OpenCV project: Chinese checkers game state detection through live video using perspective warp and colour detection

Tiago Madeira 76321 Diogo Duarte 77645

***Abstract* –This paper showcases the final product of an original project developed in the field of Visual Computation, a subject that is part of the fourth year of University of Aveiro's course Engenharia de Computadores e Telemática, as well as details of its main aspects, the main features, as well as a brief description of the work done, what was considered most important, the main goals and the problems and solutions found in order to develop this solution. The idea of creating a solution for detecting the state of a Chinese Checkers game through live video was suggested and accepted as a final assessment item for the second set of classes, which focus on openCV. Through an approach that is intended to be simple and organised, this paper starts by setting the context, scope and motivation that back this project up. Various features, ideas and problems are then presented, following roughly the real chronological order in which they came about throughout the development of the project. Finally the main conclusions are discussed, going over the acquired knowledge, the main goals and their completion and the enriching characteristics of the project, technical-wise and social-wise.**

***Resumo* –O presente artigo apresenta o produto final de um projeto original desenvolvido no âmbito da unidade curricular de Computação Visual, que faz parte do quarto ano do curso de Engenharia de Computadores e Telemática da Universidade de Aveiro, e detalhes sobre os aspetos principais do mesmo, as funcionalidades mais importantes, bem como sobre o trabalho efectuado, o que foi considerado mais relevante, os objetivos principais e os problemas e soluções encontradas, para que a solução fosse desenvolvida. A ideia de criar uma solução para detectar o estado de um jogo de Damas Chinesas através de video em tempo real foi sugerida e aceitada como elemento de avaliação final para o segundo conjunto de aulas, que se foca em openCV. Usando uma abordagem que se pretende simples e organizada, o artigo começa por estabelecer o contexto em que o projeto surgiu, o âmbito e a motivação por trás do mesmo. Em seguida são apresentadas, tirando partido da ordem cronológica real aproximada pela qual surgiram, as várias funcionalidades, ideias e problemas ao longo do desenvolvimento do projeto, colmatando com a discussão das principais conclusões tiradas do projeto, que passam pelo conhecimento adquirido, pelos principais objectivos e o seu cumprimento e pelas caracteristicas enriquecedoras do projeto, tanto do ponto de vista técnico, como do ponto de vista social.**

***Keywords* – openCV, camera calibration, perspective warp, `ArUco`, colour detection, Chinese checkers, board games**

I. Introduction

Fig. 1 - Chinese Checkers game

In the context of developing an original project in the field of Visual Computation, a subject that is part of the fourth year of University of Aveiro's course Engenharia de Computadores e Telemática, this idea of detecting the state of a Chinese Checkers game through live video was suggested and accepted as a final assessment item of the second set of classes, which teaches the usage of the Open Source Computer Vision Library or OpenCV. It is interesting to create projects that have real use across different areas of knowledge and expertise, and although the idea of the project itself seems niche, the process through which the solution was achieved can be applied to inumerous other projects and ideas. The Chinese Checkers or Chinese Chequers board is in the shape of a six pointed star. Each point of the star is a triangle consisting of ten holes (four holes to each side). The aim of the game is to be the first player to move all ten pieces across the board and into the triangle opposite. The first player to occupy all ten destination holes is the winner. Players take turns to move a single piece of their own colour. In one turn a piece may either be simply moved into an adjacent hole or it may make one or more hops over other pieces, either their own or the

opponent's. Pieces are never removed from the board. However, once a piece has reached the opposite triangle, it may not be moved out of the triangle.

II. The setup

The real world part of the project requires only a Chinese Checkers board game and a camera. However it also seemed like a good idea to have somewhere elevated to place the camera and a light source to help make the lighting more constant. Therefore, the materials used to create the real world setup were a Cayro Chinese Checkers board game, a monitor stand, an LED desk lamp and a Logitech webcam, see figure 2. A few special markers had to be printed out, glued on cardboard and attached to the board, see figure 4, these will be discussed in section IV. A special chessboard pattern also had to be printed out and glued to cardboard in order to obtain camera calibration values, as seen in figure 3, this is described in section III.

