ECE 445

Senior Design Laboratory

Final Report

AI Chess Robot with Computer Vision

Team 33

Zack Alonzo (zalonzo2)

Jose Flores (joseaf3)

Joshua Hur (jhur22)

TA: Zicheng Ma (zicheng5)

May 1st, 2024

Abstract

This final report describes the problem we proposed, the solution developed, and the implementation for ECE 445. We provided detailed designs, features, and cost

1. Introduction

1.1 Problem

Our project's goal is to address the need for a tangible and interactive chess-playing device, enabling users to play in the physical world against a chess AI rather than relying on digital platforms. Designed for both beginners and advanced players, the chess-playing robot would provide an engaging alternative to mobile apps, allowing for skill development and strategic thinking in a hands-on manner.

1.2 Solution

We plan to develop an autonomous chess-playing robot that eliminates the need for a human opponent by incorporating a chess algorithm with varying difficulty levels. Using a system involving a magnet and motors beneath the board, the computer opponent's chess pieces will move autonomously while the human player will simply pick up and place their pieces. Then, our robot will analyze the current board position by capturing an image through a camera and will identify all the pieces on the board by identifying each piece's color, associating it with the corresponding chess piece. With this updated board, we will now be able to determine the optimal move based on the chosen difficulty level and current board position. When identified, our code will output the necessary information to the system with the magnet and the motors underneath the board to move its intended piece and wait for the subsequent human player's move (additionally, a button press will "submit" the player's move).

1.3 Visual Aids

We provide a high-level overview of the solution in Figure 1. The Raspberry PI camera will take an image and send it to the Raspberry PI where it will undergo image processing to determine chess pieces and positions. Then, it will inform the ESP32 microcontroller to move the stepper motors and toggle on and off the electromagnet as appropriate to execute the chess move. Figure 2 is a physical project from the ECE Machine Shop that we planned to repurpose.

1.4 High-Level Requirements

During the demo with Professor Viktor Gruev and TAs, we met the following goals:

- Computer vision algorithm correctly identifies chess piece positions and their identity on the board with 95% ± 5% accuracy.
- Chess AI is implemented in a way that is able to identify when the human player has cheated with 95% ± 5% accuracy.

- Rail and magnet system grabs the intended chess piece to the intended location on the chess board with 95% ± 5% accuracy.

