GDAŃSK UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Telecommunications and Informatic

Student's name and surname: Denis Romasanta Rodríguez

ID: s170427
Undergraduate studies
Mode of study: full time
Field of study: Computer Engineering
Specialization/profile: Image processing

ENGINEERING DIPLOMA PROJECT

Title of project: Image processing with OpenCV - playing cards recognition.

Title of project (in Polish): Przetwarzanie obrazów za pomoca OpenCV - rozpoznawanie kart do gry.

Supervisor *signature*	Head of Department *signature*

Date of project submission to faculty office:

GDAŃSK UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Telecommunications and Informatic

Statement

First name and surname: Denis Romasanta Rodríguez
Date and place of birth: 02/10/1992 Verín/Ourense/Spain
ID: s170427
Faculty: Electronics, Telecommunications and Informatics
Field of study: Computer Engineering / Erasmus
Cycle of studies: undergraduate
Mode of studies: full-time

I, the undersigned, agree/do not agree* that my diploma thesis entitled: “Image processing with OpenCV - playing cards recognition.” may be used for scientific or didactic purposes.[11]

Gdańsk, ...
signature of the student

Aware of criminal liability for violations of the Act of 4th February 1994 on Copyright and Related Rights (Journal of Laws 2016, item 666 with later amendments) and disciplinary actions set out in the Law on Higher Education (Journal of Laws 2012, item 572 with later amendments),2 as well as civil liability, I declare that the submitted diploma thesis is my own work.

This diploma thesis has never before been the basis of an official procedure associated with the awarding of a professional title.

All the information contained in the above diploma thesis which is derived from written and electronic sources is documented in a list of relevant literature in accordance with art. 34 of the Copyright and Related Rights Act.

I confirm that this diploma thesis is identical to the attached electronic version.

Gdańsk ,..... .....

signature of the student

I authorise the Gdańsk University of Technology to include an electronic version of the above diploma thesis in the open, institutional, digital repository of the Gdańsk University of Technology and for it to be submitted to the processes of verification and protection against misappropriation of authorship.

Gdańsk,..... ……..
signature of the student

[1] [1] Decree of Rector of Gdańsk University of Technology No. 34/2009 of 9th November 2009, TUG archive instruction addendum No. 8.

[1] [2] Act of 27th July 2005, Law on Higher Education:

Art. 214, section 4. Should a student be suspected of committing an act which involves the appropriation of the authorship of a major part or other elements of another person’s work, the rector shall forthwith order an enquiry.

Art. 214, section 6. If the evidence collected during an enquiry confirms that the act referred to in section 4 has been committed, the rector shall suspend the procedure for the awarding of a professional title pending a judgement of the disciplinary committee and submit formal notice of the committed offence.

Index

1 Abstract

This document describes the project entitled Image processing with OpenCV - playing cards recognition. The aim of the project is creating the general application to detect set of player poker cards in photo using OpenCV library and Python programming platform as modern technologies and techniques as feature extractor, contours detection, K-nearest, pixel descriptor, color extraction and adaptive thresholding have been recognized and implemented.

2 Introduction

Problem of image processing is defined by using mathematical operations that transform the input that is representing by an image, a series of images, or a video, such as a photograph or video frame into the output that may be either another image or series of new images and/or a set of characteristics or parameters related to the image.

The image processing is a research discipline that has been in constant evolution in the last 20 years.

The aim of the project is given in Fig.1. We create the application to recognize or arbitrary set of cards and it.

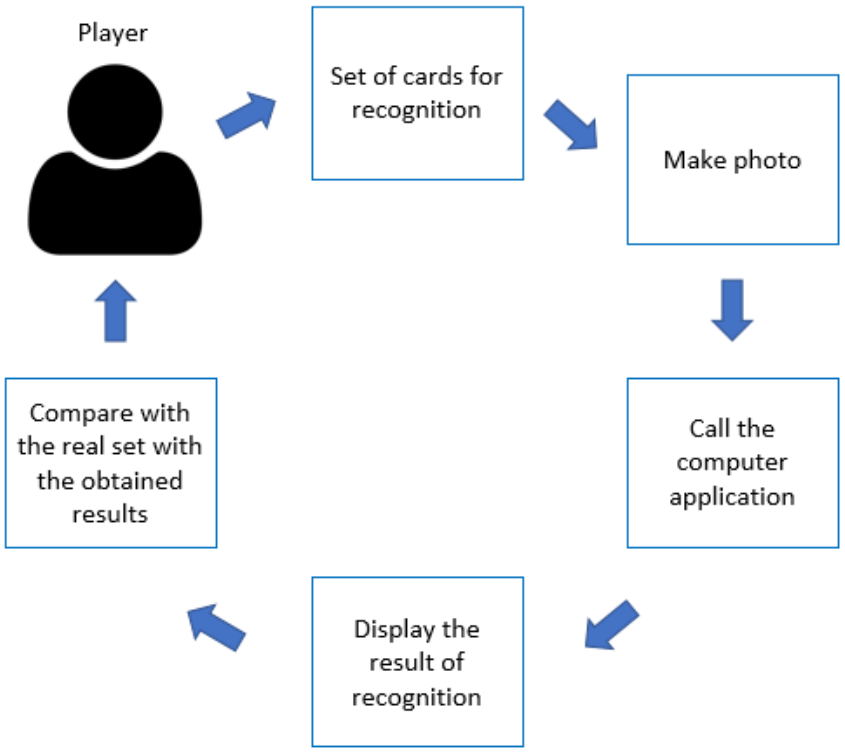


Figure 1 Rich picture of the project

At this moment, there are 3 main different ways to approach image processing, the boundaries are sometimes not clear and very often they can be used together to get better results in the sense of quality image recognition and speed of processing.

The first approach is based on fixed characteristics of the image (like feature detection), the second is based on using descriptors of the image as LBP, HOG, SURFS and a classifier as SVM or KNN with a set of samples to classify the images. The third and newest approach is based on convolutional neural networks and requires a huge amount of data to obtain good results.

