3D Chessboard and Piece Recognition

Aron Katona
Technical University of
Cluj-Napoca
Email: Aron.KATONA@student.utcluj.ro

Andrada Petcu
Technical University of
Cluj-Napoca
Email: Andrada.PETCU@student.utcluj.ro

***Abstract*—The aim of this paper is to describe a method for reconstructing a 3D chessboard and its pieces, by transforming a digital image using computer vision and machine learning techniques. This topic has been thoroughly discussed over time due to the chessboard's structured geometry.**

I. Introduction

Chess is a popular competitive board game well-known for its abstraction and challenging nature. The chessboard has 64 black and white squares arranged in an 8x8 grid. Because this game is highly popular, computer chess has rapidly evolved, therefore, chess programs compete annually in the *World Computer Chess Championship*.

The 3D chessboard recognition can be used in the AI domain for games between robots and people [5], where the human player moves a physical piece on a real chessboard, and the AI detects the move then provides a response. Another applicability represents the need to digitize the game on a chess tournament. Currently, the players must write down their moves in each turn. This may impact their performance in time-critical matches, such as in speed chess.

The chessboard recognition problem has two major follow-ups, both being thoroughly discussed: piece recognition and best next move prediction. In this paper, the mainly discussed issue will be related to the first one. As object recognition has grinded the minds of many bright engineers and computer vision enthusiasts since the late 60s, there is a great selection of scientific studies and articles on which we can construct our own contribution. OpenCV [4] (Open Source Computer Vision Library) and TensorFlow [3] (ML platform) are the two main resources used in our project iterations for image manipulation and model classification.

II. Related Work

Due to the fact that our topic of choice represents a subject of interest for many researches, intensive research has been done on chess game digitization. We can classify our work into two main areas: camera calibration (*the process of estimating the parameters of a pinhole camera model* [5]), and feature extraction (getting information about a certain image region), used for identifying the tiles and the pieces of the chessboard.In [5], M. A. Czyzewski et al. propose methods that have over 95% accuracy for chessboard alignment in an image and almost 95% accuracy for chess pieces recognition. In [8] it was provided a *combined corner and edge detection* algorithm widely used in the later years. The challenge was to find the implementation which detects the same corner in multiple similar images.

The dataset is represented by a wide selection of photos of chessboards with pieces, taken from different perspectives. The images were acquired from a web source [1].

III. Method

This paper accompanies a console application which provides a menu for choosing between various operations. The end goal is to take a 3D image of the chessboard and its pieces, and output the corresponding 2D chessboard image.

In figure 1(a) is presented the overall operation pipeline. The main steps are to detect the tiles of the chessboard, detect and identify the pieces at each tile, reconstruct the original board and visualise the final result as an image 2D chessboard with pieces.

In this approach a single camera was considered. Moreover the camera does not need to be placed perpendicularly to the board. The intent was to be able to recognise an image taken by a smartphone camera in a given perspective. Camera calibration is an optional step which can be done once, and the resulting camera matrix can be reused in future calculations. There are two possibilities:

1) Do the camera calibration with a set of pictures of empty chessboards in different angles. Then this matrix can be used for correcting the image before detecting the lattice points.
2) Obtain the lattice points of the chessboard for a set of images, do the calibration, then start again the lattice point detection with the now corrected image.