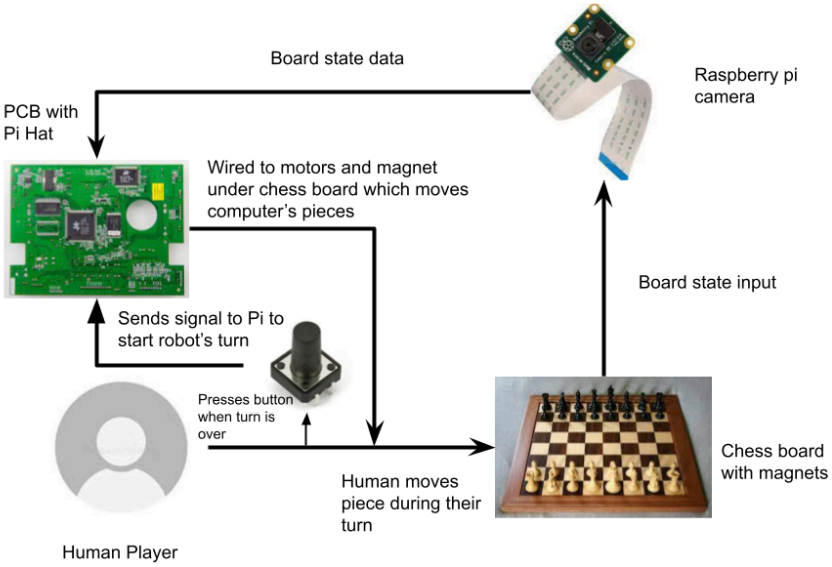

Figure 1. High-level project overview

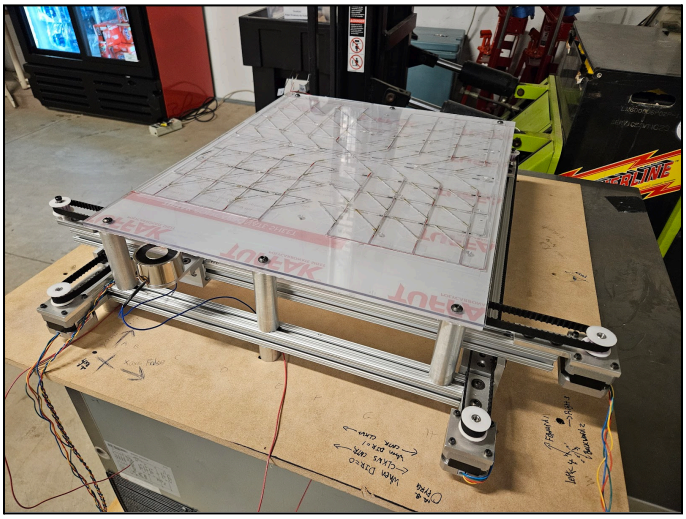

Figure 2. Physical project from ECE Machine Shop

2. Design

2.1 Block Diagram

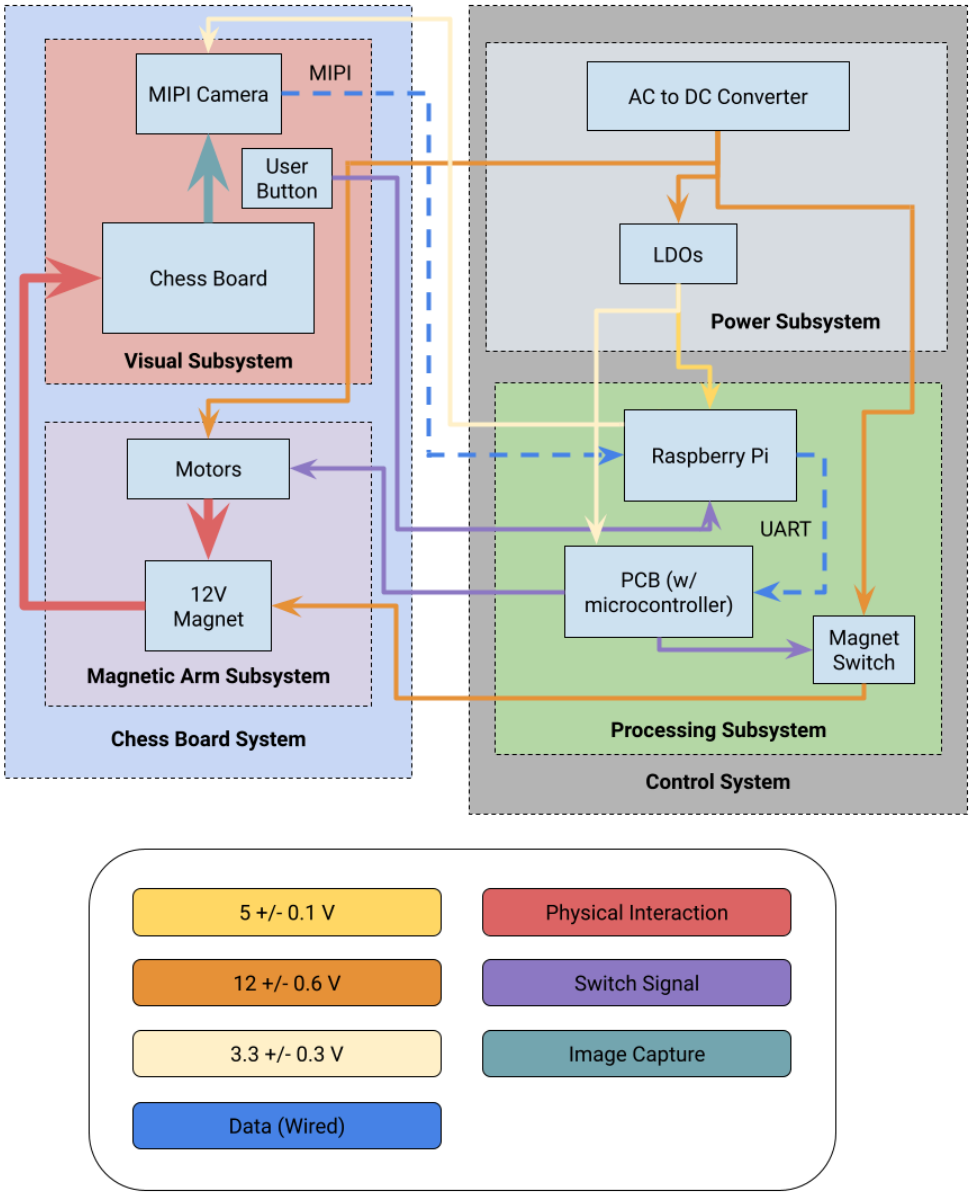

Figure 3. Block diagram of the chess-playing robot

Our design consists of four subsystems, composed of both hardware and software aspects. These subsystems are visually shown in Figure 3.

2.2 Power Subsystem

2.2.1 Overview

The power subsystem is responsible for powering all electrical and mechanical pieces associated with our project, such as the stepper motors, electromagnet, MIPI camera, Raspberry PI 4, and ESP32 microcontroller. It is comprised of:

- (1) 12.0 V ± 0.5 V Barrel Jack
- (1) 5.0 V ± 0.5 V Barrel Jack
- (1) 5.0 V to 3.3 V Linear Regulator

The main sources of power will come from a wall outlet where the AC to DC converters will output the 12 V and 5 V we need to power our project. The 12 V output will be connected straight to the BJTs in parallel that lead to the electromagnet and stepper motors, and the 5 V connected to the 3.3 V linear regulator to power the ESP32 microcontroller and stepper motor drivers; the Raspberry PI 4 connected with a 5 V, 2.1 A power adapter.

2.2.2 Design Decisions

We chose two AC to DC barrel jacks to supply power to our project because we did not consider efficiency as a part of our criteria. If we were to target efficiency, buck converters would be desirable as they could handle the power constraints required by the subsystem and are highly efficient in stepping down voltage. However, due to cost constraints and most importantly the complexity and scale of buck converters, we decided to not use them. The 3.3 V linear regulator was chosen because we did not need large amounts of current and because of its simplicity. Being a single chip to solder onto the PCB board, it reduces space and work costs to have it.

All components connected to the power sources needed to be limited within certain limits of voltage and current. As a result, we needed to verify the actual voltage and current that flows through the PCB board to their respective components. The methods and criteria we wanted to meet are in Table 1 where we describe how we measured and verified the metrics.

2.2.3 Power Subsystem RV Table

Requirements	Verification
● Maintain 12 ± 0.6 V from the AC/DC output	● Use an oscilloscope or multimeter and place the positive side terminal on the output of the AC/DC converter ● Next, place the negative side on the ground terminal of the pcb. ● Read the device display and verify that the output is within the expected ranges
● Maintain 5 ± 0.6 V from the AC/DC output	● Use an oscilloscope or multimeter and place the positive side terminal on the output of the AC/DC converter ● Next, place the negative side on the ground terminal of the pcb. ● Read the device display and verify that the output is within the expected ranges
● Maintain 3.3 ± 0.6 V from the linear regulator	● Use an oscilloscope or multimeter and place the positive side terminal on the output of the AC/DC converter ● Next, place the negative side on the ground terminal of the pcb. ● Read the device display and verify that the output is within the expected ranges

Table 1. Power subsystem RV table

2.3 Processing Subsystem

2.3.1 Overview

The processing subsystem is where all of the image processing, data analysis, and path planning, takes place. It is comprised of:

- (1) Raspberry PI 4 Model B
- (1) ESP32 S3 Microcontroller
- (2) BJT (Magnet Switch)
- (2) Red LEDs

The subsystem is split up between the 2 components, one is the Raspberry Pi, which handles the computer vision code, the python library used for the chess AI, and the path planning code used to find the best path for the chess piece to traverse through. The microcontroller is the second component which handles the execution of moves by moving the 3 stepper motors to the correct locations as indicated by the path found. These 2 devices communicate through serial via a female USB-A port on the Raspberry PI and a female Micro-USB on the ESP32 devkit connected to a USB-to-UART bridge also on the devkit. The Raspberry Pi is connected to an external camera to receive the images, and the microcontroller is connected to the motor drivers and magnet in order to execute the moves. Whenever a piece needs to be moved, the magnet will turn on by sending current straight from the power input. Whenever we need to turn the magnet off, a switch will cut off the current to the magnet which is controlled by the microcontroller.

