CV Chess

From ESE205 Wiki

Link to weekly log

This is the page for the CV Chess project.

Contents

Project Overview

This project aims to use a camera, a raspberry pi, and computer vision software to recognize the movements of chess pieces in a game of chess. The final product will be able to recognize individual pieces, as well as determine the change in positions (squares) they occupy. This will ultimately yield a project that can verify valid moves, transcribe games, and perhaps implement an AI to act as an opponent of a lone player.

- Powerpoint: Link (https://drive.google.com/open?id=1YedAReM6O2aCmyuhbexrzyZo2OzDirnc)

Team Members

Robert Goodloe
Nhut Dang
TA: Ethan Shry
Instructor: Prof. Jim Feher

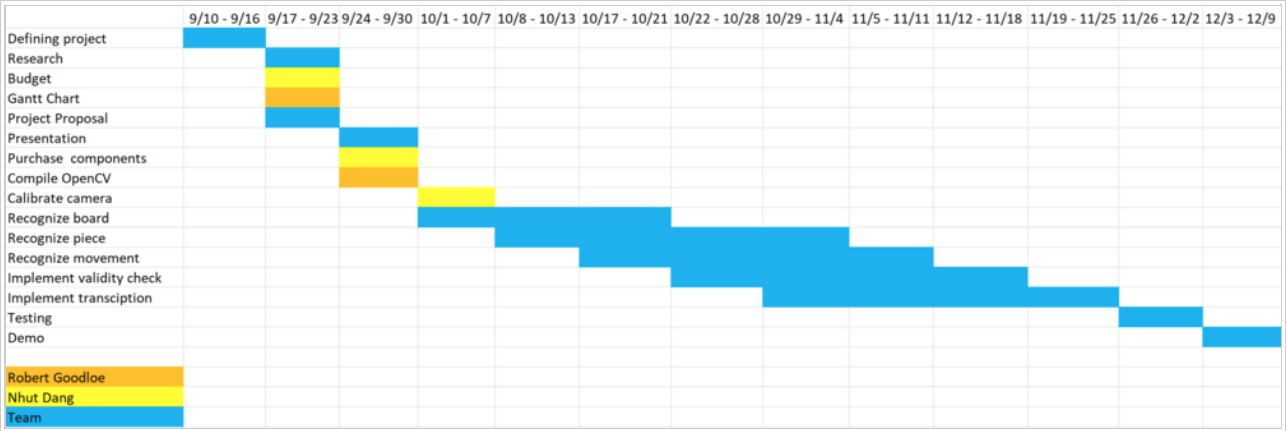
Objectives

1. Use OpenCV software to recognize chess board.
2. Use OpenCV software to recognize the chess pieces.
3. Use OpenCV software to recognize the movement of the pieces.
4. Transcribe game of chess and present in user-friendly fashion.
5. Add an AI component that responds to a users movements. It would display a move which the user must execute on behalf of the AI.

Challenges

1. Limited knowledge of Raspberry Pi
2. Zero knowledge using OpenCV or any computer vision software
3. Have been told that nobody has gotten OpenCV compiled and running in ESE 205 despite several attempts
4. Using object recognition to differentiate between similar pieces i.e. bishop versus pawn
5. testing

Gantt Chart



Gantt Chart

Budget

Item needed	Unit	Price	Description	Source Url
Chessboard	1	20.69	wood	Link (https://www.amazon.com/gp/product/B00GNG0BZE/ref=oh_aui_detailpage_o02_s00?ie=UTF8&psc=1)
Tripod	1	64.99	For mounting	Link (https://www.amazon.com/gp/product/B01LQX0P8Q/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1)
Pi touchscreen	1	0(own)	For displaying	
Spray paint	1	20	For identifying black and white pieces by maximize color different	
Raspberry Pi	1	0 (provided)	For programming	
Pi Camera	1	0 (provided)	For taking picture	
		105.68		

