ICBV Final Project - Speed Chess

Dor Litvak Gal Elgavish

1 Introduction

1.1 Background

The chess game is a long game full of details. The project goal is to summarize a chess game to its algebraic notation for players who want to learn many methods, analyze their games quickly and efficiently. We also offer a visualization of the chessboard throughout the game.

Our method can read the board and detect where the pieces are on the board. Given the board, the piece's location, and the game's order, we can determine their algebraic notation. And to present a visualization of the board at any given point of the game.

The following figure present our results over 3 chess moves ['f4', 'e5', 'fxe5'].

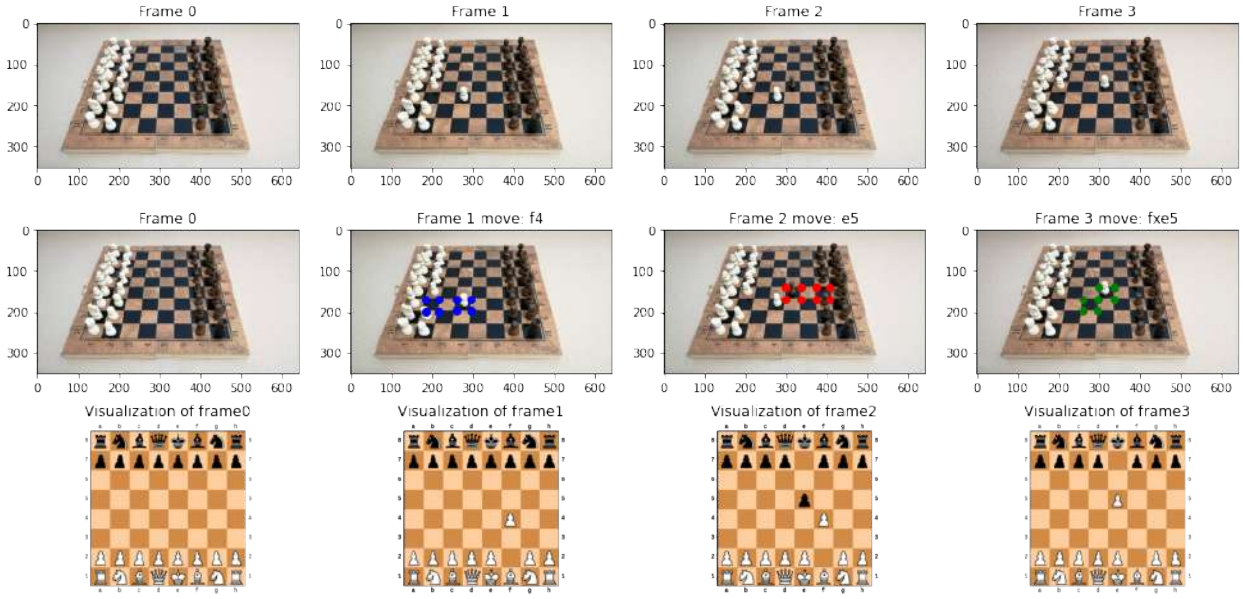

Figure 1: The first row present the moves frames extracted using our method, the second row present our analysis for each move, the third row contains visualization of the board using the algebraic notation we found.

1.2 Chess Algebraic notation

Algebraic notation is the standard method for recording and describing the moves in a game of chess. It is based on a system of coordinates to uniquely identify each square on the chessboard.

Figure 2: Chess board coordinates and initial setting.

1.2.1 Naming the squares

Each square of the chessboard is identified by a unique coordinate pair, a letter and a number, from White's point of view. The vertical columns of squares, are labeled a-h from White's left to right. The horizontal rows of squares, called ranks, are numbered 1-8 starting from White's side of the board. Thus each square has a unique identification of file letter followed by rank number. For example, the initial square of White's queen is designated as "d1", the Black's queen is "d8".

1.2.2 Naming the pieces

Each piece type (other than pawns) is identified by an uppercase letter.
K - king
Q - queen
R - rook
B - bishop
N - knight (since K is already used)

1.2.3 Notation for moves

Each move of a piece is indicated by the piece's uppercase letter, plus the coordinate of the destination square. For example, Be5 (move a bishop to e5), Nf3 (move a knight to f3). For pawn moves, a letter indicating pawn is not used, only the destination square is given. For example, c5 (move a pawn to c5).