Fig. 2 - Setup

III. Calibration of the camera

Today's cheap pinhole cameras introduce a lot of distortion to images. The webcam used in this project, as well as most of the cameras used in everyday life, suffer from this distortion. With this in mind it was an important part of the project to create an easy way to calibrate the webcam to be used for the live image capture. The function *calibWebcam* takes in two arguments, *cameraMatrix* and *distCoefficients*, and populates them with values that can be used to calibrate the camera. These are the focal distances and the camera centre coordinates (the intrinsic parameters), and the distortion coefficients. The function detects the edges of a known 9 by 6 chessboard pattern (file "pattern.png"), and through the analysis of a set of images featuring the pattern, which was printed out

Fig. 3 - Chessboard used for camera calibration

Fig. 4 - Board with `ArUco markers`

and glued to cardboard to guarantee a flat plane, returns the needed values, these are printed out to a file ("CamCalibValues") in order to be available for quick loading through the *loadCamCalib* function, as long as the same camera is used. As for how the function interacts with the user, it opens up a live video and the user can then press the space key to save a snapshot as many times as they'd like, at least fifteen of these must be captured in order to, by pressing the enter key, then process the images and obtain the camera matrix and distortion coefficients' values, the recommended number of shots is ten but a higher number usually helps get a better calibration values.

IV. Perspective warp and `ArUco` Markers

Having the values needed for the calibration of the camera, either calculated or loaded from a file, it was possible to proceed to an improved call of the *aruco::detectMarkers* function, which returns the corners of each `ArUco` marker detected. In order to warp the video to the desired perspective, four points of the original video are needed, and also four corresponding

points of where they should be placed in the frames of the new video, these were chosen in a way that would make the board fit the display nicely. Initially only one marker was used, but this would mean that all the board, being much wider than a single marker, would be quite unstable even if only one of the points was poorly detected in the slightest. From this it was concluded that the best approach would be to use four markers as far apart as possible in order to achieve a more stable warping.

A. Augmented reality

Using the information gathered this far into the project it would have been easy to take the project in another direction. With the camera matrix and distortion coefficients, it was possible to use the function *aruco::estimatePoseSingleMarkers* in order to estimate the pose of each `ArUco` marker, obtaining the rotation (`Rodrigues`) and translation vectors for each one. These were used to call the *aruco::drawAxis* function, drawing the 3d axis for the markers over the original video. This is done merely to showcase that objects could be drawn in the board, taking this project in the direction of augmented reality, however that was not the goal.

V. Detection of the pieces

The end goal is to be able to detect all the pieces, their colour and their position in the game grid and keep that information in memory.

A. Memory structure

The information about the state of the game is kept in memory in a way that would allow operations on it, such as playing the game versus the computer, it would have been much easier to take the pixels and show them somewhere else, but that would mean the computer wouldn't "know" what is happening. The structure in which the information is kept in memory is an array of 121 integers, each spot in the grid is ordered by rows from top to bottom and from left to right in each row, this is also the order in which the positions are checked in the original live video and the order of drawing the circles in the final 2d representation, if required later on, the access of a certain row and column can be made easy simply by using the constant auxiliary array provided containing the number of columns of each row (`map`). The value of the integer determines the colour that is present.

B. Colour detection

In order to detect pieces of each different colour, the original image was converted to `HSV` and thresholded using the *inRange* function for different ranges of Hue, Saturation and Value. The obtained images were then processed through morphological opening, in order to remove the `salt` noise, and morphological closing, in order to remove the `pepper` noise.

C. Piece detection

After the information of different colours had been separated by the colour detection it was time to check if a piece of each colour was located in a certain place of the game board. There was no easy way to detect a grid of the shape of this particular board, so a few tests were run using the Hough Transform in the warped video, making use of the *HoughCircles* function to detect one piece in multiple places and figure out it's coordinates, radius, and the distances between each of the places. This way, a grid with all the possible places for a piece was constructed and adjusted to the warped video by experimentation, see figure 8. It was then a matter of using each of the circles of this grid as a mask for each of the images obtained with the colour detection, obtaining points of a certain colour in a certain place. The matrix containing these points was then summed and a limit number (greater than 40000 as obtained by experimentation) was used to decide whether or not a piece was in the spot in that version of the thresholded image. In addition, because the yellow and orange colours are very similar, it is also checked if the value for one is greater than the other.

VI. Main struggles

A. Distance and angles

The main challenge of this project was to be able to detect the board and be able to move it and rotate it without losing the ability to detect the pieces. This could have been easily avoided by simply pinpointing the locating at which the board should always be placed. However, the challenge and the idea of being able to create a top view of an object from it being placed freely in a single camera's field of view was quite enticing. And as such the solution with `ArUco` and perspective warping came about.

