Card Game with Augmented Reality

João Rocha
up201806261@up.pt
João Pires
up201806079@up.pt
Maria Dantas
up201709467@up.pt

Faculdade de Engenharia
Universidade do Porto
Porto, PT

Abstract

This article documents an augmented reality card game. The implementation was made using OpenCV and Python. The developed game includes the cards segmentation and recognition, the augmentation of a virtual 3D trophy and a simple interface to interact with the game.

1 Introduction

This article is divided in several sections, each one describing in detail an important part of the project.
The selected game was *Whist*, a classic English trick-taking card game which was widely played in the 18th and 19th centuries. Although the rules are simple, there is scope for strategic play, but this latter was not subject of our work.
Both OpenCV and Python represent the technologies used (library and programming language, respectively).

2 Description

Our card game starts by calibrating the camera, thus obtaining the camera matrix and the distortion coefficients. It then starts to detect the cards and to match them with the corresponding suit and rank. After all the cards, as many as the number of players, are placed, a game algorithm checks which card was the winner. Above the winning card, a 'Winner' message is then written, and the same happens for the remaining cards, but with the message 'Loser'. The user then manually informs the game about the intention to move to the next play and the process starts all over again from the card detection to obtaining the round winner. After 4 rounds, a trophy appears on top of the marker that was generated with the Aruco Marker Generator [3], displaying which player won.

3 Game Components

Our implementation is divided in several components, represented by classes and/or files in the code, which are presented next in detail.

3.1 Calibration

The Calibration component is responsible for obtaining the camera matrix and the distortion coefficients. It was implemented based a tutorial from *Geek for Geeks* [1].

Figure 1: Example of an image used to calibrate the camera.

3.2 FpsCounter

The Fps Counter component is an auxiliary class to support the monitoring of how many FPS are currently being displayed in the image shown to the final user.

3.3 Constants

The Constants include an extensive list of variables that can be changed from an environment to another in order to obtain different results and adapt to the environment characteristics.
Here a user can also change the look of the program, such as showing the FPS, drawing the detected warped cards and changing any color.

3.4 Orientation

The Orientation class is responsible for enumerating the possible positions a card may have.

3.5 Markers

The Markers class is the main class for handling the projection of the 3D trophy by the end of the game. It contains the necessary methods that identify the marker used as the placeholder to display the 3D trophy, as well as methods to draw the 3D trophy itself. A marker is used as an auxiliary to the display and part of this code was developed following the official documentation of OpenCV for Pose Estimation [4].

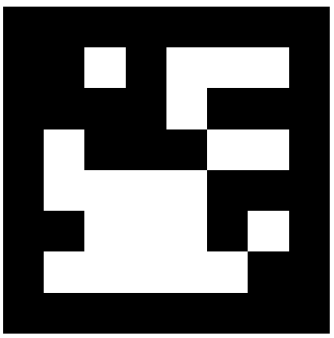

Figure 2: Marker used to display the 3D Trophy.

3.6 Deck

The Deck class contains the method that checks the winning card from a list of cards.

3.7 GameCard

The GameCard class is responsible for tracking each individual card information (suit and rank) and to compare its value with other's value.

3.8 Ui

The Ui class, as the name suggests, contains methods that are responsible for the User Interface.

3.9 Card

Unlike the GameCard, the Card class contains all the information about a card and the necessary methods to draw it.

3.10 CardCropper

The CardCropper component is not used in the game, but was used to manipulate the digitization files of the cards.

3.11 CardDetection

The CardDetection class is the one responsible for the detection of the card (detect from the obtained image where there is a card, without knowing with suit or rank it has). To achieve this we took advantage of the detect of the contours [2] in an image.

3.12 CardMatching

The CardMatching complements the CardDetection by mapping the detected card with the corresponding suit and rank. For this, a JSON file (cards.json) with the list of cards (suit, rank, value and image) is used.

3.13 Main

The Main file contains the code that represents the flow of execution of the program and that calls all the other components at the right time.

4 Solution Status

All the aims of this project were fulfilled, except the cards overlap.
Both the efficacy and performance of the developed game can be further analyzed in the next two sections.

