REAL TIME TRACKING AND ANALYSIS OF PHYSICAL CHESS GAMES USING COMPUTER VISION AND MACHINE LEARNING

By

Adolfo Israel Bugarin

A capstone project submitted for Graduation with University Honors

May 20, 2024

University Honors
University of California, Riverside

APPROVED

Dr. Salman Asif
Department of Electrical and Computer Engineering

Dr. Richard Cardullo, Howard H Hays Jr. Chair
University Honors

ABSTRACT

This project develops a robust chess vision system capable of detecting and tracking a game of chess under a variety of lighting conditions and angles consistent with the abilities of a human. It includes a graphic that displays the current state of the chess game, the last move made, and the best move to make. It works using a single camera and a continuous video stream and uses machine learning for piece recognition. It is able to adapt to different viewing angles, changes in lighting, obstructions on the board, or unexpected adjustments or movements of pieces. However, unlike a human, it runs every move it detects through a state-of-the-art chess engine to determine the best possible move to make in response. This system can be implemented in a consumer smartphone app or be used during professional chess competitions as a much more economical alternative to electronic chess computer boards.

ACKNOWLEDGEMENTS

I would like to thank Dr. Salman Asif for being my faculty-mentor and providing me his support and assistance in completing my capstone. I would also like to extend a special thanks to Dr. Bree Lang for her involvement in the early stages of my capstone and her willingness to attend my capstone presentation. I would additionally like to acknowledge Dr. Richard Cardullo for making this capstone experience possible and working with me when complications arose during the development of my capstone.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Motivation

Chess has been a common subject in the area of computing since the founding of modern computer science. Starting with its initial digitization, computer scientists have constantly been improving the performance of computers at playing chess using artificial intelligence, which has seen tremendous improvements throughout the years. Thanks to their progressive efforts, computers today are so good at playing chess that they now outperform all humans at this game. However, modern computers still fall short of being the perfect chess players due to their inability to visually detect and track a game of chess as well or better than humans can.

This project aims at bridging this great divide. With this system, a computer is actually able to view and follow a physical chess game, allowing it to interpret the moves being made by a human player and generate its own moves, which the human player makes on behalf of the computer. This system can be implemented in a consumer smartphone app that provides real-time tracking and analysis of games, which includes move validation and computer suggested moves. It can also be used during professional chess competitions as a much more economical alternative to electronic chess computer boards, which normally cost hundreds to thousands of dollars. It can additionally be utilized in conjunction with a modern chess engine to create the ultimate chess playing robot, capable of defeating any human at this game.

1.2 Background

Currently available chess vision systems are limited in their functionality and usage in one way or another. This includes only working on 2D images or single pictures, requiring a

specific camera angle or lighting, utilizing special chess pieces or board markings, and producing poor accuracy and performance [Czyzewski et al. 2020, Koray and Sümer 2016]. For example Chessify, the most popular chess vision application available today, needs the user to upload an image or take a direct photo of the board to digitize the chess position. This is impractical to use during live chess matches, as the constant process of manually taking pictures and uploading them is time consuming and negatively impacts the quality of the game. Another example KnightVision, an advanced chess vision system that works using a live video stream, still requires the camera to be positioned at a certain angle and to be mounted on a tripod. Thus, no adequate, all-encompassing chess vision solution currently exists.

1.3 Limitations

Although the goal of this project was to create an original system that significantly improved upon existing implementations to create an all-encompassing chess vision solution, the final system did not reach such high ambitions. Due to countless complications and unforeseen setbacks and delays, it was deemed necessary to shift my efforts into improving and expanding an existing implementation to accomplish most of the goals of this project. This required thorough examination, analysis, and understanding of the processes, functions, and methodologies utilized by this existing system in order to properly modify, improve, and expand its functionality. My system is an improved and expanded version of an original program by Rondinelli Morais.

