FORK-IN-SOCKECTEERS' RECHESS

A Research Paper submitted to the Department of Electrical and Computer Engineering
Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

By

Selena Pham

December 13, 2022

ADVISOR
Harry Powell, Department of Electrical and Computer Engineering

Fork-In-Socketeers' ReChess

Bryam Ayvar, Bruce Bui, Marshall Mcilyar and Selena Pham

December 13th, 2022

Capstone Design ECE 4440 / ECE4991

Signatures

Statement of Work:

Bryam Ayvar:

My main responsibility was OpenCV and the interaction between the camera and the board. I researched and developed the board detection and the piece recognition algorithms, testing multiple approaches and finding the most accurate one to have a reliable stream of data for the rearrangement algorithm to function properly. I also worked analyzing the details of the markers, such as size, distance, and camera sensor resolution. I developed the camera driver code and configured the Pi to be able to handle the resolution from our camera sensor. Finally, I designed and built the packaging of the camera. This includes around half of the woodworking aspect of the project.

My secondary responsibility was designing the motor driver portion of the PCB, 3D printing many of the parts, such as the chess pieces and the gantry parts (which Marshall designed) and setting up the breadboard for the preliminary design of the project.

I assisted on system and integration testing with Selena and Marshall for the entire project and debugging and testing the PCB with Bruce. I also prepared all the documents, parts, and board to assemble the PCB at 3W with the help of Selena.

Bruce Bui:

My main responsibility was designing, assembling, and testing the printed circuit board. This included researching the operation of motor drivers, use of MOSFETs as switches, electromagnets as inductive loads, and general board design concepts. I designed the electromagnet control and power supply sections of the PCB. After receiving the PCB, I was responsible for soldering components onto the board, and performed most of the testing on it. During integration testing, I was responsible for setting up remote access to our Raspberry Pis, to avoid having to use an external monitor and keyboard. I also enforced determinism on the Raspberry Pi, to ensure our rearrangement does not hang infinitely, and executes within a set time.

Marshall Mcilyar:

My main responsibility was designing and building the gantry system. I first researched and found the best way to achieve 2D motion (via coreXY), and then I CAD designed a gantry system under this design mechanism. I purchased and 3D printed a few parts, then assembled the gantry design by hand. Once the motors were operational, I then tested the gantry system and added features like limit switches and cable management to improve design robustness.

My secondary responsibility was working on PCB circuitry and schematics for the system inputs: the two limit switches and start button switch. This includes multithreading and debouncing the on/off switch and software level logic for the limit switches.

I spent my remaining time designing the chess pieces in CAD, iteratively testing the electromagnet + permanent magnet combination, and assisting with software improvement and full system testing.

Selena Pham:

My main responsibility was the rearrangement algorithm and the embedded code, which is the step that follows right after the image processing. I developed a program to determine if a board is rearranged, which piece to move first, prioritizing the main pieces and the shortest path for a given piece, and also handling exceptional cases, such as moving a piece out of a spot if a piece needs to move there. Then, I programmed the Raspberry Pi to send the appropriate signals for the stepper motor drivers to move one piece from a tile to the other. I also tested and determined many of the factors of the motors and the coreXY system, such as the speed, micro-stepping level, how many steps to move the carriage 1 mm and torque.

My secondary responsibility was putting Bryam image processing and Marshall multithreading code to have the completed software system. After that, I did a great deal of system tests to identify any errors if they came from mechanic system or software system. Then, I communicated with the corresponding team members to fix integration issues, such as improving the accuracy of the markers, helping with the bump-switches, testing the magnet and electromagnet strengths, and breadboard issues.

I also assisted in building the wood frame of the board, doing around half of the woodworking, and the packaging of the pieces, such as spray-painting them and designing and printing the green playing board.

Table of Contents

Table of Figures

Abstract

The ReChess is an automatic self-rearranging chess board which can be used to return the pieces quickly and efficiently to their starting positions once a match has concluded. In order to determine the current game state, a camera suspended above the board can frequently capture the positions of the pieces. This data is processed using OpenCV computer vision and used by our algorithms to determine the correct sequence of moves. The actual piece movement is handled by a gantry system underneath the board, which uses an electromagnet that is turned on and off to "pick up" pieces.

Background

Chess, one of the oldest and most popular board games, is played by two opponents on a checkerboard with specially designed pieces of contrasting colors, commonly white and black. The board consists of 64 squares arranged in eight vertical rows called files and eight horizontal rows called ranks. These squares alternate between two colors: one light, such as white, beige, or yellow; and the other dark, such as black or green. Chess first appeared in India about the 6th century and by the 10th century had spread from Asia to the Middle East and Europe [1]. After being introduced to Europe in the 15th century, chess became popular among the nobility [2]. Rules and set design slowly evolved until both reached today's standard in the early 19th century.

Since the beginning of the Information age in the 1950s [3], advancements in artificial intelligence, neural networks, and deep learning have significantly accelerated the evolution of chess. At the time, artificial intelligence playing chess was one of the most novel and groundbreaking ideas. Many studies, prototypes, and inventions pertaining to computer chess

and robotics were created. In 1997, IBM supercomputer Deep Blue's historic victory over world chess champion Gary Kasparov shook the world of chess to its core. Now, a short 25 years later, AIs employing chess engines such as AlphaZero, Stockfish, and Leela Chess Zero are commonplace, and players can even choose a level of difficulty to play against [4]. In addition to improving AI algorithms, there have been further research studies demonstrating how to create a robot to play chess with humans. At the 2011 IEEE International Conference on Robotics and Automation, a team of researchers led by AI researcher Cynthia Matuszek presented Gambit, a manipulative system that is designed to autonomously play chess against human (or robotic) opponents [5]. Matuszek and her team also developed custom robotic arm hardware, as a player, for the Gambit system [5].

As for our research, the chess industry mostly develops AI algorithms and autonomous robotic systems to play chess. In contrast, our project instead focuses on designing a chessboard with the automatic rearranging ability that supports players to reorganize their chess pieces to the original positions to start a new game. The current design is a fully integrated product between hardware and software to create a fully functioning chessboard that can move the pieces and implement the most efficient rearranging algorithm.

The strategy that intertwines hardware and software requires a great deal of knowledge accumulating through the whole engineering curriculum and some additional research and knowledge to support the design. All team members have taken the ECE Fundamentals courses which will help in creating the power supply, designing PCB, and integrating hardware testing. PCB design and signal processing would be used to create a stable power supply, such as the AC to DC transformation. Testing all the hardware components before building them to the physical system is important. The Embedded Computing & Robotics sequels (ECE 3501 and ECE 3502) cover topics in embedded computing with a focus on robotics. The courses introduced and trained us how to use a microcontroller as a "brain" to implement software programs and transfer the results to the processing signals for the hardware, especially the motor here, to move the pieces. In addition, the skills gained from the Algorithm course (CS 4102) would play the most crucial roles in designing and implementing the rearrangement algorithm.

Physical Constraints

Design Constraints

The most important constraint of our design is the capstone course constraint that our design must include a Microcontroller or Microprocessor as well as a PCB. We already had an RPi at our disposal, but the PCB needed to be designed and manufactured. The PCB design needed to be manufacturable, so it had to be under a certain maximum size, above a minimum line spacing, etc. [6]. Due to space constraints, the PCB will also need to be mountable on top of the Raspberry Pi like a HAT [7]. A time constraint for the PCB is the send out. We were only given two of three possible dates to send our board to the manufacturer: 10/10, 10/18, and 11/11.

Although these dates got pushed back, our initial PCB layout had to be complete by either 10/10 or 10/18, and it had to be finalized by 11/11.

Another manufacturing limitation was the part orders. Part orders had to be issued in groups to save money, so they only occurred once a week; for Openbuilds and Digikey, this was sometimes every other week. Furthermore, part availability was scarce, so parts being out of stock was another constraint on manufacturing. The manufacturer for creating the PCB [6] and the part assembly at 3W [8] each have a week of overhead, so this constrained our design time further.

CPU constraints were not heavily considered due to the large processing power of the Raspberry Pi 4B [9]. However, we found OpenCV to be bottlenecking our processor during testing, so we had to limit the camera resolution in order to remove this bottleneck. UVA's software licenses and open-source software made any necessary software readily available.

Cost Constraints

Our project was assigned a budget of 500 dollars. In order to maintain this budget while working in parallel, we assigned a 250-dollar budget to the gantry parts and assembly. In order to operate under these constraints, our design had to work with cheap aluminum extrusion from Zyltech [10], as opposed to more expensive cylindrical rods and bearings found at McMaster-Carr [11]. Costs were cut further by custom designing chess pieces in CAD and 3D printing them for free at UVA. The RPi was already readily available, which helped relax budgetary constraints overall. Nevertheless, the $500 budget limited the quality of the gantry and motors. Also, backup parts were not ordered due to cost constraints, so the cost constraints forced us to find workarounds for broken parts.

Tools Employed

PCB Design

KiCad [12] was used for circuit schematics and PCB design. It allowed us to create a netlist that connected components automatically, map footprints for manufacturers, and use hierarchical blocks to keep our schematics organized. KiCad's schematics editor was used first to draw up the design schematics. We then used KiCad's PCB Layout software to arrange the schematic components on an actual PCB and connect them through a netlist. Our team was inexperienced with KiCad, so it had a steep learning curve before we could start working on our board. Once our PCB Layout was complete, FreeDFM [13] was used to verify the design as manufacturable.

Gantry & Hardware Design

The gantry was designed in FreeCAD [14] before being built. Marshall, the team member in charge of gantry design, was not experienced with Cad, so he had to learn Cad from scratch before building the gantry. Designing the gantry in CAD first guaranteed that parts would fit

together for assembly. Furthermore, some of the parts used in the gantry had to be custom designed and 3D printed to fit the CoreXY design. FreeCAD was also used to design custom chess pieces. Drills, screwdrivers, hex wrenches, and pliers were utilized in the physical gantry assembly. The custom gantry and chess models needed to be created, so Ultimaker [15] and Makerbot replicator+ [16] printers were used for 3d printing them. Ultimaker Cura and Makerbot Print were used respectively to slice the CAD files into 3D printable GCode.

Software

Our OS for the RPi is the default OS, called "Raspberry Pi OS", and it is a Linux distribution based on Debian [17]. The software was programmed using the Visual Studio Code IDE [18]. Our lead programmer Selena had some experience with Python coding already which eased the process of developing functional code. For image recognition, we used a special python library called OpenCV, which has extensive functions and documentation on image processing [19]. Though our team already has experience with python, it took a lot of time to learn the OpenCV library and employ it in our final design.

Societal Impact Constraints

Environmental Impact

The product itself consumes very little power; energy consumption is not a major concern for environmental impact. However, the use of permanent Neodymium magnets is cause for concern. If disposed improperly, the magnets can be toxic and cause health problems to people near where the magnets are dumped [20]. It would be prudent to find a more environmentally friendly magnet for mass production. Plastic chess pieces are also a potential waste hazard. Though PLA plastic is recyclable, not all recycling plants can recycle it [21]. Furthermore, a new issue in microplastics has recently grown in prominence in both environmental research and the mainstream media. There is evidence that the process of recycling can cause microplastics to leak into aquatic environments and pollute them [22]. If the pollution issue of microplastics becomes severe, unnecessary or materialistic products such as this chess player may need to be redesigned. One possible solution is to swap from plastic chess pieces to custom wooden pieces when redesigning the product for mass production. Plastic is fine for rapid prototyping, but a factory with a wooden lathe would be cost efficient and environmentally friendly for mass production.

The lifespan of our product is variable dependent on frequency of use and is mainly dictated by the stepper motors. The motors drive a very light load compared to their weight limit, operating at a fifth of their maximum current, thus a fifth of their maximum torque. Thus, the motors should last for multiple years of frequent or continuous use [23]. Another potential bottleneck for product lifespan is friction of the pieces sliding along the chessboard. For a commercial product, a smooth, polished wooden board surface would help reduce friction,

9

coupled with a low friction padding on the bottom of the chess pieces. Our testing also found that the camera could fail if the piece identifiers got damaged or deformed. Providing some sort of replacement markers may help improve product lifetime, or potentially engraving the markers into the wooden pieces when manufactured.

Sustainability

Given the nature of our product and its awkward size, it's unrealistic to expect consumers to break down the product at its end of life and recycle it. However, because aluminum is highly recyclable [24] and our product contains a large amount of aluminum, our design could be refactored to allow consumers to easily detach the aluminum framing for easy recycling. Nevertheless, a life cycle assessment of some sort would be appropriate to assess the overall impact of this product on climate change with a focus on pollution from manufacturing and recyclability. The impacts may be minimal due to the small size of the product, but it would be prudent to perform an LCA regardless.

Health and Safety

Our device isn't inherently dangerous, but it does have a few potential health hazards of note. First and foremost, the chess pieces themselves could be considered a choking hazard, so this device should not be accessible to children under three years old [25]. Furthermore, the device itself should not be easily broken down in order to prevent children or unsuspecting patrons from being exposed to moving machinery. Unfortunately, this makes extracting aluminum for recycling an especially tricky task. The gantry could potentially damage a person's finger if exposed when the device is active, though the current design does not run at a high enough torque to cause any significant damage. Our current design traps the gantry inside a box that is bolted shut, but a warning label would help advise consumers against taking apart the box. Though epilepsy was initially a health concern, our design does not use flash photography, so there is no longer a concern for epilepsy triggers.

Ethical, Social, and Economic Concerns

Perceived ethical issues that could arise from this design are minimal. The chessboard does not exclude any social group except for those who might not be able to move the chess pieces or press the start button. However, a mass-produced design would be augmented to provide automatic movements through an app for voice commands, and these two accessibility options could alleviate perceived isolation from people who struggle with disabilities.

Economic disparage is the most significant ethical issue because the product may exclude anyone who cannot afford it. However, this is no different from any other unique or expensive chess set on the market. One way to maintain an ethical product is to set aside a portion of profits for a charity in which Re-Chess boards are donated to people with disabilities interested in learning chess. This would of course require more accessibility features to be added to the

design, but the current hardware acts as proof of concept for a more accessible automated chessboard.

External Considerations

External Standards

1. *NEMA motor standards* – The stepper motors we used fall under the size standard of NEMA 17. This means that the faceplate of the motor is 1.7 X 1.7 inches [26]. This standard is useful for designing any stepper motor driven actuation because a manufacturer can use a standardized motor carriage or chassis without having to worry about motor compatibility. For our design, this standard was useful because some of the most popular NEMA-17 motors were either out of stock or out of our price range.

2. *IPC PCB standards* – We used two IPC standards to design out board for manufacturability. IPC 2221 gives general guidelines on trace thickness, part spacing, solder mask clearance, etc. [27]. IPC 600 is an inspection standard [28], and it is used by Advanced Circuits, the PCB manufacturer our group used for standard PCB inspections [29].

3. *Surface Mount Standards* – Our team used standard surface mount component sizes such that we could find appropriate footprints for our PCB layout [30]. While a custom layout is possible, a standard size assures that our PCB manufacturers and assemblers don't have any problems building the board.

4. *STL (Standard Tesselation Language)* - STL is a standard used for 3D modeling purposes [31]. It is a standard file format that represents 3D models as tesselations of triangles. STL files are used to allow 3D models to be imported and exported between different 3D model programs. For example, our project used CAD models of chess pieces in FreeCAD. However, Ultimaker Cura does not understand FreeCAD files. But I can export my FreeCAD files as .STL files and then send those files into Cura, allowing me and my team to produce 3D prints of files I developed using FreeCAD.

