Computer Vision Approach for Chessboard Piece Recognition

Adriano Machado (up202105352), Félix Martins (up202108837),
Francisco da Ana (up202108762), and João Pereira (up202108848)
Faculty of Engineering, University of Porto, Portugal
(Dated: June 18, 2025)

I. INTRODUCTION

This report presents a classical computer vision approach for detecting and localizing chess pieces on a standard chessboard. Using OpenCV and common Python libraries, the proposed pipeline identifies the total number of pieces, extracts their bounding boxes in the image, and maps their positions onto an 8×8 matrix representation of the board, taking into account board orientation. The solution was developed and tested on a subset of 50 publicly available images, with results demonstrating consistent detection accuracy across varied board orientations and lighting conditions.

II. CHESSBOARD CORNER DETECTION

To help our objective, we decided to first warp the image so it only contained the chessboard and the pieces on top of it. In order to perform the perspective transform, we need to find the corners of the chessboard.

In all of the steps in this section, we worked on the grayscale image. For the main part of our corner detection algorithm, we used two different versions of the grayscale image: **(I)** a version with Gaussian Blur applied and then thresholded to isolate the color white, and **(II)** the original grayscale image.

In both of these images, we independently apply the following algorithm: **(1)** apply canny edge detection, **(2)** find the contours in the canny image, **(3)** select the contour whose convex hull has the largest area, and **(4)** merge that contour with nearby ones.

To perform the merging, we use the following iterative algorithm: **(a)** Compute the distance from each remaining contour (not their convex hulls) to the convex hull of the current largest contour. **(b)** Merge any contour that lies within a predefined proximity threshold. **(c)** Recompute the convex hull of the merged result. **(d)** Repeat the process with the remaining unmerged contours until no further merges occur.

In summary, while there exist contours with a very small distance to the convex hull of the largest contour, merge those contours with the largest one.

We use two approaches and merge their results, as each can produce noise outside the board. White thresholding is sensitive to strong lighting, while Canny sometimes captures contours of nearby pieces or even the ground. Since it's rare for both to fail on the same image, we take the intersection of their contours to reduce noise.

To finalize the corner detection, the rest of the steps are: **(5)** get the intersection of the two convex hull polygons, and **(6)** approximate the resulting polygon using `cv2.approxPolyDP`, with the objective of simplifying the result to a 4 vertices polygon, ideally matching the corners of the chessboard. For this final step, we try several error margins to control the approximation accuracy while finding the chessboard. After, we extract the corners by sorting the vertices by their x and y coordinates.

There are still some limitations, as common in traditional computer vision methods. For instance, pieces extending beyond the board under strong lighting can distort the detected contour. However, step **(6)** helps reduce these cases from affecting the final corner detection.

III. CHESSBOARD ORIENTATION DETECTION

To determine the chessboard's orientation, we locate the knight piece in the warped image using template matching. A reference image of the knight, cropped from the original board, is rotated to match each corner's expected orientation. These rotated templates are compared to the corresponding regions in the warped image, and the orientation with the highest similarity indicates the correct board alignment.

IV. CHESSBOARD GRID EXTRACTION

The grid extraction pipeline, inspired by the work of Belshe [1], begins by processing the warped image through several key enhancement techniques. Initially, Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied to enhance local contrast. Next, a Gaussian Blur is used to reduce noise artifacts, followed by Canny edge detection to outline the edges of the chessboard. Since this step can produce discontinuous edges, edge dilation is then applied to strengthen and unify the detected edges.

The Hough Lines transformation is then utilized to extract straight lines corresponding to the chessboard grid. Due to the sensitivity of this technique, extensive hyperparameter tuning was necessary, particularly of the minimum line length and maximum line gap, to reduce the number of false positives.

Despite these adjustments, the initial set of lines still included some noise and redundancies. To refine the results, we filter out lines with incorrect orientations based on an angle threshold and cluster nearby lines to merge duplicates. Even after these improvements, some expected horizontal or vertical lines may still be missing. To fill these gaps, we estimate the median spacing between existing lines and infer the positions of the missing ones, effectively reconstructing the full grid.

Once the grid is established, the intersections between the horizontal and vertical lines are calculated. As excessive intersections can emerge from peripheral noise, a final filtering step selects only the 81 intersections that are closest to the center of the board and maintain a minimum distance from one another, ensuring the creation of a precise square matrix that accurately represents the chessboard. If there was no minimum distance required between the points, the selected intersections would result in a circle around the center of the image.

