OpenCV Playing Cards Detection

Simon Müller, Mateusz Roganowicz

***Resumo* - Este documento apresenta o trabalho feito para criar uma detecção de cartas de jogo usando o OpenCV C++. Este projecto é o resultado de várias aulas de OpenCV no âmbito da Computação Visual da Universidade de Aveiro. No início este trabalho foca a ideia principal e OpenCV em geral, em seguida descreve a ideia principal do programa, até finalmente focar em detalhes das características mais importantes do software e sua implementação.**

***Abstract* - This paper presents the work done to create a playing cards detection using OpenCV C++. This project is a result of several classes of OpenCV as part of Visual Computing at the University of Aveiro. In the beginning that paper focuses on the main idea and OpenCV in general, then describes the main idea of the program, until finally focusing on details of the most important features of the software and their implementation.**

I. INTRODUCTION

This article presents the main ideas and techniques that were provided to us during four practical classes of OpenCV as a part of the Visual Computing class. During these classes we were digging into the topic of Image Processing. The result of that is this project, which main functionality is the detection and recognition of playing cards on a live video image. Therefore the playing cards have to be placed on a plane surface in front of a webcam connected to the computer. Then the software processes the captured images to detect the borders of cards and subsequently their content.

OpenCV ("Open Source Computer Vision Library") is an open source, cross-platform library used for real time image processing. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies [1]. While the library was natively implemented in C++, it also provides interfaces for use with other programming languages like C, Python or Java. This project was written using C++.

II. MAIN FEATURE DESCRIPTION

Having a card placed on a desk, the image is captured by the camera and the program detects and recognises every card individually.

The first part is detection, the image that the camera captures is blurred and then thresholded. From such image, the program finds contours and by that we already are able to know how many cards are placed on a desk.

We have two windows, one which presents color version of the captured image, the other one which shows the result of the edge detection process. From the source image (color version), we detect edges using the built in function Canny. Then using another built in function, findContours, from the original image we find contours and store it as a vector of points. Having the contours, the next step is to process them and check if it is a card. The program approximates the corners of the card and if there is 4 corners and the card is within given size limits it is considered as a playing card. Then each detected card is compared with the one saved on the computer to detect the exact suit. Then card information is displayed on the main window.

III. APPLIED TECHNIQUES

A. Canny Edge Detection

The Canny edge detector[7] is an edge detection operator that uses multiple steps to detect a wide range of edges in images. It was first developed by John F. Canny in 1986, and explained in his paper [2].
There are several steps to get edges:
First a Gaussian Filter is applied to smooth the image and remove noise, then using the preprocessed image, the gradients are computed using a edge detection operator like Sobel. After a non-maximum suppression, having two threshold values (minVal, maxVal) and the intensity gradients, this information is used to determine potential edges. Edges with a gradient higher than maxVal are considered as a strong edge, while edges with intensity lower than minVal are considered as non-edge. If gradient intensity of any edge is between minVal and maxVal, its membership depends if they are connected to an edge or

non-edge. As long as there is a strong edge in the pixel's surroundings, it will be preserved.
All these above steps are combined in the OpenCV function Canny(), which takes as arguments the input and output image matrix, minVal, maxVal, and aperture_size. The latter is the size of the Sobel kernel used to find image gradients. By default it is 3. The last argument is L2gradient which specifies the equation used for finding gradient magnitude.
The two threshold values play an important role for the detection algorithm. A two high threshold cand miss important information while a too low threshold can lead to false identification of irrelevant information as important. To tackle this problem, the developed program shows the user a preview window of the image after the Canny detection was applied and allows to adjust the threshold parameters via a slider. The upper threshold is fixed as twice the value of the lower threshold.