5. *V Slot Extrusion Standard* – Though not guided by a formal standard, V-slot extrusion is a sort of standardized guide rail created by openbuilds [32]. The use of a standard guide rail allows our team to buy parts from multiple manufacturers, in this case both Zyltech and Openbuilds.

6. *Choking Hazard* – The only standardized health hazard of our project is the standard on small parts used in toys and games [25]. Our project has chess pieces small enough to be considered a choking hazard according to the CPSC because the pieces are smaller than 1.25" in diameter and 2.25" in height. If our project were mass produced, we would need a clear warning for children under 3 to steer clear of the chess set.

Intellectual Property Issues

The project's patentability is dependent on many factors. The idea is not entirely unique; there is already a chessboard that uses a similar mechanism to move pieces available on the market: the SquareOff board [33]. It also uses an electromagnet to pick up pieces, but uses an HBot gantry system rather than a CoreXY. Additionally, our project uses image processing to detect pieces on the board, whereas the SquareOff board makes use of capacitive sensors and sound feedback to make sure a piece has been touched. Another board similar to our project is the US 11 369 862 patent [34], which also has a framework to obtain an abstraction from the physical world and place it into a computer. However, the patented board uses radio frequency identification (RFID) antennas arranged in registration with the 64 squares on the board to detect pieces, which is wholly different from how we accomplish piece recognition. Figure 1 below shows a diagram of the pieces being detected on the board.
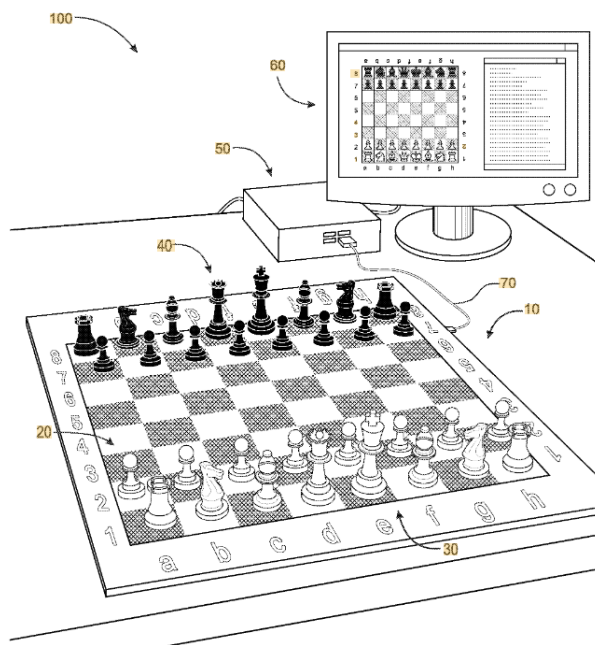
Figure 1: US20190314715A1 smart chessboard patent piece detection

A more general concept of electronic game boards and game piece detection is described by patent US 10 456 660 [35]. The patent is not specific to chess, but instead covers all board games which have some method of tracking piece characteristics and storing/updating them in memory. Our board falls under this category, since it uses a camera to track piece positions, and stores this data in the Raspberry Pi's memory. A patent for our project may not be approved due to its derivative nature. However, we argue that the piece detection system is only one part of our project, and that the piece rearrangement constitutes enough of a difference.

Overall, we believe that our project is sufficiently non-imitative when compared with other patents. The intent of our project is different from those, in the sense that ours focuses only on rearrangement, while existing patents aim for players to play games autonomously, or do not cover autonomous piece movement at all. The mechanisms by which we achieve our goal are also sufficiently different. In conclusion, our project would most likely be marketable as a self-rearranging board.

Detailed Technical Description of Project

What It Is

Our project's objective was to automatically reorganize a chessboard. The process starts when the push button is pressed, then our arrangement algorithm selects a piece and generates a queue of stepper motor moves to return the pieces to their initial place using image processing to locate the board and ArUco markers to pinpoint the exact locations of the pieces. Until the entire board was changed, this process would be repeated numerous times. A Raspberry Pi 4 served as the computer and processor and was employed for stepper motor control and image processing.

Instead of using the traditional 8x8 chess board, the board has a border of 1 tile to allow for capture squares which made the board be 10x10 in total (Figure 2).
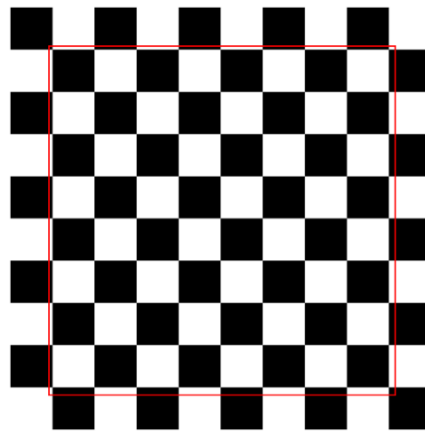
Figure 2: The chess board and its captured pieces architecture

The chess board is put on top of a wood frame which contains a CoreXY system to move the chess pieces. The Core XY system is composed of a gantry and two stepper motors that move an electromagnet which will be enabled with a logic high whenever a piece needs to be moved. Each chess piece has a permanent magnet attached to its base so that they can be grabbed by the electromagnet. Figure 3 describes the first draft of the product.
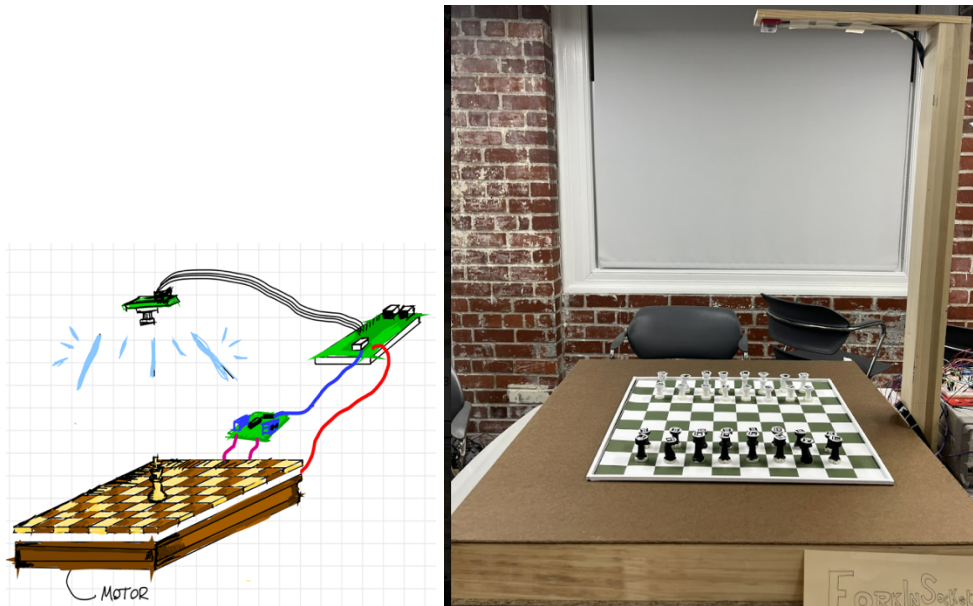
Figure 3: The first draft of the product (left) and the final product (right)

The system design was broken down into the following sections

- Software (Raspberry Pi)
    - Board Detection
    - Marker Detection
    - CoreXY execution: motors and electromagnet logic
    - Origin abstractions: Bump switches logics
    - Start Button Logics
    - Rearrangement Algorithm
- Hardware (PCB)
    - Power Supply
    - Motor Driver Circuit
    - Electromagnet Control
    - Switches Circuit
    - Connection to Raspberry Pi
- Mechanical
    - Piece Design
    - Wood-frame
    - Core-XY gantry design

How It Works
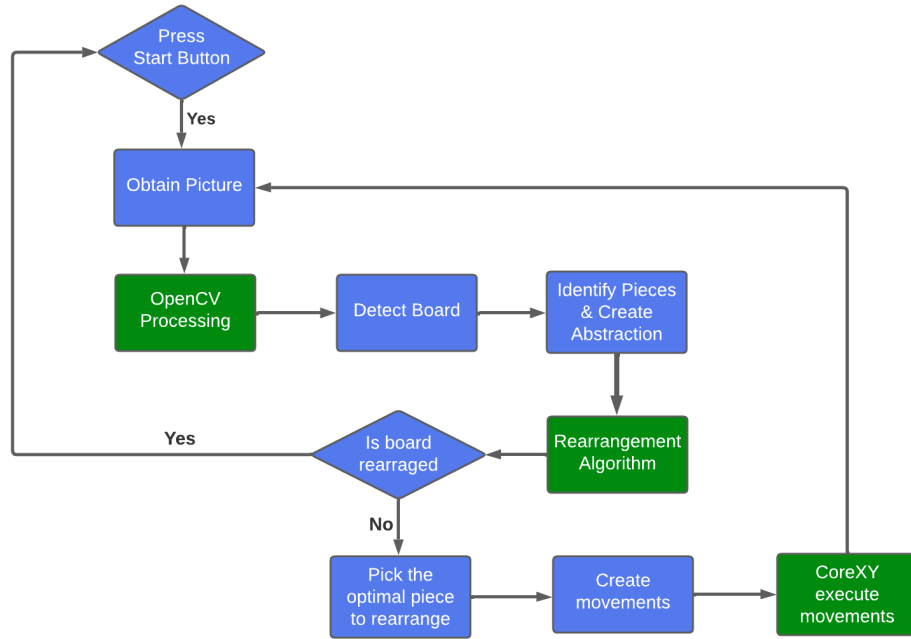
Figure 4: General Process of ReChess

Figure 4 describes the general software process of ReChess. When users press the start button, the Raspberry Pi is instructed to take a picture to capture the board's current configuration. The photo taken is processed by OpenCV [36] to detect the board, identify pieces, and create programming abstractions of the board and chess piece states. There are two abstractions that we use in the program: a 2D array and a "piece" class list that contains the approximate x and y coordinates of the piece, the name of the piece, and the tile it is located in.

The rearrangement algorithm takes these abstractions as input to determine whether the chess board is rearranged or not. If the board is not rearranged, the algorithm picks the optimal piece to rearrange, calculates the path to move the piece to its destination, and creates a queue of movements. The CoreXY executes the movements to physically move the optimal pieces from its current position to the destination. Then, the camera takes a new picture reflecting the new state of the chess board. The process is repeated until the board is rearranged.

Software

Image Processing & Generating Abstraction

Once the Raspberry Pi [37] obtains a picture of the chessboard, it needs to create an abstraction of the current state of the board for algorithm processing. To achieve this, we can further divide the steps into smaller goals:

1. Detect the chessboard
2. Identify the chess pieces

3. Find the location of each piece

We will use the following image as an example of our image processing algorithm:
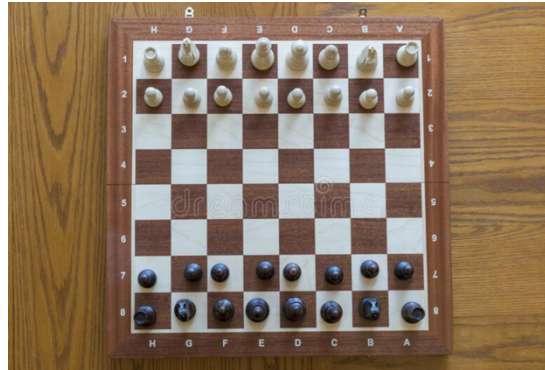
<u>Detecting the chessboard</u>

To detect the chessboard, we will use a string of well-known image processing algorithms from the openCV library [38]. We mainly need to use the Canny algorithm, which detects the edges in a given picture. However, some preprocessing needs to be done before we can apply it, thus the program first applies a Gaussian blur with a 7x7 kernel and turns the image into gray-scale, as these are better for thresholding, which is the next step. OpenCV has an adaptive thresholding algorithm we can use, in case there are changes in the amount of lightning in the picture, to enhance the system's robustness [38]. After these, we can apply the Canny algorithm on the image, as seen by the 1st image below:
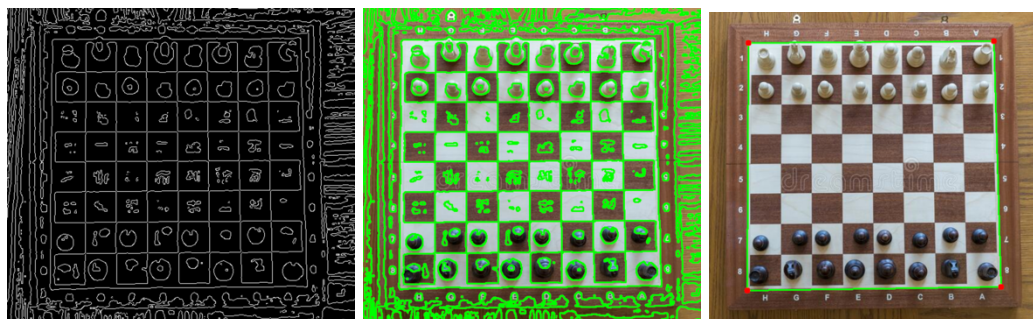
Figure 6: From left to right - Canny result, findContour result, and Board Detector output

After obtaining the result, we can obtain all the contours in the image using findContour, this process is shown by the middle figure. Then, we can approximate these contours to a geometrical shape, using an OpenCV function called **approxPolyDP** [36], and check for the one that has the largest area and four sides - the chessboard. The result can be seen in the last picture. There are a few points of weakness to keep in mind when using this method, such as requiring a bird's eye view, a fixed distance from the chessboard, and having a distinct background under the

chessboard. These are the reasons we decided to apply a static holder configuration for the camera to the final product.

Figure 7: Camera above Board

Finally, we can apply an image warp to zoom in the chessboard and account for any sideways movement in the camera:
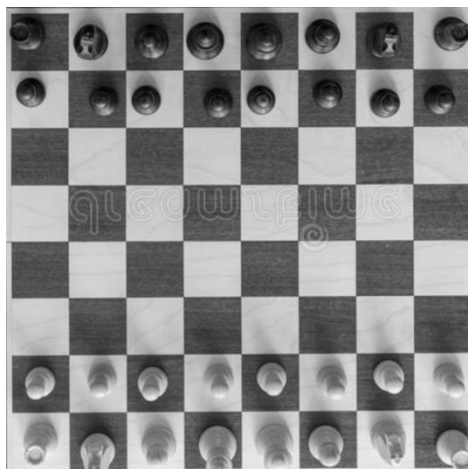
Figure 8: Warped image result

By dividing the pixels in this image by 8, or 10 in our actual project because of the capture squares, we can approximate the bounds for tiles of the chessboard, which will be needed to determine in which tile each piece is located once they are recognized. Further, through a simple conversion with a reference measurement, we also obtain physical distance.

