Computer Vision Project - Task 1

Chessboard representation and pieces detection

Miguel Lima (up202108659) - Martim Maciel (up202400313) - Jaime Fonseca (up202108789) - Pedro Ferreira (up202409828)

The problem was divided into 5 main steps, described below in more detail: board segmentation, correct board orientation, grid segmentation, identify occupied tiles, define pieces' bounding boxes. All figures are present in the Annexes.

Step 1 - Board Segmentation

To avoid detecting unwanted features in the background, we started by creating a **mask of the table**, filtering pixels with its wooden color. Over that color selection, we applied the following operations:

- **Morphological erosion** to reduce mask noise
- Selected the **biggest area** contour
- Created a **convex hull** to obtain the rectangular shape of the **table**, filling in occasional holes in the initial color selection.

Then we converted the **original image** to grayscale and applied a **bilateral filter** to reduce noise while keeping edges sharp. Using the **Canny algorithm** with the table mask, we identified edges in the image, **applied dilation to make edges more prominent**, and used a **morphological close to join disconnected edges** in the chessboard.

In the **resulting image**, we identify all **contours** and **select the one with the biggest area**, since in our dataset it is always the **board**. We calculate a convex hull of the contour and approximate it to a square, through the *convexHull* and *approxPolyDP* functions, to mitigate cases where border pieces create protrusions in the chessboard contour.

Finally, using the corners of the chessboard, we computed the warp matrix to apply a perspective transformation to the original color image. This process is represented in Figure 1.

Step 2 - Rotation Process

To achieve the correct orientation, we use the small horse that appears in the corner of the chessboard, and rotate the image so the horse lies in the bottom-left position.

For every image, some **preprocessing was necessary,** both on the small horse image (which we cut out) and on the original image; grayscale conversion, a Gaussian filter to reduce noise, and the CLAHE algorithm to increase contrast in the image.

With the **images preprocessed** we applied the **SIFT algorithm** to find the keypoints and descriptors of the two images, then we applied the **FLANN matcher** algorithm to match the descriptors. Using the **RANSAC method** we obtained an accurate homography matrix, from which we **obtained the rotation angle** between the correct horse position image and the position of the horse in the image we are comparing. With this angle, we rotated the warped image to the correct orientation (horse on the down left corner), represented in Figure 2.

This process worked for all scenarios in the dataset. Curiously, there was a period where, with different OpenCV versions, the board would/would not rotate, because of the SIFT operation. Changing the horse image to contain more identifying elements solved that issue.

Step 3 - Grid Area Board Detection

After initially just doing a cut of fixed length on the borders, a more robust and generalized approach was adopted. On the warped image from step 2, the following is applied:

- Canny Edge Detection and Hough Lines
- Those lines are then drawn into a blank image, let's call it a density image.

The **density image is divided into 4 quadrants**, and for **each quadrant, the point with the highest score is found**. That score is calculated based on a heuristic over the density of the Hough lines and the distance of that point to the center (there were many points with high density in the middle of the board as well, so we had to account for that). With the 4 corners, we do another warp, now including only the playable tiles section of the board, represented in Figure 4, allowing us to break it down into individual tiles.

Step 4 - Matrix Representation

To reason about individual tiles, we divided the image grid into 8 equal parts in each direction, and applied the "houghCircles" function to check if there is a piece occupying a tile, since all pieces have a **common circular base**. We apply a heuristic to **only consider circles whose center is relatively near the center of the tile**. Additionally, in this step, the color of the piece is calculated.

Step 5 - Bounding Boxes

Leveraging the matrix representation previously obtained, we **calculated the 4 corners of each occupied tile** in the warped image, then calculated the inverse warping and rotation matrices to **transpose the points' positions to the original image**.

For each occupied tile, we defined an initial bounding box approximation, whose left, right, bottom, and top borders corresponded respectively to the smallest x, biggest x, biggest y, and smallest y - factor, among the 4 corners' coordinates. The factor is inversely related to the tile's height (y-biggest and y-smallest), since in most images, when the tile's height is smaller, the angle at which the picture is taken is more sideways, meaning the pieces should have a bigger height.

To better distinguish black pieces and black tiles in the board, we apply a bilateral filter over the original image to reduce noise and perform gamma correction to increase contrast in lower-regions of the image.