B. Lighting and colour

One of the main struggles was the change in ambient light from room to room. The source of direct light used (the desk LED light) helps stabilise things quite a bit but its effect is diminished by changes in the distance and angle at which the board is placed. Having that said, and not wanting to control all the lighting, possibly making the experience of playing the game intolerable, the need arose of being able to quickly adjust the values used to threshold each colour. So the control windows discussed in subsection VII-B were created, making it possible to change all the values for the colour detection, but organised in a way that makes it easy to change only the needed values, which are few most of the time. The most affected values are naturally the orange and yellow hues, which incidentally fall right next to each other, thus the end of one and beginning of the other tend to deviate a little. However if the change in lighting is really significant it could throw off the white and black value, since the empty spaces may become too dark or too light.

C. Putting it all together

The active struggle of putting all the different methods and information together was constant and also led to the creation of solutions for problems that didn't really exist in an attempt to solve something else. However, the documentation for `OpenCV` was quite helpful and allowed for the finding of key functions quickly, keeping the project constantly growing and on track. Piecing everything together also had some influence on the information shown to the user. For example, when one or more of the `ArUco` markers cannot be captured by the camera, the internal state of the game in memory is not altered and the window that normally shows the warped video shows instead the normal video with a warning over it, this is a custom image that is blended with the current frame, this is showcased in subsection VII-D. Blocking the markers while playing the game and projecting shadows on the board are also a reality, the discussed measures help keep a more consistent experience, even if some miscalculation may still occur.

VII. User interface

The user interface consists of a few windows with the original video, the warped video, the 2D final display of the information in memory and controls for each of the colours detected, see figure 5.

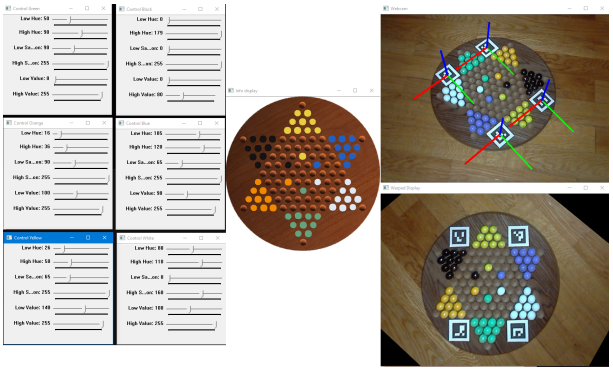

Fig. 5 - User interface

A. Game state display

The display of the game state that is kept in memory is done through a 2d representation of the board. In order to keep this representation easy to change and improve upon, as it wasn't the main focus of the project, the grid of the used image (where the markers for the pieces should show up) is detected through Hough Transform, making use of the *HoughCircles* function. A control window with sliders to control the distance, threshold and radius of the circles to be detected is provided in the code (commented). This way new images can be used easily.

B. Colour HSV controls

The user interface presents a windows of controls for each colour to be detected (green, orange, yellow, black, blue and white). The ranges of *hue*, *saturation* and *value* used to detect the multiple colours, can be changed and adjusted through sliders in these windows, making it possible to adapt to very different environments with varying ambient light and even to change the used pieces completely.

C. Original live video and axis

A window showcasing the original video being captured is shown with axis calculated for each `ArUco` marker drawn on top.

D. Warning

In case of not being able to detect the four `ArUco` markers needed to perform the warping of the video captured, the window dedicated to it shows the original video and a warning over it, see figure 6.

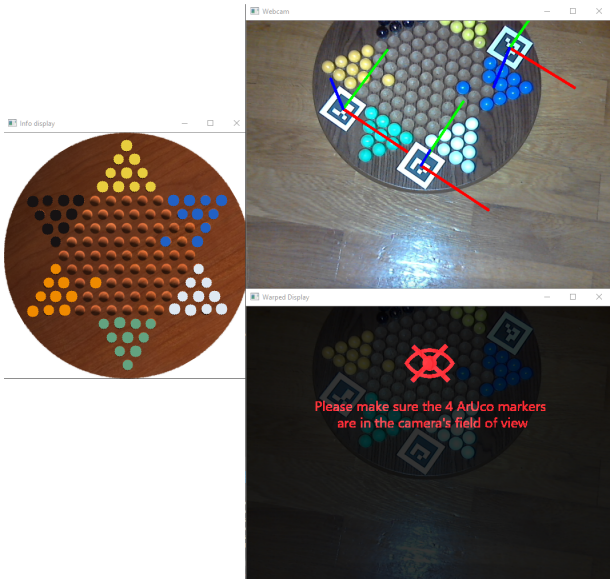

Fig. 6 - No warping warning

VIII. Technical specification

A. Modules

For this project, `OpenCV` was built and installed in windows 10 using `CMake` so that the `ArUco` module could be used, this was added as an extra from *opencv_contrib*. The version of `OpenCV` is 3.3.1

B. Debugging

A few parts of code that can be of use in the future were kept commented. This includes the control panel for the detection of circles in the image for the 2d display, the display of the colour threshold, see figure 7, the display of the complete grid over the warped video, see figure 8, and the code that was used to build the grid, which prints out the coordinates of detected circles in the warped video to a file upon pressing the space key.