To develop an algorithm to image processing there are several tools in the market, among them we can highlight the following:

- OpenCV: Opensource library for image processing. Available for several languages. [1]
- Scikit-image: Python library to image processing. [2]
- Tensorflow: Machine learning platform supported by google. [3]
- Matlab: Software with several options for machine learning and image processing. [4]

The role of the proposal approach is shown in the following Fig.2

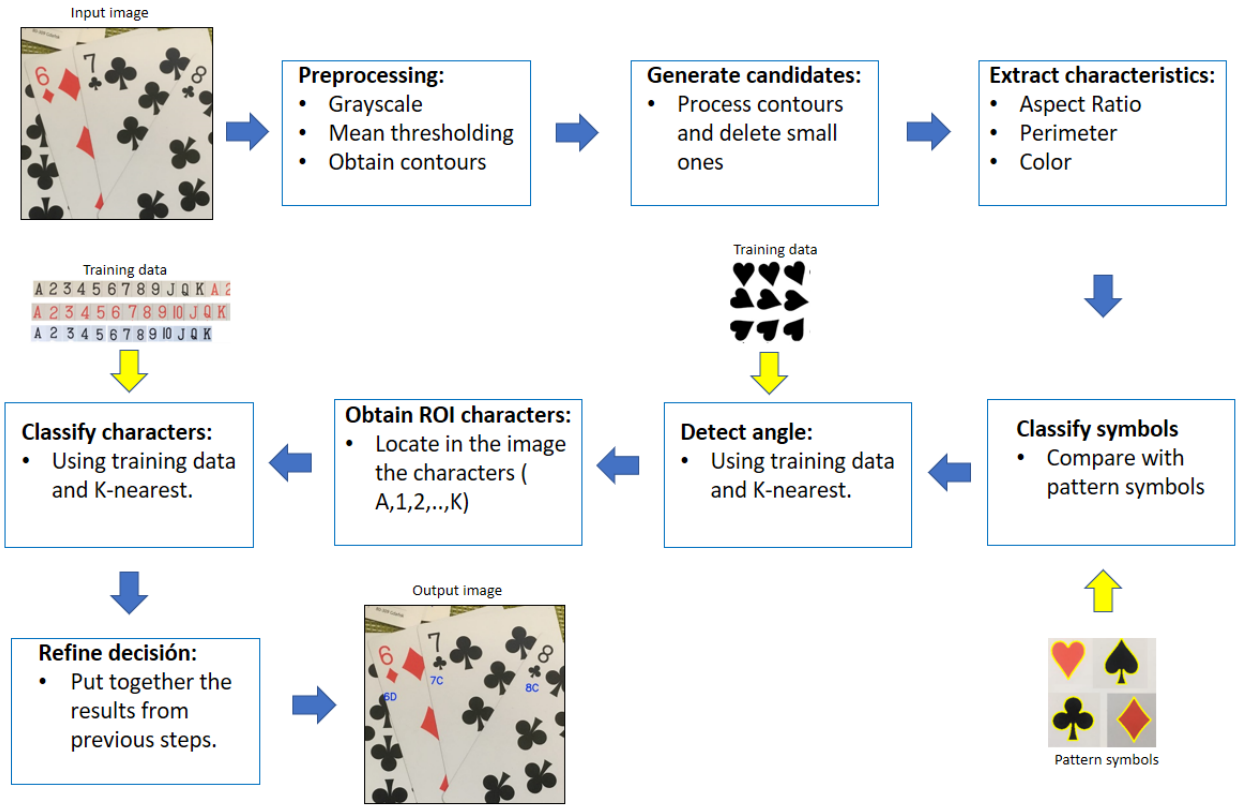


Figure 2 General approach to the card detection

3 OpenCV and Python

For the development of the application we will use OpenCV and Python.

The reasons for choice of such development platform are the following:

- Powerful and complete library with some of the most common algorithms for image processing already implemented.
- Allows to have plenty control over the algorithm that is being developed.
- Allows quick coding using Python.
- Multiplatform (Available for Windows, OS and Linux)
- It has been extended debugged and testing.
- Great internet community
- Good documentation including a huge set of examples.
- Can be integrated easily with other software.
- Not restricted for commercial applications.
- Tensorflow: It is focus in machine learning and neural networks, for use this tool is better to have a huge database of images to improve the results. In our project, we had not this situation so the tool was discarded.
- Matlab: It is a very good software with several options for traditional image processing and also for machine nevertheless it is not a open source tool and some of its modules can be really expensive.
- Scikit-Image: This tool was very considered for the development of this project and also I test some of its features however OpenCV was enough for the scope of this project.

During this project, we decide to use OpenCV and concentrate on the following algorithms:

- Feature extraction: Characteristics as position in the image, grayscale image, binary etc.
- Contour detection: Detect the contours in the image using the method provide for OpenCV.
- K-Nearest algorithm: Algorithm for classification and regression of samples.
- Pixel extraction: Obtain local information about pixels in the image.
- Color extraction: Detect the amount of each color in a part of the image.
- Image thresholding: Binary thresholding in order to perform a better contour detection.

In the following sections, we also will discuss deeply the following concepts.

4 Description of the algorithm

Our General approach according to Fig. 2 is as follows.