In the previous step, we predicted whether there is a black or a white piece in a tile. Here, for each initial bounding box, we try to find a piece solely of that specific color, by f**iltering pixels with a yellowish or black color range**, followed by morphological closing and opening operations, as well as discarding any pixels that fall outside of the contour of the chessboard (considering both protruding pieces, and the board edges), to clean the masks. This way, if there is a black and a white piece simultaneously, the bounding box is

not extended thinking they belong to the same piece. In case a piece of that color is not found (possibly due to miscalculation in the previous step), we attempt to search for one with the opposite color. We consider a **piece to be the biggest area contour, within a maximum distance from the lower third of the bounding box,** and then we calculate the **minimum bounding rectangle of the contour, to obtain a final refined bounding box of the chess piece**.

This solution **works for general cases**, assuming step 4 is correct, but fails in scenarios where equal color pieces overlap, or in specific lighting conditions that cause the black tiles' colors to fall inside the filter range, or the piece's highlights/shadows to fall outside the filter range. This process is represented in Figure 4. In Figure 5, there is the initial image of a board, with the bounding boxes drawn.

Results and testing

To evaluate and improve the pipeline, we hand-made solutions for all chess boards, in the solutions.json file, achieving the following final results. These results only pertain to tile occupation, ignoring the piece color prediction.

Accuracy	Precision	Recall	F1 Score
94.97%	89,12%	87,45%	88.27%

This was especially useful, as in the beginning we had high accuracy and precision values but relatively low recall values, meaning we had a lower number of positive rates. Tweaking the parameter values (for example, to recognize more circles, to recognize more pieces,..) was enough to increase almost 20% of the recall value, maintaining other metrics. These values represent the most optimal solution found.

Notes and AI Usage

As previously mentioned, as an extra feature, we have a method to recognize the color of the pieces (black or white), which we describe below. After having the individual tile and the circle recognized, the tile is cut from the center with a distance equal to 2*radius of the circle recognized, to cut as close as possible to the center of the piece. The image is on a gray scale, and a simple threshold is enough to determine if the color of the piece (which is the majority of this new image) is white or black. Updating the metrics to also consider color correctness, the FP (False Positives) now means incorrect occupied prediction, or correct occupied prediction and a mismatch in color, and TP (True Positives) a correct occupied and color prediction. These are the results we have obtained for recognizing pieces' colors:

Accuracy	Precision	Recall	F1 Score
94.03%	84.71%	86.88%	85.78%

To better develop the project and test different approaches, a **Pipeline Design Pattern** was applied, allowing the combination of steps in different orders.

AI tools were used for:

- Brainstorming ideas when some approaches were not working the way we expected
- Some of the code was written with AI, mainly for things we knew how to do (we are lazy) and, always with critical thinking - that is why, in terms of architecture, the project makes sense.
- Understanding bugs