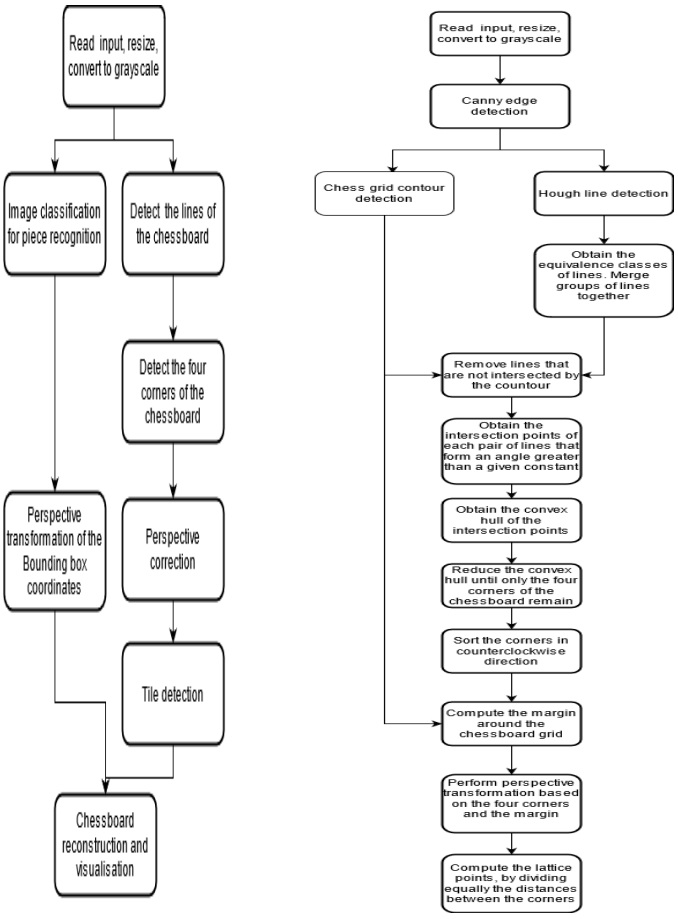
In this program both variants are implemented. In case someone takes its own photographs, pictures with empty chessboards can be taken easily and the result procedure will be simpler. However, in case of images being taken from a dataset which does not contain empty chessboards, then only the second variant can be used.

We used the pinhole camera model for representing the relationship between world coordinates and image coordinates through the perspective transformation.

A. Corner detection for empty chessboards

The first topic addressed was the corner detection problem. This represents the main approach in computer vision to draw

[1] https://public.roboflow.com/object-detection/chess-full/24



((a)) Overview of the operations which need to be performed on the input image to obtain the final result.

((b)) Detailed view of the operations which are needed for detecting the chessboard tiles and for performing perspective correction.

Fig. 1: Operation Pipeline

out features of an object. We applied this mechanism for detecting the corners of an empty chessboard for a given input image. For this step we used the *cv::findChessboardCorners* which takes as parameters the input picture, the size of the image, various operation flags and a vector of 2 dimensional coordinates structure (*Point2f*) in which the function stores the position of all detected corners. In order to have a visual representation of how our approach actually works, we called a predefined function *cv::drawChessboardCorners*. This modifies the source by having the corners colored and connected if the chessboard was successfully detected.

B. Camera calibration

The lenses of the camera distort the image, thus the last must be pre-processed. The objective of this task is to find the most accurate mapping between the camera image and the perspective camera model. There are two types of lens distortions: *radial*, denoted by *k* and *tangential*, denoted by *p* [10].

Radial distortion implies that some or all the straight lines in the chessboard image appear to be curved. On the other hand, tangential distortion occurs because the image-taking lens is not perfectly parallel aligned to the imaging plane. Therefore, some areas in the image may come across as being nearer than expected [10].

For obtaining the coefficients of the radial distortion we used the following formula:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

For the tangential factors we used the following formula:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Some other useful information is connected to the camera parameters: the focal length (f_x - units of horizontal pixels, f_y - units of vertical pixels) and the optical centers (c_x, c_y). These represent the intrinsic camera parameters, used to eliminate distortion. The rotation ($r = \begin{bmatrix} R_x & R_y & R_z \end{bmatrix}^T$) and translation ($t = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}^T$) vectors are the two extrinsic parameters used translate 3D points into the camera coordinate system [9].

$$Camera\ Matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The projection equations is:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

This implementation loads a set of images, detects their corners and matches them to the 3D chessboard corners. This way we obtain the correspondence between them, which is represented by the above mentioned parameters.