2.3.2 Design Decisions

We chose a Raspberry PI 4 Model B to handle the computationally expensive operations since it was the most cost effective single board computer for the computational speed it provided. We chose the ESP32 as the microcontroller we used due to it also being the most powerful microcontroller we could use that was provided to us for free.

Originally, we had planned to use a N-Channel MOSFET as our magnet switch. However, this did not turn out well and we damaged the MOSFET while adding it to our circuit. Luckily, Jason, the head TA, engineered a solution using 2 BJTs in parallel which had the functionality we wanted for our magnet switch and we were able to move forwards with our project.

Lastly, we used the OpenCV and EasyOCR libraries to help us write the computer vision code to detect the chess board and the pieces. We originally were going to distinguish between the chess pieces solely on color instead of OCR, but we ran into trouble with distinguishing between the black and white pieces as the glare on the plexiglass from the lights ruined our original plan for this detection (since we were originally going to detect balance between the number of black and white pixels in the square). A very detailed and in-depth explanation of the design of the computer vision code can be found in Appendix D.1.

2.3.4 Processing Subsystem RV Table

Requirements	Verification
● Computer vision algorithm correctly identifies chess piece positions and their identity on the board with 95 ± 5% accuracy (results can be found in Appendix D.2)	● Randomly generate a chess board state using an online tool and place the pieces as shown. ● Once set up, press the button to signal to the algorithm to capture a screenshot of the board and begin processing. ● Check the internal representation of the chessboard and ensure that it is correct by Forsyth-Edwards Notation (FEN) standards. ● Repeat 100 times and ensure that the accuracy fall within expected performance
● Microcontroller plans a path to move the necessary pieces and executes it with 95 ± 5% success rate	● Play chess games against the robot and keep track of its successes and failures. ● Continue to play until the robot reaches 100 moves performed and count up all the robot's successes. Ensure that its accuracy falls within expected performance.

Table 2. Processing subsystem RV table

2.4 Visual Subsystem

2.4.1 Overview

The visual subsystem serves as the part of the project that the human interacts with and the part where the AI receives its visual input to the algorithm via the camera. The system is comprised of:

- (1) Chess Board created by machine shop

- (1) Arducam for Raspberry PI
- (32) Colored chess pieces
- (1) User controlled button

The chess board is provided by the machine shop, and it includes the magnetic arm subsystem underneath the board. Additionally, there is a camera that is hung above the board looking down that will be used as input to the Raspberry PI. What the camera sees is what our image processing code will work on and send to the chess AI. There is a button that the user will press at the end of every turn to signal that it is the end of their turn and the robot will begin analyzing the board. This will loop until a stalemate or victory.

2.4.2 Design Decisions

We initially chose to process chess pieces with OpenCV's HSV color manipulations to view and determine piece locations. However, due to issues with the library and identification difficulties, we decided to use OCR. Because of the computer vision change, we decided to change from six colors, each side having their lighter and darker colors, for the chess pieces to white and black pieces with their chess piece type printed on the top of them. This allowed OCR to recognize the piece's type and color. For the button, we decided to utilize the keyboard connected to the Raspberry PI to inform the computer side that the human player finished their turn because of its simplicity.

For the chess board, it was returned by the ECE Machine Shop after a couple of modification requests. One is that the chess board had to span the range of the magnetic arm subsystem which is 23x23 inches. The Arducam needs to view the entire chess board so the Machine Shop constructed an adjustable arm to house the camera. In Figure 4, the top is the Arducam held onto by the adjustable arm, chess board, and the magnetic arm subsystem below the board. One last issue with the chess board was by itself, the middle of the board would sink downwards causing increased friction with the magnetic arm subsystem. To fix this problem, we inserted paper wedges underneath and above the bolts to induce a force to push the middle of the board upwards to counteract the downward bow. In Figure 4, this is seen as the white paper inserts in the four corners and two sides of the chess board.

We followed a series of steps to check and verify that the visual subsystem was working as intended. These steps and verifications can be seen in Table 3.

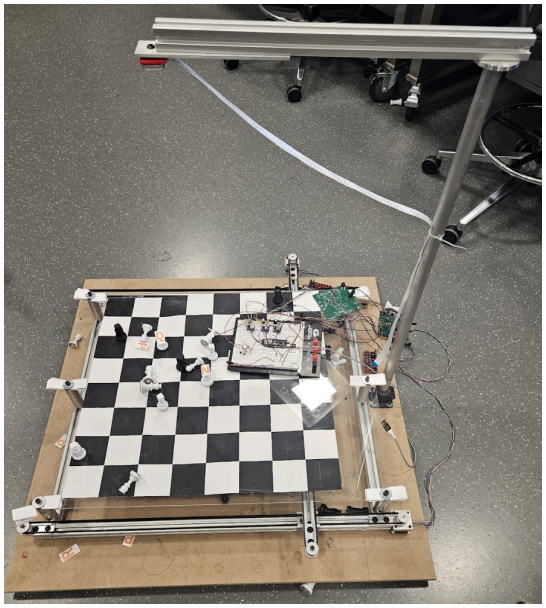

Figure 4. Finished chess board by ECE Machine Shop

2.4.3 Visual Subsystem RV Table

Requirements	Verification
• Raspberry Pi can recognize the camera connection	• Plug the two device into each other • Check the Raspberry Pi's connection recognition • If it does not recognize the camera, troubleshoot with a Raspberry Pi/MIPI camera manual
• Camera only sees the board to avoid distractions	• Mount the camera to the centering apparatus • Output what the camera sees onto a separate window or jpg/png • Adjust the height of the apparatus accordingly until the outer edges of the camera sees the outer border of the chess board
• Pictures of the board are able to be sent to the Raspberry Pi	• Power the Raspberry Pi and connect the camera to the Pi • Once powered, check the read value from the Raspberry Pi

Table 3. Visual subsystem RV table

2.5 Magnetic Arm Subsystem

2.5.1 Overview

The magnetic arm subsystem receives data from the processing subsystem which tells it where to move its motors and when to turn the magnet off or on in order to grab and move the AI's pieces effectively. It is comprised of:

- (3) Mercury Motor SM-42BYG011-25 2 Phase 1.8° 32/20
- (1) KK-P35/30 30 kg Electromagnet

There are two motors in parallel to each other and dedicated to operating in the same direction along the Y axis of the board. They are responsible for moving the third motor along the X axis of the board which rests perpendicular to the other two motors. This allows the rail system to move in 4-directions. The electromagnet rests on the third motor's axis and it is fed voltage via the BJTs to turn on and off. The magnet picks up and drops off pieces by attracting the magnetic washers that are secured underneath each chess piece. To allow special movements from the knight chess pieces, the magnet can drag the pieces along the lines of the chess board to maneuver around them.