- Detect symbols (hearts, clubs, spades, diamonds)
- Detect angle of rotation of each symbol
- Obtain the ROI of the character (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K)
- Identify the character

The steps of general algorithm are the following:

1. Image preprocessing
2. Generate symbols candidates
3. Extract characteristics of symbols candidates
4. Classify symbols candidates
5. Angle detection
6. Obtain characters ROI
7. Process character
8. Refine decision

We will explain each one of these steps in the following sections:

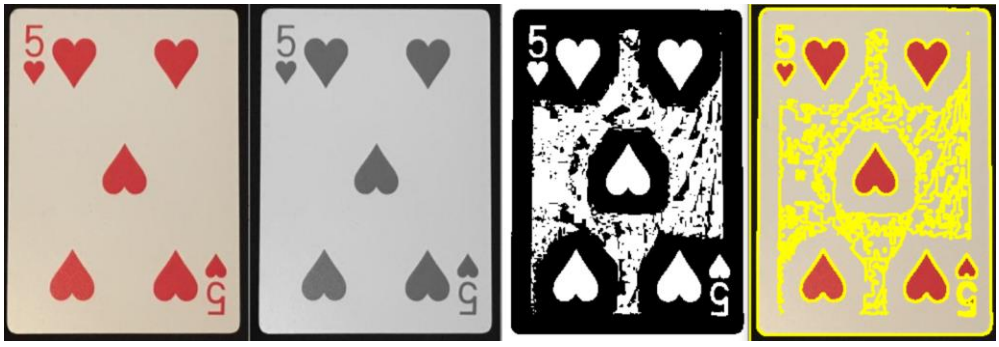
4.1 Image preprocessing

This part of the algorithm has as main goal to obtain a good thresholding of the image that allows us to detect the contours that will be used determine the location of to locate the symbols and characters in the image.

The steps are the following:

1. Convert the image to grayscale
2. Threshold with the mean value (block size of 41). All pixels with an intensity greater than the mean will be turn as white and otherwise black in the neighbourhood.
3. Obtain all contours

All steps are illustrated in Fig.3:

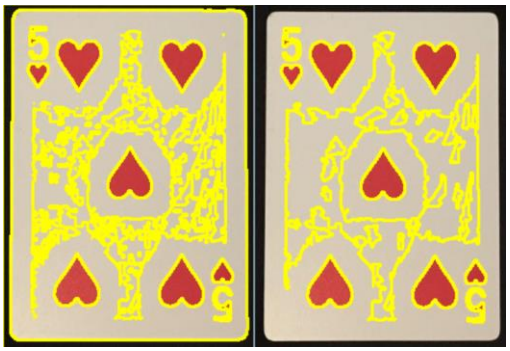


a) b) c) d)

Figure 3 Image preprocessing: a) Original b) Grayscale c) Mean Thresholding d) Contours Detection

4.2 Generate symbol candidates

The image preprocessing allows us to get a lot of contours in the picture, some of them are too small to be main symbol of playing cards so our first step will be delete those contours with less area than 50 pixels. In consequence, we erase a lot of unnecessary contours, as it is shown in Fig.4.



a) b)

Figure 4 a) Generate candidates b) For symbols of playing cards elimination of some contours.

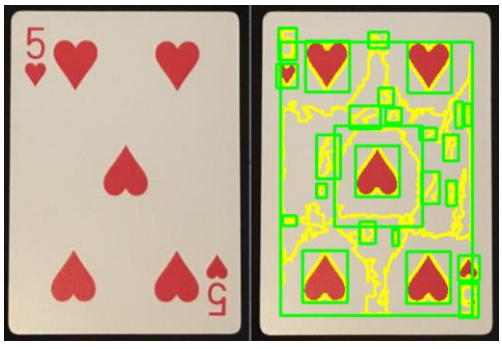
The extracted rest contours can be analyzed on the next steps of the procedure.

4.3 Extract characteristics of symbols

In this step that we have delimited our contours and regions of interest (ROIs). We take into account the data about these contours extracting their main characteristics:

1. Aspect ratio: It is the ratio between the width and the height of the bounding rectangle of the contour.
2. Extent ratio: It is the ratio between the area of the contour and the area of the bounding rectangle.
3. Solidity: It is the ratio of the contour area to the convex area.
4. Relation perimeter: It is the ration between the contour perimeter and the bounding rectangle perimeter.
5. Amount of black and red color in the bounding rectangle.

The output we obtained is shown in figure 5.



a) b)

Figure 5 Obtain characteristics of the rectangle areas. a) Base image. b) Results after extract characteristics of symbols

4.4 Classify symbols candidates

Our current candidates are the obtained results from the previous step. For each candidate, the obtained in the previous steps we calculate the properties that we mentioned before and compare it with the reference values, taking as margin a 20% of the value.

For example, the reference values for a club symbol are:

```
Amount Black = 0.34
Amount Red = 0.0
Relation area = 0.52
Relation perimeter = 0.93
Aspect Ratio = 0.75
```

If a contour has all these properties +/- 20% we will classify it as club.

With using this as classifier can give us a lot of fake positives, to avoid that we will add also another criterion, following the comparison between the detected contours and a reference contour of ours pattern symbol, as it shown in Fig.6.

Figure 6 Reference contours of ours pattern symbols

OpenCV comes with a function *cv2.matchShapes()* which enables us to compare two shapes, or two contours and returns a metric showing the similarity. The lower the result, the better match it is. It is calculated based on the hu-moment values. Using such function, we can compare every contour with a reference contour of a club, heart, spade or diamond. If the result of the function is less than the threshold value (0.15 in our case) we will classify the sample as its symbol corresponding. As pattern symbol, with the highest similarity.

In that way the picture from Fig 5-b is transform to the picture showing in Fig 7.

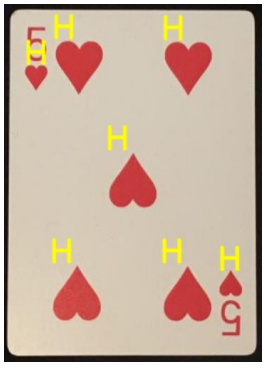


Figure 7 Result of classify candidates by contour features and match shape function

4.5 Angle detection

After identification of the symbols we should proceed to identify the angle of each one. The purpose of this step is being sure about where the characters (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K) will be in the picture.