Captures - When a piece makes a capture, an "x" is inserted immediately before the destination square. For example, Bxe5 (bishop captures the piece on e5). When a pawn makes a capture, the file from which the pawn departed is used to identify the pawn. For example, exd5 (pawn on the e-file captures the piece on d5).

2 Our Method

Given a chess game video, we extract the frames from the video and use various computer vision methods to detect the algebraic notation of the game.

2.1 Find chess board full corners

We assume that we have a clean chess board at the beginning of the game. And we can extract an empty board image, denote clean board.

Extract 7x7 chess board corners:

Using open-cv method *find_chess_board_corners*, we extract the 7x7 internal corners coordinates of the chess board. The clean chess board image and the 7x7 coordinates found is presented in the next figure.

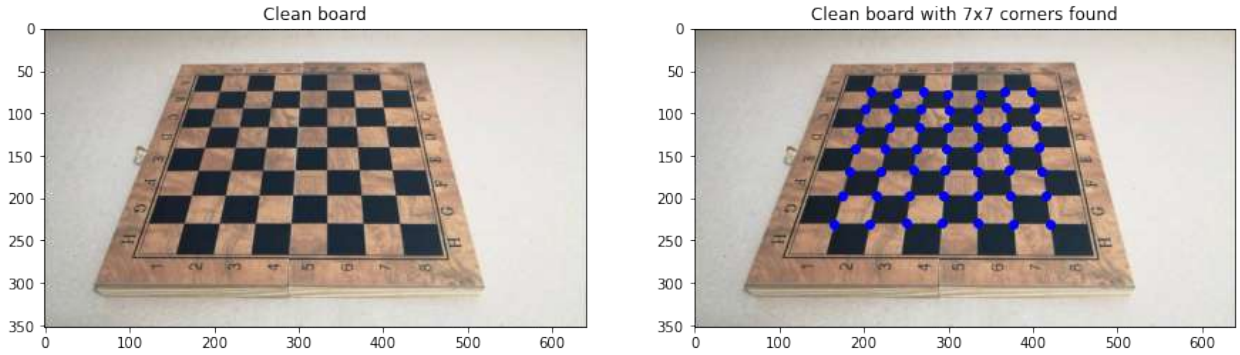

Figure 3: From left the clean chess board image, from right the clean board with the 7x7 coordinates found in blue.

Extrapolate corners:

We can find the full 9x9 board coordinates using the 7x7 coordinates we found using open-cv. For every two adjacent points in the two outer layers of the board coordinates, denote as (x1,y1) and (x2,y2) when (x1,y1) exterior than (x2,y2). The new outer coordinate (x3,y3) would be:

$$(x3, y3) = (x1, y1) - ((x1, y1) - (x2, y2)) \tag{1}$$

The following figure illustrates the extrapolation procedure. The red and the yellow coordinates on the left image are the coordinates used for extrapolating the green coordinates on the right image.

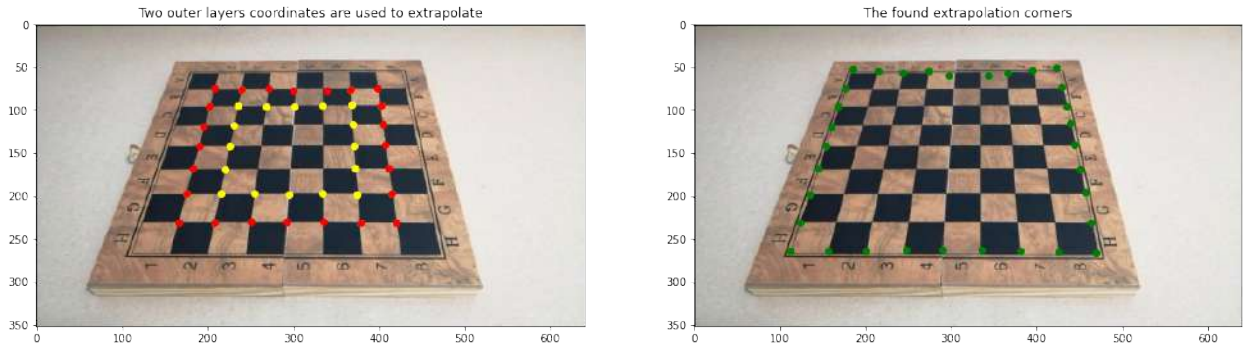

Figure 4: From left the extrapolating input from right the extrapolation output.