2. METHODS

2.1 Board Detection

To detect and produce a mapping of a board, this system has automated and improved the process used by Morais. This process heavily utilizes functions provided by the OpenCV library, a library of programming functions for real-time computer vision. This process consists of first converting the image to gray scale, smoothing the image using a Gaussian blur, and locating the biggest contour on the image. From this contour, Canny edge detection and a Hough transform is applied to create a grid of lines. Lattice points are subsequently determined and utilized to attempt to create a 9x9 matrix of points. If such a matrix is created, each 1x1 cell in the matrix corresponds to a square on the board, and the image is mapped accordingly. This results in a successful detection of a chess board. This process is depicted in Figure 1.

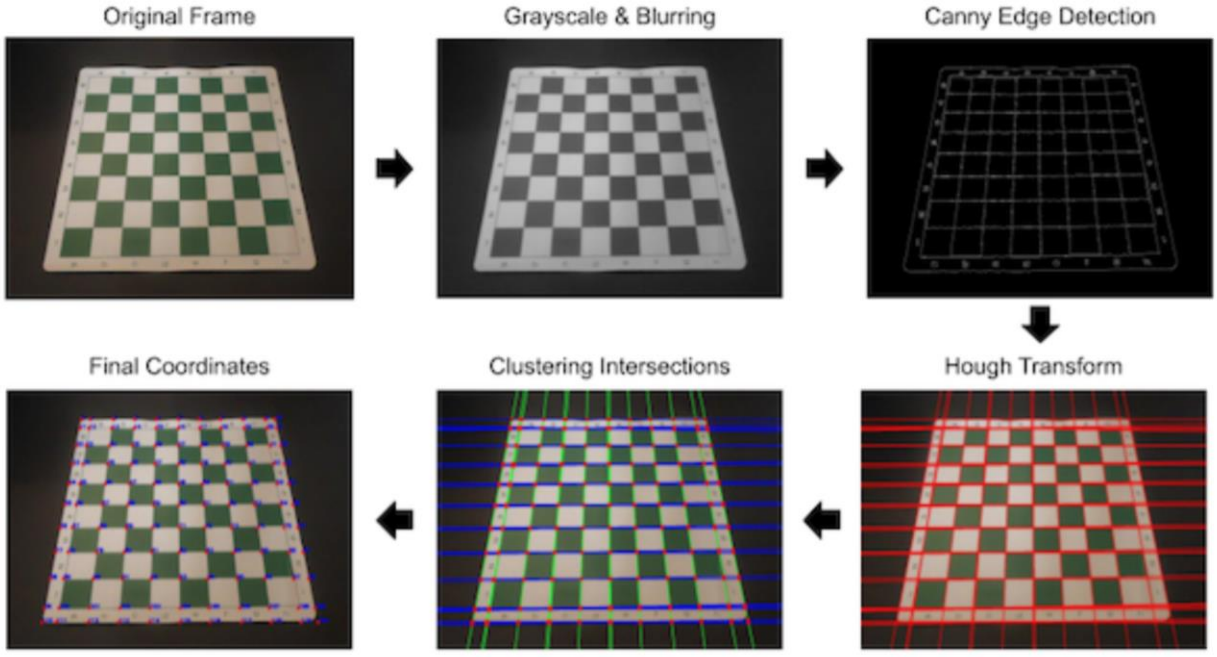

Figure 1: Process used to detect and produce a mapping of the board

This original board detection and mapping process needed to be manually called and was repeatedly prone to failure, resulting in a very slow and unintuitive process. This process was automated and improved by adding preemptive checks for correct degree number and convexity of the contour to the process to ensure that a contour incapable of being a chess board is not selected. The process was also thoroughly inspected and secured from failing due to uncaught exceptions or failed board mappings, which previously ended the board detection process without producing a mapping. These changes significantly sped up and improved the board detection process and increased its usability greatly.

2.2 Piece Recognition

Morais created and labeled a custom dataset of 258 images of different chess board configurations taken from directly above the board in a well-lit environment. A sample of some of the images in the dataset are in Figure 2. All these images use the same type of chess pieces and board. These images were used on the Darknet framework to train the YOLOv4 neural network to produce a model that recognizes chess pieces from images and classifies them into one of 12 classes depending on the piece color (black or white) and type (pawn, knight, bishop, rook, queen, and king).