2.5.2 Design Decisions

The original dimensions of the chess board provided to us by the machine shop was 18 x 19 inches with the magnetic arm rail system having a reach of about 18 x 18 inches. These dimensions were too small for our idea of moving the magnet along the lines of the chess board. We requested the board and reach of the magnetic arm to be 24 x 24 inches then decided to make the playing area 23 x 23 inches to give an inch of extra space around the edge for the camera detection and to have an area to move captured pieces to. We also decided to downsize our magnet to a smaller one that is about half the diameter of the original magnet to help with the problem of the arm accidentally grabbing unwanted pieces on its way to the destination.

2.5.3 Magnetic Arm Subsystem RV Table

Requirements	Verification
● Rail system can move to specific chess board positions with 95 ± 5% accuracy	● First verify that it can move to specified chess board positions with the processing subsystem's software which will be similar to computer numerical controller (CNC) ● Then, generalize the chess positions with a random number generator ● Run the program for 100 trials

• Rail system can move from one chess tile to another while holding onto a chess piece without bumping into other pieces with 95 ± 5% accuracy	• Use the program from the magnetic arm subsystem's first verification process, but power the magnet when it travels from one tile to another • Have a chess piece over the magnet to be dragged around, and place pieces around the board to test for bumps • Run it for 100 trials
• Magnet will grab and hold on to desired chess piece with 95 ± 5% success rate	• Use the program from the magnetic arm subsystem's first verification process, but power the magnet while it travels from one tile to another • Have a chess piece over the magnet to be dragged around • Run it for 100 trials

Table 4 . Magnetic arm subsystem RV table

2.6 Physical Design

Our design used one ⅛ inches plexiglass to act as the chess board and acrylonitrile butadiene styrene for the chess pieces, which is environmentally friendly and not toxic according to [6].

2.7 Tolerance

To calculate our tolerances, we need to decide on how we are going to power our chess-playing robot. From the power subsystem, we decided to use a 12 V and 5 V barrel jack alongside a 3.3 V linear regulator.

Other points of information to consider:

- Raspberry Pi 4 has a recommended input voltage of 5 V with a range of -0.5 to 6 V

	Voltage (V)	**Current (A)**	**Min Power (W)**	**Max Power (W)**
Stepper Motors	12	0.33	3.96	3.96
Electromagnet	12	0.83	9.96	9.96
Raspberry Pi 4	5 ± 0.1	3.0	14.7	15.3
ESP32 S3	3.3 ± 0.3	0.5	1.5	1.8

Arducam IMX219	3.0 ± 0.3	0.3	0.81	0.99

Table 5. Power Draw Calculation Table

As shown in Table 5, our maximum current draw will be 4.96 A. The 12 V barrel jack outputs 12 V and 5 A or 60 W and the 5 V barrel jack outputs 5 V, 3 A or 15 W. From the calculated tolerances, both the 12 V and 5 V have satisfactory amounts of power for the various components of the project.

Next, we will talk about the stepper motors. According to the datasheet for the Mercury Motor SM-42BYG011-25 2 Phase 1.8° 32/20, its time constant τ is inductance (*H*)/resistance(Ω).

$$Inductance\text{: } 46 \pm 9.2\ mH\ (36.8\ to\ 55.2\ mH) \quad (2.1)$$

$$Resistance\text{: } 34 \pm 3.4\ \Omega\ (30.6\ to\ 37.4\ \Omega) \quad (2.2)$$

$$\tau(electrical\ time\ constant) = \frac{inductance}{resistance} \quad (2.3)$$

$$\tau_{lower} = \frac{0.0368}{37.4} = 0.984\ ms \quad (2.4)$$

$$\tau_{upper} = \frac{0.0552}{30.6} = 1.804\ ms \quad (2.5)$$

This means it will charge up the coil to 63% of its rated value in 0.984 ms to 1.804 ms.

For comparisons, we viewed Sanyo Denki SS2422-5041 from [7] to observe the difference of capabilities.

$$Inductance\text{: } 2.9\ mH \quad (2.6)$$

$$Resistance\text{: } 5.4\ \Omega \quad (2.7)$$

$$\tau(electrical\ time\ constant) = \frac{inductance}{resistance} \quad (2.8)$$

$$\tau = \frac{0.0029}{5.4} = 0.537\ ms \quad (2.9)$$

$$Percent\ Change_{lower} = \left(\left|\frac{0.537}{1.8044}\right| - 1\right) * 100 = -70.23\% \quad (2.10)$$

$$Percent\ Change_{upper} = \left(\left|\frac{0.537}{0.984}\right| - 1\right) * 100 = -45.42\% \quad (2.11)$$

Comparing the time constant of our stepper versus the compared stepper, we see that the compared stepper has a time constant that is 57.83 ± 12.41% faster at charging its coil than ours. This difference does not matter for lower speed torques, as time constant does not matter as much for these values (“Stepper Motor Basics”). However, for higher speed torques, this does matter. Unfortunately, these are the stepper motors provided to us by the machine shop, and we do not have the budget for new ones, so our chess robot may move chess pieces slower than what is ideal.