Refined tuning of the corners:

We found a full board coordinates 9x9. But we can notice inaccuracies in the coordinates, caused both from the original open-cv coordinates found and from the extrapolation we performed. We can fix those inaccuracies using Linear Least-squares. The board consists of nine vertical lines and nine horizontal lines. We are using Linear Least-squares first to find the upper and lower lines passing through the outer points we found on the board. When we saw the a1,b1, a2,b2, the parameters correspond with those lines. We project the coordinates to find a better approximation of them. We perform the same process over each of the nine vertical lines of the board. And lastly, we are making the exact approximation again to the nine horizontal lines of the board. The result of the refined tuning process is present in the next figure.

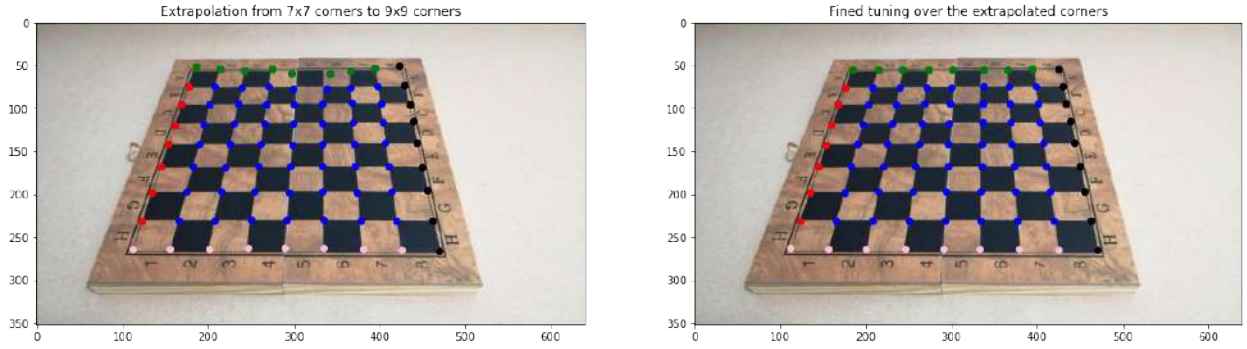

Figure 5: From left full coordinates before refine tuning, inaccuracies in the upper board, from the right the coordinates after the refine tuning, the upper coordinates are aligned.

Find squares using corners

We have full-size 9x9 chessboard corners. Each square on the board has four coordinates representing it. Together they create a parallelogram. We will later want to use the chessboard squares' mean color, and it would be hard to do so using a parallelogram. For this reason, we want to find for each location on the board the minimum square is contained in the parallelogram representing it. We find it using the smallest diagonal of the parallelogram.