However, because this model was trained on such specific examples, it only produced accurate recognitions and classifications of chess pieces of the same specific shape, color, lighting, and angle as the ones it was trained on. This made it nearly impossible to use the program to analyze a game of chess on a chess board with different pieces without training it on

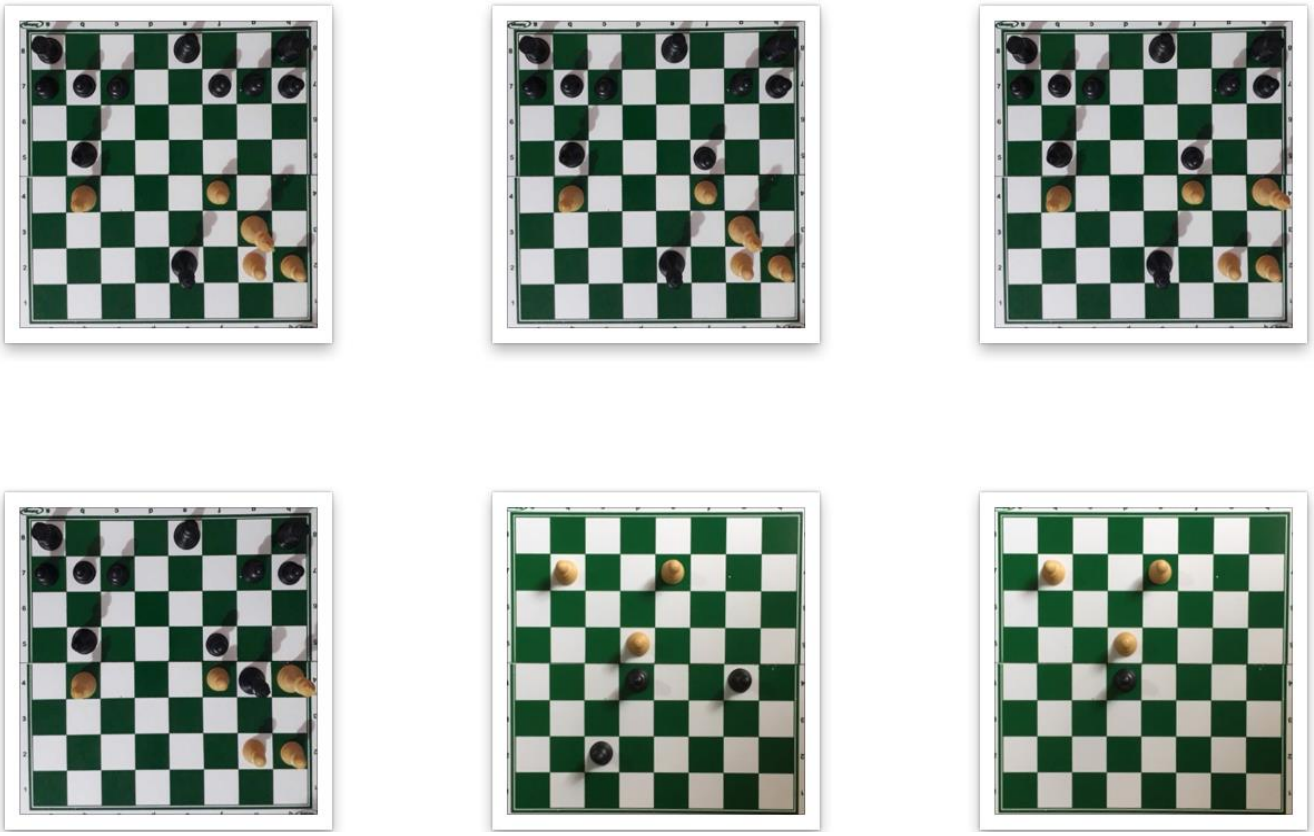

Figure 2: Sample of custom dataset

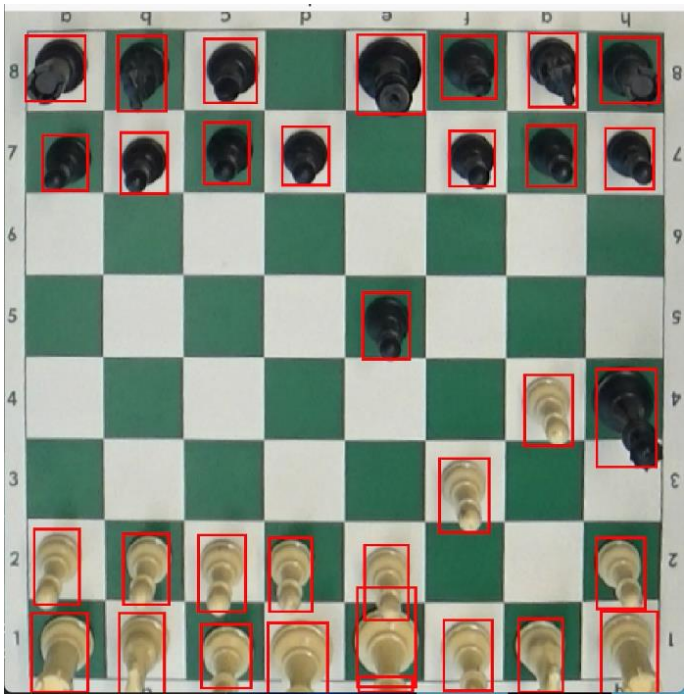

Figure 3: Chess piece recognition

a new model. It also failed to recognize chess pieces properly when lighting conditions changed. To solve these problems and make the program much more versatile, the classification was changed to be binary (2 classes) instead of the 12 classes for each piece type. The classification threshold was also lowered to allow recognitions of a greater variety of chess pieces. An example of chess piece recognitions in is Figure 3.

2.3 Hand Perception

The system scans the board continuously (every 800 ms), but only attempts to recognize chess pieces every time a hand detection cycle occurs, i.e., when it perceives human hand was present and then no longer present on the board. This typically signifies that a movement of chess pieces was completed. To perceive a hand, the system employs the method used by Morais, which also utilizes OpenCV functions. It first applies a layer of masks to the image to nullify both the squares and pieces from the board, then converts the image to grayscale and applies a series of erosions to the image. From this toned-down image, contours are attempted to be extracted. If a contour is found, this signifies that a human hand is present on the board. However, this method was heavily prone to false positives, so an extra layer of Gaussian blur was added to the image and the number of erosions and threshold was increased. This resulted in a reduced number of false positives but still sufficiently sensitive enough to detect a human hand on the board.