C. Line detection

If we consider images which have pieces on them, the problem of detecting lattice points becomes much harder because of shadows and hidden corners. Therefore we decided to firstly detect the lines of the chessboard, then find out the intersection points of the horizontal and vertical lines. This way the lattice points are detected even if they are hidden by a piece.

The first step is to detect the lines of chessboard. To do this, the Canny edge detection algorithm is used for obtaining the image with the edges. Before calling the algorithm, a Gaussian blur is applied on the image to smoothen it.

Then the the probabilistic Hough line transform algorithm is used for detecting a list of lines. However for a single line in real life, multiple segments could be detected by this algorithm. Therefore these lines need further processing. First the very small segments are removed. Then the remaining lines are partitioned into equivalency classes based on the relative angles. In the end a reduced list of lines are obtained which appears to be almost ideal.

D. Line cancellation based on the contour of the chessboard grid

The previously obtained lines are reasonably good. However the image does not contain only the chessboard grid. It can contain other objects whose edges may be detected as lines. For our particular dataset, there is a margin around the chessboard grid which contains markings about the tile coordinates. This way most probably an edge is detected which is outside the grid.

To solve this problem our approach was to obtain the contour of the chessboard grid, then intersect each line with it. Finally keep only those lines which are indeed intersecting with the contour.

To obtain the contour of the grid, first all contours are found, then the one with the maximum area is kept. This would not be a good solution for any dataset. However for the particular images that was used in this project, the lighting helped to hide some edges of the chessboard margin, thus the largest contour became approximately the chessboard grid.

The next step is to cancel those lines which are not intersecting with the contour. To do this, the contour and the lines are drawn on separate images. Then, for the contour image and for each line images individually, the binary and operator was applied to compute those pixels which are present both in the contour and in the line. If the resulting pixel is all black, there is no intersection and the line can be discarded.

E. Finding the four corners of the chessboard

The next step is to intersect each horizontal line with each vertical line. To do this, before intersection the angle is verified between the two lines to be greater than a fixed value.

Because line detection may detect more lines than it should, the intersections may provide also some additional points that are not real lattice points. However this is not a problem yet. The goal is now to find the four corners of the chessboard.

First these intersection points are filtered to not have another point in their fixed radius. Then for the remaining points, their convex hull is computed. This convex polygon certainly contains the four corners of the chessboard, given if all four edges of the grid was found by the line detection algorithm.

Then the size of the convex hull is reduced to four elements, by removing those elements whose distance to the line, formed by its immediate neighbors, is smaller then the length of that line. This is repeated in a loop until only the four corners of the chessboard remain.

F. Sort the corners in counterclockwise order and rotate the image

To obtain known order of the corners, they are sorted in counterclockwise order. This is achieved by sorting them based on the slope of the line from the corner to the center of mass of the polygon defined by the corner points. After this operation, the corners are in counterclockwise order, the bottom-right one being on the beginning of the list.

The image needs to be rotated to a specific orientation, according to the chess rules. The H1 tile was chosen to be placed in the bottom-right corner. To do the rotation, one should rotate the list of corners before doing the perspective correction. Each unit of displacement will result in a turn of 90°. Therefore the list is rotated once which will correctly position the board.

G. Obtain an optimal margin around the reprojected image

By doing directly the perspective transformation, the four corners of the board will become the four corners of the image. This way if a piece overlaps the grid, it will be clipped.

To entail completely all the pieces in the transformed image, a margin is created around the grid, which will also contain the perspective corrected pixels.

To obtain the right dimensions of the margin, the contour points are verified to fit inside the reprojected image. First a perspective correction of the contour point occurs with 0 margin in all directions, based on the corners. Then the bounding rectangle of the contour is verified: if any point of this rectangle is outside the image, the difference is added to the margin of the corresponding direction.

Thus if we redo the perspective correction with the newly computed margin, it will contain completely all the pieces.

H. Perspective transformation and lattice point computation

The image and the corner points are transformed to achieve perspective correction, by using the *cv::warpPerspective* and *cv:perspectiveTransform* respectively.