B. Finding Contours

For finding contours, OpenCV offers a function called findContours, which uses a binary image to find them. It takes several arguments, like the input image, on which should be used for example the Canny() function to create a binary image. Secondly a contour retrieval mode, in our program RETR_EXTERNAL was used to retrieve only the extreme outer contours and finally a contour approximation method, CV_CHAIN_APPROX_SIMPLE was used to compress horizontal, vertical, and diagonal segments and leave only their end points.
The output of the function is a list of detected contours, which are stored as a vector of points and optionally a hierarchy describing the image topology. [3]

C. Color Detection / HSV Color Space

At the beginning, to detect color of the card we had our frame captured and stored in variable, our goal was to detect the red color on a card. The image was converted from the RGB to the HSV color space (Hue - color information, Saturation - intensity of the color, Value - brightness of the color). In this situation the advantage of HSV over RGB was that in HSV color is represented just by one (Hue) component. After detection of red color, that information was printed on the detected card, if not, the card color was considered black. This approach was later replaced by using similarity measurements which will be explained later.

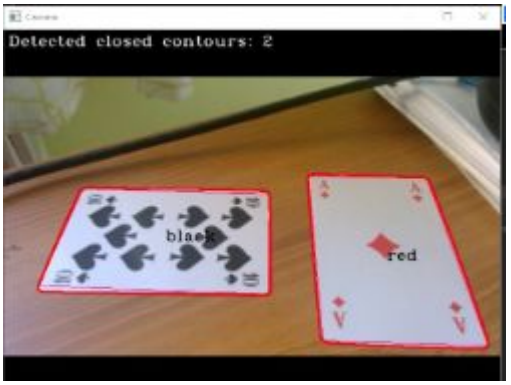

Fig. 1 Color Detection Prototype

D. Perspective Transformation

To properly detect and compare the image content, a flat, rectangular image of the detected cards was needed. Therefore on the distorted image from the camera view a perspective transformation must be applied.
OpenCV offers built in functionality for the calculation of a perspective transformation between pairs of corresponding points. The method uses Gaussian elimination to solve the equation and find the proper transformation matrix. This can be used to then warp the skewed image section into a rectangle for later usage.

E. Similarity Measurements

To measure the similarity between two images, two algorithms where used. A easier way is to calculate the "Peak signal-to-noise ratio", which is based on the calculation of the mean squared error. But while this similarity check is fast and easy, the outcome is not very consistent with what the human eye perceives.
Because of that, a structural similarity algorithm was used to calculate the "Structural Similarity Index".
Because the details of how this evaluation works and is implemented greatly exceed the scope of this project, an already finished implementation was used and there will be no further explanation of the inner workings of this algorithm in this document. For more information please refer to the sources [4] and [5].

IV. Software Implementation

Before actual implementation we did a research to get an overview on that problem. We encountered many articles about detectors, most of which were created using OpenCV-Python, for example for credit cards. The one that grabbed our attention the most was a video on the YouTube platform, where a hardware engineer created the playing cards detection using OpenCV-Python running on Raspberry Pi 3 with a PiCamera attached to it. To understand it more deeply we took a glance on that video, which presented the functionality of the program and the

python code on GitHub showing features of that.[6] Having theoretical background we moved to implementing the code.

For this application, a imperative approach was chosen over using object-oriented programming which would also be possible using C++. This decision was made because the goal of this project was to develop a simple prototype applying different image processing algorithms without any plans for future enhancements or intentions to adapt the software into a larger scale application which would benefit of the more structured object-oriented approach.
For simplicity's sake the whole program was built in one single main.cpp file.

To measure similarities and recognize playing cards, the user first has to provide those images for comparison. For this reason, the program may be executed using the *"-capture"* command line argument. Once the program detects cards, those extracted images can be saved by pressing the *Spacebar* on the keyboard. Then, all images will be saved in the */images/* subfolder, which will be created if it does not exist yet. In the later use of the program, those images will be used for comparison. The filenames are numbered and tagged using the current timestamp. For the correct display of the detected card at a later stage, the filename of the image will be used, so the captured images have to be manually renamed to the value and suite of the playing card shown in the image.