2.4 Move Construction and Validation

To construct and identify a move, the system combines the board mapping retrieved from the board detection process with the binary chess piece recognitions, along with the coordinates of each recognition. From this combination, each piece's square is ascertained and converted into an 8x8 binary representation of the board. Each time the system scans and recognizes chess pieces, a new 8x8 binary representation of the board is produced and saved for comparison. These representations are then compared to determine which squares have changed in value. If no squares have changed value, no move has been made. If more than two values or only one value has changed (with exception of a capture), this signifies that an invalid board operation has occurred and the board needs to be fixed. If only two values have changed (with exception of a capture), this signifies that a move has been made.

To identify the move that was made, it is first determined what type of changes occurred. A change of a square from a 0 (no piece present) to a 1 (piece present) signifies that a piece was added. The opposite case signifies that a piece was removed from the square. The addition of two pieces to the board is an invalid operation. The removal of two pieces from the board indicates a capture is taking place, and the subsequent move must only have an addition of a piece to the board. The last and most common case, one addition and one removal, indicates a regular chess move. The coordinates of the location and destination squares are determined from the row and column of the squares in the 8x8 representation, and these are compiled together as one move in coordinate notation.

To validate the move, the move is run through the Stockfish chess engine, which determines whether the move was legal or not. If it is legal, the virtual representation of the

board is updated accordingly. The Stockfish chess engine is also used to determine the best move to make, which is discussed in the following sub-section.

