Problem definition

The primary goal of this project is to design and implement an algorithm that is capable of detecting and following a 2D chess game and outputting the according moves on the chessboard.

The following requirements were implemented:

1. Detect and crop out a 2D chessboard on an image.
2. Determine the rotation of the chessboard.
3. Separate the squares and assign corresponding coordinates.
4. Tracking the location of different pieces on the board.

Algorithms

1. Detect and crop out a 2D chessboard on an image.

   - Experiment:
     - Harris Corner Detection Algorithm

Due to the nature of a graphical screen, there are way too many corners. Because of the way the pixels work, sharp corners are everywhere (corner of an open windows, almost every letter or character has corners along them.

There is too much noise as the Harris Corner Detection Algorithm was not meant to be used on graphical screen. It works much more efficiently in real-life images.
• **Line Detection by Hough Transformation**

Again, due to the nature of a graphical screen, there are way too many lines/edges. Because of the way the pixels work, sharp edges are everywhere, almost everything has edges. Provably, it’s totally possible to tell what the original image was just by just looking at the edge map because most of the details remain.

Yet, again, there are too much noise as the Line Detection by Hough Transformation was not mean to be used on graphical image neither.

**Actual Algorithm:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x, y)</td>
<td>(x+5, y)</td>
</tr>
<tr>
<td>(x, y+5)</td>
<td>(x+5, y+5)</td>
</tr>
</tbody>
</table>

Checking for this very pattern where a pixel’s color is different than the ones to the right and below it but the same as the one down across to the right.

This simple algorithm yields a much better result. It filters out almost all of the corners except for the chessboard corners.
Note: some character with a positive slope would easily get matched with the pattern, like /

Similar colors were determined by calculating the Euclidean distance from the 3 color channels.

**Figuring out the chessboard**

The set of coordinates where corners were found is analyzed to pick out the most common distance between corners. Ideally, there should be a subset with a majority of the corners having an almost constant distance between each other. That set is figured out to be the corners of the chessboard.
**Determine board orientation**

The cropped board is divided in 4 vertically.

The average colors of the top and the bottom regions are compared to figure out the orientation of the board.

Since they have pretty much the same background. The region with the lighter average color must be the side with white pieces. Knowing which side is which, it was very straightforward to assign corresponding coordinates to individual squares.

**Tracking pieces**

Image subtraction was used constantly on the streaming frames to detect movements. Note that all of the squares were located and assigned correct coordinates beforehand. Thus, once a movement occurs, i.e. a difference in image subtraction is found, the program can check right away on which squares the changes occurred.

Ideally, there should be changes on 2 separate squares, for example

From this, the beginning position and the destination square is distinguished by calculating the “blankness” of the square. The blank square is determined to be the beginning position of the piece while the non-blank square is determine to be the destination square.
Results:

Example of a detected chessboard.

Discussion

The program works very good but not consistent as it has not been thoroughly tested.

It is capable of detecting chessboards, determine the orientation of the board and assign corresponding coordinates to the squares/pieces, and following pieces around the chessboard to output the correct chess notation.

There are still edge cases that have not been handled, like the castle move in chess where 2 pieces move at the same time or when part of the chessboard is not visible.

Future work:

If I have more time on this project, I would definitely continue what I started but did not get very far with, chess piece recognition. Currently, the program has to assume the beginning layout as it cannot recognize chess pieces. This is very confined because it can only work properly if it see a chessboard from the beginning, which does not happen very often in online chess videos.

Conclusion

This is a simple yet very interesting project. They key of this project was to design simple rules that could be applied consistently.

I was able to combine a chess engine and a GUI automation library with the functions defined to create a virtual chess player (a bot) on chess.com that is capable of playing against other players based only on the visual inputs, which I consider somewhat a success.
Credits and Bibliography

Python libraries used:

- openCV (image manipulations)
- stockfish (chess engine)
- pyautogui (GUI automation)
- numpy (array manipulations, mostly used to for colors’ values computation)
- collections.Counter (counting objects)

Harris Corner Detection tutorial

[https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html)

Hough Line transformation

[https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html](https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html)

Stockfish 10

[https://stockfishchess.org/](https://stockfishchess.org/)

OpenCV color conversions

[https://docs.opencv.org/3.3.0/de/d25/imgproc_color_conversions.html](https://docs.opencv.org/3.3.0/de/d25/imgproc_color_conversions.html)