This pipeline reliably extracts the chessboard grid, allowing us to iterate over each square for further analysis.

V. PIECE DETECTION

To find the board squares where there are pieces, we treat each square independently. We tried several approaches and the best one in terms of performance is what we will describe next.

For each square on the board, we crop its image using the coordinates from the previous step and apply `cv2.grabCut` with a hint mask to guide the segmentation of the foreground and background. The initial mask consists of two ellipses positioned slightly over the center of the square, as the shape of a chess piece is generally tall and narrow. The inner ellipse is labeled as `cv2.GC_FGD`, and the outer one as `cv2.GC_PR_FGD`, to define the probable area of the piece for extraction. A piece is considered present if the resulting foreground ratio falls within a specified range. This step is necessary because GrabCut may either fill the entire square or leave it blank when no piece is present. Therefore, we filter out cases where the foreground ratio is too low or too high. This approach proves effective, as a chess piece typically occupies only a portion of the square rather than the entire area.

To **detect bounding boxes**, we initially used HSV thresholding and contour detection. This method worked well for white pieces, since their hue is distinguishable. Nevertheless, to refine the results, we applied morphological operations: `cv2.erode` and `cv2.dilate`. However, black pieces were much more challenging to isolate using HSV thresholding, leading to many false positives and negatives. To address this, we dilated the resulting mask and used a distance transform-based threshold. We then applied `cv2.watershed` for region growing, which improved results but was still not ideal.

Given our good results for the board, we developed a second method for bounding box detection, which uses the GrabCut mask result as a starting point. The main idea of our algorithm is to start with this initial GrabCut mask and subsequently apply `cv2.watershed` to refine it, allowing the mask to extend outside the corresponding board square. In each iteration, we used the resulting mask to build the next hint mask, using erosion for the sure foreground and dilation to build the sure background. After several iterations, this process aimed to produce masks that fully covered each piece.

From these two approaches to extract the piece bounding boxes, we chose the second one due to its slightly better performance. However, since it heavily relies on accurate corner detection, it can sometimes produce poor results. Since we can detect if the corner detection was inaccurate, in those cases we simply revert back to the original approach, which does not depend as much on a well detected board.

Finally, to extract the bounding boxes, we apply the inverse warp transform, previously used on the board, to the contours (not directly to the bounding boxes). Once transformed, we then compute the bounding boxes from the adjusted contours.

VI. RESULTS

Performance was evaluated against ground truth labels for the 50-image test subset.

Corner detection accuracy, measured by Minimum Mean Squared Error (MSE) was useful in a first stage as it directly quantified the quality of the initial perspective warp, which is fundamental in our approach for reliable grid extraction and piece localization.

Board occupancy prediction achieved high accuracy, with a mean F1-score of **0.9039** (median: **0.9387**) across all squares in the test set. This demonstrates reliable identification of piece presence under varied conditions.

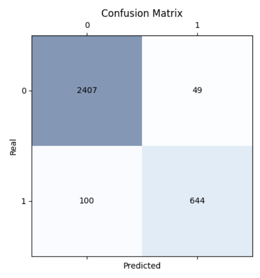

FIG. 1: Confusion Matrix for Board Occupancy Prediction.

For **bounding box detection**, the F1-score based on Intersection over Union (threshold of 0.5) resulted in a mean of **0.7336** (median: **0.7560**). As expected, precise localization is more challenging than occupancy detection, reflected in the lower score, particularly due to factors like piece overlap and perspective distortion.

REFERENCES

[1] Craig Belshe. *Chess Piece Detection*. https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1617&context=eesp. Senior Project, Electrical Engineering Department. 2021.

[2] Andrew Chen and Kevin Wang. “Robust Computer Vision Chess Analysis and Interaction with a Humanoid Robot”. In: *Computers* 8 (Feb. 2019), p. 14. DOI: 10.3390/computers8010014.

[3] Athanasios Masouris. *Chess Recognition Dataset (ChessReD)*. https://data.4tu.nl/datasets/99b5c721-280b-450b-b058-b2900b69a90f/2. Sept. 2023.

[4] Daylen Yang. *Building Chess ID: Featuring Computer Vision! Deep Learning!* https://medium.com/@daylenyang/building-chess-id-99afa57326cd. Jan. 2016.