If the resulting image is set to be a square, all the lines will become approximately parallel to the image edges, thus the horizontal lines will become approximately perpendicular to the vertical edges. Moreover the size of each tile will be approximately the same.

This important result helps to correctly obtain the corners of the image. The width and height of the image is divided by 8 (the number of tiles in a row/column). This value will be the distance between two consecutive lattice points. Thus with a nested loop all the lattice point positions can be calculated. Moreover the errors in the line or intersection point detection algorithms will not effect the result, given that the four corners are obtained correctly.

Therefore the chessboard is detected, and the four corners of each tile are known.

I. Piece detection

We used machine learning for detecting the chessboard pieces and we decided that the best approach is to work with *Faster R-CNN* (Region Based Convolutional Neural Networks). As a base for this project iteration we used a web notebook which was modified in order to meet our project's requirements [2]. The model was trained on the input data set using the TensorFlow Object Detection API. The Faster R-CNN is a two-stage detector which firstly identifies regions of interest and, secondly, feeds these detected regions to a convolutional neural network. Further, the resulting features are passed to *SVM* (Support-Vector Machine) for data classification and regression analysis. Subsequently, the model is

exported in a protobuf format, and loaded into the program. For using the model, the Tensorflow C++ API was formerly compiled and linked with the application.

The model receives as input the source image and it returns:

- The number of detected objects
- The class (piece type) for each detected object
- The score for each detected object
- Two 2D points per object, defining its bounding box.

The detected objects are filtered to keep only those which have a score above a certain threshold.

J. Locating pieces in the final chessboard

The points of the bounding boxes of each object are transformed through perspective correction, based on the corners. Based on the observation that all the images are taken from the same angle, it was deduced that only the maximum X and Y values of this bounding box must be considered in the computation of the final location. Equation 2 gives the formula for computing the chessboard location. $x_{top-left}$ is the cx coordinate of the top left corner. dx is the distance between lattice points in the horizontal axis, i.e. the width of a tile. The same formula can be used for computing the y values, by replacing all occurences of x with y, and all width with height.

$$dx = \frac{width}{8} \tag{1}$$

$$x_{chessboard} = \lfloor \frac{x_{max} - x_{top-left}}{dx} \rceil - 1 \tag{2}$$

K. Digital Chessboard Visualization

After the model is loaded into the program, the detected pieces are added, together with their attributes (class and position), to a storing data structure. Our final representation was developed according to the world-wide recognized standard, therefore, we used the chessboard and piece images previously downloaded from web [1]. Our implementation iterates over every tile of our 512x512 board and in case a piece is found on a certain position, the corresponding digital tile is occupied by the detected chess piece. Because the images are in an *.png* format, for this iteration we took into account only the non-transparent pixels of our matrix representation.

IV. Evaluation and Results

A. Corner detection for empty chessboards

On figure 2 one can observe the result of calling the corner detection algorithm on an image from the image set. The chessboard contains 8x5 tiles, which means that it contains 9x6 corners. All the corners of this chessboard were successfully detected, and each vertical line was colored with a different color.

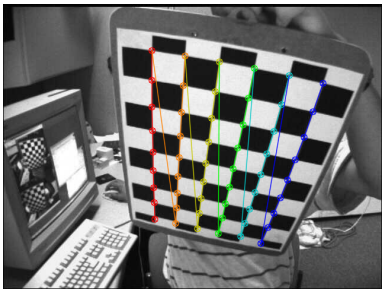


Fig. 2: The effect of the corner detection. The corners of the chessboard of size 8x5 was detected successfully.

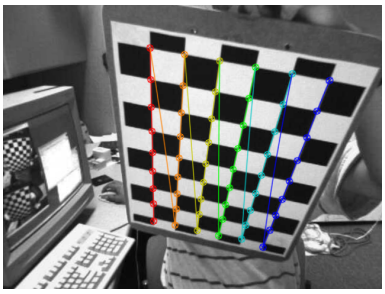


Fig. 3: Camera calibration reduces the distortion effect given by the camera lenses. One can observe that all edges of the chessboard became straight. The image is cropped.