ANNEXES

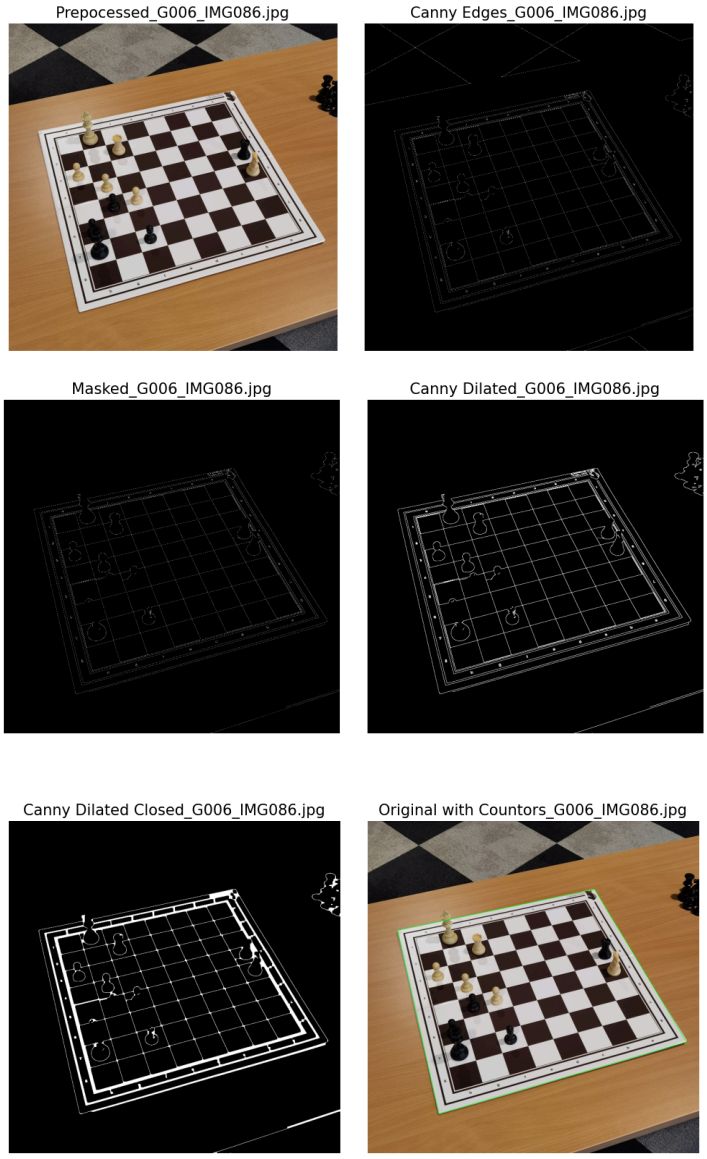

Figure 1: Pipeline of images for the board segmentation component

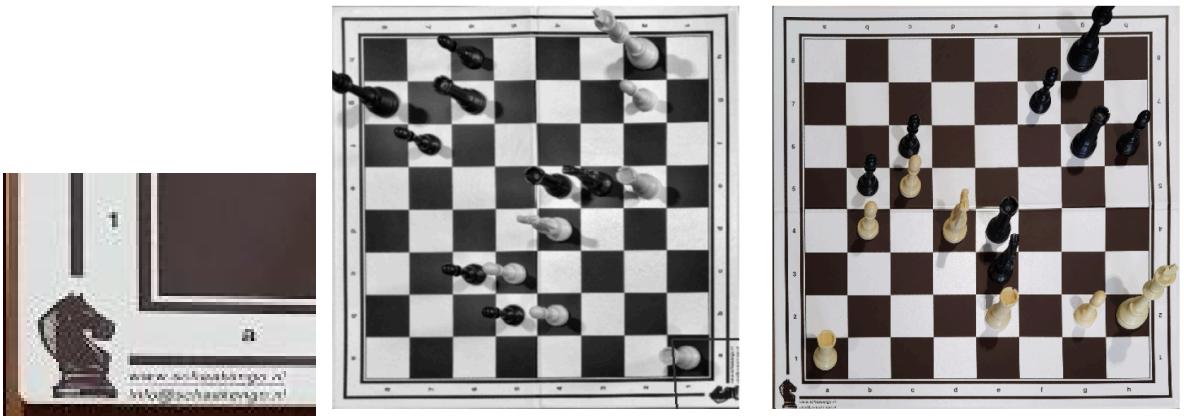

Figure 2 - Cutout of the small horse, sideways warped chessboard, and warped chessboard with corrected orientation