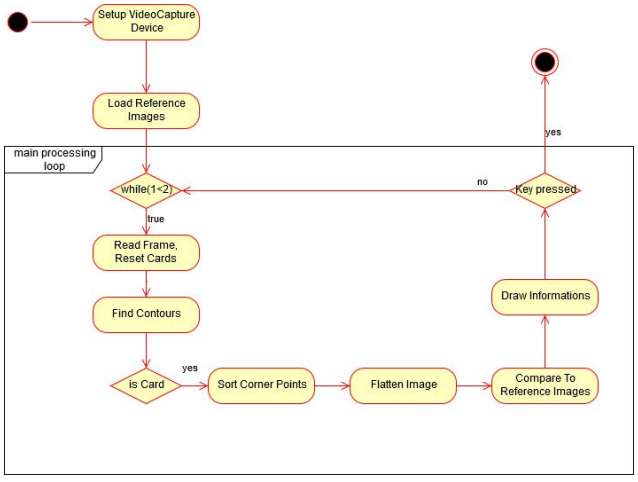

Fig. 2 Activity Diagram

Before starting the main processing endless loop, the application executes its setup.
For this a VideoCapture object is created, which represents the webcam used for capturing the image frames. To gain access to a higher resolution than the default 640x480, the camera API has to be set to use the Microsoft Media Foundation API, which allows the usage of High Definition images (1280x720).

Afterwards, images captured in the previously described process and their filenames will be loaded for later comparisons.

Once the setup has finished, the main "Grab and write" loop runs indefinitely or until the user presses a key on the keyboard to terminate it.
The list of cards gets cleaned and refreshed in each iteration of the loop. Once the Canny Edge Detection is executed, the program tries finding external contours in this binary image.

Each of the contours found will be approximated by a polygon curve, whose area is checked for minimum and maximum limits and is only considered a card if it is a rectangle with 4 corner points. By checking the lengths of the outer edges, the corner points are ordered so that the extracted image always ends up showing a upright playing card.

Once the corner points are in the correct order, this perspectively skewed contour is used to extract the camera image, which will then be transformed into a rectangular one of fixed size. This standardized image is then used for the comparison to the reference images. Using the same size and transformations for capturing the reference images and then each frame assures that the similarity check delivers stable results.
The captured frame extract is compared to all reference images found in the */images/* subfolder, and if the maximum SSI (structural similarity index) found is greater than 0.5 - the index delivers values between 0 (no similarity) and 1 (the same image) - this index and the appropriate filename is returned. The resulting name is finally drawn onto the center of the card.

Fig. 3 Card Detection Camera View Output

Finally, all detected cards are displayed in their own window and the detected contours are drawn onto the original frame.
The application also shows a second video displaying the binary image created by the Canny edge detection algorithm. A slider allows the user to adjust the threshold to better fit the environment for best detection.

V. Conclusions

The project of creating a program to detect and recognise playing cards from the very beginning sounded really interesting, but on the other hand demanding. It was a great opportunity to combine knowledge from the classes and do something that we really wanted.
In the end, the main goal to capture the video, process it and recognise playing cards was achieved.
There is still room for improvement, like further improving the recognition algorithm and implementing some software on top of the detector to use the recognized information for example for counting cards in a game of Blackjack.

References

[1] OpenCV team, "About", https://opencv.org/about/, 2019, last visited 04.01.2020

[2] J. Canny, "A Computational Approach To Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679–698, November 1986.

[3] OpenCV team, "OpenCV: Structural Analysis and Image Descriptors", https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0, 2019, last visited 04.01.2020

[4] OpenCV team, "OpenCV: Video Input with OpenCV and similarity measurements", https://docs.opencv.org/master/d5/dc4/tutorial_video_input_psnr_ssim.html, 2019, last visited 04.01.2020

[5] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Image Quality Analysis (IQA) API assessment: From error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004

[6] EdjeElectronics, "OpenCV Playing Card Detector: Python program that uses OpenCV to detect and identify playing cards from a PiCamera video feed on a Raspberry Pi", https://github.com/EdjeElectronics/OpenCV-Playing-Card-Detector, 14 January 2018, last visited 04.01.2020

[7] Alexander Mordvintsev & Abid K., "Canny Edge Detection — OpenCV-Python Tutorials 1 documentation", https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html, 2013, last visited 04.01.2020