<u>Identifying chess pieces</u>

To identify the pieces, we will make use of a special type of fiducial markers, also known as ArUco markers. There are a substantial number of functions in openCV [36] that facilitate the detection of these markers. Additionally, this method was preferred over an original machine learning because of speed and accuracy purposes. On the original Machine Learning prototyping phases, the board only had an accuracy

An ArUco marker is first found applying an additional adaptive filter, OpenCV internally detects the corner of each possible square candidate, then it divides the candidate by the width of the border + the resolution of the marker. The border is the black padding of the marker. This is In our case, we opted to use the original resolution of an ArUco (5x5) because we did not need many markers (which is the upside of having a higher resolution) and having a lower resolution of markers makes it easier for the camera to detect a piece. Once the candidate has been divided up, each square has a color filter applied to it. The white squares signify a 1 and the black squares signify a 0. This arrangement is hashed into a decimal number that the computer uses to identify the marker. In our case,
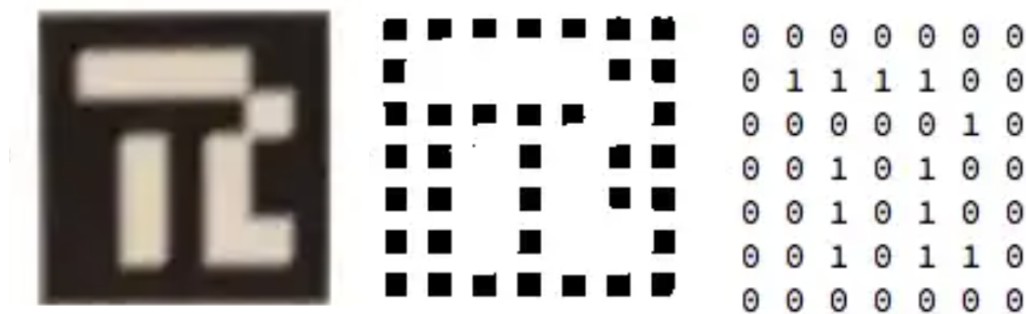
Figure 9: ArUco Marker Breakdown

In our project, we chose the specific ID configurations of the markers that were mostly solid, constant lines, as they were better detected by the camera. Markers that had uneven edges of single squares floating in the middle were more susceptible to being rejected by the computer. We suspect this was because the warping portion of the process makes the color filter to be less certain about whether the bit is 1 or 0.

The next factor which affected the process was the quality of the camera. According to a drone-project at Budapest University of Technology and Economics, a marker should cover 20x20 or 30x30 pixels of the photo. We found that a resolution of 1980x1980 pixels was enough

to detect 15mm markers from around 0.7meters. However, we increased the resolution to 3070x3070 and left the 15mm to make sure the system was robust and did not miss any pieces. The only downside of this design decision was that processing the higher-resolution pictures would take around half a second longer than the other kind. We considered that our project was static and preferred accuracy over speed. Additionally, to improve accuracy, we made use of the built-in sub-pixel refinement method to detect ArUco markers, which is able to detect the corners more accurately.

There are a total of 12 different markers IDS, and they are placed on top each of the pieces. Each of the markers is only unique to one of the pieces, so all eight white pawns would have the same hat, the two black bishops would have the same hat, and so on. Figure 10 displays a few preliminary designs of the pieces (left) and the final pieces (right).

Figure 10: Preliminary pieces vs final pieces

The next step is to write a piece of code that will identify each of the markers. Figure 11 showcases the output of such code on the right window, with the input on the left. The markers are swapped with images of the corresponding piece that they decoded.
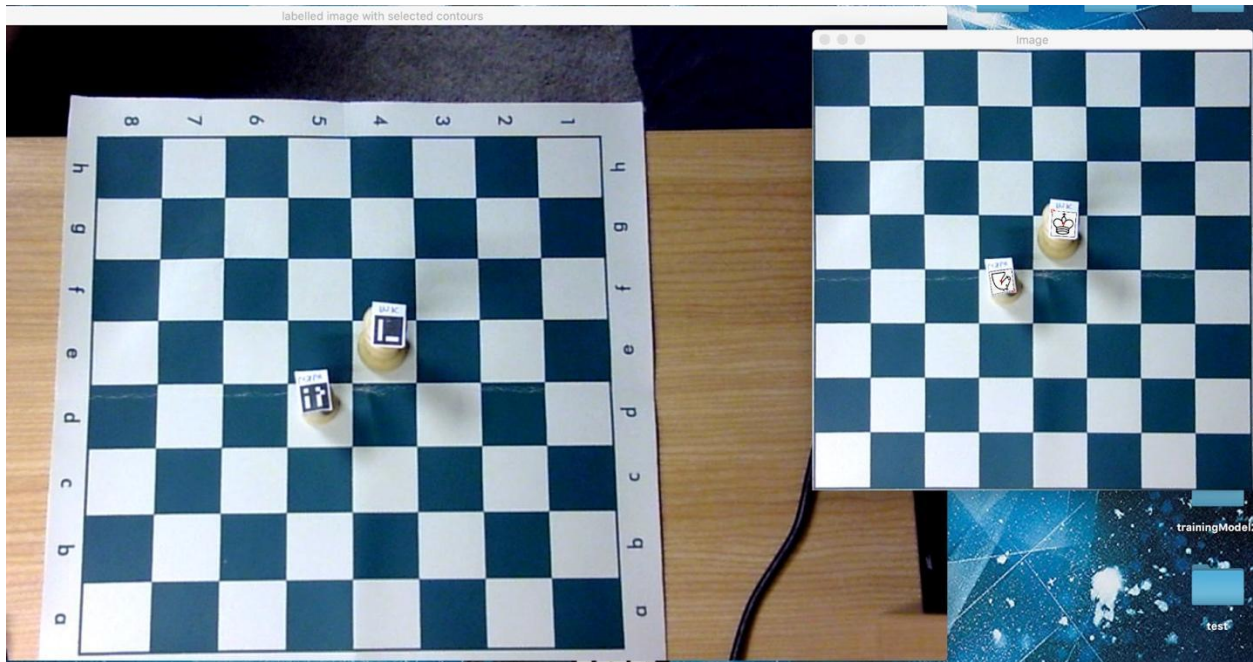
Figure 11: Machine view of the game of chess

Finally, we can simply apply a mid-point formula to find the coordinates in the center of the ArUco marker. This will be a single point, shown in red in the figure above, which will tell us the approximate location of the piece. With the methods and algorithms applied above, our result is a zoomed in and warped picture of the chessboard from the top view with the markers identified in a specific location.

Find the locations of each piece

After obtaining this information, we can simply divide the picture on the right of Figure 11 by the number of tiles of the chessboard and obtain the bounds for each tile (in pixels). Then, we can approximate the location of each piece by finding the midpoint of the squares that they enclose. Using these two pieces of data, the points and the bounds, we can apply a modified version of binary search in the two directions to segment the points and find out their exact locations in the chessboard. Finally, output an abstraction for the rearrangement algorithm to process. The abstraction is the following:

Figure 12: Low-level abstraction result

There is also a secondary abstraction we make use of - a piece array that contains the piece name, row, column, x, and y coordinate. This abstraction is used by the rearrangement algorithm to find the nearest edge and path for the piece to move. The primary 2D array is used to simply determine whether a spot is taken or not. Note that the figure above is from the preliminary testing on the abstraction, with the final abstraction being 10x10 rather than 8x8 to account for capture squares.

<u>Rearrangement Algorithm</u>

Check if the board is rearranged

After receiving the programming abstractions which are the 2D array describing the current board and the list of the piece object orientation, the algorithm checks whether the board is rearranged or not. There is a fact that if there is one piece at an incorrect spot, the board is not rearranged. By scanning through the piece list received from the image processing, the algorithm determines whether every piece is at the right position. If a piece is not where it should be, the board is not rearranged and the algorithm sets the boolean board_rearrange to be False. Figure 13 describes the high level for checking board is rearrange.
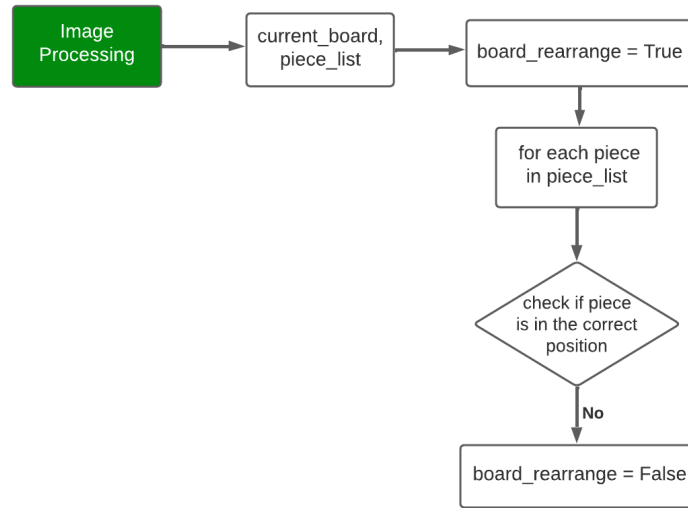
Figure 13: Checking board is rearranged flow chart

Check if piece is in the correct position

As Figure 13, we used another function to check if a piece is in the correct position to support checking is a board is rearranged. We would like to discuss this function here. As we know, each piece has a name and can have multiple correct positions, such as a black knight piece having two possible right destinations, and a white pawn having up to eight correct destinations. Recognizing these facts and saving the runtime for the algorithm, we create two constant dictionaries storing piece names as keys and piece correct destinations as the values. One dictionary is for all the black pieces and the other one is for all the white pieces. In addition, we have a dictionary to store the number of pieces that have different names. Figure 14 shows the three dictionaries from the code.

```python
# R is Rook, N is Knight, B is Bishop, Q is Queen, K is King, P is Pawn
# Dictionary stores the right squares of black pieces. This acts like the
# constants for all algorithm.
BLACK_SQUARE = { "BR1" : [1,1], "BN1": [1,2], "BB1": [1,3], "BQ1" : [1,5],
        "BK1": [1,4], "BB2" : [1,6], "BN2": [1,7], "BR2": [1,8],
        "BP1": [2,1], "BP2": [2,2], "BP3": [2,3], "BP4": [2,4],
        "BP5": [2,5], "BP6": [2,6], "BP7": [2,7], "BP8": [2,8]}

# Dictionary stores the right squares of white pieces. This acts like the
# constants for all algorithm.
WHITE_SQUARE = { "WR1" : [8,1], "WN1": [8,2], "WB1": [8,3], "WQ1" : [8,4],
        "WK1": [8,5], "WB2" : [8,6], "WN2": [8,7], "WR2": [8,8],
        "WP1": [7,1], "WP2": [7,2], "WP3": [7,3], "WP4": [7,4],
        "WP5": [7,5], "WP6": [7,6], "WP7": [7,7], "WP8": [7,8]}

NUMBER_OF_PIECES = {"WR": 2, "WN": 2, "WB":2, "WK": 1, "WQ": 1, "WP": 8,
                    "BR": 2, "BN": 2, "BB":2, "BK": 1, "BQ": 1, "BP": 8,}
```

Figure 14: Pieces and number of pieces dictionaries

We number the pieces that have the same name to identify their possible destinations in the dictionaries. The function uses these numbers as index to scan through the dictionaries for every specific piece name without being numbered. When a piece is passed into the function from the check-board-is-rearranged function, the algorithm checks if the piece is black or white to use the appropriate dictionary. Then, it looks up the maximum number of pieces that have the same name as the current piece and uses the value as max_index to scan through the dictionary. The function aims to find the closest destination for the piece, so we have min_distance, dest_row, dest_col, occupied variables to keep track with the process. The function returns a boolean that determines if a piece is at the right position, the destination that includes dest_row and dest_col, a boolean occupied that determines if the destination is occupied, so the algorithm can move the occupied piece out of the destination before putting the piece there, and the min_distance from the piece's current position to the destination. Figure 15 shows the high-level algorithm for this function in a flow chart.
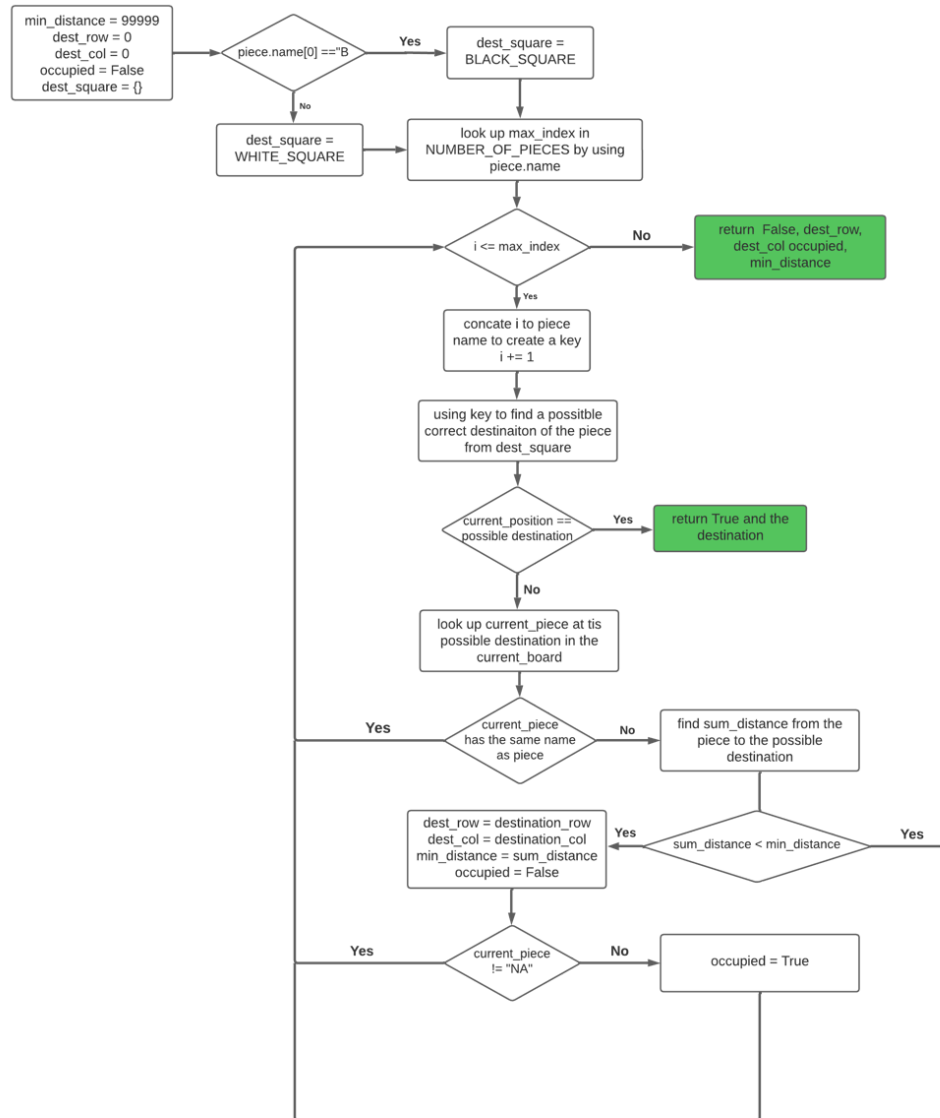
Figure 15: Check if piece is at the correct position flow chart

Pick a piece to move

Acknowledging that finding the optimal piece requires scanning through all the pieces in the piece_list, we expand the check-if-board-is-rearranged function to pick a piece to move to save memory and run time. The function adds extra variables to track the process: min_distance, piece_to_rearrange. As Figure 16 shows, after receiving results from the check-if-a-piece-in-the-right-position, the function compares the result to some standards to find the optimal piece to move. These standards are:

+Prioritizing to rearrange the outside pieces that include King, Queen, Knight, Rook, and Bishop pieces before rearranging Pawn pieces.