Finally, we will talk about an analysis of our linear regulator. We want to make sure that our linear regulator, which is powering our stepper motors, is not overheating. We will be following the specifications provided in the linear regulator's data sheet ("LM3940"). Here is a schematic of the linear regulator:

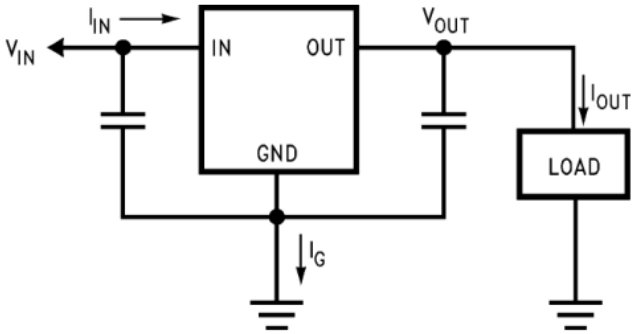

Figure 5. Linear regulator schematic

As preliminary knowledge, we chose the 3 A variant of the buck converter because the Raspberry Pi can draw a maximum of 3 A. Since the 5 V from the buck converter that goes to the Pi also goes to the linear regulator, our I_{In} = 3 A. Now, let's calculate the power dissipated by the regulator (Pd):

$$I_{Out} = 1 \text{ A} \quad (2.12)$$

$$I_{In} = 3 \text{ A} \quad (2.13)$$

$$V_{In} = 5 \pm 0.5 \text{ V (4.5 V to 5.5 V)} \quad (2.14)$$

$$V_{Out} = 3.3 \pm 0.099 \text{ V (3.201 V to 3.399 V)} \quad (2.15)$$

Lower Pd:

$$Iin = Iout + lg \quad (2.16)$$

$$Ig = Iin - Iout = 3 - 1 = 2 \quad (2.17)$$

$$Pd = (4.5 - 3.399) * 1 + 4.5 * 2 = 10.101 \quad (2.18)$$

Upper Pd:

$$Iin = Iout + Ig \quad (2.19)$$

$$Ig = Iin - Iout = 3 - 1 = 2 \quad (2.20)$$

$$Pd = (5.5 - 3.201) * 1 + 4.5 * 2 = 11.299 \quad (2.21)$$

Next, we will calculate TR(max), which is the maximum allowable temperature rise. We will do this using the formula:

$$TR(max) = TJ(max) - TA(max) \quad (2.22)$$

where TJ(max) is max ambient temperature (which we will assume to be 20°C in ECEB), and TA(max) is max allowable junction temperature, which for commercial grade parts is 125°C. Plugging into the formula, we get

$$TR(max) = 125 - 20 = 105°C \quad (2.23)$$

Lastly, we will use this formula to calculate the max allowable value for the junction-to-ambient thermal resistance:

$$R_{\theta(JA)} = \frac{T_R(max)}{P_D} \quad (2.24)$$

For lower Pd:

$$\frac{105}{10.101} = 10.400\frac{°C}{W} \quad (2.25)$$

For upper Pd:

$$\frac{105}{11.299} = 9.293\frac{°C}{W} \quad (2.26)$$

Both of these values are lower than 23.3°C/W, which is the max allowable value for the junction-to-ambient thermal resistance for the TO-220 package of the LDO, which is what we used. This means we will need a heatsink. However, the datasheet recommends we use either a standard heat sink or a copper plane on our PCB. Since we already have this, we should be dissipating the extra heat that the linear regulator is not able to dissipate, meaning our stepper motor drivers will be powered with 3.3V by a device that will not be overheating.

2.8 Schedule

Week	**Jobs**	**Person**
February 26th - March 3rd	Designed first PCB	Everyone
	Fix design document with feedback from design review	Josh, Jose
	Prototype 3D printed chess pieces with different sizes and different sized washers	Zack
	Order all parts needed for project including backup parts	Josh, Jose
March 4th - March 10th		
	Revise PCB if necessary	Everyone

	Begin coding an outline for interfacing our image processing code with Chess AI library	Zack
	Design instructions to send to microcontroller from Raspberry Pi (almost like Gcode for 3D printers)	Josh
	Paint prototyped chess pieces in preparation for testing computer vision once we get our parts/PCB	Zack
	Gather data on how to communicate between microcontroller and motors/magnet switch	Jose
	FIRST PCB ORDER MARCH 5TH	Everyone
March 11th - March 17th (Spring Break)	Continue tasks from previous week if necessary or get a head start on next week's tasks	Everyone
March 18th - March 24th	Finalize physical chess board setup (sheet on top of plexiglass)	Everyone
	3D print our finalized design for the 32 chess pieces, assemble with washers, and paint them	Zack
	Begin work on image processing code to identify chess pieces	Zack
	Begin work on chess AI implementation using the python-chess library.	Josh
	SECOND PCB ORDER MARCH 19TH	Everyone
March 25th - April 2nd	Begin working on programming microcontroller to decode Raspberry PI instructions to move motors and flip magnet switch	Jose
	Test and finalize the chess AI implementation using the python-chess library.	Josh
	Continue work on and test image processing code to identify chess pieces	Zack
	THIRD PCB ORDER MARCH 26TH	Everyone
April 3rd -April 9th	Added barrel jack and removed buck converter on PCB	Everyone
	Begin work on the image processing to chess AI pipeline	Josh

	Test and finalize programming microcontroller to decode Raspberry PI instructions to move motors and flip magnet switch	Jose
	Test and finalize work on image processing code to identify chess pieces	Zack
	FOURTH PCB ORDER APRIL 4TH	Everyone
April 10th - April 16th	Test and finalize the image processing to chess AI pipeline	Zack
	Full project testing	Everyone
	Made new PCB with new stepper motors	Jose
	FIFTH AND FINAL PCB ORDER APRIL 11TH	Everyone
April 17th - April 23rd	Integrated all individual parts and debugged each section	Everyone
April 24th - April 30th	Final Demo	Everyone

Table 6. Schedule of project

3. Cost Analysis

3.1 Parts/Materials

For this project, we made three PCB orders with stencils for each one. We used two PCBs combined with a breadboard for our final demo. One PCB supplied power to the second PCB which powered our microcontroller and 3.3 V level logic for cheat detection.

For a complete overview of the parts we used on our PCBs and breadboard, please refer to Appendix C.

3.2 Estimated Hours of Development

We estimate that we averaged 15 hours per week on the project and a total of 40 hours for the final week leading up to the demo. Therefore, we estimate 250 hours per person for a total of 750 hours.