2.5 Best Move Determination

To determine the best move to make, every board configuration is run through a state-of-the-art chess engine. The chess engine utilized in this system is Stockfish, a free and open-source chess engine and the strongest CPU chess engine in the world. Stockfish uses a neural network board evaluation function and heuristic tree search to determine the best move to make with strategic understanding of the game. This determined move is then shown to the user as a computer suggested move.

3. RESULTS

3.1 Requirements

The chess vision system requires a live video feed with a clear, complete, and stationary view of the board. There must be sufficient lighting for pieces to be recognized. The camera needs to be positioned directly above or within 45˚ of the orthogonal to the plane of the board so that moves are detected properly.

For the system to run, the dependencies inside “requirements.txt” must be installed. This can be done by running the command “pip install -r requirements.txt” and only needs to be done the very first time the system is run. Once the dependencies are installed, the system can be started by running the command “python3 src/main.py —start” inside the root of the project.

3.2 Description of System Functionality

The system first prompts the user to input whether or not the board has padding present (y or n). It then continually scans video frames until a contour corresponding to a chess board is found, and attempts a mapping. If this mapping fails, the process is repeated until a successful mapping is found. Once a mapping is found, it is applied to every video frame received. This mapped image of the board is displayed to the user. The system then prompts the user to enter the number of 90˚ clockwise rotations to perform on the image to correct its rotation (0 - 3). For the system to work, the image needs to be oriented with white’s position on the bottom and black’s on top. Once a value is entered, the game will initiate and the initial game position is displayed on the game graphic (Figure 4, (a)).

Once the game initiates, the system scans the board to detect pieces. The system will rescan every time a hand is detected and removed. After the scan, if any missing or additional pieces are detected on the board, these are shown to the user on the game graphic as red squares. The message “Please fix board.” is displayed on top of the game graphic (Figure 4, (f)). If all the pieces detected are in their correct positions, the best move computed by the chess engine is shown as a green arrow. If a legal move was made, the game graphic will be updated accordingly and displayed as two green squares corresponding to the before and after location of the piece with respect to its move. The move will also be displayed underneath the board in coordinate notation (Figure 4, (b)). If an illegal move was detected, the game graphic will not update and two red squares will be displayed instead corresponding to the before and after location of the piece with respect to its illegal move. The message “Illegal move:” is displayed on top of the game graphic along with the illegal move in coordinate notation (Figure 4, (c)). If a legal move resulted in a check, the king who is in check will be highlighted in red on the game graphic (Figure 4, (e)).

To capture a piece, the user needs to first remove both the piece they are moving and the piece they are capturing from the board. If these two pieces are of opposing color, the game graphic will show these two squares as yellow and display “Capturing…” on top of the game graphic (Figure 4, (d)). The user then needs to put the piece they are moving on the board in the location of the piece they are capturing. The move will be subsequently registered as either a legal or illegal move in the same manner as all other moves.

If a move made by the user results in a promotion, the user is prompted to enter the promotion piece type (q for queen, r for rook, b for bishop, or k for knight). Once the type is entered, the game graphic is updated appropriately to reflect the piece chosen by the user.

Once the game ends, the appropriate game ending message will be displayed on top of the game graphic. If a checkmate, “Checkmate!” will be displayed along with the color that won (Figure 4, (e)). If the game resulted in a stalemate, “Stalemate!” will be displayed. No more moves will be registered on the board until the game is reset. The game can be reset at any time during or after the game by returning the pieces to their starting positions. To terminate the system, the user needs to simply close both the game graphic window and the window showing the video feed of the board.

3.3 Game Graphics

The game graphics were produced using the python-chess library and displayed using the matplotlib library. The usage of these libraries necessitated the reading of their documentation in order to determine which functions to use and understand how to use them. Figure 4 illustrates several examples of different scenarios and messages that can be displayed on the game graphic.