B. Camera calibration

One can observe that the edges of the checkerboard are curved in figure 2. This radial distortion effect is given by the lenses of the camera which took the photograph. After applying camera calibration the camera matrix and the distortion coefficients are obtained. These matrices are used for undistorting the image.

The *cv::calibrateCamera* function returns the root mean squared (RMS) error of re-projection [10]. If this value is closer to 0, the values of the found parameters are more accurate. For the example image of figure 2 the obtained RMS error is 0.392583.

Figure 3 shows the first approach for undistorting an image using the *undistort* function of OpenCV. One can observe that the edges of the chessboard became straight. Another observation is that the image is cropped to have the same size as the input image.

Figure 4 shows the second approach for undistortion, by computings the undistortion and rectification transformation map and remapping the source image. One can observe that the edges of the chessboard are indeed straight, however now the image is not cropped, instead it appears like if someone tried to fold it in half horizontally.

C. Line detection

Figure 5 shows the input image after it is resized and converted to grayscale.

Figure 6 shows the result of the Canny edge detection. One can observe that the largest contour appears to be indeed the

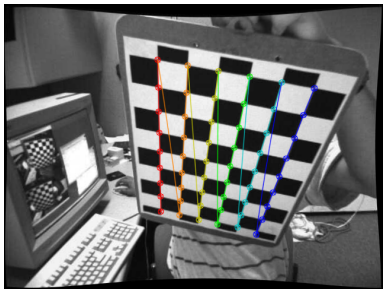


Fig. 4: All the edges of the chessboard became straight. The image appears like it is folded in half horizontally a small amount.

Fig. 5: Input image converted to grayscale

grid of tiles, and not the physical board itself with the margin. This happened because of the lighting of the input image.

Figure 7 shows the result of the Hough line transform algorithm and the detected lines. One can observe that it is not perfect. Because of the white pieces in the leftmost column ate close to the left edge, the algorithm detected multiple lines in that region. In total it detected 38 lines.

Figure 8 shows the reduced number of lines after the equivalence partitioning algorithm. It is visible that that there are much less lines, the green colored regions is much thinner in the left part of the board. In total it detected 22 lines;

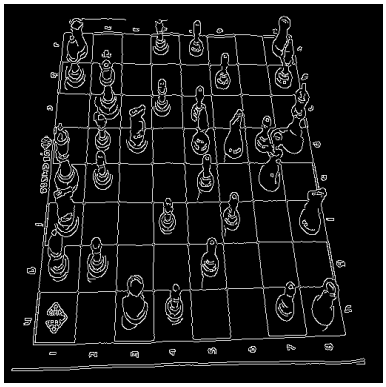


Fig. 6: Result of Canny edge detection

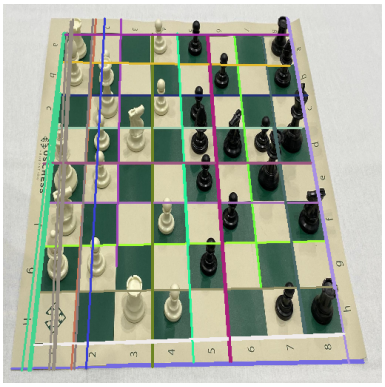


Fig. 7: Detected lines by the Hough line transform algorithm

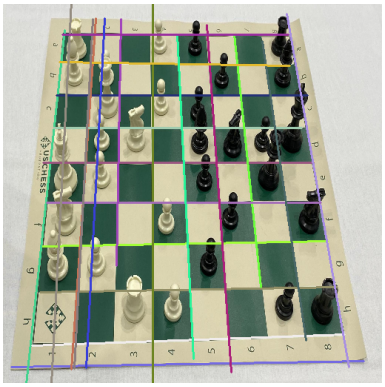


Fig. 8: The reduced number of lines after using the equivalence partitioning algorithm

D. Line cancellation based on the contour of the chessboard grid

Figure 9 shows the detected maximum area contour. As mentioned in section IV-C, this contour will be approximately the grid of tiles, the edges of the physical board don't appear entirely in the image of detected edges.