3.3 External Materials and Resources

We obtained a couple of components from the following resources provided to us by the ECE 445 class.

For a detailed overview of the components obtained and used in our final design, please refer to Appendix C.

- Machine Shop
    - 24 x 24 inch homemade chess board with camera mount and magnet rail system
    - 32 Washers for chess set.
- ECE E-Shop
    - All SMD and through hole resistors on PCB and breadboard
    - Barrel Jacks
- Senior Design Lab
    - We utilized the soldering irons and PCB oven to bake our PCB.
    - We utilized specific equipment for testing in the Senior Design Lab including a multimeter, oscilloscope, and DC power supply.

3.4 Approximate Total Cost

For the labor expenses, the estimate we received from the machine shop was 60 hours at $50/hour, totalling $3000. For the labor costs of our group, the average starting salary for a computer engineer at UIUC is $109,176. At 40 hours a week, this comes to $52.49/hour. We estimated that the project will took 250 hours per group member so the total cost of our labor is 3 x 52.49 x 2.5 x 250 = $98,418.75.

In total, the cost of materials plus the cost of labor would be approximately:
Materials: $414.94
Labor: $98,418.75
Total cost = $98,833.69

4. Conclusion

4.1 Conclusion

We were able to successfully implement our proposed solution to our problem statement, and we were able to satisfy our three high-level requirements. It is able to move intended pieces to and from specified locations and locate and verify chess pieces both with at least 90% accuracy. Because the cheating algorithm for the chess AI was solely based on output from computer vision, if computer vision operated with at least 90% accuracy, then the chess AI's cheating algorithm could run with the same accuracy as the computer vision. All subsystems

were operating with intended functionality except the power subsystem. Originally, it was meant to run on a 5.0 V buck converter instead of a 5.0 V barrel jack and power adapter, but due to shipping delays for both components and PCB boards, we were unable to ultimately combine everything onto a PCB board in the end.

If we were able to complete this project again, equipped with present-day knowledge, we would begin on the PCB design immediately and verify its functionality and design with teaching assistants and professors to ensure its logical soundness. Once the PCB is correct and finalized, the parts need to be thoroughly investigated to ensure proper compatibility between the different components and modules as using incompatible components could lead to the usage of deprecated versions of software that may have their own share of problems.

4.2 Ethics and Safety

As we worked on the project, we carried out the ethical guidelines from COPE's code of ethics [1] which are not limited to:

- PRINCIPLES, Principle 1: PUBLIC, Section 1.01:
    - Accept full responsibility of the work
- PRINCIPLES, Principle 6: PROFESSION, Section 6.06:
    - Obey all laws governing the project and its components

For full details regarding ethical practices and the legal precautions, please refer to Appendix A.

Appendix A Ethics & Safety

A.1 Ethics

Our group works in accordance with the Committee on Professional Ethics (COPE) [1] and they promote ethical conduct amongst the computing professionals with a code of ethics. In accordance to their code of ethics, which include but not limited to:

- PRINCIPLES, Principle 1: PUBLIC, Section 1.01:
    - Accept full responsibility of the work
- PRINCIPLES, Principle 1: PUBLIC, Section 1.03:
    - Approve software if it does not diminish the quality of life, privacy, or harm the environment
- PRINCIPLES, Principle 1: PUBLIC, Section 1.06:
    - Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods and tools.
- PRINCIPLES, Principle 3: PRODUCT, Section 3.02:
    - Ensure proper and achievable goals and objectives for any project on which they work or propose.
- PRINCIPLES, Principle 5: MANAGEMENT, Section 5.01:
    - Ensure good management for any project on which they work, including effective procedures for promotion of quality and reduction of risk
- PRINCIPLES, Principle 6: PROFESSION, Section 6.06:
    - Obey all laws governing the project and its components

A.2 Legal Precautions

A.2.1 Python-Chess

Creating a chess algorithm from scratch to evaluate countless chess moves and how optimized they are for victory can be challenging and time-consuming; it may require time that is out of scope of a semester's worth of time. As a result, we will be assisted by Python's chess library "python-chess" [13] to compute moves and their varying efficiency. Because we are using a library, there is a need to be aware of the potential licensing conflicts. The library has a GPL v3 license which means that it can be involved in commercial use. In accordance to a GPL v3 license [2], but not limited to:

- Terms and Conditions, Section 4, Paragraph 2:
    - You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.
- Terms and Conditions, Section 7, Group C:
    - Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version.

- Terms and Conditions, Section 8, Paragraph 1:
    - You may not propagate or modify a covered work except as expressly provided under this License.

A.2.2 Raspberry PI

We are utilizing a Raspberry Pi to act as a microcontroller in our project's design and with it comes their terms for usages [4]. Under Raspberry Pi trademark rules and brand guidelines, they explicitly mention a list of allowances and prohibitions that deal with Raspberry Pi and all they own. If our project ever decides to commercialize, we will need to contact them to obtain a license. We can use the Raspberry Pi's word mark to refer to products or services, or to describe that there is compatibility between products. We cannot use the logo unless it is connected to sale or distribution of genuine products. The Raspberry Pi marks must be less prominent than what it is used for/connected to.

A.2.3 MIPI Camera

For the MIPI camera, we can use the product for personal use according to the MIPI Alliance's Frequently Asked Questions [3] for the ECE 445 project, but if any desires for commercialization occur, we will stay within their boundaries for intellectual property and more.

A.2.3 Chess Piece CAD Models

For the CAD models of the chess pieces, the designs we used can be found at [5] and are under a BY-NC-ND 4.0 DEED creative commons license. Since we are not selling this chess robot, will give credit, and will not distribute our modifications, we are following the terms of the license agreement. If we were to sell this commercially, we would end up hiring someone to design the chess pieces or find a different design online that would allow for commercial use.

Appendix B Design & Subsystems

Appendix C PCB Parts & Costs

The following table, table 7, is a list of all the components we purchased using the budget provided to us by the ECE department for ECE 445.