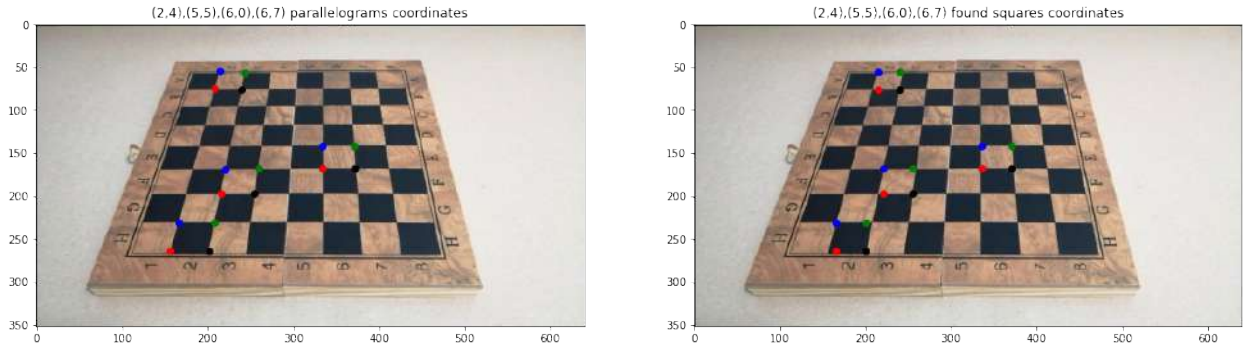

Figure 6: The following figure shows four parallelograms created from the corners (on the left) and the squares found for those parallelograms from the right.

2.2 Extract moves from video

In order to extract images of the different moves we are performing the following procedure:

- Get the different frames from the video every *#suspension* milliseconds
- Preprocess each frame by converting it to gray-scale and applying a Gaussian-blurring filter using cv2.cvtColor and cv2.GaussianBlur respectively.
  The gray-scaling is needed for the thresholding function, and the Gaussian-blurring will help to avoid small light changes between frames (that can be falsely detected as a movement)
- We computed the binary-changes-image by computing the absolute difference between the current frame and the last one and thresholding for significantly changed pixels only, using cv2.absdiff and cv2.threshold respectively
- We computed *pct_changed* which is the percentage of binary-changed pixels from all pixels
- We determined that there was a movement between two following frames if *pct_changed* was greater than a certain threshold (0.1%)
- Finally, if there was a movement in the previous frame and in the current it stopped, we save the frame as an image of the next move in the game.

Note that we made an assumption that a hand (or any object creating a movement) can appear in the video only in order to make a move and then exit the frame.

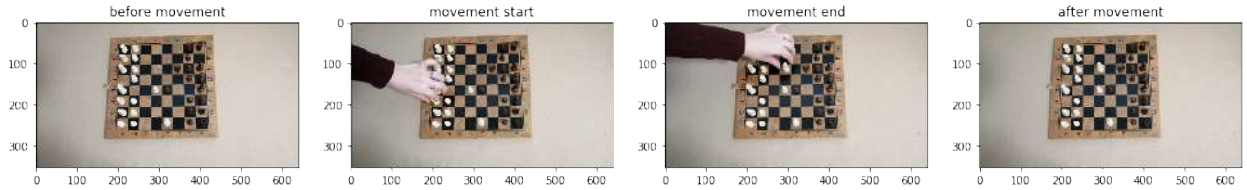

Figure 7: The following example shows four (1 second suspension) consecutive frames. The 'before movement' frame is the base frame. Then, the algorithm ignores the 'movement start' and 'movement end' frames, and saves only the 'after movement' frame as the newly made move.

2.3 Find chess piece movement

Find chess board composition

Find scaling parameter - a parameter that represents the upper-right coordinate location, can get the values of "a8" or "h1" (white on left, or white on right). We can find the white and black sides (left-right or right-left), using the mean color of the left squares and right squares of the board.

Initialization

- scaling parameter - found previously.
- squares_names - an 8x8 board contains each square name, a letter and a number.
- pieces_locations - an 8x8 board contains the initial setting of the board, with respect to the squares_names and the scaling parameter.
- Game_Algebraic_Notation - an empty list, in every iteration of the loop a new chess move will be added to it.
- clean board mean - for comparison reasons.

The main loop

For every two frames of a chess move, denote as frame_i and frame_{i+1}, we will calculate the distance between them. To find the origin of the moving piece, we will subtract frame_i - frame_{i+1} + clean board. When subtracting the frames, we also subtract the board, then we add it back. We then compute the mean color of each square on the board. The coordinates would be the coordinates with the maximum color change compared to the clean board. We will do the same procedure to the moving piece's domain, using frame_{i+1} - frame_i + clean board.

The result is the origin and destination coordinates.

We will calculate the algebraic notation of the move for those coordinates we found, considering the last state of the board (for capturing) and the piece type (pawn vs. other pieces).

The algebraic notation of the move is then added to the list of algebraic notations. Using this list, we can create a visualization of the game moves.

Example over 3 moves video.