To obtain the angle of each symbol we will use as classifier knn-nearest with k=1 and as characteristic vector the pixel values. A more detail explanation can be found in the following lines:

4.5.1 K-nearest neighbors algorithm

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In our case we will use this algorithm for the following classification purposes:

In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

4.5.2 Training samples

The main way of obtain great results with this algorithm if have a quality training samples.

For our training samples, we will use our symbols in different angles. The Figure 8 illustrate the training set of heart symbols for the heart training.

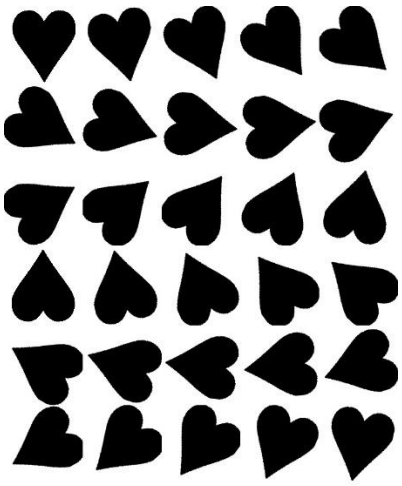


Figure 8 The example of training samples for angle detection of heart symbols. Angle step = 12º

Note: That whole algorithm, we have been use 360 samples for each symbol with angle step of 1º.

4.5.3 Obtaining training data

For each symbol that we have in our training sample we will obtain the contours and its bounding rectangle. Then we will use every bounding rectangle as our ROI (region of interest in the image).

Figure 9 Example of ROI for angle detection

For each analyzed ROI, we will resize it to a shape of 10x10, to make its characteristics invariant to size. As characteristics of every ROI we will use its pixel intensities so we will store its values. Also for the image training we know the angle of each sample so we will save it with the characteristics that we have been extracted.

The obtaining training data is described in Table 1:

Angle	**Pixel values**					
0	0	217	255	…	…	178
12	124	255	126	…	…	125
24	256	123	234	…	…	213
.	.	.	.	…	…	.
.	.	.	.	…	…	.
.	.	.	.	…	…	.
348	2	219	248	…	…	217

Table 1 Training data for K-nearest for a specific symbol. Pixel values is an array of length 100.

We will store this values in a file for use as training data for the k-nearest algorithm, and then we will use this data to train our algorithm to recognize the angle of every symbol.

This process with OpenCV consist of loading the sample data that we obtained previously and using the function to train our model:

```
cv2.KNearest.train(trainData, responses[, sampleIdx[, isRegression[, maxK[, updateBase]]]]) → retval
```

A more detailed description about k-nearest can be found at [7].

4.5.4 Applying k-nearest approach to our samples

For each symbol that we have detected in the step: **Classify symbols candidates** we have to process it with the k-nearest algorithm to obtain the angle.

For each symbol, we will use as ROI the bounding rectangle of its thresholding.

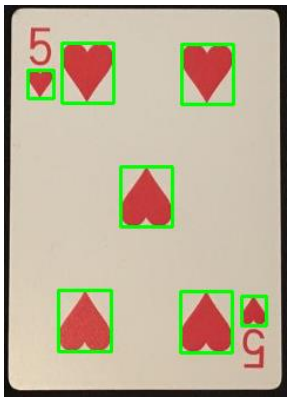


Figure 10 ROI of each detected symbol. The ROI for the k-nearest will be the thresholding of this image

To make it invariant to size we will resize this ROI to 10 x 10 (for a right behavior of the k-nearest algorithm we should process our symbols in the same way as our training data).

With this process for each symbol we will obtain data similar to a showing the Figure 10.

The k-nearest algorithm will process this data and will give as response the closer training sample and the distance between this sample and our symbol.

The distance is a metric that describes how far our sample is from the closer element from the training set, it is the distance from the input sample to the corresponding neighbor. This value has to be adjust manually and in our case, we define it with a maximum of 999.999. If the distance is greater than that value we will assume that we are not analyzing a valid symbol image, and we will discard the sample. Applying this the angle detection will work also as validation for the previous symbol detection.

The results that we obtain for the previous image is shown in Fig 11:

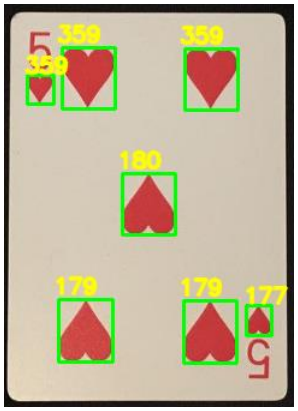


Figure 11 Angle detection for each symbol in the image

4.5.5 The diamond symbol angle

With the diamond symbol the k-nearest method with pixel values as classifier gives poor results so we have developed another algorithm to obtain the right angle.

The first step is detecting the extreme points of the diamond, this can be achieved using the contours of each symbol and taking the characteristics points that are most to the up, down, right and left: (See figure 12).

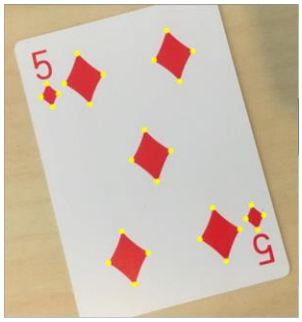


Figure 12 Detect extreme points in diamond symbols

After that we compare the distance between the upper and the bottom point and the right and the left point. The bigger of these distances will be taken as reference.

Comparing the angle between this reference line (green) and a vertical line (yellow) we can obtain the angle of the diamond. (See figure 13)

Figure 13 Lines to calculate the angle of a diamond symbol

With these 2 lines, we can obtain the angle of each symbol in the image. (See Fig 14):

Figure 14 Angle for diamond samples

4.6 Obtain ROI characters

After detection of the symbols and the angle we have to determine where are the characters in the image. We have all the information that we need, the positions, angles and sizes of the symbols that we have detected. For each sample, we will rotate the image by the degrees that we have been detected previously using as reference point the centroid of the sample.