Description	**Manufacturer**	**Quantity**	**Unit Price**	**Link**
Raspberry Pi 4 Model B - 4 GB RAM	Raspberry Pi Ltd	1	$55.00	Link
USB-C to 2 Pin Bare Wire	Maixbomr	1	$8.99	Link

Arducam for Raspberry Pi IMX219 Camera Module with ABS Case, 1080P IMX219 Camera Module	Arducam	1	$17.99	Link
ESP32-S3 Microcontroller	Espressif Systems	1	$1.85	Link
12V 5A Desktop AC Adapter	TT Electronics	1	$16.79	Link
Buck Switching Regulator IC Positive Adjustable 0.8V 1 Output 3A SOT-583	Texas Instruments Incorporated	2	$1.00	Link
Universal 90 Degree Power Cord	TNP	1	$7.89	Link
Black and White PLA 1.75mm Filament	Dikale	2	$6.99	Link
MicroSD card for raspberry PI	SanDisk	1	$7.35	Link
First order Stepper Drivers	Texas Instruments	3	$3.93	Link
5 V to 3.3 V Linear Regulator	Texas Instruments	1	$1.74	Link
N-Channel MOSFET	Nexperia	1	$0.40	Link
6.8 µH Inductor	Murata Electronics	1	$0.32	Link
Total Cost			$144.35	

Table 7. List of parts purchased using ECE 445 team budget

Table 8 is a list of components that we acquired for free using ECE resources such as the Machine Shop, ES-Shop or the Senior Design Lab in ECEB, 2070. Table 9 is a different list of components that we had to acquire using our own money.

Description	**Supplier**	**Quantity**	**Unit Price**	**Our Price**	**Link**
Zinc Flat Washers	Machine Shop	32	$0.06	Free	Link
30kΩ resistor	ECE ES Shop	1	$0.12	Free	Link
1.3kΩ resistor	ECE ES Shop	1	$0.13	Free	Link
51.0kΩ resistor	ECE ES Shop	1	$0.06	Free	Link
1.5kΩ resistor	ECE ES Shop	1	$0.06	Free	Link
10kΩ resistor	ECE ES Shop	2	$0.06	Free	Link

4.7uH Inductor	ECE ES Shop	1	$1.50	Free	Link
6.8uH Inductor	ECE ES Shop	1	$1.80	Free	Link
33uF Capacitor	ECE ES Shop	4	$0.70	Free	Link
N-channel MOSFET	Senior Design Lab	1	$0.80	Free	Link
2.1mm Female Barrel Plug	ECE ES Shop	2	$0.70	Free	Link
5V 1A DC Power Supply Adapter	ECE 110 Honors Lab Project (Already Owned)	1	$7.59	Free	Link
2 oz. 12-Color Acrylic Craft Paint Set	Already Owned	1	$7.98	Free	Link
LED (for displaying cheat detection)	ECE ES Shop	2	$0.17	Free	Link
Button	ECE ES Shop	1	$2.30	Free	Link
Mercury Motor SM-42BYG011-25 2 Phase 1.8° 32/20	Machine Shop	3	$17.99	Free	Link
KK P-50/27 50 kg Electromagnet	Machine Shop	1	$21.37	Free	Link
Schottky Diode	Senior Design Lab	2	$0.42	Free	Link
1/4 Inch Thick Plexiglass Sheet 18” x 24”	Machine Shop	2	$18.83	Free	Link
Camera Mount	Machine Shop	1	$39.99	Free	Link
FT232R USB 2.0 to UART Interface Evaluation Board	Senior Design Lab	1	$17.50	Free	Link
ESP32-S3-DEVKITC-1-N8	Senior Design Lab	1	$15.00	Free	Link
USB-A to USB-Mini Cable	Senior Design Lab	1	$2.98	Free	LInk
Total Cost			$218.23		

Table 8. Components acquired for free through ECE 445 resources

Description	Manufacturer	Quantity	Unit Price	Link
DRV8428PWPR Stepper Driver	Texas Instruments	3	$3.27	Link
Buck Switching Regulator IC Positive Adjustable 0.8V 1 Output 3A SOT-583	Texas Instruments Incorporated	2	$1.00	Link
SCHOTTKY DIODE 40V 5A	Comchip Technology	1	$0.44	Link
0.47 µF Ceramic Capacitor	YAGEO	4	$0.27	Link
Bipolar (BJT) Transistor NPN	Comchip Technology	2	$0.29	Link
Tactile switches	Omron Electronics Inc-EMC Div	2	$0.75	Link
DRV8825 Stepper Motor Driver Module	HiLetgo	1	$14.49	Link
Power Cord - NEMA 5-15P to IEC 60320	Monoprice	1	$6.48	Link
300 pcs Multilayer Monolithic Ceramic Capacitor Assortment kit	EEEEE	1	$9.99	Link
Micro HDMI to HDMI Cable 6 Feet	iBirdie	1	$5.99	Link
Total Cost			$52.36	

Table 9. List of parts purchased using team member's own funds

Appendix D Computer Vision

D.1 Computer Vision Code Design

Here is an in depth explanation on the most technically complicated part of the code for our project, that being the computer vision portion. Though the OpenCV [11] and EasyOCR [12] libraries are doing a lot of the heavy lifting, this was a very challenging portion of the project to complete, and the code we wrote can be found in the python file 'board_detector.py' [15]. We will discuss how it works and the design choices made while writing the code.

First, the main part of the code was board detection. A couple different attempts were made, including one using shi-tomasi corner detection, one using Douglas-Peucker algorithm, and the probabilistic Hough Lines Transform, but we ultimately decided on the non-probabilistic Hough Line Transform [10] offered by OpenCV, with the help of a discussion of using the function for a similar use on a sudoku puzzle [9]. It is more demanding on the Raspberry Pi than the probabilistic Hough Lines, but is necessary to get the lines of the board and to extend them all the way to the edges of the image to make sure all 81 corners could be detected as the probabilistic one would not finish the line sometimes. Some filtering was also required, as we could assume the lines were going to be roughly the same every time since the camera was fixed. Therefore, we ensure the lines are mostly vertical and horizontal, and also the midpoints are far enough away, where these thresholds were found just by tuning based to make it work with our specific image and camera distance away from the board. With the lines found, we remove the lines that go past the edges of the board through contour manipulation, get the intersection points of the lines, and warp the image based on the 4 outermost corners of the board. We only perform this board detection once so that each subsequent image is being warped the same each time based on the first image's corners, which will become important later when we take the difference between the current and previous images.