First frame, piece: White pawn, from square: f2, move to square: f4, where there was a piece: None. In the first row are the captured frames that contain moves. The second row contains the subtracted frames. The third row contains the origin and destination locations found from the maximum change in color.

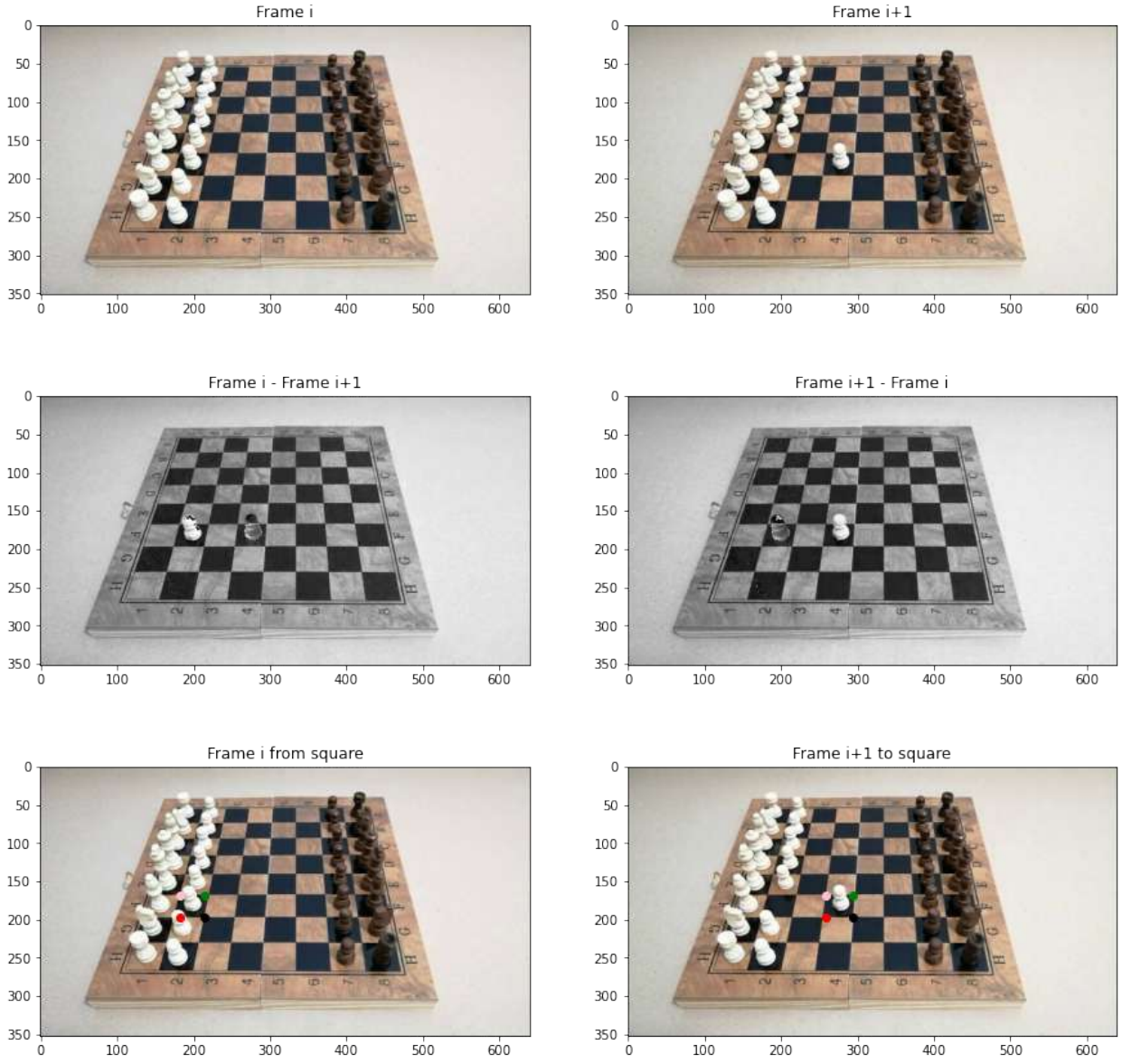

Figure 8: First iteration, the algebraic notation found 'f4'.