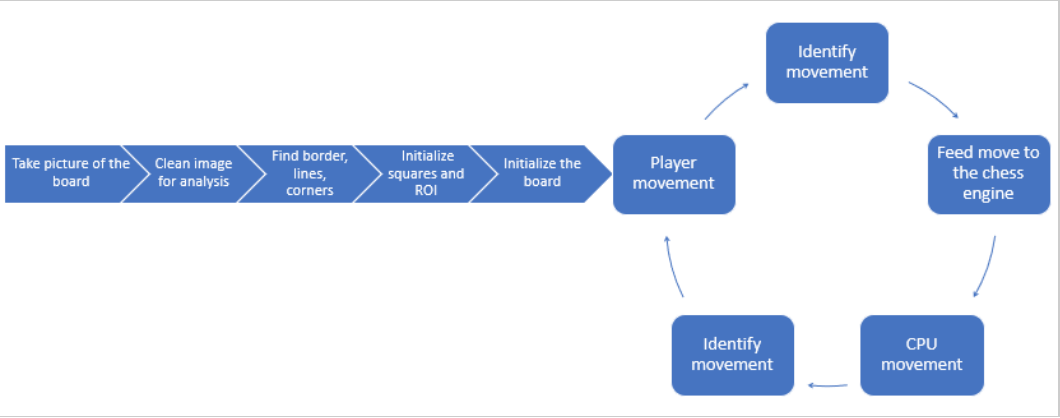
Design Solutions

Hardware

The project used a Raspberry Pi model 2b and an accompanying Raspberry Pi camera module. The camera was connected to the Pi using a 24-inch cable and mounted to the bottom of a tripod. The original design was intended to have a 3D printed tripod to mount the Pi camera, but this was changed. The tripod was able to stand directly above the chessboard and the camera directed straight down at the board managed to capture it in its entirety.

Software

The program was written in python and implemented using Object-oriented programming. The program had a graphical user interface created using the tkinter[1] library. The GUI allowed the user to start a new game, choose the difficulty of the CPU opponent, and choose which color they would like to play.

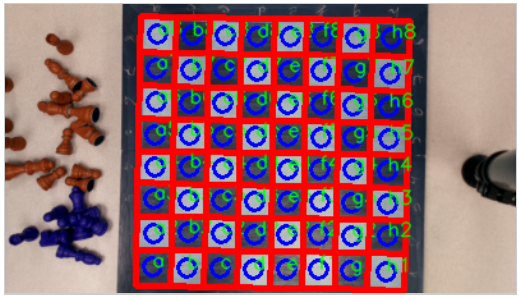


Program Flowchart

Board Recognition Algorithm[2]

- At the start of the program the GUI prompts the user to clear the board so that it may initialize the board. This process took a picture of the empty board and performed an analysis of the image in which it could properly map the chessboard. In doing so, the photo was first turned into a grayscale copy of itself.

Using the OpenCV library's built in algorithm Adaptive Thresholding (https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html)[3], pixels above a threshold rgb color value were turned white and those below turned black. This binarization of the image allowed for contours (shapes) within the image to be more easily found. The largest of these contours found was the edge of the chessboard.



Board initialization

- Given the edge of the chessboard, a black copy of the image was created and the original image within the edge of the chessboard was written onto the black image. This effectively created a new image with the chessboard surround by black. This is important because it excludes any noise that may interfere with the following OpenCV algorithms.
- Next, using the OpenCV Canny Edge Detection (https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html)[4] , the edges of the chessboard squares were inferred from the sharpest gradient of the color values in the image.
- Given these edges, the Hough Line Transform (https://docs.opencv.org/3.4.0/d9/db0/tutorial_hough_lines.html)[5] was able to infer the lines in the image, mapping a grid onto the image. The corners of the squares were determined by the intersection of the vertical and horizontal lines. After removing duplicates and sorting the corners by both row and column, the squares were created. Using these squares, the program was able to initialize a board that would hold much of the information about the game. before explaining the piece movement detection algorithm, it is best to describe the objects used by the program and the information that they stored.
  - For more instruction on Open CV, visit our tutorial:Open CV (https://classes.engineering.wustl.edu/ese205/core/index.php?title=OpenCV4#Capturing_image)

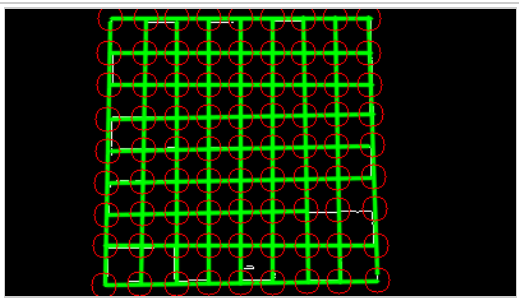
Objects

- Camera[6]:The camera takes pictures.
  - For more instruction on pi camera, visit our tutorial:Open CV (https://classes.engineering.wustl.edu/ese205/core/index.php?title=OpenCV4#Capturing_image)

- Line[7]: Line objects held the lines endpoint x and y values as well as the orientation of the line (vertical or horizontal).