+Looking for the closest distance to move among those pieces.

+Not considering the movements needed to move unexpected pieces out of the destination.

Figure 16 describes the high-level algorithm to pick an optimal piece to move.
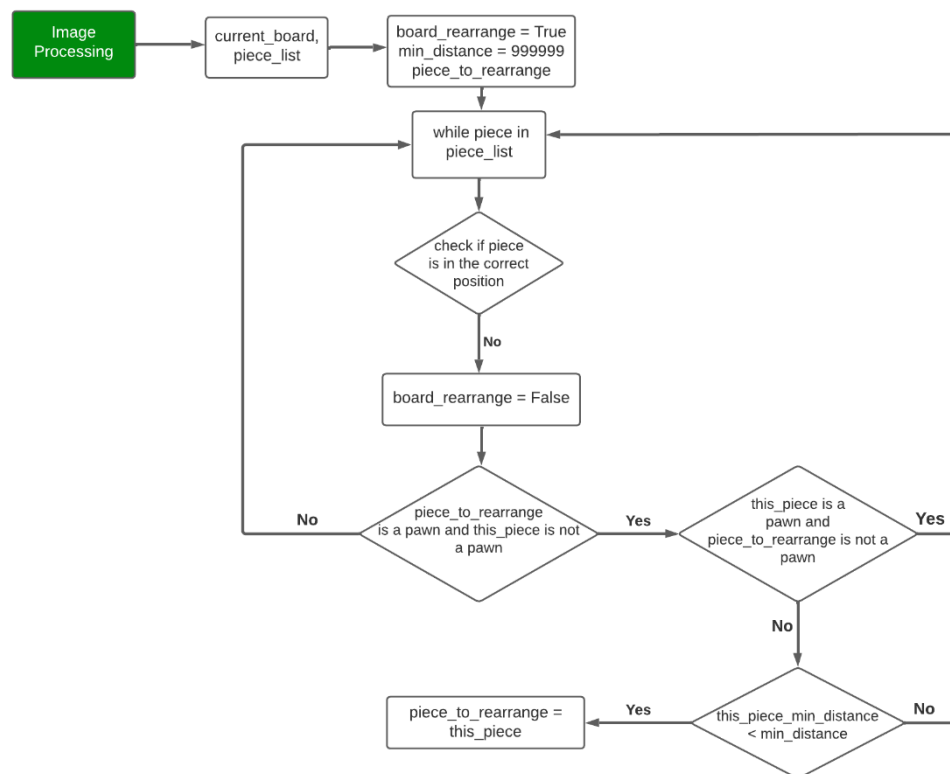
Figure 16: Check if piece is at the correct position flow chart

Create piece movement

After determining whether board is rearranged and finding which piece to rearrange, the algorithm creates movements to move the piece from its current position to the destination. There are two types of movements that the algorithm generates:

+Step movement: this movement is ordering the CoreXY to move the carriage in a direction (up, down, left, and right) and the number of steps that the two stepper motors need to rotate.

+Electromagnet movement: this movement is ordering the CoreXY to turn on or turn off the electromagnet to grab or release a piece.

The system aims to move the pieces in the middle of the square in the up, down, left, or right destination to avoid colliding among the pieces because most of the times, players put their pieces in the squares. However, if there are pieces on the way, the permanent magnet under the moving piece will push the standing pieces out of the way. We tested many magnets and designed the chess pieces in the right size to make this be possible. In the case that if the electromagnet may grab the standing piece and unexpectedly release another piece, it will move the wrong piece to the wrong destination and later on, the pieces will be rearranged eventually. We have tested the probability of happening this situation. The probability is 1 out of 10 times because the moving piece always have tendency to move pieces out of the way. If the wrong grabbing piece situation happened, the pieces are always rearranged to their right destination.
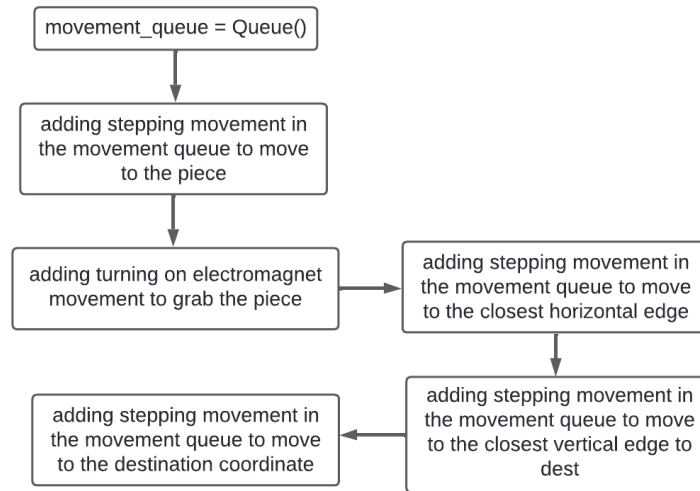
Figure 17: The high level of creating the piece movement

In addition, the algorithm uses the occupied Boolean variable from the check-if-a-piece-in-the-right-position to determine whether there is a piece occupying the destination. If there is an occupied piece, the algorithm uses breadth-first-search to find an empty square near the destination to move the occupied piece to the empty square by adding extra movements before adding movements to move the right pieces.

CoreXY execution

After receiving the movement queue from the rearrangement algorithm, the CoreXY executes each movement in the queue to manipulate the motors and electromagnets to rearrange the pieces. Taking into the consideration of motor step lost and any errors during the process, such as, the carriage hit into the gantry or stuck in the middle of the process, we create a bump

switch system to set up an origin to reset and robust the movement process. After every four pieces are rearranged, the CoreXY moves the carriage back to the origin. If an exception occurs, the CoreXY moves the carriage back to the origin.

The bump switch system includes:

+bumpX: if the coreXY carriage hits the switch while moving to the left, the CoreXY stops the movement.

+bumpY: if the coreXY carriage hits the switch while moving down, the CoreXY stops the movement.

We create a function move_to_origin() where it keeps running the carriage down until it hits the bumpY then it keeps running the carriage to the left until it hits the bumpX. This is where the carriage reaches the origin.

Finally, we have a start button that using multithreading to run parallel with the main program to check if the start button is pressed. We also implement the debouncing function to filter distorting signals from pressing the button. Figure 18 shows the finite state machine of pressing the start.
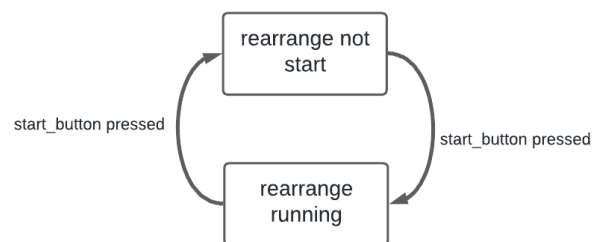
Figure 18: The start button logic

As we see that there are two states for the button: not start state and running state. We use two Boolean variables to describe the two states. When the button has not been pressed, not_start is True and running is False. When users hit the button, not_start becomes False and running becomes True in the side threading. The True running flag triggers the main program to start rearranging process. During the rearranging process, if users hit the button, the running flag becomes False and the not_start flag becomes True. These changing behaviors signal the main program to stop the rearranging process and move the carriage back to the origin with the move_to_origin() function.

How to run the software

We put all the software implementation in the Code folder. Please download all the files. You can use any IDE that can run Python code, such as, Visual Code Studio, to run the code. The only file that you need to run is main.py. Please keep in mind that you need to download OpenCV software to have the full function.

Block Diagram for the Hardware and Mechanical
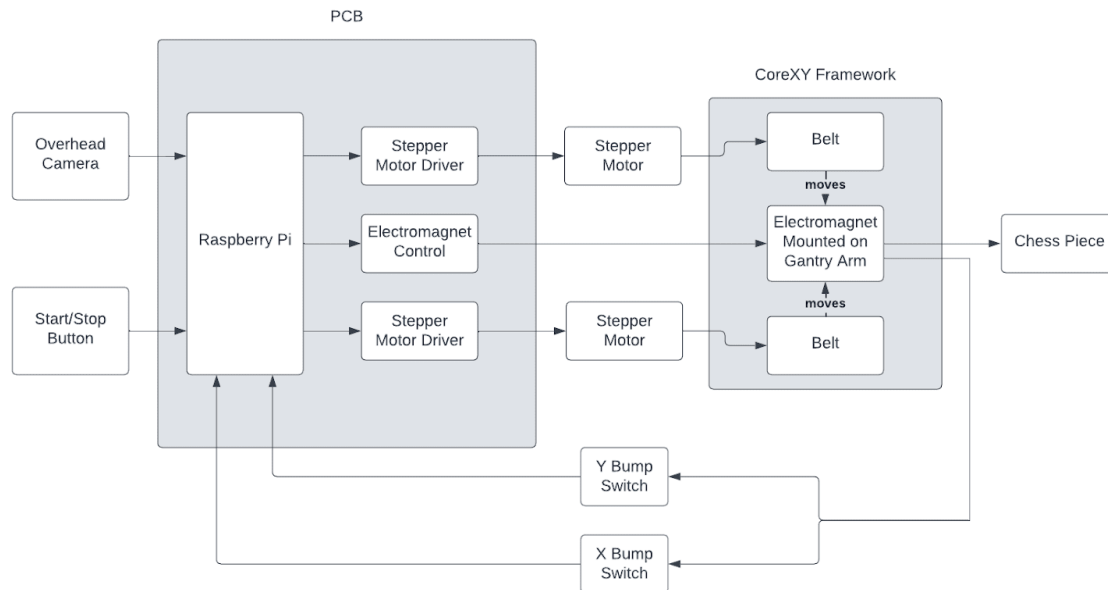
Figure 19: Hardware/Mechanical Block Diagram

Mechanical

Our mechanical system consists of a 2D linear actuator that uses CoreXY, an electromagnet to move the chess pieces, the chess pieces themselves, and a wooden frame to hold everything together. The core of our mechanical design is the CoreXY framework that moves our electromagnet payload. CoreXY is a belt and pulley system that uses a clever arrangement of pulleys to achieve 2D motion [39]. There are two stepper motors from zyltech [10] located in the bottom left and bottom right of a grid that achieve diagonal motion. These diagonal actuators are linearly independent under the coreXY system, so they cover the entire range of 2D motion within the area covered by the pulleys. One can then simply add or subtract the two diagonal vectors together in order to get motion in the X or Y direction.

Our CoreXY gantry was first designed in CAD in order to make sure it could be assembled properly. In this CAD assembly process, a few parts were designed to help fit the extrusion together and hold the pulleys in place. CoreXY is a design technique and not a specific design itself, so the CAD design depicted in Figure 20 was designed from scratch. The parts in green were designed and 3D printed models, while everything else was an imported model of an existing product (motors, extrusion, magnet, etc.).
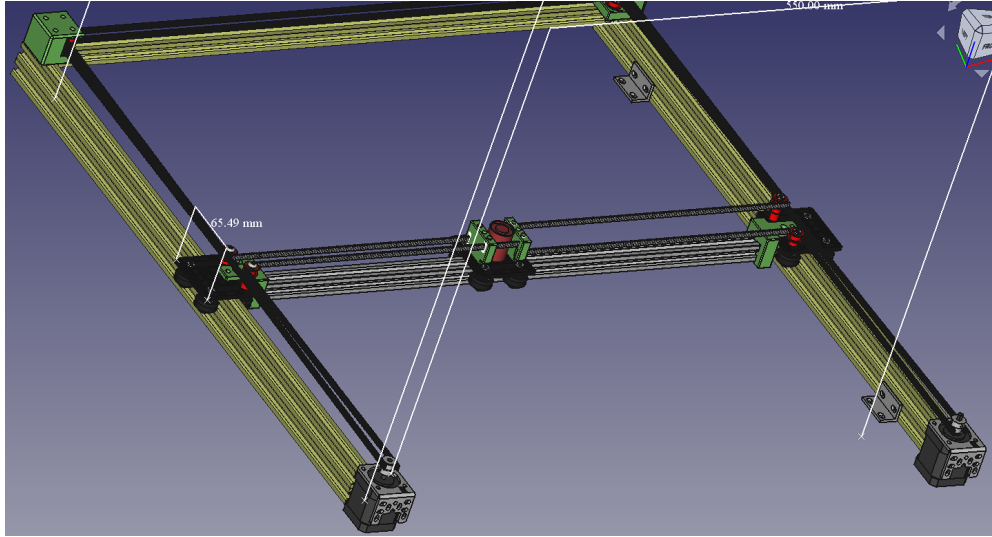
Figure 20: Isometric view of CoreXY CAD Design

Motion is enabled by gantry carts [10], which are small wheeled platforms that can slide freely up and down a piece of aluminum extrusion [10] (shown in yellow). The belts wrapped around the motors are attached to both ends of the middle gantry cart, so the motors pull on the gantry cart when they rotate, like a game of tug-of-war. The belts are wrapped around pulleys that are mounted on the side gantry carts in order to provide smooth vertical motion. A key tenant of CoreXY is keeping the pulleys parallel to the guide rails, as this ensures that the tensions across the belts stay balanced, given that Core XY takes advantage of components of the diagonal vectors superimposing destructively.
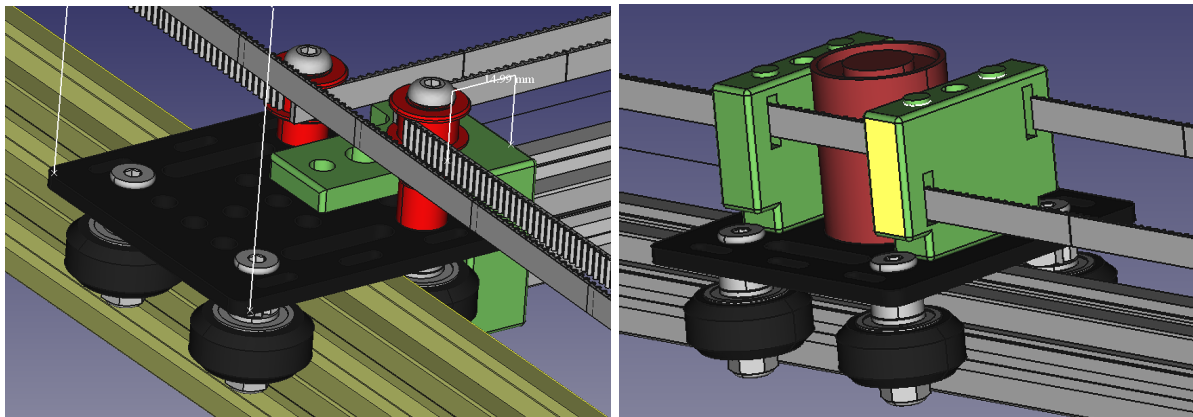
Figure 21: (a) Side gantry cart with idler pulleys (b) Middle gantry cart with belt holders

The custom designed CAD parts each serve unique purposes. The square block is designed to hold pulleys in place that route the belts from one side of the gantry to the other. The thin T shaped block holds the cross beam in place that allows for motion in the X direction. The Fat T shaped block acts as a restraint for the belts. Though not shown in Figure 21, the belts in

the final design are wrapped around the block with clamps that hold the belts in place, maintaining a firm tension in the belts that prevents belt skipping on the stepper motors.
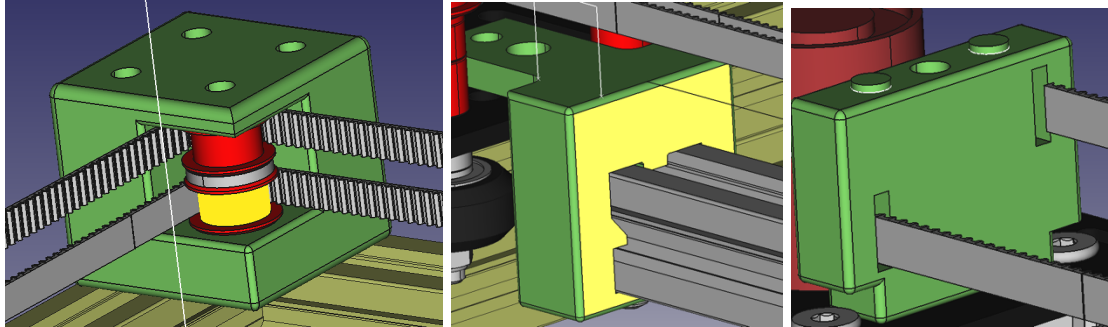
Figure 22: (a) Corner pulley block (b) Extrusion connector block (c) belt clamp block

The whole point of this gantry is to move an electromagnet in a 2-D grid. The electromagnet is glued down to the center gantry cart that can move anywhere in 2 dimensions. The gantry simply takes inputs from the code telling it how long to run the motors, and with proper software this moves the electromagnet directly underneath any piece the code wants.