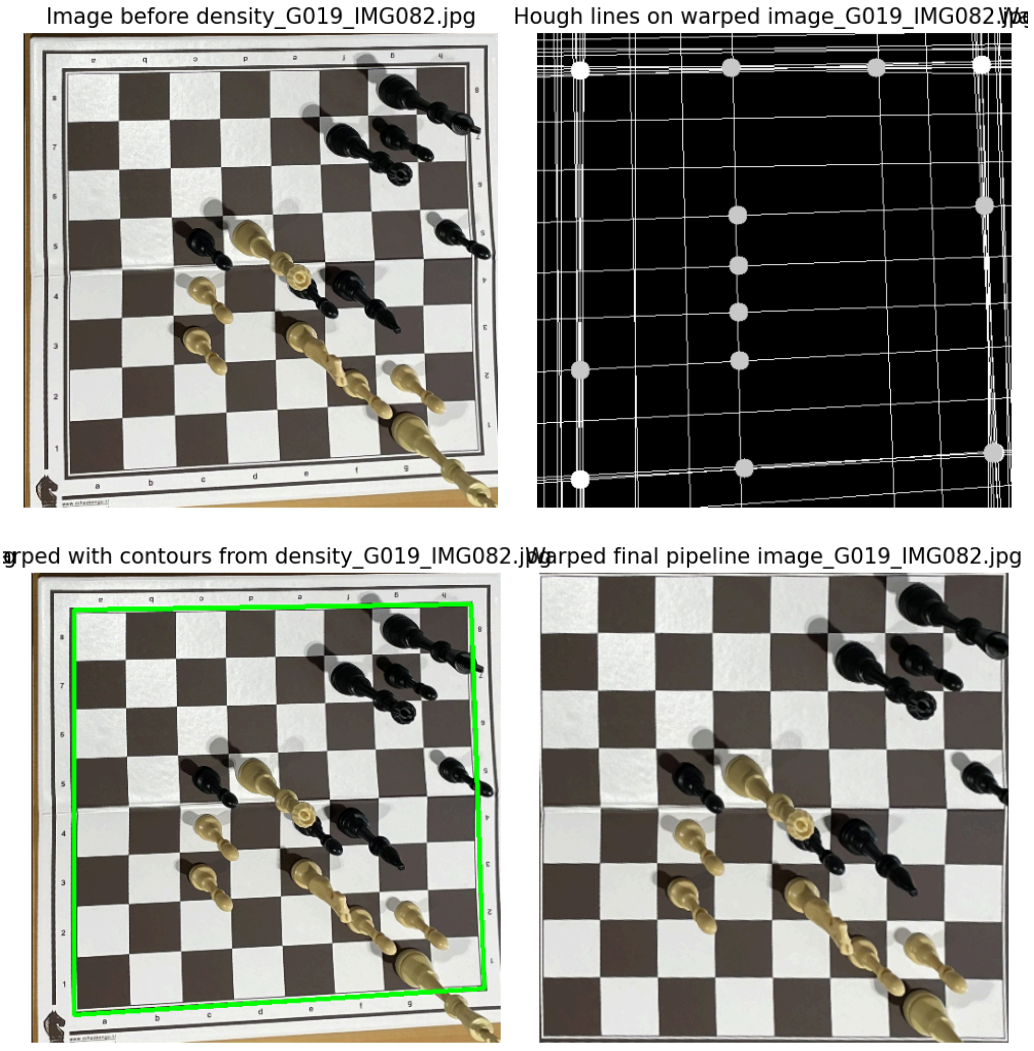

Figure 3 - Example of an image where the board segmentation did not work well enough. However, the approach through density allowed for a correct division of the board into separate tiles. In the second image, the darker circles present points of high density, and the white points represent the ones chosen to cut the image

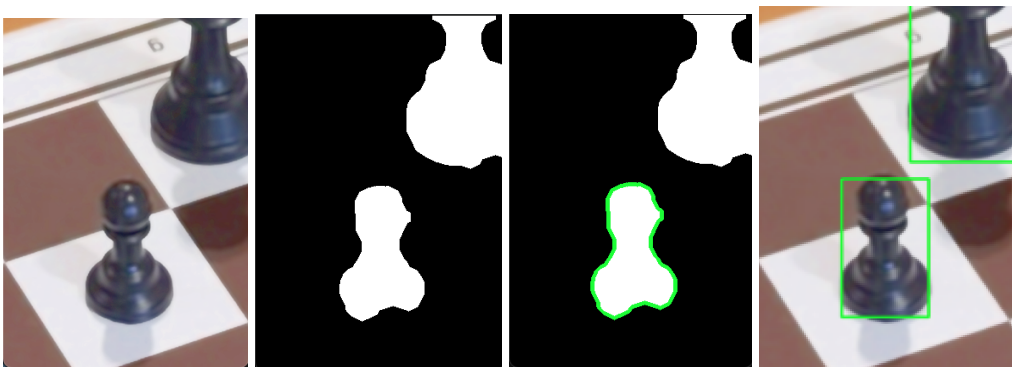

Figure 4 - Cutout of initial bounding box in colored image, in piece mask, and in piece mask with selected contour; final piece bounding box

Figure 5 - Bounding boxes drawn in the initial image