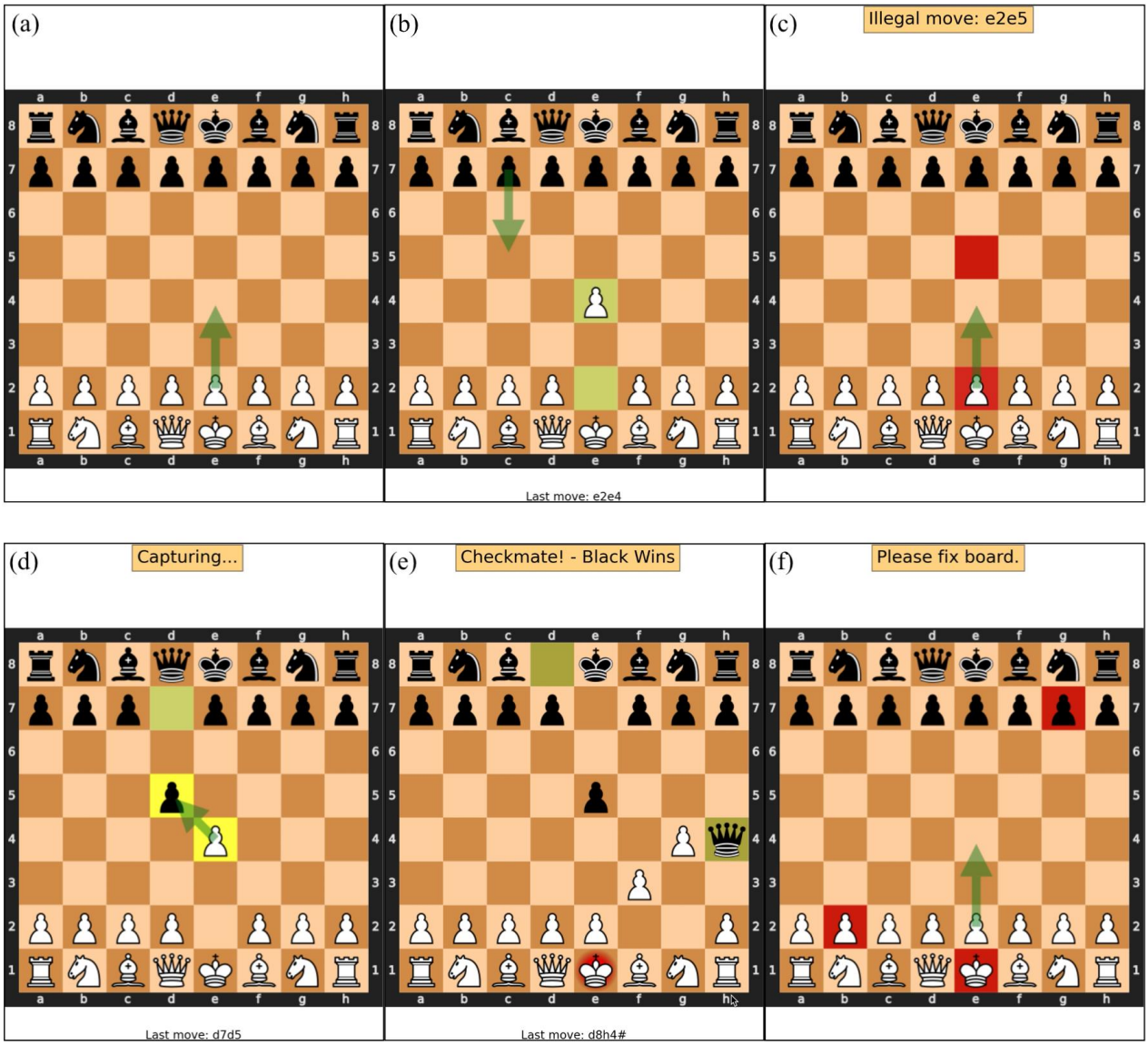

Figure 4: Examples of different game graphic scenarios: (a) Initial game position (b) User made legal move (c) User made illegal move (d) User is capturing a piece and removed both pieces from board (e) Game has ended in a checkmate (f) Initial game position has pieces missing

3.4 Further Improvements

The current system supports all legal chess moves with the exception of castling and en passant. Support for these moves will be added with future modification and expansion of the

move construction methods. The current system assumes that pieces displayed on the game graphic are the same pieces that are actually on the board, i.e., any chess piece can be used to represent any other chess piece. This assumption will be relaxed by using an unsupervised machine learning technique such as k-means clustering on the fly to separate the pieces into two classes, one for each color. This will allow the user to directly capture pieces on the board without having to remove both pieces from the board, and enforce the use of correct color pieces on the board by each player.