In order to actually move the pieces, however, there needs to be a mechanism of movement for the pieces as well. Thus, the pieces have a small magnet embedded in them. Custom chess pieces were designed in order to fit this need (as well as the need for space to hold the arUco markers mentioned previously). With a permanent magnet glued or taped to the bottom of the piece, the chess piece is now attracted to the electromagnet whenever a switch is flipped.

Figure 23: Chess piece with bottom cut out for magnet

In order to hold up all the pieces, a thin board is laid out on top of the gantry. The board is thin enough and low enough friction such that when the electromagnet is turned on, there is still a firm attraction between the chess pieces and the electromagnet. As the electromagnet moves, it pulls a chess piece with it, and this complete system creates 2-D motion for any chess piece on the board. There are issues with the magnets from chess pieces interfering with each other, so the permanent magnets went through an extensive testing process until the right magnet size was found that minimized interference while maximizing attraction to the electromagnet.

Figure 24: Chess board on top of hidden gantry

The frame itself consists of a pine wood two-by-four cut and bolted on top of plywood (purchased from Lowes), with a thin hardboard material laid out on top to hold the chess pieces. For temporary usage, a large piece of paper was printed out to form the chessboard. A mass-produced design would have custom painted and stained wood to reduce friction even further and increase product longevity.

The final mechanical aspect of our design is the bump switches [40]. These bump switches e situated in the corner of the gantry, and they tell the software when the electromagnet is at the origin. They calibrate the electromagnet to make up for any lost steps in the stepper motor such that error doesn't accrue over time. These switches had to be moved around a few times because they kept getting tangled up with the wires of the electromagnet, but they are currently glued down in the bottom left corner of the gantry. An overall shot of the finished gantry is displayed in Figure 25. The differences between this image and the CAD design are apparent, specifically with new stepper motor holders [41] and all the input wirings being in place.
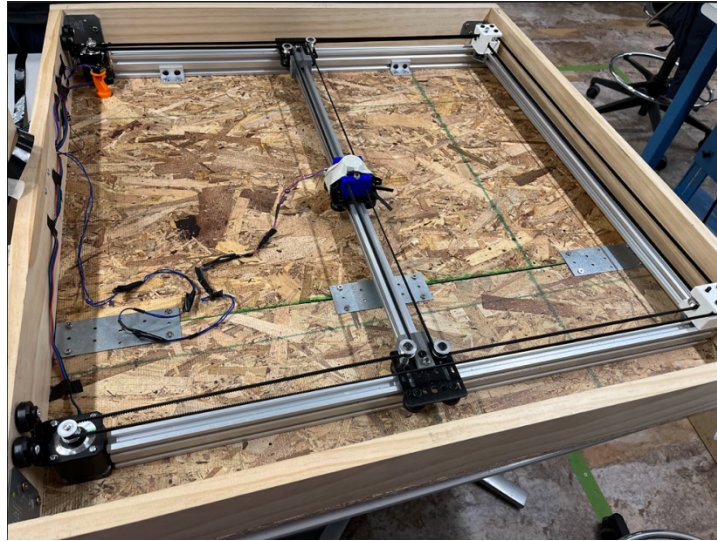
Figure 25: Finished gantry design

Hardware

      Our printed circuit board was designed to interface between the code and the CoreXY motor system as a Hardware Attached on Top (HAT) which rested on top of the Raspberry Pi. The PCB was also designed to provide power to the motors and electromagnet. Initially, we did not realize the need for the bump switches and start button, so they were not included until later. The preliminary KiCad schematic for the PCB is shown below in Figure 26.
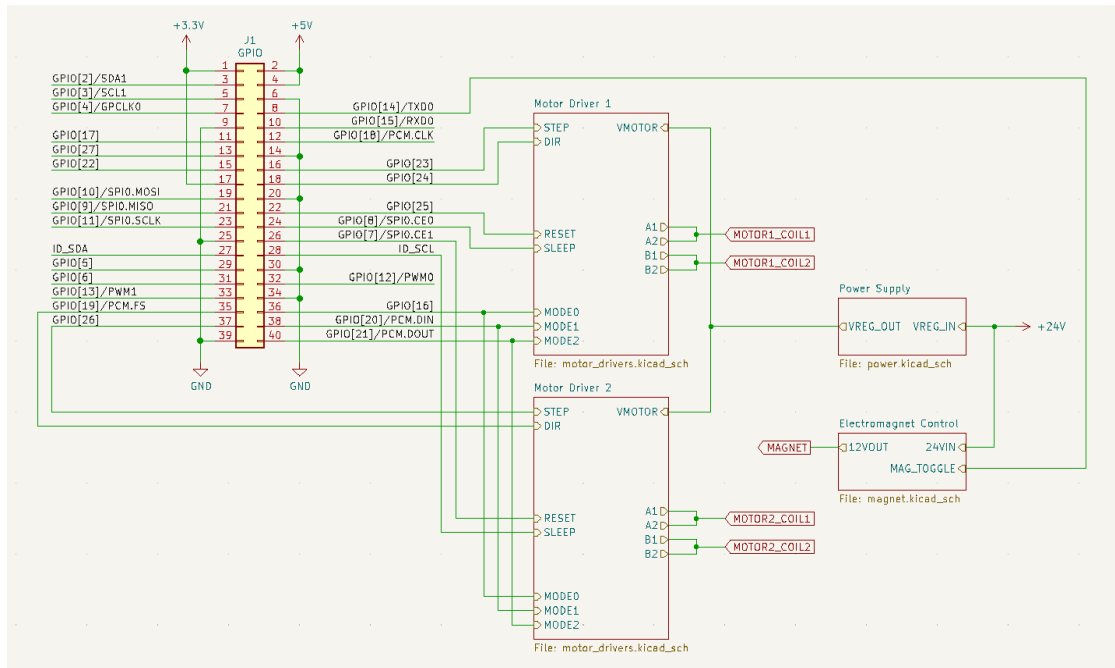
Figure 26: Full Schematic

After receiving our first PCB, we discovered several flaws. Some components in the schematic had footprints which were not realistic for their value and were subsequently not able to be purchased. For example, our schematic contained 100μF capacitors rated for 50V, but the footprints chosen for them were a size 1206 package, which is far too small for capacitors with those specifications. As a result, we were unable to solder all the components on our PCB to test it fully. Additionally, the buck converter we used to step 24V down to 5V ran out of stock at our suppliers, and we had to select a new buck converter. After updating our design to include the switches, we made some more modifications to resolve the footprint issues and replace the old buck converter circuit. We also added a switch to turn the power on and off, and several LEDs which indicated the status of the rearrangement process. The final schematic is shown below in Figure 27.
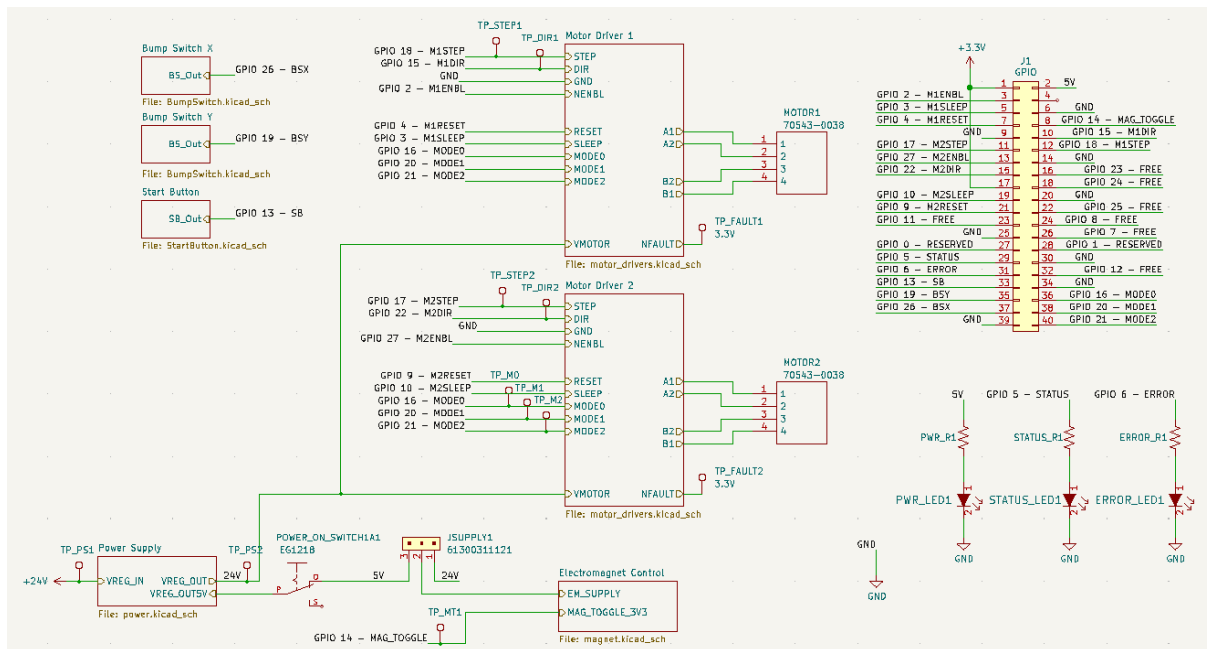
Figure 27: New Final Schematic

Component J1 is a GPIO header which allows the PCB to sit on top of the Raspberry Pi, and MOTOR1 and MOTOR2 are the 4-pin connectors where the motors are plugged in. When we designed this version of the PCB, we were deciding between two types of electromagnets to use, each powered by 24V and 5V respectively. Because of this, we added component JSUPPLY1: a jumper between 5V, 24V, and electromagnet power supply, which allows us to place a shunt to connect the electromagnet to the correct source.

We will now examine the different subsections of the schematic, starting with the power supply, as shown below in Figure 28.
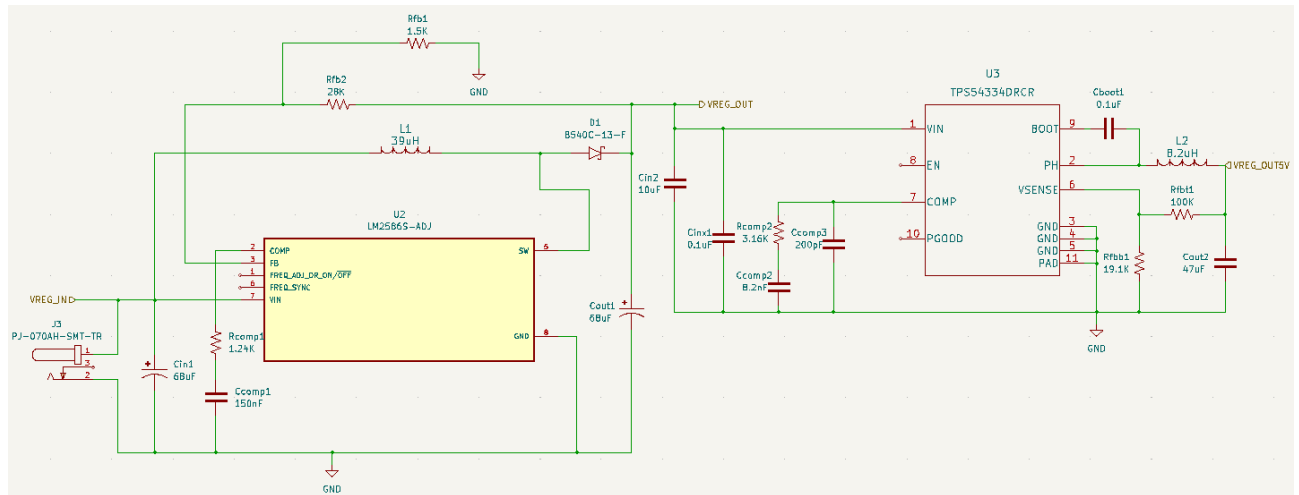
Figure 28: Power Supply Schematic

The power adapter we chose to use (not shown on schematic) supplied 24V and 2.5A to our circuit and was connected to the PJ-070AH-SMT-TR power jack. The motors we used required 24V, which is the reason for our adapter choice. In this schematic, we used a LM2586S-ADJ boost converter to provide input protection to our PCB. It is supplied with 24V by the power jack, and outputs 24V and 2A to our motor driver chips, as well as the TPS54334DRCR buck converter. This converter steps the 24V down to 5V and outputs 2A, which would be used to supply our Raspberry Pi and electromagnet. Passive components were connected based on the datasheets of the converters, and their values were determined by the TI WEBENCH power designer tool to give the correct current and voltage output.