Chessboard

- Square[8]:
  - These Square objects held much of the information used throughout the entirety of a program during a game of chess. The Squares stored their standard algebraic notation (i.e. a1, a2,...) as a variable called position. A square object also stored the piece located within in it or empty if it was empty. This variable called the state held a character representing the piece.
  - Capital letters for white (R N B Q K B N R/ P P P P P P P P), lower case for black (r n b q k b n r/ p p p p p p p p) and and period (.) for an empty square. The Squares had a circle of radius 7 at their center, in which the color value of the square was found. This circle is called the region of interest or ROI. The Squares stored the empty color value, found when the board was cleared for initialization, and their current color values throughout the game.



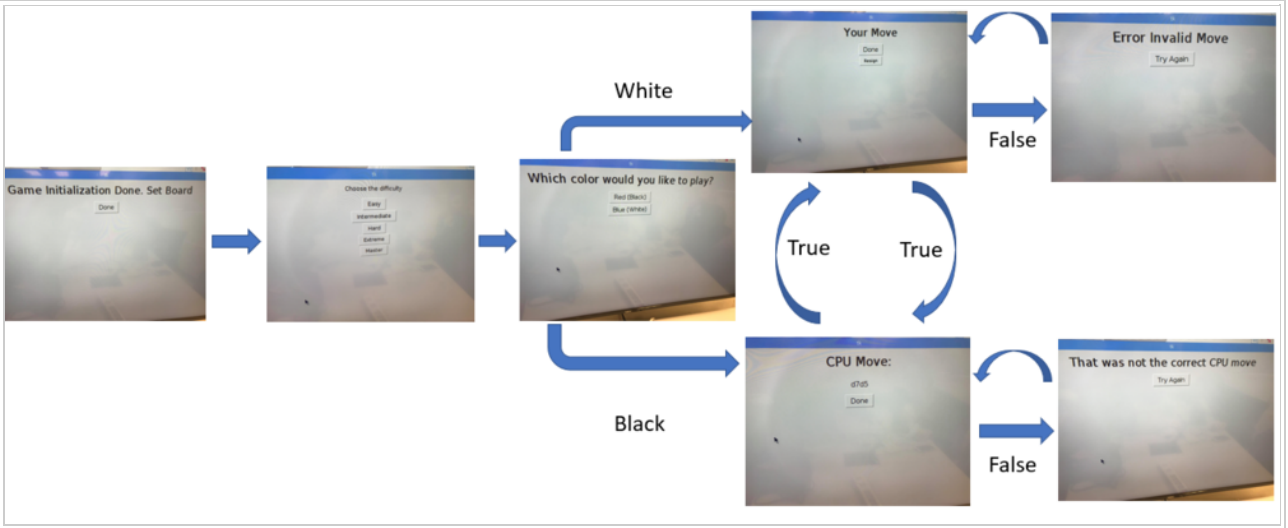
x,y lines and Corners

- Board[9]:
  - The Board objects stored all of the squares in the board. It assigned the squares' states. It is also where the piece movement detection algorithm was written.

- Chess Engine[10]:
  - The chess engine class imports the chess module from the python-chess library which can be found at Pychess (https://pypi.org/project/python-chess/)[11]. The Chess Engine class instantiates a local chess engine using the open source chess engine Stockfish (https://stockfishchess.org/)[12]. The class communicates with the chess engine using an open communication protocol that allows chess engines to interact with user interfaces. It is called the Universal Chess Interface (UCI) (https://python-chess.readthedocs.io/en/latest/uci.html)[13].
  - The Chess Engine class holds an instance of the chess board that can be updated when pieces are moved. The methods in this class include updating the board after a player move and getting a move from the CPU. It also notifies the user when they make an invalid move or they are in check/checkmate/stalemate...

- Game[14]:
  - The Game class is the main bridge between the user interface and the underlying image/game analysis. It contains the Board, the Camera, the Chess Engine, and two images for image analysis (before a move, and after). This class calls the Board initialization. It houses the Player Move method and the CPU method, handling image updating and calling other classes for image analysis as needed.

GUI[15]



GUI flowchart

- If the player choose white, player move first; if the player choose black, CPU move first.
- The code will check for error in every move. If there's an error, an error GUI will pop up, preventing the player to move to the next phase, and ask the player to make an valid move. The flow from player move to CPU move and vice versa will continue when the player make a valid move.