The current system requires the four corners of the board to be visible and for the user to enter whether or not padding is present on the board and to correct its rotation. It is also currently unable to find a board mapping or takes too long to do so when the background of the board contains multiple lines or too many objects. This will be resolved by applying the algorithm developed by Czyzewski et al. that digitizes chess board configurations with an accuracy of over 99.5%. The current hand perception method is also prone to false positives and negatives. This will be improved in the future by utilizing a supervised machine learning model to correctly identify human hands.

3.5 Project Links

- Link to Code: https://drive.google.com/drive/folders/1zR4FZk69F-FBT_9lnd5EjM2MkxAYdMOa?usp=sharing
- Link to Video Demos: https://drive.google.com/drive/folders/1xaCNKd-Z14J_xQDINW9GHcU_YyT7I0gg?usp=sharing

- Link to Original Program by Rondinelli Morais:

  https://github.com/rondinellimorais/chess_recognition?tab=readme-ov-file

4. REFERENCES

Czyzewski, Maciej A. et al. “Chessboard and Chess Piece Recognition With the Support of Neural Networks.” Foundations of Computing and Decision Sciences 45 (2020): 257 - 280.

Ding, Jialin. “ChessVision : Chess Board and Piece Recognition.” (2016).

Koray, Can and Emre Sümer. “A Computer Vision System for Chess Game Tracking.” (2016).

Omran, Alaa Hamza and Yaser M. Abid. “Design of smart chess board that can predict the next position based on FPGA.” Advances in Science, Technology and Engineering Systems Journal (2018): n. pag.

de Sá Delgado Neto, Afonso and Rafael Mendes Campello. “Chess Position Identification using Pieces Classification Based on Synthetic Images Generation and Deep Neural Network Fine-Tuning.” 2019 21st Symposium on Virtual and Augmented Reality (SVR) (2019): 152-160.

Christie, Dennis Aprilla, et al. “Chess Piece Movement Detection and Tracking, a Vision System Framework for Autonomous Chess Playing Robot.” 2017 Second International Conference on Informatics and Computing (ICIC), (2017): https://doi.org/10.1109/iac.2017.8280621.

Simonyan, K., & Zisserman, A. “Very Deep Convolutional Networks for Large-Scale Image Recognition. Computer Vision and Pattern Recognition.” (2014): http://export.arxiv.org/pdf/1409.1556.

Wölflein, Georg, and Ognjen Arandjelović. “Determining Chess Game State from an Image.” Journal of Imaging, vol. 7, no. 6, June 2021, p. 94. Crossref, (2021): https://doi.org/10.3390/jimaging7060094.

Game Image Recognition Using Neural Networks.” Medium, Towards Data Science, 22 Oct. 2020, (2020): https://towardsdatascience.com/board-game-image-recognition-using-neural-networks- 116fc876dafa.

B, Saurabh. “Convert a Physical Chessboard into a Digital One.” Bakken & Baeck Tech, 13 Nov. 2019, (2019): https://tech.bakkenbaeck.com/post/chessvision.

Orémuš, Zoltán. “Chess Position Recognition from a Photo.” Masaryk University, Master’s Thesis. (2018).

Menezes, Richardson & Peixoto, Helton. (2023). An Intelligent Chess Piece Detection Tool. 60-70. 10.5753/semish.2023.229800.

https://python-chess.readthedocs.io/en/latest/

https://pypi.org/project/stockfish/

"Wikipedia." Wikipedia, Wikimedia Foundation, www.wikipedia.org/.

Charlette, Stéphane. “Darknet/YOLO FAQ.” https://www.ccoderun.ca/programmingdarknet_faq/ (2024).

https://docs.opencv.org/3.4/index.html

https://stackoverflow.com

https://www.geeksforgeeks.org

https://computerchess.org.uk/ccrl/4040/index.html

https://stockfishchess.org/about/