The next subsection we will cover is the electromagnet control, shown below in Figure 29.
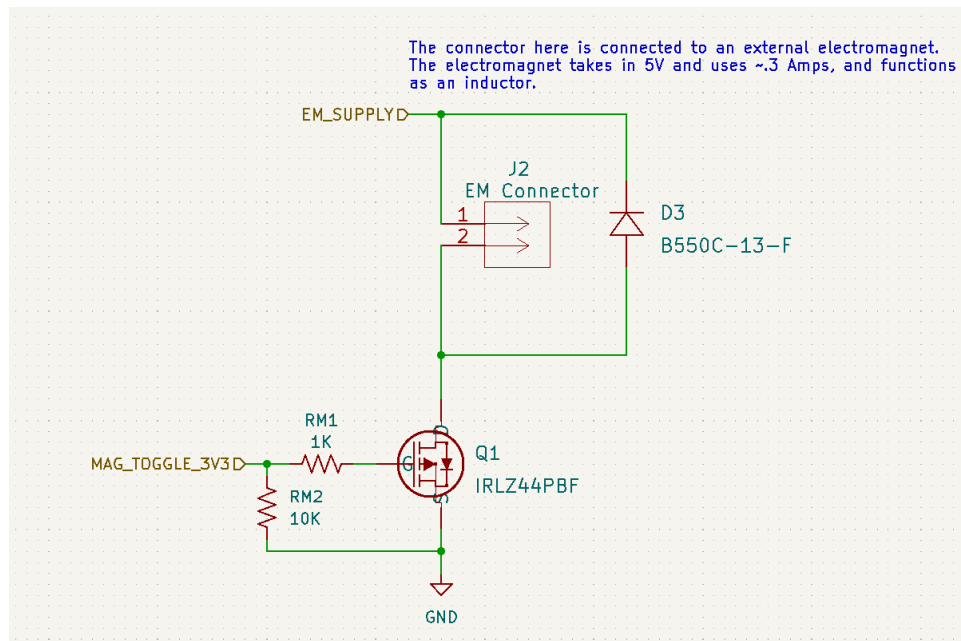
Figure 29: EM Switch Schematic

In this circuit, Q1 is an n-channel logic level MOSFET that receives a signal from the Raspberry Pi to turn the electromagnet on and off. J2 is the electromagnet connector, and D3 is a Zener diode which protects the transistor from voltage spikes when switching the magnet. When the corresponding pin on the Raspberry Pi GPIO header is set high, the MOSFET will allow current to flow to the magnet, turning it on. Setting the pin low will block current flow, thus turning the magnet off.

Next, we will look at the motor drivers, shown below in Figure 30.
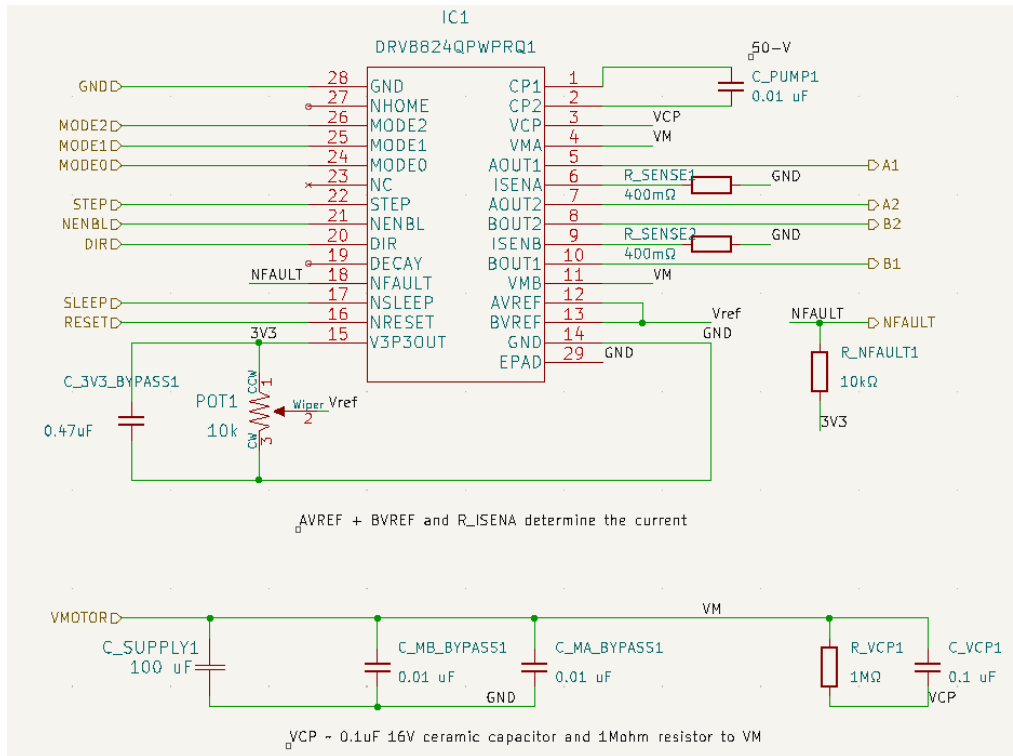
Figure 30: Motor Driver Schematic

To drive our motors, we used the DRV8824QPWPRQ1 chips, which were readily available to us. We used two motors, and thus needed two identical motor driver chips. The capacitors at the bottom of the schematic serve as input protection for the chip, and various pins on the chip are connected to the Raspberry Pi via the GPIO header. All the components in this section were connected according to the recommendations in the chip's datasheet, except POT1, which is a potentiometer we included to allow the tuning of V_{ref}'s value. Adjusting V_{ref}'s value changes the motor current, as defined by the equation $I_{motor} = \dfrac{V_{ref}}{5 \cdot R_{sense}}$. The MODE0, MODE1, and MODE2 pins are used to determine the stepping resolution of the motors and are set using GPIO pins. NENBL, NSLEEP, and NRESET are pins which turn the driver chip on and off. The DIR and STEP pins control the motor operation by setting the motor's spin direction and running the motor respectively.

Lastly, we will examine the bump switch and button circuits by looking at the schematic for the X-axis bump switch.
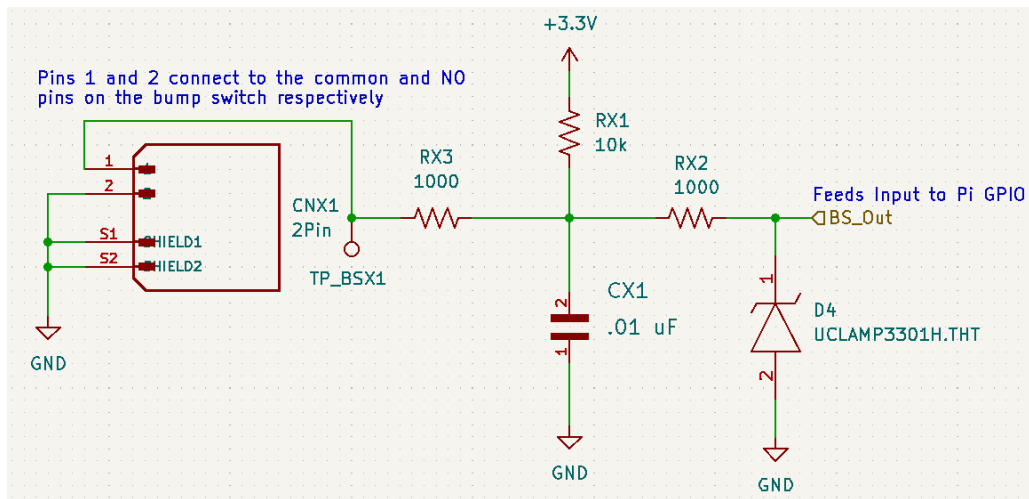
Figure 31: Input Switch Schematic

The bump switches and start button all share the same schematic. The large component at the left of the schematic is the switch/button connector. RX1 acts as a pull-up resistor to 3.3V, and the Zener diode, capacitor, and remaining resistors provide input protection for the Raspberry Pi.

Now that we have finished examining the different subsections in the schematic, we will look at the physical layout of these sections on the PCB.
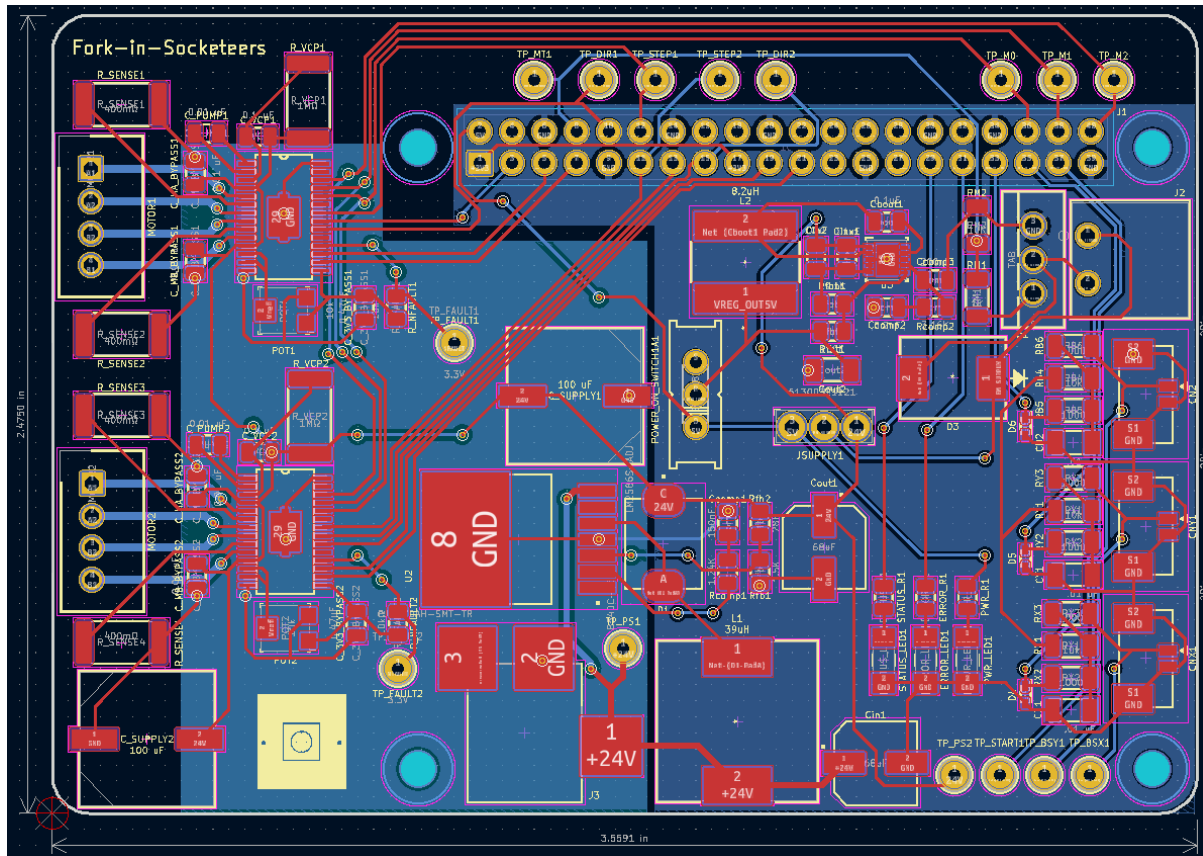
Figure 32: PCB Layout

As detailed in Figure 32, there are test points placed in several locations around the board, which we used to test the PCB. The motor connectors were placed on the left of the board, and the electromagnet and switch connectors were placed on the right to avoid wire tangling and to fit on the Raspberry Pi. A noteworthy feature of the layout is the presence of the two ground planes, shown in the figure as filled in blue areas. These were included in order to separate the high current flow of the driver chips and boost converter from the logic of the circuit. However, in practice, the benefits of this were negated by the fact that some passive components connected to the boost converter were placed on the right ground plane. Another feature which is not visible from the figure is the cutout on the bottom solder mask. An area matching the shape and location of the left ground plane was cut out of the bottom solder mask layer to provide cooling for the motor driver chips.

One major issue encountered with the PCB was the thickness of the traces used for high current connections. This can mainly be seen in the connections surrounding the 5V buck converter.

Figure 33: Zoomed in 5V Buck PCB Layout

These connections needed to be much thicker to accommodate the 2A of current flowing through them. As a result of this design, several of these traces were burnt during testing. Overall, our main problem with the PCB was time. The process of designing the board dragged on for too long, which resulted in us getting our first PCB on the second board sendout, therefore delaying our testing. Additionally, the fact that our first board had mismatched footprints prevented us from fully testing all the board's functions. This led to design flaws being propagated to the second board.

Other than the circuit board, we also worked with hardware for setting up piece recognition for input to our code. The input hardware used a light above the board to get clear vision of the pieces and a camera to take pictures of the board periodically for computer vision. The board itself had white and green squares, and white and black pieces. An essential part of computer vision is getting a proper image, so the input hardware and chessboard are physically configured (and colored) such that the images taken from the CV camera can be processed by the Raspberry Pi effectively.

Initial Design Decisions and Tradeoffs

At the beginning of the project, we planned on creating a chess board that allowed users to play chess matches fully autonomously without touching a single piece. We decided to move away from this concept, as we felt it would be difficult to complete within our allotted time. Instead, we decided to limit the scope of our project to rearranging pieces to their starting

positions after players conclude a match. This reduces the possibilities for user error, at the cost of the design's usefulness. During the ideation process, we iterated through several versions of our chess board, with different mechanisms for recognizing and moving pieces.

<u>Piece Movement</u>

All our piece movement designs were similar, in that they were based around a stepper motor gantry system which would move a carriage that picks up pieces. Our initial designs housed the gantry above the board, like an arcade claw machine. The "claw" would move around the X and Y axes to navigate to pieces and descend to pick them up. For this version of the board, we did not find a good method of picking up the pieces. This fact, coupled with flaws such as the overhead system obscuring players' vision and pieces being easily knocked over during rearrangement, led us to move on from this design. Our later designs relocated the gantry to below the board, removing the need for vertical movement. The mechanism would then only move in the X-Y plane and pick up pieces using magnetic force. The pieces would have a permanent magnet embedded in the bottom, and the carriage would contain an electromagnet that is turned on and off to attract the pieces. By hiding the gantry away, the area above the board is freed up, so players would not have to worry about accidentally damaging it. For these reasons, we settled on this movement mechanism, and proceeded to design the gantry system.

<u>Piece Recognition</u>

We also considered several different methods of recognizing pieces on the board, each corresponding to the different piece movement designs. On our "claw" design, we initially planned on embedding five small magnets in the bottom of the chess pieces, which would be detected by the Hall effect sensors placed on the gantry carriage under the board. The orientation of the magnets would be read differently by the sensors and allow our board to check which piece is at the square where the carriage is currently located. However, upon further consideration, it became clear this idea would not work. Since we would be using a separate set of magnets to pick up the pieces and identify them, they would most likely interfere with each other's operation. Additionally, the pieces may be rotated during rearrangement, which would change the positions of the recognition magnets and cause pieces to be misidentified. Robustness was also not present in this design, since the sensors would only be able to read one square at a time. This would not be sufficient to detect error states such as pieces being knocked over. To resolve this, our next idea emphasized being able to read the entire board state at once, and used photoelectric sensors placed on each square to detect when the square is covered by a piece. While this design could capture the entire board state at once, it could not tell which piece was blocking the sensor. In our last design for a board with gantry on top, we planned to build the board out of transparent material, place identifying markers on the bottoms of the pieces, and suspend a camera below the board that would read the markers. This system would be able to read the entire board state and identify the pieces on each square but was ultimately infeasible due to the camera not having enough light and contrast to read the markers. At this point, we started considering placing the gantry underneath the board instead of on top, and our piece

41

recognition system design changed to accommodate this. In this design, the camera is suspended above the board, which eases the lighting requirements. The identifying markers are placed on top of the pieces to be read by the camera. As an alternative to the camera and markers, we investigated using RFID tags in the pieces, but opted not to use them due to the potential interference between 32 RFID tags. We eventually finalized our design for the physical board, with the CoreXY gantry underneath the board, and the camera for piece recognition above.