Figure 10 shows the lines after filtering by intersection. It detected 21 lines. The only line which is removed was the bottom edge of the physical board, because it was entirely outside the contour.

After the line detection steps 21 lines were obtained which are more than the optimal number of 18. However this error is resolved in the future steps.

E. Finding the four corners of the chessboard

The horizontal and vertical lines which were detected above are now intersected. Each intersection point is verified to be inside the image, and each pair of lines are verified to have an angle at least 40 degrees. 216 points are obtained. After discarding the points that are in a nearby neighborhood of other points (figure 11), 86 points remained. The optimal number is 81. The difference of 5 is obtained because of the 3 lines which were not supposed to be detected. In most of the examples the number of lines were detected correctly in the

Fig. 9: The detected contour

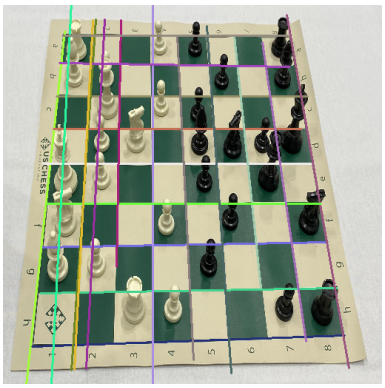


Fig. 10: The filtered lines after checking the intersection with the contour

previous steps, thus in this step the correct number of points were detected.

Again this error will be resolved in the future steps. The current goal is to find the four points of the board which is one of these points.

Figure 12 shows the points of the convex hull. One can observe that indeed the four corners of the board are part of this polygon.

Figure 13 shows the result after reducing the size of the convex hull to 4. Thus obtaining the four corners of the chessboard grid.

F. Sort the corners in counterclockwise order and rotate the image

The program prints to the console the coordinates of the corners (listing 1). One can observe that they are indeed in counterclockwise order. After doing the perspective transformation (figure 14) the image will be correctly oriented, because of the rotation of the corners.

Listing 1: Corner coordinates of the chessboard

```
Corners: [387.931, 41] [82.3471, 41]
    ↪ [36.6344, 461.084] [459.262,
    ↪ 446.062]
```

Fig. 11: The intersection points of horizontal and vertical lines

Fig. 12: The convex hull of intersection points

G. Perspective transformation and lattice point computation

Figure 14 shows the image after perspective transformation is applied on it and on its corner points. One can observe that the resulting chessboard is approximately similar to a regular 8x8 grid with equally large cells.

Based on these observations the new points are computed by dividing the horizontal and vertical distances by 8, which will be the approximate sizes of the tiles. On figure 16 one can observe the result of these calculations, and that it can be

Fig. 13: The reduced convex hull results in the four corners of the chessboard grid

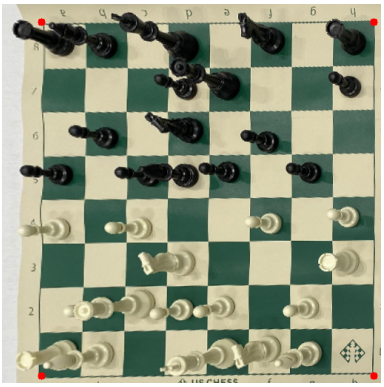


Fig. 14: Projection transformation of the image based on the four corners

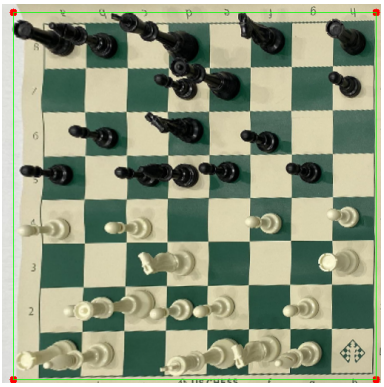


Fig. 15: The bounding box of the contour after perspective correction. One can observe that all figures are visible completely.

considered as a reasonably good result.

