range(len(images_zoom)):
    normal = cv2.normalize(images_zoom[zoom], images_cards[zoom], 0, 255,cv2.NORM_MINMAX)
    ret, thresh_card = cv2.threshold(normal, 127, 255, 0, cv2.THRESH_BINARY_INV)
    final_pro.append(thresh_card)
    contour_zoom, hierarchy_zoom = cv2.findContours(thresh_card, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Figure 18: Example code of contour after normalize

Figure 19: Example picture of contour after normalize

```python
def crop_contour_rank(contour, img):
    # Crops the rank contour from the image as a new image
    x1, y1, w1, h1 = cv2.boundingRect(contour)
    rank_roi = img[y1:y1 + h1, x1:x1 + w1]
    dim = (RANK_WIDTH, RANK_HEIGHT)
    rank_sized = cv2.resize(rank_roi, dim, 0, 0)

    # Returns a cropped out image of the rank
    return rank_sized
```

Figure 20: Example code of rank contours being cropped

Figure 21: Example picture of a final cropped rank

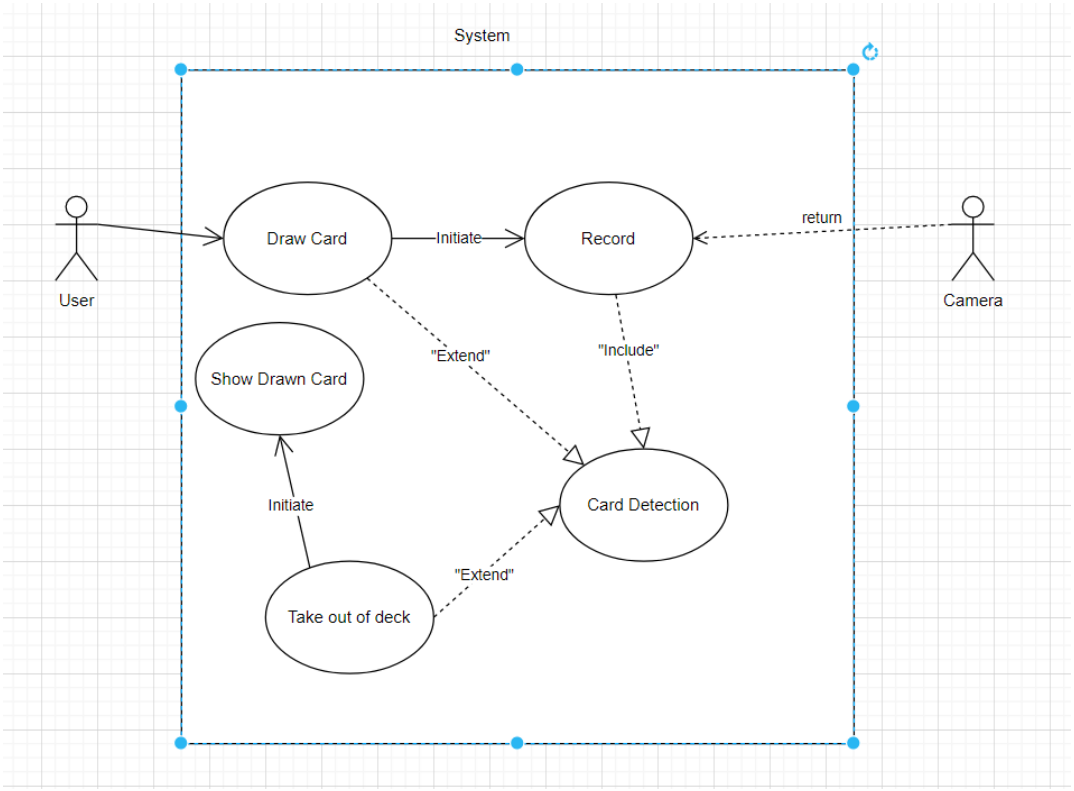

Figure 22: Use Case Diagram for drawing/scanning a card

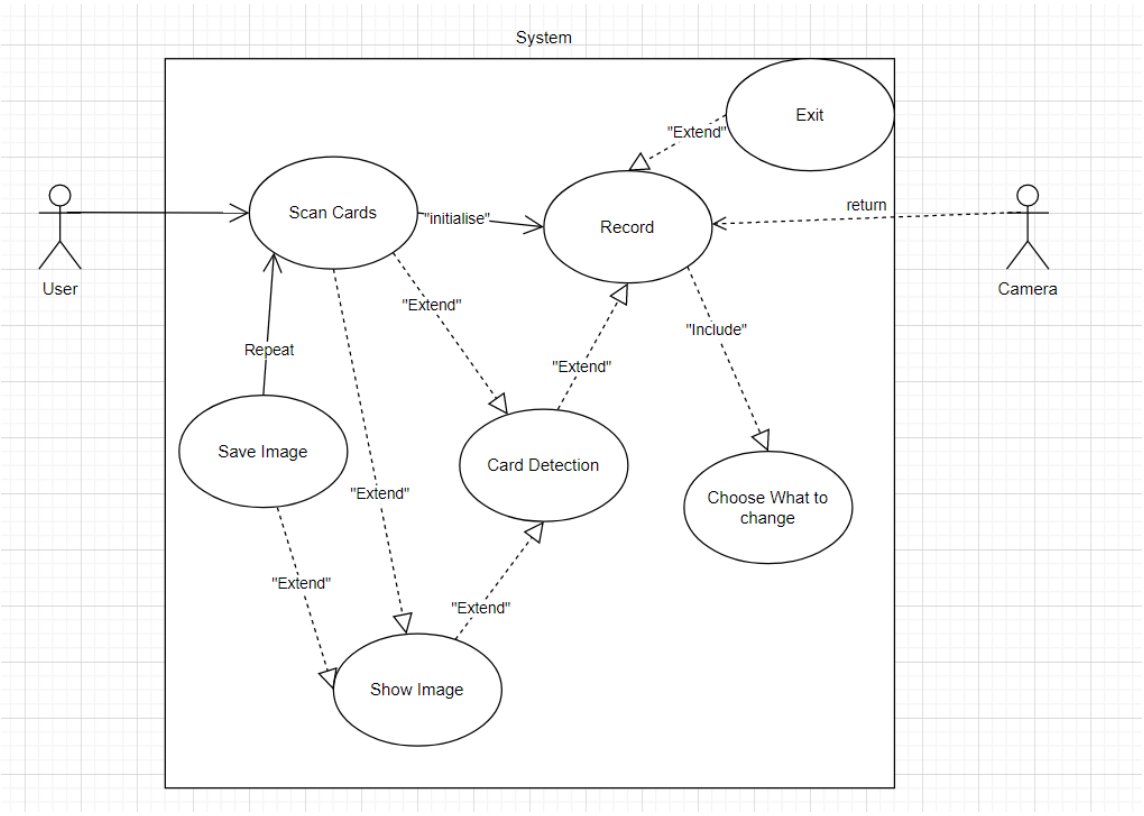

Figure 23: Use Case Diagram for scanning a rank or a suit and saving it

```
# Difference the ranks images from each of the train rank images,
# and store the result with the least difference
for Training_rank in training_ranks:
    difference_image = cv2.absdiff(rank_image, Training_rank.img)
    rank_difference = int(np.sum(difference_image) / 255)

    if rank_difference < best_rank_match_diff:
        best_rank_match_diff = rank_difference
        best_rank_name = Training_rank.name

# Difference the suits images from each of the train suit images,
# and store the result with the least difference
for Training_suit in training_suits:

    difference_image = cv2.absdiff(suit_image, Training_suit.img)
    suit_difference = int(np.sum(difference_image) / 255)

    if suit_difference < best_suit_match_diff:
        best_suit_match_diff = suit_difference
        best_suit_name = Training_suit.name
```

Figure 24: The matching of the absolute difference

```
R_img, S_img = card_finder(frame)
if R_img != -1 and S_img != -1:
    for i in range(len(R_img)):
        r, s, r_check, s_check = match_card(R_img[i], S_img[i], train_ranks, train_suits)
        if r_check < 1300 and s_check < 1300:
            if r != "Unknown" and s != "Unknown":
                print(r, s, r_check, s_check)
                return r, s
```