5 Efficacy of the Solution

We want our project to be responsive and every FPS we can get will improve the users' experience when using our program.
To improve the efficacy of the card matching we are loading and saving every card we have in our database at the beginning of the execution. From what we tested, we noted that this was far better than loading the cards only when we used them because the program was doing way more heavy tasks then at the start of the execution.
We also tried to use the matchTemplate function that is in the OpenCV library but our program was getting very slow. So we ignored this function and went with another approach instead, which was comparing the warped image we got from the camera with the ones we have stored.
Overall we think our project is fairly efficient and gives the user a nice experience.

6 Performance Analysis

In one hand, the accuracy of the card detection is really high in a controlled environment, i.e. with a black background to increase the contrast between the cards and the surrounding environment, as well as the light present in the environment.
On the other hand, the card matching is not always accurate. As we can see from the bellow example, one of the suit of a card was wrongly recognized. It doesn't happen every execution, but it's influenced by the environment conditions.
In the above play, the winner card was the seven of spades since the trump was spades for this execution.
The augmented 3D Trophy is displayed correctly and without any detected problem.

7 Conclusion

Although the start of the development was a bit scary because we didn't have much experience working with OpenCV, after research we found that it was easy to work with and we got a good result.
With this project we learned more about OpenCV and applied methods and techniques we had previously learned in the classes.

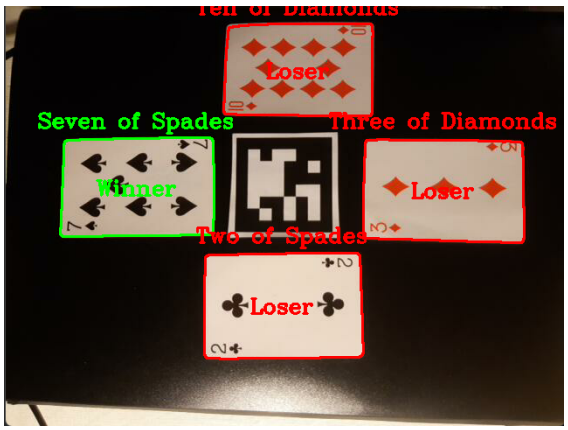

Figure 3: Example of the card detection, matching and winner/loser information.

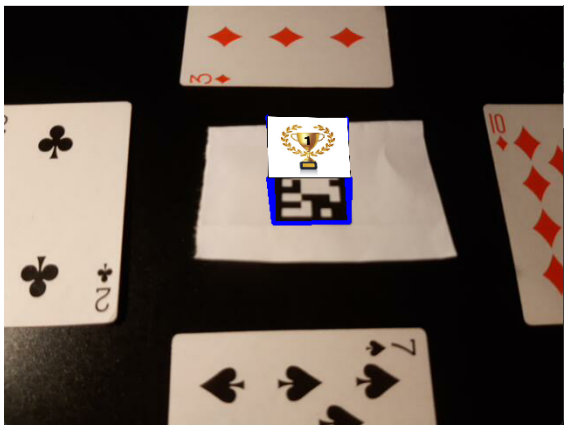

Figure 4: Example of the 3D Trophy shown by the end of the game.

References

[1] Camera calibration guide from geek for geeks. https://www.geeksforgeeks.org/camera-calibration-with-python-opencv/. Last Accessed: 2022-10-24.

[2] Opencv more about contours. https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html. Last Accessed: 2022-10-24.

[3] Marker generator. https://chev.me/arucogen/. Last Accessed: 2022-10-24.

[4] Pose estimation in opencv. https://docs.opencv.org/3.4/d7/d53/tutorial_py_pose.html. Last Accessed: 2022-10-24.

A User Manual

In order to run the game, OpenCV must be installed in the computer where the game is going to be executed.
Before running the game, the user must open the file 'Constants.py' in the root folder and change the variables according to the environment characteristics in which the game is going to be executed.
Then, inside the root folder of the project, the command 'python main.py' must be executed to run the game.
A new window will open with the visualization of the camera. After the user places the same number of cards as the number of players defined in the file 'Constants.py', each card will then have a contour and a word on top of it whether it is a winning or losing card.
The user must then hit the button 'n' to move to the next play and repeat the card placement process.
After 4 consecutive plays, the card recognition process ends and, by pointing to the marker the user can then see the 3D trophy in augmented reality, indicating which player won.
To finish the game, the user can press either 'n' or 'q'. By pressing 'q', at any stage of the execution, the game will stop and close.