One can also observe on figure 15 that all the pieces are visible completely. This is achieved by setting the margin to the optimal value.

H. Piece detection and visualization

Figure 17(a) shows the input image, an actual photo of the real chessboard, in 17(b) one can see the predicted pieces with

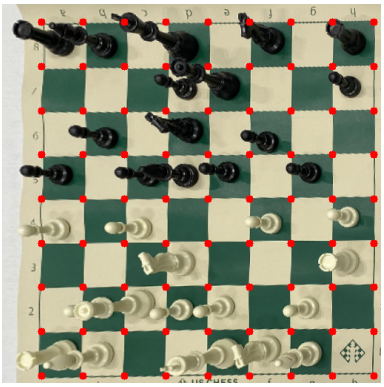
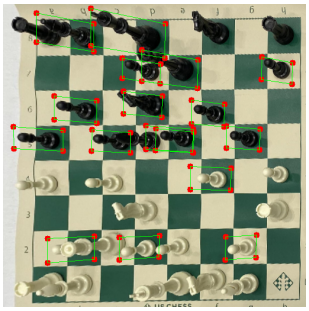
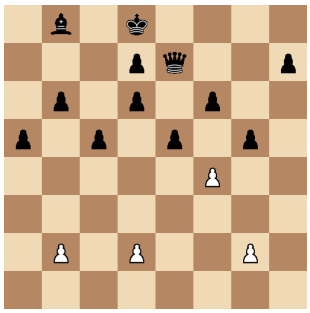


Fig. 16: Newly calculated lattice points

((a)) Source



((b)) Predicted pieces



((c)) Digital chessboard

Fig. 17: Final result visualization

their bounding boxes while figure 17(c) shows the final result, the digital representation of the given chessboard.

The model did not predicted perfectly. The causes can be the limited number of training epochs, shadows, piece overlapping and other factors. However it predicted correctly a noticeable number of pieces, thus a relatively good result is obtained.

V. Conclusion

The scope of this paper and the accompanying program was to present a method for obtaining a digital representation of a 3D chessboard and its pieces.

The goal was reached partly. The model needs to be further improved to provide better predictions.

Another limitation of our approach is that assumptions were made about the input data. The algorithm is specific to the features of the images:

- Camera position. This also implies the elongation direction of the pieces in the image.
- Lighting: the maximum contour contains only the chess grid because the lighting makes the borders of the chessboard edges undetectable for the Canny edge detection.

A great achievement from our perspective is that through this project we have broaden our horizons beyond the *Image Processing* domain, due to machine learning integration. We are looking forward to improving our approach so that a more general solution and a better prediction is obtained.

References

[1] Chessboard and piece images source.

[2] Training faster r-cnn object detection on a custom dataset. https://colab.research.google.com/drive/1U3fkRu6-hwjk7wWIpg-iylL2u5T9t7rrscrollTo=uQCnYPVDrsgx. Accessed: 2021-05-14.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Maciej Czyzewski, Artur Laskowski, and Szymon Wasik. Chessboard and chess piece recognition with the support of neural networks. *Foundations of Computing and Decision Sciences*, 45:257–280, 12 2020.

[6] Maciej A. Czyzewski, Artur Laskowski, and Szymon Wasik. Chessboard and Chess Piece Recognition With the Support of Neural Networks. *Dr. Dobb's Journal of Software Tools*, 2020.

[7] J. Ding. Chessvision : Chess board and piece recognition. 2016.

[8] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages 23.1–23.6. Alvety Vision Club, 1988. doi:10.5244/C.2.23.

[9] Sergiu Nedevschi. Lecture slides from the image processing course. 2021.

[10] OpenCV. *The OpenCV Reference Manual*, 3.4.13 edition, December 2020.