After that we will use as region of interest (blue in the next image), the content that is upper from the symbol, using as width 1.2 times the width of the symbol, as height 1.4 times the height of the symbol and as offset in the y axis 1.6 times the height of the symbol. (See Fig 15)

Figure 15 Original image, rotated image, ROI Character

In the case of diamonds the angle for a sample and 180 degrees more is the same, so for each diamond symbol we will check up and down of the symbol. The posterior analysis of the blue rectangles will discard that ones which not contain a character. (See Fig 16)

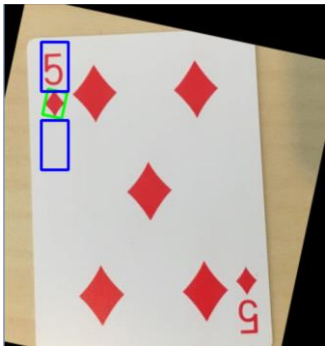


Figure 16 Possible ROIs with diamond symbols

4.7 Process ROI character

For each ROI that we have detected in the previous step we will process it to know if there is a character in that region and which one is.

For solve this problem we will use the same approach that we used before in the **Angle detection part**.

We will apply the k-nearest algorithm to our ROIs to find an answer.

The possible valid answers are the next ones: A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K

So, for our training data we will use the next picture showing in Fig 17:

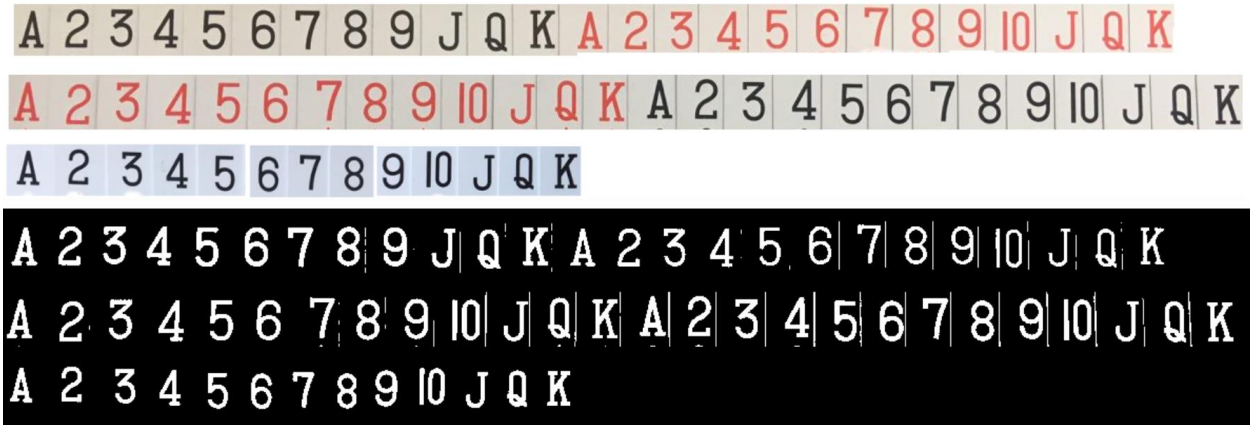


Figure 17 Training samples for character recognition. Original image and thresholding image

For each ROI character that we detected in the previous step we will apply the k-nearest algorithm to find the closer solution, this process will be similar as the one used to detect the angle in the samples.

4.8 Refine decision

For each ROI, we will obtain as result the closest character and its distance to the closer neighbor.

If this distance if greater than 1.500.000 we will discard that ROI.

If as response we obtain a single character and it is a 0 we will guess that the right answer is Q due they are really similar, and an error of this type is common.

If as response we obtain a single character and it is 1 we will discard it as no character, due to that the 1 character can be easily a straight line in the picture.

5 Evaluation of the algorithm

The main purpose of this step is to satisfy some preconditions:

- It is assumed that for this algorithm. The conditions of the picture will be without a lot of noise and that the resolution of the camera with be at least 6 megapixels.
- The brightness will be enough to distinct easily the different cards for a human.
- The algorithm can deal with some noise as shadows and different lighting as we will show in the following examples.
- The algorithm is designed for our current set of cards from sportsdirect.com (see bibliography, GitHub and following images to get examples), in order to apply this algorithm to another type of cards we should make some changes in it, what we will be discussed this in the remarks section.

5.1 Quality metrics

For the measure of the quality we will use several photos taken with an iPhone 6s in different conditions.

As main parameter to determine the quality of the analysis we will use the following metric:

$$Sensibility = \frac{Number\ of\ cards\ detected\ properly}{Total\ number\ of\ testing\ cards}$$

The following example illustrate this concept: (See Fig 18)

The character close to the number in the Figure 18 is the suit of the card: **C**lubs, **H**earts, **S**pades, **D**iamonds respectively.

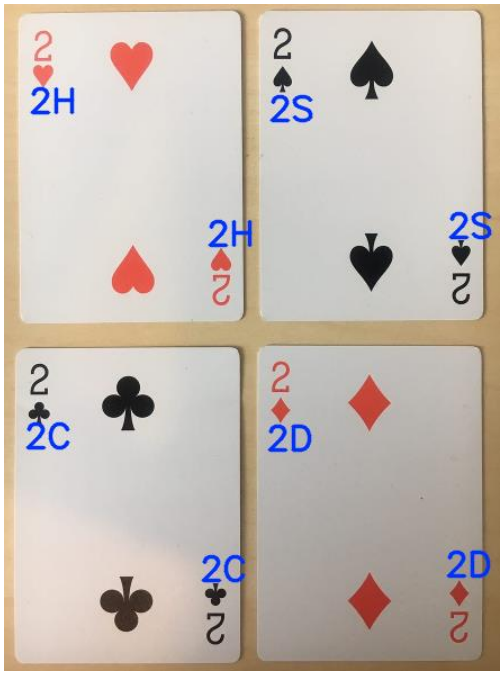


Figure 18 Example of card detection

In such case the sensibility will be 1 since because the total number of cards is 8 is the number of detected cards is also 8. We considered that if the symbol and the number of the card appears twice in the image we should process it two times, so for a complete card image we have 2 results.