Piece Movement Detection Algorithm

- This algorithm requires two photos: one before a move and one after. So, after a move is made, the user hits 'Done', telling the Camera to take a picture which is passed to the algorithm.
- Next, the program iterates through all of the squares on the board, determining the change in color value from the previous photo to the current photo. It does so using the distance formula for each color value. In a typical move, two squares have the largest change in color, the square from which a piece moved and the square to which it moved. A similar color differencing is done on these two squares, however this time their color value is compared to their empty color value. Whichever square has a color value closer to its empty square color is now empty. Consequently, the move can be determined to be the position of the now empty square to the other square ('a1a2').
- The states of these squares are updated appropriately, the occupied square is given the previous state of the now empty square and the empty square is now set to empty. There are a few special cases for piece movements:
  - The first is castling. In castling, both the king and the rook move. This leads the algorithm to find more than two squres changing, in fact four squares change. In this case, the algorithm ensures that it is the king and rook moving, rather than two other pieces, for which it would throw an error. It then determines whether the castling takes place on the short side (king side) or long side (queen side). The move returned for castling as per the UCI protocol is the movement of the King so an example might be 'e1g1' (white short side castling).
  - Another special case is pawn promotion. In the game of chess, when a pawn reaches the other end of the board, the player is offered to promote that pawn to any piece of their liking. While it is common to simply promote it to a queen, one should have the ability to choose any piece. So, when detected a GUI window pops up asking the user to choose which piece to which they would like to promote. As per UCI protocol, the character associated with their choice is appended to the end of the move. For example, a move 'e7e8' with the choice of promoting to a queen becomes 'e7e8q'.


Castling

Tutorial

Follow this link to check out our tutorial[16]:Open CV (https://classes.engineering.wustl.edu/ese205/core/index.php?title=OpenCV4#Capturing_image)

Source Code and CAD files

Code

Open CV chess code: Github[17]

CAD File

Raspberry pi camera case:Thingverse[18]

Results

The product works:

1. Use OpenCV software to recognize chess board : Check
2. Use OpenCV software to recognize the chess pieces : Check
3. Use OpenCV software to recognize the movement of the pieces: Check
4. Transcribe game of chess and present in user-friendly fashion: Check
5. Add an AI component that responds to a users movements(the user must execute on behalf of the AI): Check

Shortcomings

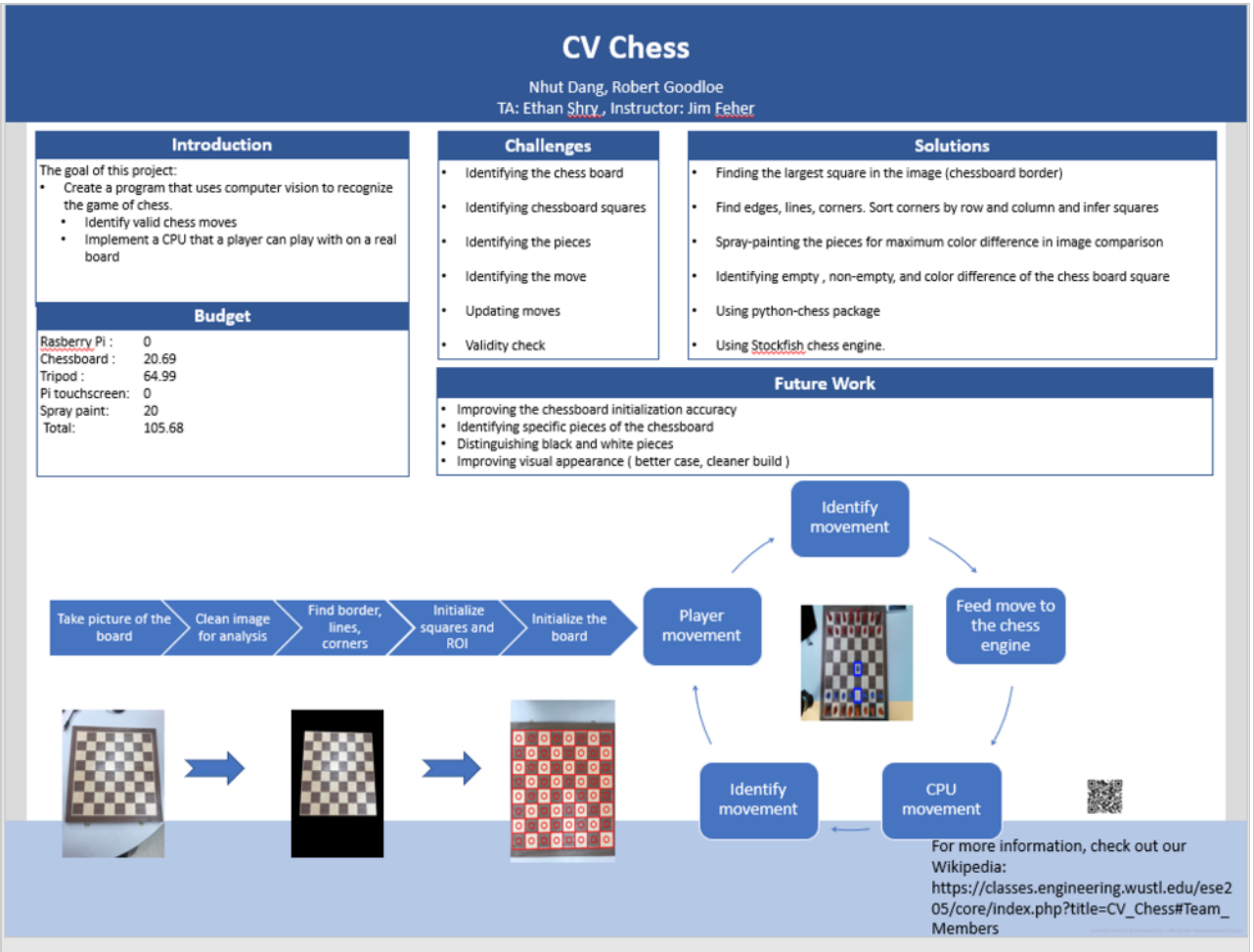
The pi case was not included in the project. Having a pi case will improve the aesthetic of the project.