Second frame, piece: Black Pawn, from square: e7, move to square: e5, where there was a piece: None

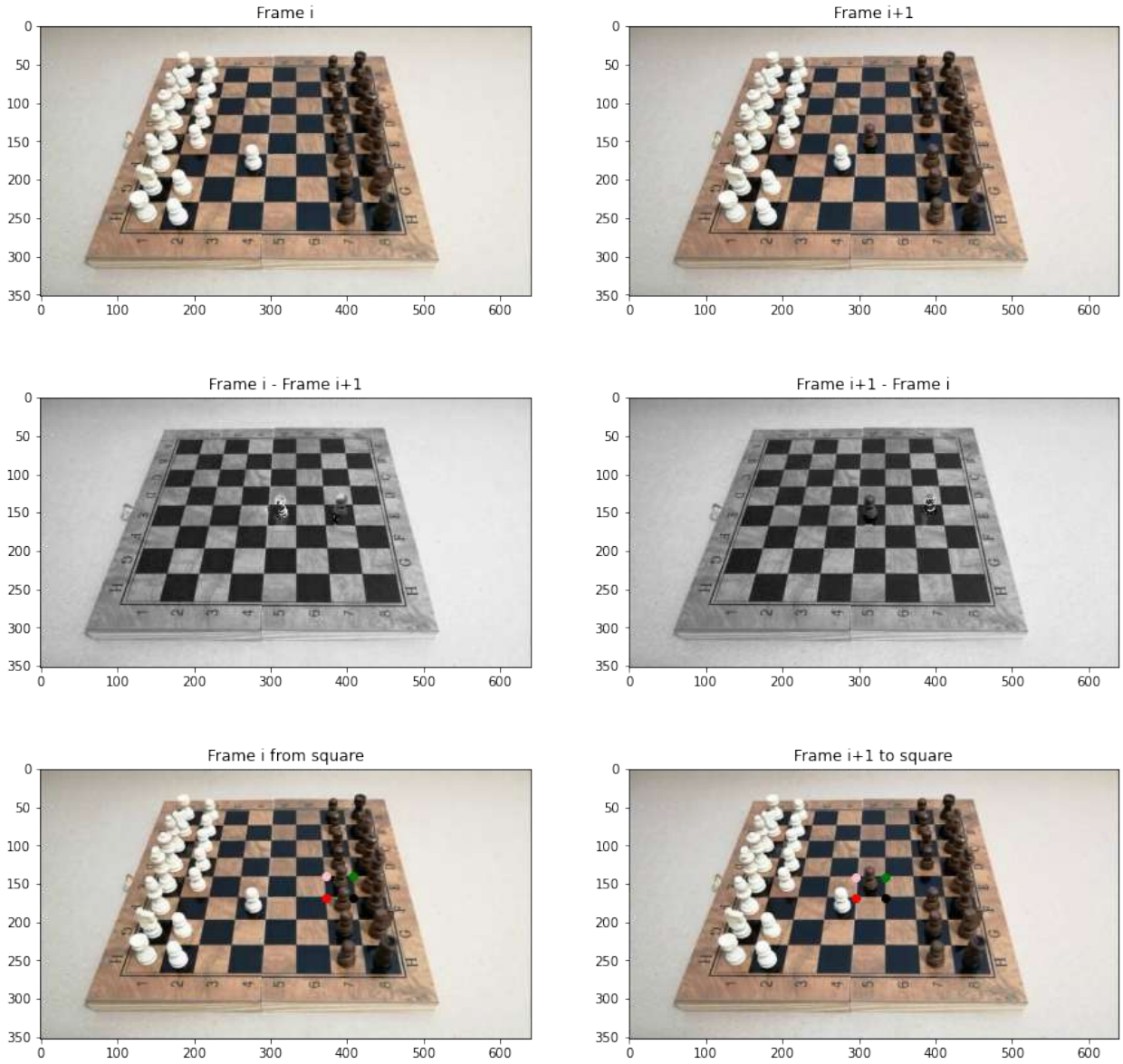

Figure 9: Second iteration, the algebraic notation found 'e5'.