Project Timeline

ID	Name	Timeline
40	▼ Important Dates	
41	Fall Break	
59	Poster Session	
42	Midterm Design Reviews	
43	Thanksgiving Break	
44	Demo Day	
45	▼ Choose Components	
47	Choose microcontroller	
48	Choose electromagnet	
49	Choose chess pieces	
52	▼ Mechanical	
53	▼ Chess Piece Mechanism	
55	Design & Make Prototype	
60	Test w/ electromagnet	
61	Make batch	
3	▼ Design Motor Framework	
7	Decide Architecture	
46	Preliminary Sketches	
19	Research & Order Parts	
22	▼ Construct Mechanical Model	
23	Standalone System	
24	Interface with Electrical Com	
20	▼ Add Homing Mechanism	
21	Attach switches to mechanic	
26	Interface with Raspberry Pi C	
25	Test out with Motor Driver	
63	Packaging	
1	▼ Hardware	
10	▼ Motor Driver / Power Supply PC	
11	Research Components	
12	Design Prototype 1 & Layout	
13	Group Review	
14	Send out board 1	
15	Receive / Test out board	
16	Design Prototype 2 & Layout	
17	Group Review	
18	Send out board 2	
2	▼ PWM Embedded Code	
6	Research microcontroller	
27	▼ Learn / Prototype Code	
33	Look at example code	
34	Develop own code	
35	▼ Test Code	
36	Motor Driver	
37	Attach to system	
57	▼ Firmware	
58	Write to Motor Drivers Goal	
4	▼ Software	
38	Image Processing	
39	Object Detection	
5	Piece Rearrangment Algorithm	
62	Positioning Algorithm	
56	System Testing	

Figure 34: Proposed Gantt Chart

43
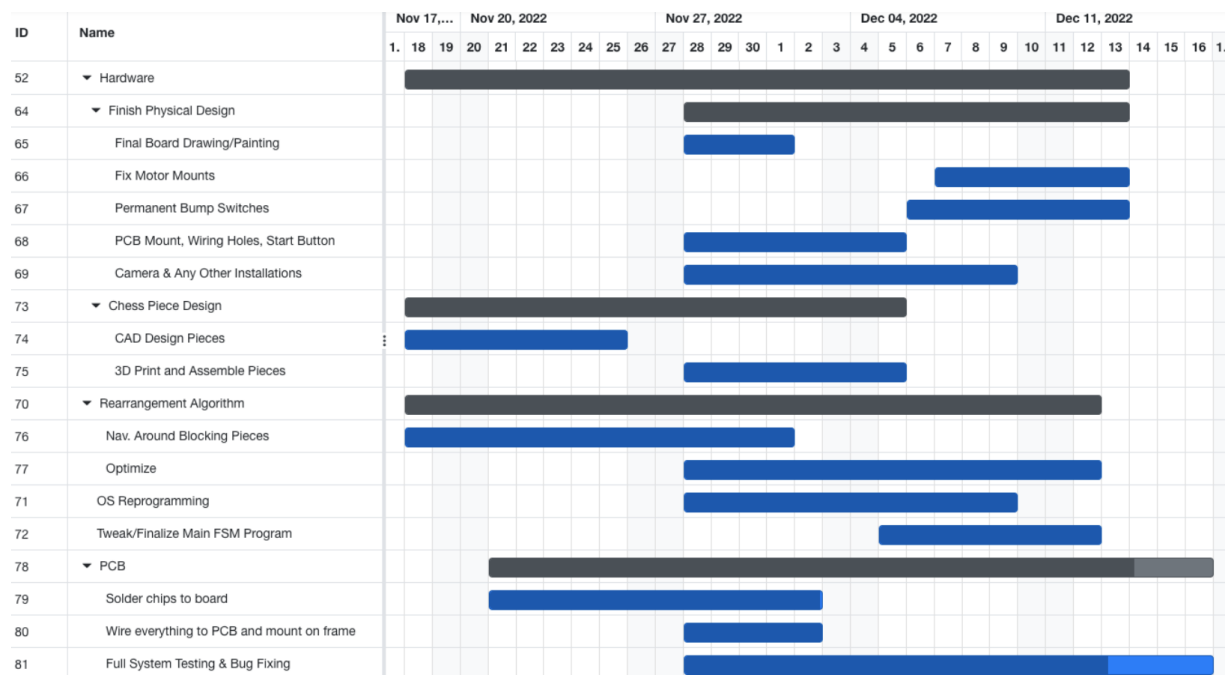
Figure 35: Updated Gantt Chart

The first proposed Gantt chart can be seen in Figure 34. The original timeline sought to complete the whole project before December 12th where the system test would have been the last task to do. We had completed all the tasks before December 12th except the PCB. Due to many problems in the PCB, we have not completed the PCB design task. Instead of using the PCB to test the whole system, we designed temporary circuits on breadboards to check other subsystems. As checking, all other subsystems work exactly like we wanted them to be and we have a full functioned automatic ReChess without the PCB to demo on December 12th. We received an extension for completing until December 16th, so we hope that we can finish the PCB on December 15th.

Bryam was the primary group member for implementing image processing with the OpenCV, researching cameras, detecting the pieces as well as their coordinates, and building the wood frame. He acted as the secondary group member for designing the PCB for the motor driver portion, 3D printing many of the parts, such as the chess pieces and the gantry parts (which Marshall designed) and setting up the breadboard for the preliminary design of the project. Bruce was the primary group member for designing the PCB that includes the power supply, PCB layout, and PCB debug. Marshall was the primary group for designing and building the gantry system, designing the bump switches and start buttons in the mechanic as well as software part. His secondary tasks are designing the chess pieces, printing 3D them, and implementing the multithreading for the software system. Selena was the primary group member for implementing the rearrangement algorithm and the embedded code for the motors and

44

CoreXY system, using Bryam image processing software and Marshall software tasks to build the whole system code. She is also responsible for testing the whole system and identify errors or malfunctions. Her secondary tasks are helping Bryam build the wood frame, painting 3D printed chess pieces, and decorating the product.

Test Plan

In our proposal and midterm design review, we did not have a well-defined test plan. During testing, we did not strictly follow a test script, but did use common sense testing procedures. Our system had three main sub-modules which were tested independently of each other: software, hardware, and mechanical. The testing performed for each module is listed below:

Software

Extensive testing was performed in order to determine the marker sizes, the resolution, and the right camera to determine whether the camera is correct or not. During the first few strides, the camera was only able to detect a few pieces, and we attempted to only make them around 7x7mm, which turned out to be too small. We gradually increased the dimensions and the resolution to determine which would be best for speed and accuracy purposes.

In terms of the motors, finding the correct speed was also essential. In testing, the delay for the step waves being too frequent sometimes made the magnet come lose and let go of the piece that was being moved. Through exhaustive testing, we found the correct delay that balanced motor speed and piece attraction.

Hardware

This testing module covered the printed circuit board, and the various subsystems on it which interfaced with other areas of the project.

Power Supply

To power the PCB, we used a 24V wall adapter which was plugged into a power jack. To test that the power jack was functioning, we used a digital multimeter, and measured the voltage at the test point connected to the jack. After successfully measuring 24V, we then measured the voltage at the test point connected to the output of the 24V boost converter. Again, we were able to measure the correct voltage, and so we moved on to testing the 5V buck converter. We again measured the expected 5V, and thus concluded that our power supplies were working.

Electromagnet Control

To test the electromagnet control, we used a breadboard to simulate the GPIO output from the Pi which would toggle the magnet on and off. The electromagnet was plugged into its connector, and the power jack was connected to supply the magnet with 5V. By connecting and

disconnecting 3.3V to the MOSFET, we were able to verify that the magnet turns on and off when supplied with the correct voltage.

<u>Motor Drivers</u>

In order to independently test if our PCB could drive the motors correctly, we again used a breadboard to simulate the outputs of the various GPIO pins. For the STEP pin which is toggled on and off repeatedly to spin the motors, we used an MSP432 [reference] to provide a square wave input which simulated the switching on and off behavior. In doing so, we found that not enough current was given to drive the motors and used the potentiometers to increase the current until the motors functioned correctly. We also tested each motor driver chip individually, before testing both drivers running concurrently. After this testing, we were able to verify that both motor driver chips were functioning.

<u>Bump Switches and Start Button</u>

To test these functionalities, we wired the switches and button to their respective connectors. We then pressed the buttons and measured the outputs that would be fed to the Raspberry Pi GPIO using a multimeter. Initially, the switch resistor values did not allow our switches to function, and after replacing them with various different values on a breadboard, we were able to successfully read 3.3V from the switches when pressed.

Mechanical

The gantry, electromagnets, and piece magnets were initially tested in parallel. First, the gantry was constructed with tape holding it down temporarily to make sure all the parts fit together. Once the motors were functional, the gantry was tested briefly with the motors wired on a breadboard. This was a hybrid test of the motor drivers and the gantry, as the tension and friction of the belts and pulleys can change how much current the motor drivers need in order to drive the gantry. Testing was difficult to get perfect with the temporary gantry setup, but performing tests this way allowed everything to be taken apart quickly in case design changes needed to be made.

Fortunately, the gantry-motor system worked the first time, so the gantry was bolted together once it was confirmed to be working. The belts were also tensioned at this time, except the frequency measurements originally planned for testing did not work very well. Instead, we simply tested the belt tensions by plucking it with our hands and listening to the sounds/watching the vibrations. So long as the tension in each belt is approximately equal, the gantry is functional [42]; any small errors in kinematics could be corrected by the bump switches or adjusted in software. The bolted gantry was tested with the electromagnet with "stress testing". We ran the gantry faster and faster with higher torques until it broke down. We used half of the maximum speed as our designated max operational speed in software.

An unforeseen issue cropped up during testing in that the electromagnet's wiring was getting tangled with the gantry belts during gantry operation. As a temporary solution, fishing

weights were crimped and then taped to the wires of the electromagnet to hold the wires down. Surprisingly, this worked so well it was kept in the final product.

In parallel with gantry testing, the electromagnet and permanent magnets for the chess pieces needed to be tested. Because the pieces hadn't been designed yet, our team found chess pieces of approximately equal weight and size and used those as a substitute. This turned out to be a mistake, as designing the chess pieces first would have resulted in a better final product. Nevertheless, we purchased a variety pack of permanent neodymium magnets with different strengths and sizes.

With each magnet, we emulated the motion of a gantry by hand, holding the electromagnet underneath our chess board while on top of the board was a chess piece and magnet. With the magnet moving at our planned gantry speed, it was considered functional, and was put in a list of functional magnets. Out of those functional magnets, we then put two magnet-piece pairs next to each other and measured how close they could get without repulsing each other and knocking one another over. This is a key part of testing our permanent magnets because our software was designed to slide chess pieces between each other. Thus, we needed to find permanent magnets that can slide between the predetermined gaps between chess pieces without knocking pieces over.

With proper permanent magnets selected and the gantry bolted down, the gantry-magnet system was functional enough to test the software that rearranged one or two pieces. However, the bump switches needed to be tested before full system testing, as the piece rearrangement tended to accumulate error after four or five cycles.

The bump switches and start button were first tested on a breadboard to make sure the software logic was working properly. With proper code guaranteed, the bump switches were then temporarily glued into their respective places and tested with software. The bump switches were found to be finicky at first, but after removing the debouncing code (which turned out to be unnecessary), all the buttons were working as intended. Once the PCB arrived, all the buttons were tested on the PCB rather than just the breadboard. Though our PCB itself was having issues, the testing didn't show any issues in the switches or the circuitry that dictated their behavior.

Integration Testing

After verifying the operation of the individual parts of the project, we began integrating and testing them. We connected the Raspberry Pi and motors to a breadboard, where the motor driver chips were placed. With this setup, we were able to test the operation of the motors using our written code and verify that the software and mechanical portions of our project functioned.

Final Results

Overall, our project was a success. During the demo session, our project was functioning for the entire three hours. The final product is shown below in Figure 35.

Figure 36: ReChess Finished

Users were able to move pieces away from their starting positions, and even play games on the board, before initiating the rearrangement process. They were able to watch as the robot moves pieces around the board, navigating them to their starting positions. They could even move pieces during rearrangement, and the board would detect the new pieces' positions, and include them in the rearrangement. Now, we will evaluate our final product based on the criteria outlined in our proposal, shown in the table below.

Letter Grade	Criteria
A+	- An image processing script that can detect and label out a chessboard. - Object detection script that identifies chess pieces. - An algorithm that finds the movements required to rearrange the pieces and outputs commands for the motor. - Embedded code to adjust motors' PWM

	• A working gantry system that is able to move to any x,y pair in a chessboard and electronically move pieces via a magnet system. • A circuit that is able to deliver power to the whole system • A printed circuit board that is able to correctly drive two stepper motors.
A	Device does not perform one of the tasks required for an "A+"
A-	Device does not perform one of the tasks required for an "A+" AND one other task only works partially
B+	Device does not perform two of the tasks required for an "A+"
B	Device does not perform three of the tasks required for an "A+"
B-	Device does not perform four of the tasks required for an "A+"
C	Device does not perform five of the tasks required for an "A+"
D	Device does not perform six of the tasks required for an "A+"
F	Device does not perform seven of the tasks required for an "A+"

Criterion 1: An image processing script that can detect and label out a chessboard

Our final product met this criterion. Our code for image processing was able to successfully detect the outline of the board, and used this to determine the boundaries for where to look for pieces.

Criterion 2: Object detection script that identifies chess pieces

The product was able to detect and identify pieces on the board with **100%** accuracy during the demo.

Criterion 3: An algorithm that finds the movements required to rearrange the pieces and outputs commands for the motor

Our algorithm successfully rearranged the board every time it ran. The motors never made a misstep, and the board always ended up in the correct state.

Criterion 4: Embedded code to adjust motors' PWM

Our code preemptively adjusted the motor's stepping resolution and controlled the speed of the motor. Although this was not demoed, adjusting the resolution of the motors in the code would change the speed at which the motors moved. We used these characteristics to determine the speed and delay time to move the motors to match with the system. The full functioned motors system can be a reference for the embedded code working.

Criterion 5: A working gantry system that is able to move to any x,y pair in a chessboard and electronically move pieces via a magnet system

Our gantry system also worked according to our expectations. It was able to move to any piece on the board, regardless of its location, and successfully move it to any position around the board as well.

Criterion 6: A circuit that is able to deliver power to the whole system

In testing, our circuit board was able to successfully power the motor drivers and electromagnets.

Criterion 7: A printed circuit board that is able to correctly drive two stepper motors

In testing, our circuit board was able to successfully drive the two stepper motors. The operation of the motors matched our code. This meant that when we ran code for the motors to move the gantry up, left, down, and right, the motor performed those actions.

Costs

For our project, we went slightly over the $500 budget provided to us. A general breakdown of the costs may be found in Appendix A, with each category of costs broken down further in the following appendix entries. Most of our budget went toward ordering the parts for the gantry, such as the metal bars along which the gantry would move. Part orders for the PCB were the second most expensive component of the project. The remaining budget was used to pay for PCB manufacturing and the soldering done by 3W.

If we were to mass-produce our project in 10,000-unit quantities, our PCB fabrication and assembly costs would scale accordingly. Each board fabrication costs $33 individually, and the soldering done by 3W also costs $55 per board. Since each finished product would require a PCB, we can multiply the cost for one board by 10,000 and find that simple construction of the boards would cost $880,000. Each product would also require a gantry system, which would cost $2,445,600 in total. PCB parts would then add approximately $800,000 to the total cost. We do not believe that automated equipment could be used to assemble our device, as it required very precise adjustments, such as tuning the current supplied to the motors via the potentiometers and tensioning the belts on the gantry. Thus, labor costs from assembly of the products would also need to be accounted for.