With the board detection and warping done, we can now move on to piece detection. First, we convert the image to HSV color space and threshold to get the brightest and most saturated parts of the image, which will be the colored letters on top of the chess pieces. Again, this simply required tuning based on test images in the lab room. We then take the difference between the previous image's letters and the current image's letters, which allows us to essentially define the squares that we want to run our OCR detection on. This is important because this ensures we do not have to run OCR on every single square with a letter in it, as that is how we were doing it originally and it was incredibly slow on the Raspberry Pi (roughly a minute and 30 seconds per turn originally, down to about 10 seconds per turn using this new optimization). With our 81 corner intersections found, we can now loop through all the squares of the board. When a square is found to have a large enough contour from the difference between the two images, we mark it with a flag to run OCR detection on it and decide which color the contour is based on the average hue (this contour, since we are thresholding on size, should be the letter in the square).

The process for OCR detection is as follows: get the contour in the square (which is the letter), straighten the contour based on its bounding box [14], run OCR (making sure to only look for the letters 'KQRNBP'), rotate by 90 degrees, run OCR again, and use whichever OCR result was better. The reason we are running OCR detection twice is because when we set the OCR detection to run on the 0, 90, 180, and 270 degrees axes, it would confuse the letters sometimes (such as it detecting a Q when there was actually a P). It ran much better only using the 0 and 180 degree axes, but also when we straightened the letter, it would sometimes rotate it to the 90 and 270 degree axes, making it sideways. Essentially, we're artificially including all 4 axes ourselves because we were having trouble getting it to work with EasyOCR and having trouble

making it only rotate to be rightside up or upside down (0 and 180). Regardless, we take the best of the 2 results as our piece, which will either be one of the letters ‘KQRNBP’ or nothing, and we mark the color. Later, the board becomes updated, where we add the piece based on its color and letter (or remove the piece at that board location if OCR detected no letter).

D.2 Computer Vision Tests/Verifications

This section includes a brief description of some of the demo tests used to confirm that the computer vision was working correctly, which can be found in the Computer Vision Demo Tests folder [16]. This is not representative of all tests, as we had many more while we were finalizing the code leading up to our demo. Each test shows a game starting from the initial state of a board with the white pieces moving first (the player) and switches back and forth between the player and the chess robot. Also, these images have the real state of the board after the magnet moved the piece around. However, these images were saved and ran through our code again, so it is worth noting that the move selected by stockfish will not be the same as the move played by the chess robot since it will probably not pick the same move as when we took the images. From these images we’ve obtained and the extensive tests where we did not save the images, we believe under ideal conditions (not bumping the board/camera or touching pieces that are not part of the actual move being played), the piece detection is very accurate and above our goal of 90% accuracy. In fact, these test images have an accuracy of 100%, and our tests that we did not save had similar results. The only issues were castling and en passant, which we omitted from the project for the final demo due to time constraints and bugs we could not finish before the demo.

References

[1] “Software engineering code - ACM ethics,” ACM Ethics - The Official Site of the Association for Computing Machinery’s Committee on Professional Ethics, https://ethics.acm.org/code-of-ethics/software-engineering-code/ (accessed May 1, 2024).

[2] “GNU general public license v3.0,” Choose a License, https://choosealicense.com/licenses/gpl-3.0/ (accessed May 1, 2024).

[3] “Frequently asked questions,” MIPI, https://www.mipi.org/resources/frequently-asked-questions#f (accessed May 1, 2024).

[4] Raspberry pi trademark rules and brand guidelines - raspberry pi, https://www.raspberrypi.com/trademark-rules/ (accessed May 1, 2024).

[5] Thingiverse.com, “Chess set - print friendly by tetralite,” Thingiverse, https://www.thingiverse.com/thing:378322 (accessed May 1, 2024).

[6] “Home,” Mueller Corp, https://muellercorp.com/abs-plastic-and-how-we-use-it/ (accessed May 1, 2024).

[7] “SS2422-5041 - stepper motor, single shaft, 42 mm, bipolar, 1.8 °, 0.186 N-M, 1 a,” Farnell, https://uk.farnell.com/sanyo-denki-sanmotion/ss2422-5041/42mm-slim-pancake-stepper-motor/dp/1708573 (accessed May 1, 2024).

[8] “OpenCV-Python: Sudoku Solver - Part 2.” *OpenCV-Python*, 3 June 2012, opencvpython.blogspot.com/2012/06/sudoku-solver-part-2.html. (accessed 28 Mar. 2024)

[9] “How to Remove Convexity Defects in a Sudoku Square?” *Stack Overflow*, stackoverflow.com/questions/10196198/how-to-remove-convexity-defects-in-a-sudoku-square/11366549#11366549. (accessed 28 Mar. 2024)

[10] “Line Detection in Python with OpenCV | Houghline Method.” *GeeksforGeeks*, 9 May 2017, www.geeksforgeeks.org/line-detection-python-opencv-houghline-method/.

[11] “OpenCV: OpenCV-Python Tutorials.” *Docs.opencv.org*, docs.opencv.org/4.x/d6/d00/tutorial_py_root.html.

[12] “Jaided AI: EasyOCR documentation,” www.jaided.ai. https://www.jaided.ai/easyocr/documentation/

[13] “Python-Chess: A Chess Library for Python — Python-Chess 1.5.0 Documentation.” *Python-Chess.readthedocs.io*, python-chess.readthedocs.io/en/latest/.

[14] “How to straighten a rotated rectangle area of an image using OpenCV in Python?,” Stack Overflow. https://stackoverflow.com/questions/11627362/how-to-straighten-a-rotated-rectangle-area-of-an-image-using-opencv-in-python (accessed April 22, 2024).

[15] “board_detector.py,” Google Docs. https://drive.google.com/file/d/1DBpt3vKHdgMPGldsZKuyxcVkKDZcNjf9/view?usp=sharing (accessed May 02, 2024).

[16] “Computer Vision Demo Tests - Google Drive,” *drive.google.com*. https://drive.google.com/drive/folders/1DDXIVdKum35UD4oqHp7FucpXLTzem7w_?usp=sharing (accessed May 02, 2024).