We have test different situations:

- Changing Brightness
- Changing the distance of the cards
- Rotating the cards
- Overall

The whole set of images for testing is available in the GitHub of the project see bibliography.

5.2 Changing Brightness

The Fig. 18 will illustrate the results of the algorithm with different illumination:

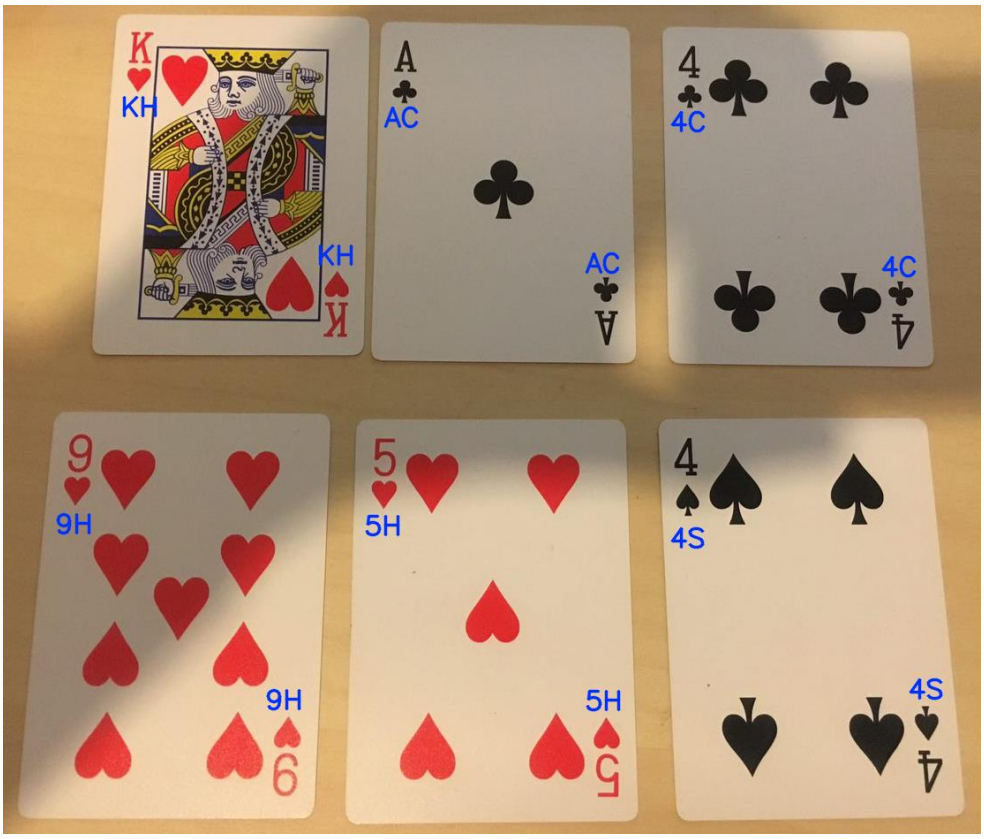


Figure 19 Picture with different lightning

Even with some shadows the algorithm is able to recognize the cards without problem. Under the folder *CardImages/BrightnessTest* in GitHub we can find the results about different test with different lights.

5.3 Changing distances

All the images used for the testing of the algorithm with the cards at different distances from the cameras. The complete test set can be found in the GitHub under the folder *CardImages/DistanceTest*.