Figure 25: The check of the reliability of the card

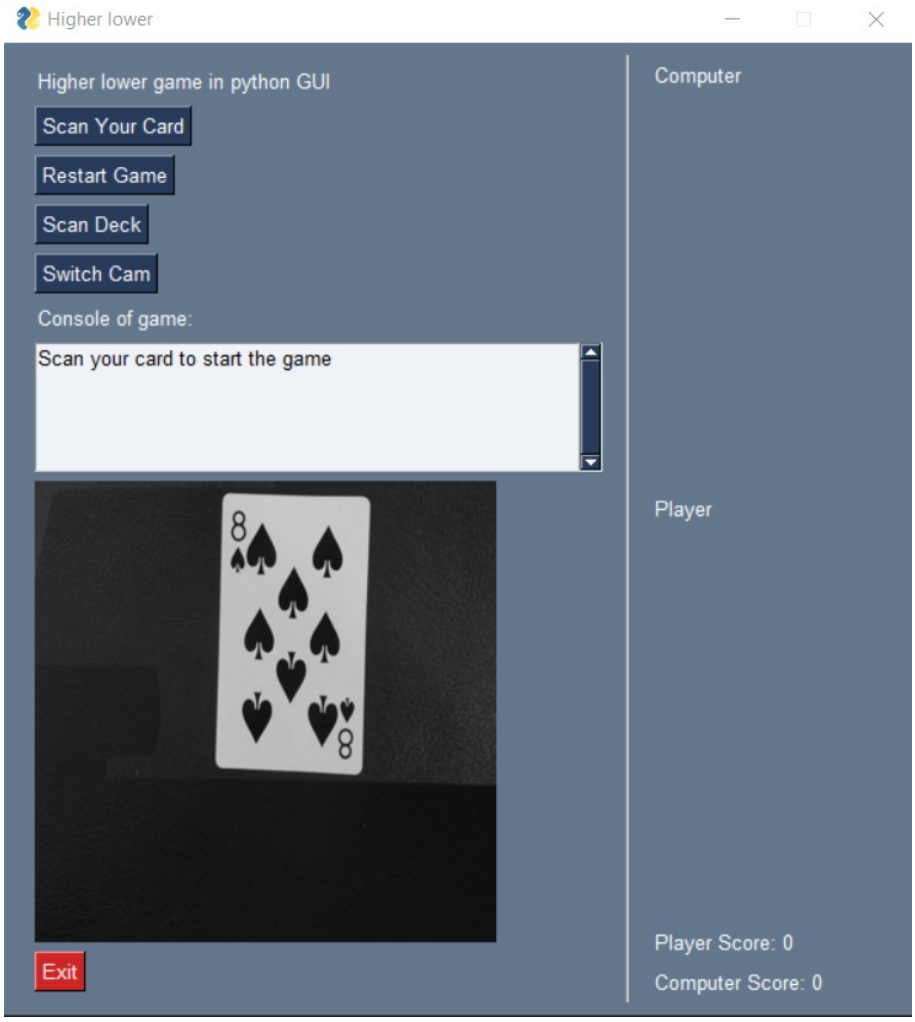

Figure 2: The GUI

Figure 3: Round played in the GUI

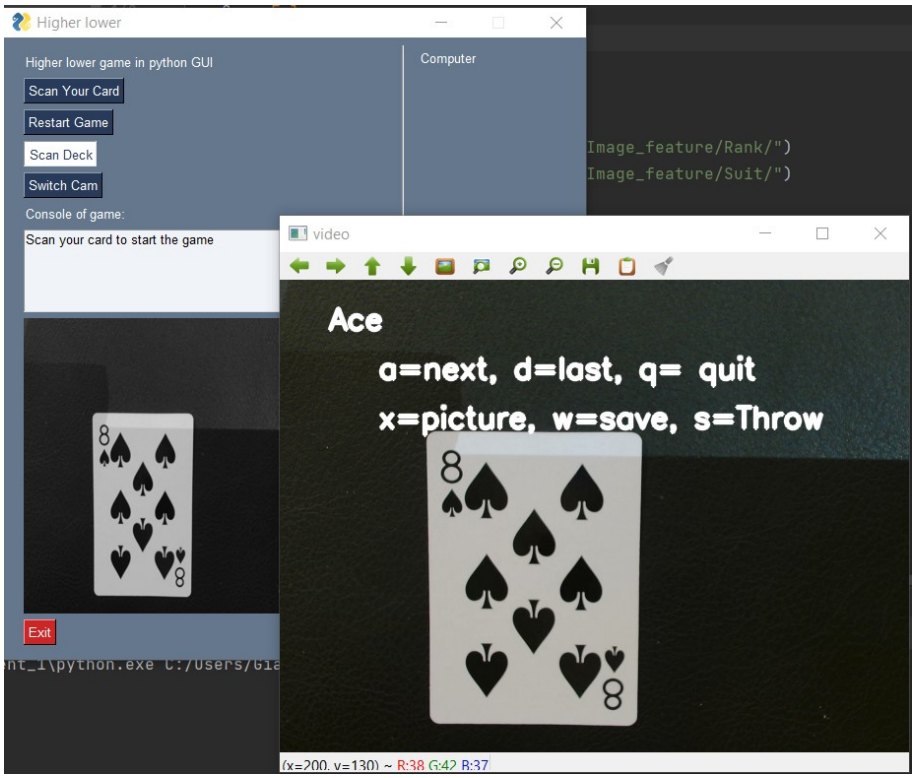

Figure 4: GUI of Scan Deck, user can iterate through ranks & suits, take a picture and save it or discard it

References

[1] EdjeElectronics. *Python program that uses opencv to detect and identify playing cards from a picamera video feed on a Raspberry Pi.* **november** 2022. URL: https://github.com/EdjeElectronics/OpenCV-Playing-Card-Detector.

[2] Krunal Krunal. *Python CV2: Filtering image using Gaussianblur() method.* **september** 2020. URL: https://appdividend.com/2020/09/19/python-cv2-filtering-image-using-gaussianblur-method/.

[3] Rohit Kumar. *CV2 normalize() in python explained with examples.* **july** 2021. URL: https://www.pythonpool.com/cv2-normalize/.

[4] Oskar Lars Giar. *Assignment Project Work: First release - report.* **november** 2022. URL: https://uia.instructure.com/courses/11713/assignments/44542?module_item_id=396837.

[5] Adrian Rosebrock. *4 point opencv warp.* **july** 2021. URL: https://pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/.

[6] Adrian Rosebrock. *OpenCV thresholding (cv2.threshold).* **may** 2021. URL: https://pyimagesearch.com/2021/04/28/opencv-thresholding-cv2-threshold/.

[7] Unknown Unknown. *Contours : Getting Started.* **november** 2022. URL: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

[8] Unknown Unknown. *Operations on arrays.* **november** 2022. URL: https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga6fef31bc8c4071cbc114a758a2b79c14.