The pawn promotion has not been tested out in the demo day as no player has been able to promote their pawn.

The shadow of people walking around causing the code to produce wrong movement outputs.

Poster

Product



Poster

Future Considerations

- 3D-print raspberry pi case is necessary to improve the appearance of the projects. There should also be a special mark which will indicate which location of the chessboard will work best for the camera to recognize the chessboard and its corners. A fixed 3D printed camera stand can potentially be one of the solution as the stand will allow the camera to have fixed camera focus.

- The code for board recognition definitely need some improvement for better performance as it has difficulty recognizing the chessboard if the chess board does not place in the middle of the focus.
- `sleep` may be useful when taking a picture as it will eliminate the double clicking problem (taking the picture twice) since it will recreate some delay before taking another pictures.
- There should be lighting directly above the chessboard so that code can run better due to less shadow which cause a problem in the color difference algorithm

References

1. Tkinter:Link (https://docs.python.org/3/library/tkinter.html)
2. Board Recognition:Link (https://github.com/rjgoodloe/ESE205-CVChess/blob/master/Board.py)
3. Adaptive Thresholding :Link (https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html)
4. Canny Edge Detection: Canny Edge Detection (https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html)
5. Hough Line Transform:Link (https://docs.opencv.org/3.4.0/d9/db0/tutorial_hough_lines.html)
6. Camera:Link (https://github.com/rjgoodloe/ESE205-CVChess/blob/master/Camera.py)
7. Line:Link (https://github.com/rjgoodloe/ESE205-CVChess/blob/master/Line.py)
8. Square:Link (https://github.com/rjgoodloe/ESE205-CVChess/blob/master/Square.py)
9. Board:Link (https://github.com/rjgoodloe/ESE205-CVChess/blob/master/Board.py)
10. Chess Engine:Link (https://github.com/rjgoodloe/ESE205-CVChess/blob/master/ChessEng.py)
11. Pychess: Link (https://pypi.org/project/python-chess/)
12. Stockfish:Stockfish (https://stockfishchess.org/)
13. UCI: Link (https://python-chess.readthedocs.io/en/latest/uci.html)
14. Game:Link (https://github.com/rjgoodloe/ESE205-CVChess/blob/master/Game.py)
15. GUI:Link (https://github.com/rjgoodloe/ESE205-CVChess/blob/master/OOPGUI.py)
16. Open CV tutorial:Tutorial (https://classes.engineering.wustl.edu/ese205/core/index.php?title=OpenCV4#Capturing_image)
17. Github: Github (https://github.com/rjgoodloe/ESE205-CVChess/)
18. Thingverse:Thingverse (https://www.thingiverse.com/thing:92208/)