Third frame, piece: White Pawn, from square: f4, move to square: e5, where there was a piece: Black Pawn.

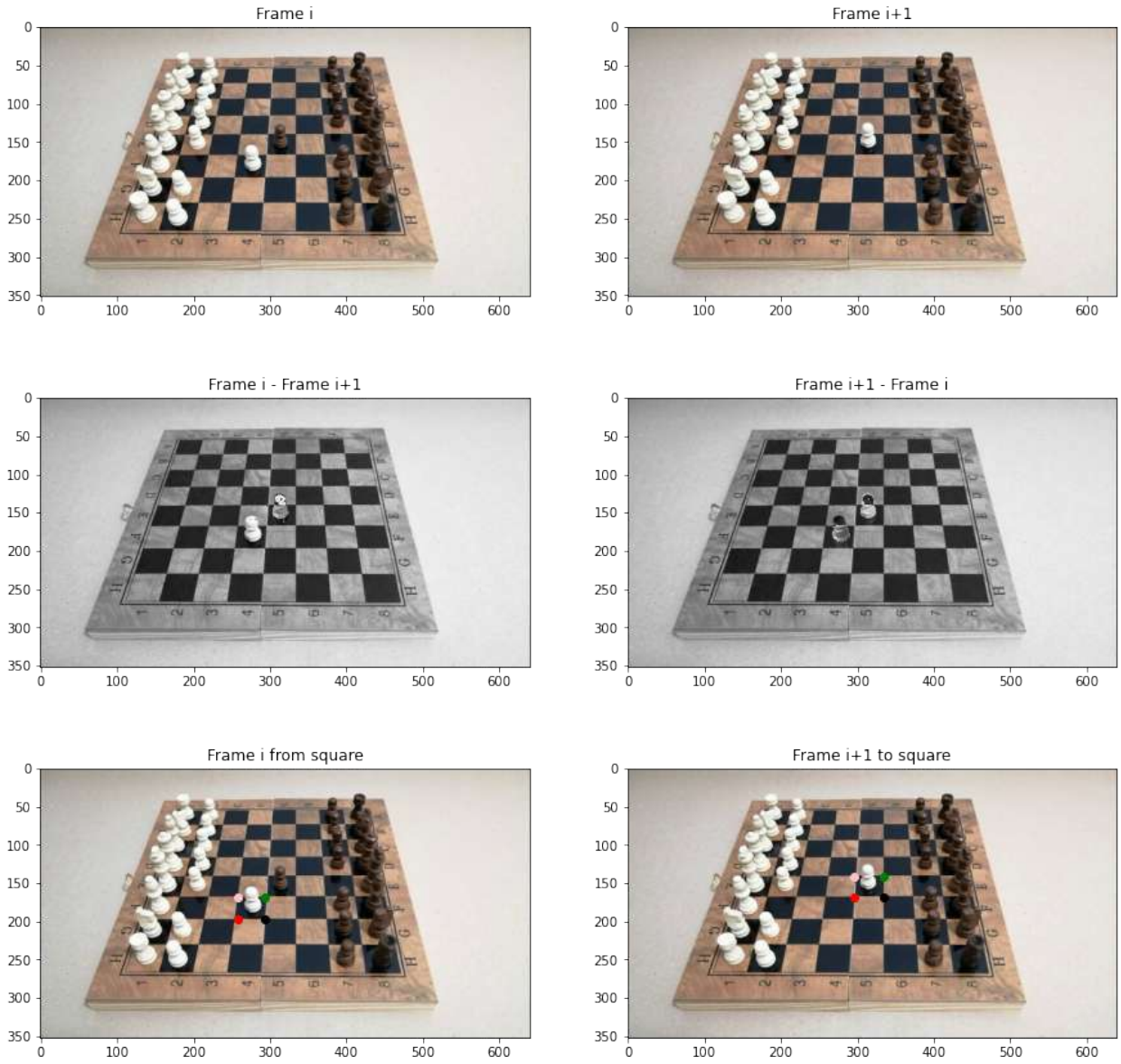

Figure 10: Third iteration, the algebraic notation found 'fxe5'.