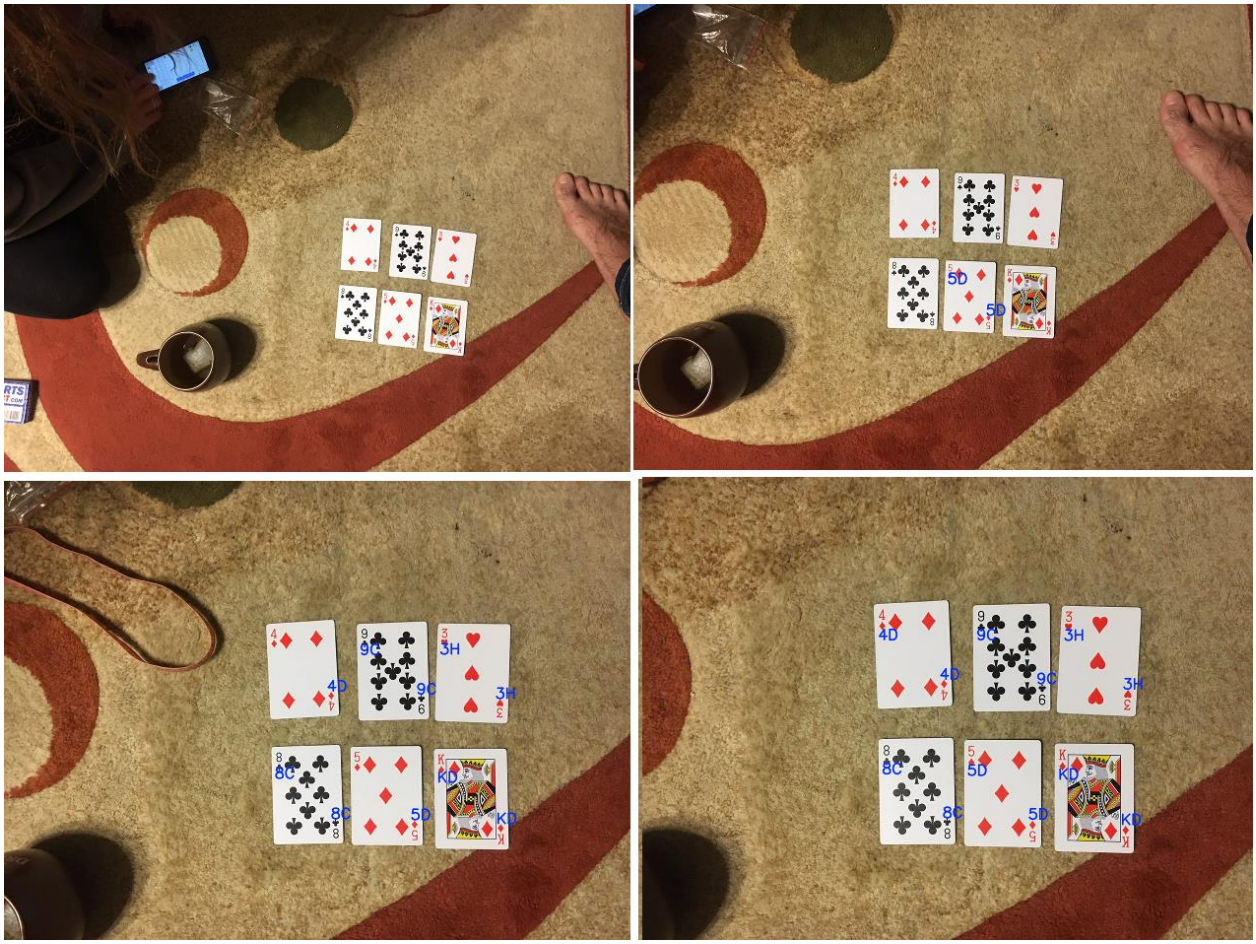


Figure 20 Distance test: 100, 70, 52,46

Number of cards		72	
Distance (cm)	**True positives**	**False negative**	**Sensibility**
100	0	12	0%
70	2	10	17%
52	10	2	83%
46	12	0	100%
39	12	0	100%
33	12	0	100%
25	12	0	100%
Overall	**60**	**72**	**86%**

Table 2 Relation between distance and card detection

The results with less than 50 cm of distance are satisfactory but the influence of the distance is really negative for the results of the image processing.

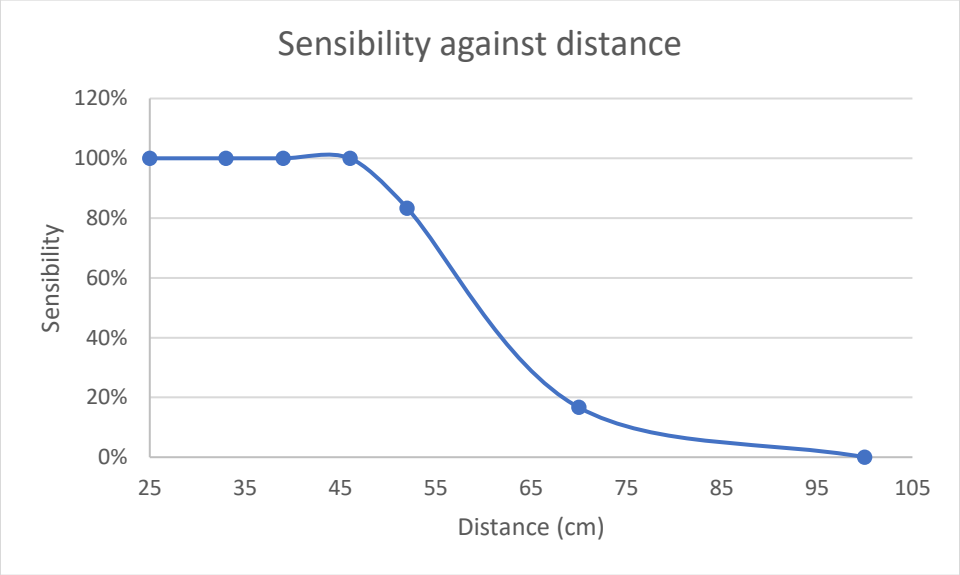


Figure 21 The relation between distance and sensibility

5.4 Rotation

The images for rotation test can be found in GitHub under the folder *CardImages/RotationTest*

The influence of the rotation in the card detection is almost irrelevant as we can see in Fig. 22:

Table 3 Influence of the rotation in the card detection

5.5 Overall results

The images and the results of the image processing can be found in the GitHub under the folder: *CardImages/OverallTest and CardImages/Results* (see Fig 23),

