Computer Vision Project - Group 05

Graillet Arthur Gardier Simon Van de Vyver Eri

1 Problem Statement

The project goal is to identify chess pieces on a board, based on their movements. White pieces are all represented by a same object of same color, same for black pieces. The initial, classic, placement cannot be assumed.

The algorithm must update the game state as moves are made. If a piece cannot be confidently identified, it should be marked as "unknown". The expected output is a `JSON` file containing all game states.

2 Methodology

The main classes of the program are: `Program`, `VideoProcessing`, `Chessboard` and `Solver`.

The video processing section (see 2., ..., 7.) of the project has been parallelized to improve resolution speed for the competition.

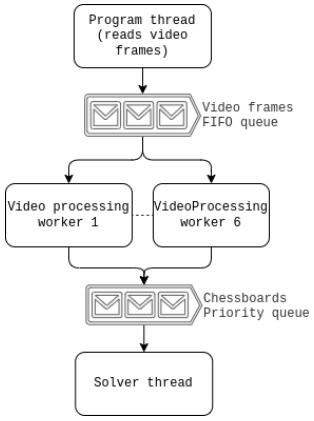

Figure 1: Parallel architecture

The two speed bottlenecks in the project are the video read from disk and the video processing. Parallelizing the video processing stage allows us to reach the same pace as the read from disk, which became the only bottleneck, that can not be solved. Our program finishes as soon as the video read finishes, providing the fastest analysis possible.

Here are the important steps of the program:

1. **Read video:** The video is read and the frames are stored in a queue.
2. **Detect stickers:** The stickers are detected using a mask on a hsv version of the frame.
3. **Detect black square:** The black squares are identified using thresholding, opening, contour detection and filtering the contours found.
4. **Detect corners:** Using the black squares, the diagonal a1-h8 is detected. We can assign each black square center a position in the chessboard using the homography of the corners of a1 and a8. That gives us enough points to compute the homography to find the corners.
5. **Assign corners:** To assign the corners to their respective position (a1, a8, h1, and h8), we use the stickers detected previously. We draw a line between the two stickers and a perpendicular one that gives us four quadrants. Each corner should be in a different quadrant that defines which position it is on the chessboard.
6. **Crop frame:** The frame is cropped to only keep the chessboard in a square frame.
7. **Detect pieces:** The pieces contours are detected using bluring, dilation/erosion, adaptative thresholding, masking, opening, closing, and hulls.

8. **Compute position matrix:** Each contour is assigned a position on the chessboard.

9. **Compute variation matrix:** When a capture is detected a variation matrix is computed between the current board image and the previous one. For each square containing a piece, the following normalized variation score :
$(\frac{norm(mean\_color\_diff)}{\sqrt{3}\times 255}+$
$I(team\_swapped = 1) * 0.5+$
$\frac{norm(piece\_center\_diff)}{\sqrt{2}\times square\_size})/3$
The team assignment at this stage is done using KMean (k=2) clustering on the pieces mean color.

10. **Analyse the movement:** The difference between two frames is computed to detect the movement of the pieces. For capture moves, we use the variation matrix defined above and a verification of legal moves to detect where the piece attacking ended. Castling and enpassant also have specific rules to detect them.

11. **Update piece color:** Temporary colors are given to each piece that move until we know the real color. Once we know the real color we can backtrack and replace the temporary color by the real one. We can be sure of the color in case of castling, enpassant and pawn detection. Capture is also used to determine the color.

12. **Update piece types:** The pieces start with all possible types. When a piece is moved, we filter the possible types based on the movement and the color if it is already defined.

The limitation of this methodology is that black square detection is not perfect on every video. If more than two turns are played between two frames analysed, the program will not be able to determine anything after that frame. We also had to make the hypothesis that each type had a certain number of possible instances for each team even if pawns could potentially be promoted.

Once finished, the resulting game states and frames are placed in a JSON file. Evaluation is made separately from the resulted JSON files.

3 Results

As a comparison, the implementation had been compared with JSON files that contain the true game states and their corresponding frames.

For the analyses, we only used the videos from our own group because our code was primarily based on them.

A first approach was to compare the number of mistakes that were made between the created JSON file and the "true" JSON file. The number of mistakes have thus been calculated.

We observed that the number of errors initially remained close to zero, but after one error, the number of errors rapidly increased.

In addition, we have evaluated how fast the program converge to a state where there are no unknown pieces anymore.

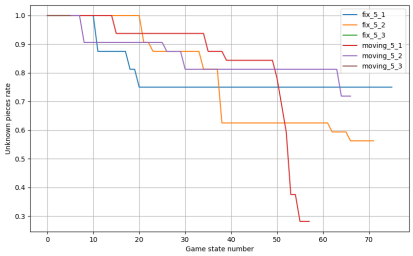

Figure 2: Rate of unknown piece in function of the gamestate

We note that the number of unknown pieces is never zero. For fix_5_1, the number of unknown pieces remains high even if no errors were made.

4 Contributions

Graillet Arthur contributed to: Chessboard detection, piece detection, game logic, and report.
Gardier Simon contributed to: Stickers detection, variation detection, legal move analysis and parallelization.
Van de Vyver Eri contributed to: Game logic, evaluation, and report