The total algebraic notation found is: ['f4', 'e5', 'fxe5']

Using this algebraic notation and the chess-python package we can produce demo board containing all the game moves.

3 Conclusion and Results

Conclusion

Using the method we build, we can create a full chess game analysis. We detect the board coordinates and analyze them, then we extract moves from the given video, detects pieces location and composition. Using this information, we find the current movement algebraic location. After we have the full algebraic notation of the game, we can build a new synthesis speed chess game visualization.

Further Work:

- Improving the chess move detection by extracting the "suspicious" squares that have moved, and decide on the chess move by exploring possible moves for each piece or square.
- Performing the chess move detection in real-time (for example, with the computer's camera) and add recommended moves for each player (which can be useful for people who wants to learn)

Results

Our method can deal with videos taken from different directions. We succeeded in dealing with different moves and different scenarios.

For example, the following figure shows a chess game scene from a side camera. The first row contains the detected moves frames. The second row includes the detected squares coordinate moves (the headline of each figure in this row is the algebraic notation found by our algorithm). The third row contains the visualization we extract to that algebraic notation move using the python-chess library.

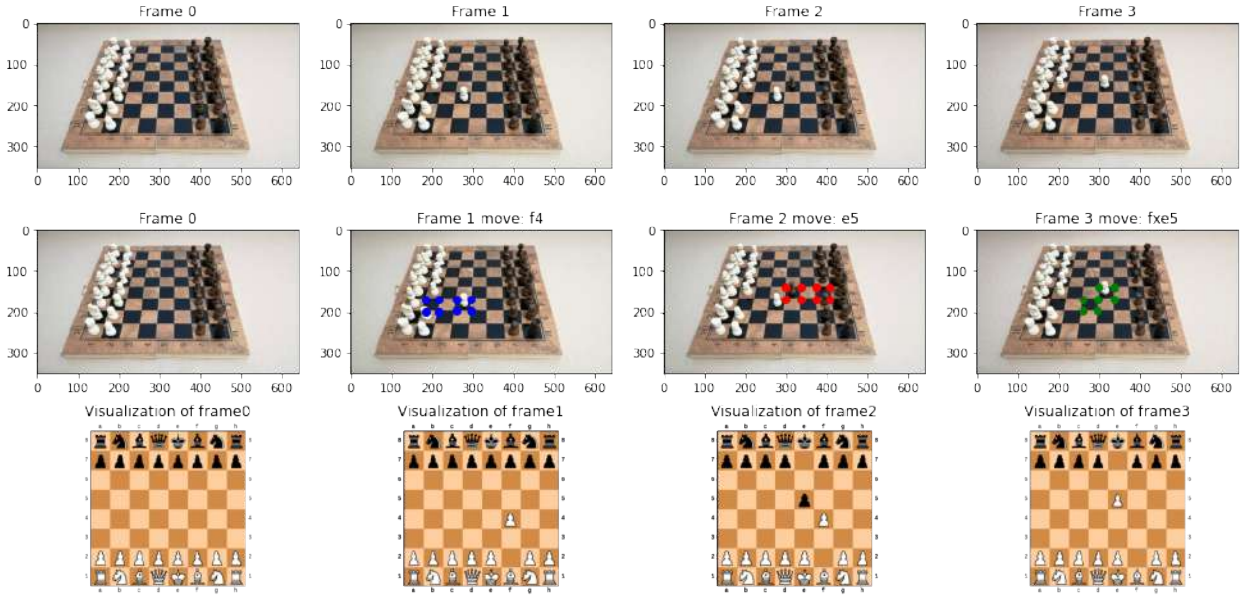

Figure 11: The following figure present our results over 3 chess moves ['f4', 'e5', 'fxe5'].

Other experiment videos can be found in the additional material or on our project video.

4 References

[1] Linear least squares - https://en.wikipedia.org/wiki/Linear_least_squares

[2] Linear least squares - numpy linear algebra:
https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html

[3] Visualization using python-chess library: https://github.com/niklasf/python-chess.git

[4] open-cv https://opencv.org/

[5] Chess board algrbraic notation: https://en.wikipedia.org/wiki/Algebraic_notation_(chess)