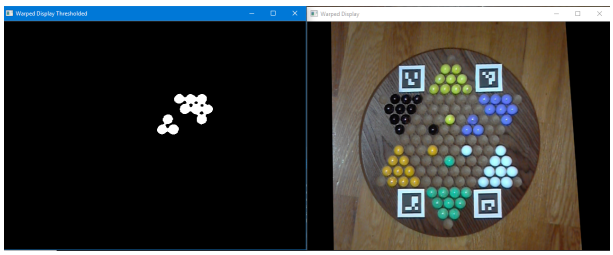

Fig. 7 - Threshold of the colour blue

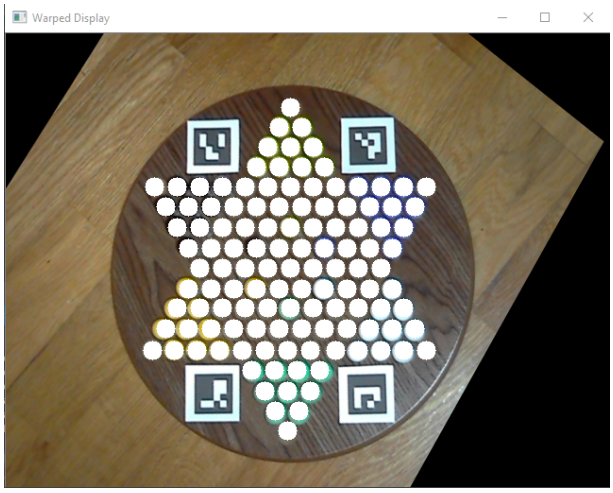

Fig. 8 - Grid used to detect pieces

IX. Conclusions

Throughout the development of this project, it was possible to learn how a multitude of techniques work and match them together in a greater scale than merely lessons and tutorials can provide, with a greater motivation backing it up. It was all planned with a very specific target goal in mind but the solutions for the different smaller problems in which the greater task was divided ended up being very interesting and there's multiple scenarios in which they could be applied. It was a very enriching project. The tools used are extremely valuable, OpenCV, C++ and Visual Studio, furthermore it was possible to experiment with a contribution library (`ArUco`), which is also new and exciting. Having that said, the goals set for the project were achieved. It is also of course important to go over the whole social aspect of working in a team, even if only a two-man one, soft-skills become imperative, cooperation and understanding is crucial and something to always be worked on granted the opportunity, which this project has.

X. Grades

Estimation of the contribution of each element of the group in percentage, considering the different nature of the tasks selected:

Diogo Duarte	Tiago Madeira
40	60

Estimation of the relative effort of each element of the group:

Diogo Duarte	Tiago Madeira
40	60

References

[1] opencv.org, "Opencv modules".
URL: `https://docs.opencv.org/master/`

[2] opencv.org, "Camera calibration and 3d reconstruction (calib3d)".
URL: `https://docs.opencv.org/master/d9/d0c/group__calib3d.html`

[3] opencv.org, "Aruco marker detection".
URL: `https://docs.opencv.org/master/d9/d6a/group__aruco.html`

[4] Victor Eruhimov, Bernát Gábor, Edgar Riba, and Vladislav Sovrasov, "Camera calibration and 3d reconstruction (calib3d module)".
URL: `https://docs.opencv.org/master/d6/d55/tutorial_table_of_content_calib3d.html`

[5] Sergio Garrido, "Aruco marker detection (aruco module)".
URL: `https://docs.opencv.org/3.1.0/d9/d6d/tutorial_table_of_content_aruco.html`

[6] Shermal Fernando, "Color detection and object tracking".
URL: `https://opencv-srf.blogspot.pt/2010/09/object-detection-using-color-seperation.html`

[7] opencv, "opencv contrib modules".
URL: `https://github.com/opencv/opencv_contrib/tree/master/modules/aruco`

[8] opencv.org, "Adding (blending) two images using opencv".
URL: `https://docs.opencv.org/2.4/doc/tutorials/core/adding_images/adding_images.html`

[9] Masters Traditional Games, "The rules of chinese checkers".
URL: `https://www.mastersofgames.com/rules/chinese-checkers-rules.htm`