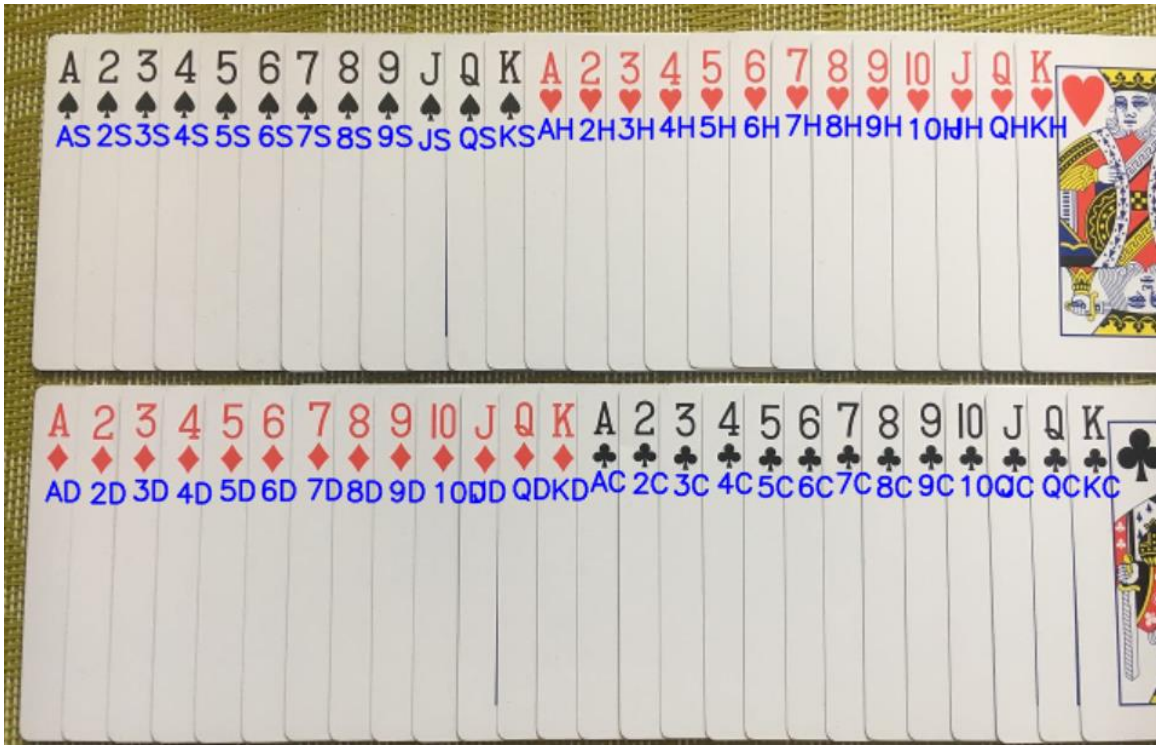


Figure 22 Set of cards

After running all the test with the images that we can found in the GitHub under the folder *CardImages/OverallTest* the results are shown in Table 4:

Number of cards	357	
True positives	**False negative**	**Sensibility**
321	26	93%

Table 4 Overall results of testing

If we do not include card images that are further away than 50 cm from the camera the results improve considerably (see Table 5):

This assumption has sense because during playing card games the cards are usually closer than 50 cm from the player eyes (the same also for camera).

Number of cards	321	
True positives	**False negative**	**Sensibility**
309	2	99%

Table 5 Overall results without images further away of 50 cm

6 Remarks

6.1 Applications of this work

The image processing application that has been descripted in this document can be easily applied to other set of cards or even to different image processing problems.

It can be really used to recognize card images during card events (for example during the famous poker competitions around the world) or more ambitious task to create a bot to play card games with humans or between them.

6.2 Adapt to different set of data

The algorithm works with a specific deck of cards, if we would like to use another we will have to change the following:

- The contour metrics for the classification of symbols (suits of the deck) should be changed.
- The training samples for the k-nearest algorithm should be changed to the new desk.
  - Change the samples for symbols in all the possible angles
  - Change the samples for the new font of characters
- The distance between the number and the detected symbol.

The code of the program available in GitHub contents two classes for the image training in the machine learning package:

- extractor.py
- knearest.py
- roi_detector.py

This classes can be used to train new data, to use the training data we should pass it as parameters to the classes angleDetector.py and processROICharacter.py, used in our project.

6.3 Other alternatives evaluated during this project

Before using this approach to solve the problem related to image processing several extra trials were done, in case of highlight the OpenCV cascade classifier.

This approach was tried using LBP and HAAR as descriptors and with Gentle AdaBoost, the results obtained with this technique were really poor in comparison with the results of the algorithm that has been explained in this document. To get more info about this topic a link to the official documentation is provided in the bibliography.

6.4 Conclusions

The results of the image processing applying to our set of cards have really good results. The use of k-nearest as classifier and the pixel and contour information as descriptor have been tested as a good approach to the problem of card images detection. An overall result of 93% (99% with playing cards closer to the camera than 50 cm) of sensibility shows that this algorithm classifies the card images with acceptable quality.

7 Bibliography

1. **Tensorflow: Consulted at 23/06/2017**
   https://www.tensorflow.org/
2. **Matlab: Consulted at 23/06/2017**
   https://www.mathworks.com/product/matlab.html
3. **Scikit-Image: Consulted at 23/06/2017**
   http://scikit-image.org/
4. **GitHub of the project: Consulted at 05/06/2017**
   https://github.com/notanumber11/Image_Processing
5. **OpenCV Contour features: Consulted at 03/21/17**
   http://docs.opencv.org/trunk/dd/d49/tutorial_py_contour_features.html
6. **OpenCV Match Shape: Consulted at 04/19/17**
   http://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#gaadc90cb16e2362c9bd6e7363e6e4c317
7. **OpenCV K-nearest: Consulted at 05/15/17**
   http://docs.opencv.org/2.4/modules/ml/doc/k_nearest_neighbors.html
8. **OpenCV Thresholding: Consulted at 03/17/17**
   http://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html
9. **K-nearest general information: Consulted at 05/17/17**
   https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
10. **K-nearest practical example: Consulted at 05/20/17**
    https://stackoverflow.com/questions/9413216/simple-digit-recognition-ocr-in-opencv-python
11. **Set of cards Consulted at 02/16/2017:**
    http://www.sportsdirect.com/sportsdirect-playing-cards-898027?colcode=89802791
12. **Cascade classifier training Consulted at 05/16/207**
    http://docs.opencv.org/2.4.13.2/doc/user_guide/ug_traincascade.html

7.1 Images used for testing.

- **Overall images: Consulted at 17/06/2017**
- https://github.com/notanumber11/Image_Processing/tree/master/CardImages/OverallTest
- **Rotated images: Consulted at 17/06/2017**
- https://github.com/notanumber11/Image_Processing/tree/master/CardImages/RotationTest
- **Different distances images: Consulted at 17/06/2017**
- https://github.com/notanumber11/Image_Processing/tree/master/CardImages/DistanceTest
- **Different brightness images: Consulted at 17/06/2017**
- https://github.com/notanumber11/Image_Processing/tree/master/CardImages/BrightnessTest
- **Results of the image processing: Consulted at 17/06/2017**

  https://github.com/notanumber11/Image_Processing/tree/master/CardImages/Results

8 Index of Figures

9 Index of tables