Future Work

In the future, if this project were to be expanded or improved, there are many directions which could be taken. Currently, our board only performs rearrangement, and, excluding this feature, the board is identical to a normal chess set. The most interesting feature that could be

added is the ability to actually play games of chess on the board autonomously. Our initial vision for the project included this, but the feature was removed from our proposal to accommodate time constraints. There are some minor improvements that could be made, such as finding a better method of reducing friction on pieces, or further refining the rearrangement algorithm to optimize time taken. Additionally, at the start of the project, we had not foreseen the difficulties we would encounter with the PCB. It is advised that as much time as possible is dedicated to designing the PCB, such that the first board send out deadline is met. Additionally, frequent checks with the teaching staff should be performed to ensure the validity of the designs. It would also be wise to consider the budget more carefully at the beginning, as by the end of the project, money was very tight.

References

[1] H. J. R. Murray, "History of chess". Skyhorse Publishing, 2015.

[2] J. Siikala, P. M. Autio, T. Tammaisto, and H. Wilenius, "Culture and history in the Pacific". Helsinki: Helsinki University Press, 2021.

[3] "Beyond the limits: flight enters the computer age," *Choice Reviews Online*, vol. 27, no. 02, pp. 27-092627-0926, Oct. 1989, doi: 10.5860/choice.27-0926.

[4] G. Matthew Sadler, "The TCEC16 computer chess superfinal: A perspective," *ICGA Journal*, vol. 41, no. 4, pp. 253–258, Feb. 2020, doi: 10.3233/icg-190123.

[5] C. Matuszek et al., "Gambit: An autonomous chess-playing robotic system," 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 4291-4297, doi: 10.1109/ICRA.2011.5980528.d

[6] "2 & 4 Layer PCB Special Pricing Options," *Advanced Circuits*. [Online]. Available: https://www.4pcb.com/pcb-prototype-2-4-layer-boards-specials.html. [Accessed: 13-Dec-2022].

[7] J. Adams, "Introducing Raspberry Pi HATS," *Raspberry Pi*, 18-Sep-2021. [Online]. Available: https://www.raspberrypi.com/news/introducing-raspberry-pi-hats/. [Accessed: 13-Dec-2022].

[8] WWW Electronics Incorporated, "Manufacturing." [Online]. Available: https://3welec.com/#Manufacturing. [Accessed: 13-Dec-2022].

[9] Raspberry Pi, "Raspberry pi 4 model B specifications," *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. [Accessed: 13-Dec-2022].

[10] "Zyltech Engineering LLC," [Online]. Available: https://www.zyltech.com/. [Accessed: 13-Dec-2022].

[11] "Linear Motion," *McMaster-Carr*. [Online]. Available: https://www.mcmaster.com/linear-motion/. [Accessed: 13-Dec-2022].

[12] *KiCad*. [Online]. Available: https://www.KiCad.org/. [Accessed: 13-Dec-2022].

[13] "FreeDFM," *FreeDFMNet*. [Online]. Available: https://www.my4pcb.com/net35/FreeDFMNet/FreeDFMHome.aspx. [Accessed: 13-Dec-2022].

[14] *FreeCAD*. [Online]. Available: https://www.freecadweb.org/. [Accessed: 13-Dec-2022].

[15] UltiMaker. [Online]. Available: https://ultimaker.com/. [Accessed: 13-Dec-2022].

[16] MakerBot. [Online]. Available: https://www.makerbot.com/3d-printers/replicator/ [Accessed: 13-Dec-2022]

[17] Raspberry Pi, "Raspberry pi OS," *Raspberry Pi Documentation*. [Online]. Available: https://www.raspberrypi.com/documentation/computers/os.html. [Accessed: 13-Dec-2022].

[18] Microsoft, "code editing. redefined," *Visual Studio Code*, 03-Nov-2021. [Online]. Available: https://code.visualstudio.com/. [Accessed: 13-Dec-2022].

[19] "OpenCV-python tutorials," *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html. [Accessed: 13-Dec-2022].

[20] A. Kumari, M. K. Jha, and D. D. Pathak, "An innovative environmental process for the treatment of scrap ND-Fe-B magnets," *Journal of Environmental Management*, vol. 273, p. 111063, Nov. 2020.

[21 M. Griffin, "Is pla recyclable? – simply answered," *All3DP*, 20-Jul-2021. [Online]. Available: https://all3dp.com/2/is-pla-recyclable/. [Accessed: 13-Dec-2022].

[22] Suzuki G, Uchida N, et al., "Mechanical recycling of plastic waste as a point source of microplastic pollution," Environmental Pollution, vol. 303, p. 119114, 2022.

[23] "5718 series5718x-01shybrid Stepper Motor," 5718X-01S | Hybrid Stepper Motor | Lin Engineering. [Online]. Available: https://www.linengineering.com/products/steppermotors/hybrid-stepper-motors/5718-series/5718X-01S.

[24] S. K. Das, J. A. S. Green, and J. G. Kaufman, "Aluminum Recycling: Economic and Environmental Benefits," 2011

[25] "Small parts for toys and children's products business guidance," U.S. Consumer Product Safety Commission. [Online]. Available: https://www.cpsc.gov/Business--

Manufacturing/Business-Education/Business-Guidance/Small-Parts-for-Toys-and-Childrens-Products.

] "OpenCV library," *Opencv.org*, 2019. https://opencv.org/

Raspberry Pi Foundation, "Raspberry Pi — Teach, Learn, and Make with Raspberry Pi," *Raspberry Pi*, 2019. https://www.raspberrypi.org/

G. Bradski, "OpenCV." [Online]. Available: http://roswiki.autolabor.com.cn/attachments/Events(2f)ICRA2010Tutorial/ICRA_2010_OpenCV_Tutorial.pdf

"Core[X,Y]," *CoreXY*. [Online]. Available: https://corexy.com/theory.html. [Accessed: 13-Dec-2022].

"SS-5GL2," *Digi-Key*. [Online]. Available: https://www.digikey.com/en/products/detail/omron-electronics-inc-emc-div/SS-5GL2/137204. [Accessed: 13-Dec-2022].

"2266," *Digi-Key*. [Online]. Available: https://www.digikey.com/en/products/detail/pololu-corporation/2266/10449890?s=N4IgTCBcDa4IwA4DMBaMYBsGUDkAiIAugL5A. [Accessed: 13-Dec-2022].

Promega, "Belt tensioning," *Promega*. [Online]. Available: https://promega.printm3d.com/repair-and-maintenance/belt-tensioning. [Accessed: 13-Dec-2022].

Appendix

Expense	Balance	Cost	New Balance
Non-PCB Misc. Parts	$500.00	$14.00	$486.00
Gantry Parts Order	$486.00	$244.56	$241.44
PCB Parts Order 1	$241.44	$78.18	$163.26
PCB Parts Order 2	$163.26	$57.85	$105.41
PCB Sendout 1	$105.41	$33.00	$72.41
PCB Sendout 2	$72.41	$33.00	$39.41
W3 Assembly	$39.41	$55.00	-$15.59

Appendix A: Project cost breakdown

Index	Manufacturer Part #	Digikey Part #	Qty in Stock	Qty Req'd	Per Unit Price	Cost

1	3,873	1528-2689-ND	51	1	$9.95	$9.95
2	8,173	469-1048-ND	2,298	5	$0.81	$4.05
					Total Cost	$14.00

Appendix B: Miscellaneous non-PCB parts order

Index	Manufacturer Part #	Digikey Part #	McMaster Part #	Open Builds Part #	Amazon	Zyltech	Qty in Stock	Qty Req'd	Per Unit Price	Cost
1	1704HS168A-OB			623			N/A	2	$17.98	$35.96
2	EXT-2040-REG-700-VGRV					EXT-2040-REG-700-VGRV	N/A	2	$11.45	$22.90
3	EXT-2020-VGRV					EXT-2020-VGRV	N/A	2	$7.45	$14.90
4	EXT-CUT-SERV					EXT-CUT-SERV	N/A	2	$1.50	$3.00
6	HW-20CARR-BLK					HW-20CARR-BLK	N/A	3	$9.95	$29.85
5	$2,266.00	2183-2266-ND					318	2	$4.95	$9.90
7	LXMY0382				B07L9QDVC8		N/A	1	$13.99	$13.99
8	$210.00			210			N/A	2	$5.99	$11.98
9	470-By-the-Foot			470-By-the-Foot			N/A	20	$2.49	$49.80
10	90-Pack			90-Pack			N/A	1	$3.39	$3.39
11	225-Pack			225-Pack			N/A	1	$3.89	$3.89
12	$5.00			5			N/A	4	$0.31	$1.24
13	760-Pack			760-Pack			N/A	1	$1.99	$1.99

Index	Manufacturer Part #	Digikey Part #	Mouser Part #	Qty in Stock	Qty Req'd	Per Unit Price	Cost
14	140-Pack		140-Pack	N/A	1	$1.79	$1.79
15	$185.00		185	N/A	2	$0.19	$0.38
16	536-Pack		536-Pack	N/A	2	$2.99	$5.98
17	10-Pack		10-Pack	N/A	2	$0.99	$1.98
18	490		490	N/A	2	$1.49	$2.98
19	946-Pack		946-Pack	N/A	2	$0.99	$1.98
20	485		485	N/A	4	$1.49	$5.96
21	922-Pack		922-Pack	N/A	2	$1.19	$2.38
22	2165-Pack		2165-Pack	N/A	1	$2.79	$2.79
26	135-pack		135-pack	N/A	1	$1.69	$1.69
23	92005A120	92005A120		N/A	1	$4.26	$4.26
24	92005A223	92005A223		N/A	1	$6.87	$6.87
25	90695A035	90695A035		N/A	1	$2.73	$2.73
						Total Price	$244.56

Appendix C: Gantry parts order

Index	Manufacturer Part #	Digikey Part #	Mouser Part #	Qty in Stock	Qty Req'd	Per Unit Price	Cost
1	CSRN2512FKR400	CSRN2512FKR400CT-ND		14,739	10	$0.45	$4.54
2	12065C103KAT2A	478-1542-1-ND		262,764	13	$0.15	$1.95
3	$885,012,208,034.00	732-7690-1-ND		3,337	4	$0.20	$0.80
4	CL31B104KBCNNNC	1276-1017-1-ND		1,376,463	4	$0.11	$0.44
5	35211M0FT	A116104CT-ND		3,459	4	$0.57	$2.28

				Qty in Stock	Qty Req'd	Per Unit Price	Cost
6	50THV10 0M10X10. 5	1189-2059 -1-ND		221,143	4	$0.89	$3.56
7	TC33X-2- 103E	TC33X-10 3ECT-ND		115,772	4	$0.28	$1.12
8	RMCF120 6JT10K0	RMCF120 6JT10K0 CT-ND		1,747,089	15	$0.03	$0.38
9	$192,020, 013.00	WM18731 -ND		75,247	4	$0.33	$1.32
10	$5,444,26 2.00	277-11313 -ND		109,498	1	$1.00	$1.00
11	$5,447,86 1.00	277-11343 -ND		81,238	1	$1.00	$1.00
12	UCLAMP 3301H.TC T	UCLAMP 3301HCT- ND		118,654	2	$0.64	$1.28
13	SS-5GL2	SW156-N D		36,445	2	$3.10	$6.20
14	PR144C1 900	EG4699- ND		6,503	1	$2.30	$2.30
15	$2,181,12 0,204.00	900-21811 20204-ND		427	3	$1.92	$5.76
16	$533,980, 271.00	WM7606 CT-ND		206,079	3	$0.68	$2.04
17	RMCF120 6FT1K00	RMCF120 6FT1K00 CT-ND		2,944,824	6	$0.10	$0.60
18	C1206C10 3KAREC AUTO	399-17174 -1-ND		6,255	3	$0.23	$0.69
19	GSM60U 24-P1J		709-GSM 60U24-P1 J	1,039	1	$37.76	$37.76
20	PJ-070BH -SMT-TR		490-PJ-07 0BH-SMT -TR	838	2	$1.58	$3.16
						Total Costs	$78.18

Appendix D: First PCB parts order

Index	Manufact urer Part #	Digikey Part #	Mouser Part #	Qty in Stock	Qty Req'd	Per Unit Price	Cost

1	VL-HDW-1 01	1241-1053 -ND		106	1	$5.00	$5.00
2	$2,223.00	1528-1385 -ND		284	1	$2.50	$2.50
3	$61,300,3 11,121.00	160-1404- 1-ND		750,211	2	$0.13	$0.26
4	QPC02SXG N-RC	S9337-ND		988,614	2	$0.10	$0.20
5	160-1404- 1-ND	LTST-C150 KGKT		750,211	3	$0.35	$1.05
6	B540C-13- F	B540C-FDI CT-ND		178,568	4	$0.54	$2.16
7	$705,430, 038.00	WM4826- ND		51,465	1	$1.18	$1.18
8	$533,980, 271.00	WM7606C T-ND		184,518	3	$0.65	$1.95
9	IRLZ44PBF	IRLZ44PBF -ND		944	2	$3.01	$6.02
10	UCLAMP3 301H.TCT	UCLAMP3 301HTR-N D		60,840	3	$0.64	$1.92
11	$5,001.00	36-5001-N D		1,643,773	25	$0.34	$8.52
12	EG1218	EG1903-N D		79,551	1	$0.76	$0.76
13	$5,444,26 2.00	$5,444,26 2.00		196,337	1	$1.00	$1.00
14	SRR1210- 390M	SRR1210- 390MCT-N D		2,438	1	$1.41	$1.41
15	SRN8040- 8R2Y	SRN8040- 8R2YCT-N D		16,096	1	$0.78	$0.78
16	RNCF0805 DTE330R	738-RNCF 0805DTE3 30RCT-ND		15,327	1	$0.14	$0.14
17	RNCP0805 FTD200R	RNCP0805 FTD200RC T-ND		124,279	2	$0.10	$0.20
18	EMK316B BJ476ML- T	587-5425- 1-ND		408,452	1	$0.64	$0.64

19	C0805B10 6K050T	4587-C08 05B106K0 50TCT-ND		11,997	1	$0.47	$0.47
20	EEE-FN1H 680XP	10-EEE-FN 1H680XPC T-ND		7,680	2	$0.87	$1.74
21	08055A20 1JAT2A	478-10509 -1-ND		75,496	1	$0.23	$0.23
22	C2012C0G 1H822K06 0AA	445-14325 -1-ND		4,479	1	$0.30	$0.30
23	CC0805KR X7R8BB15 4	311-4315- 1-ND		5,721	1	$0.19	$0.19
24	CL21B474 KOFNNNG	1276-6483 -1-ND		245,311	2	$0.10	$0.20
25	C0805C10 4M5RAC7 800	399-C080 5C104M5 RAC7800C T-ND		821,952	4	$0.10	$0.40
26	C1206C10 3KARECAU TO	399-17174 -1-ND		6,138	3	$0.23	$0.69
27	CL21B103 KBANNNC	1276-1015 -1-ND		3,259,318	6	$0.10	$0.60
28	RMCF120 6JT10K0	RMCF120 6JT10K0C T-ND		1,087,474	4	$0.10	$0.40
29	AC1206JR- 071KL	YAG3927C T-ND		37,787	7	$0.10	$0.70
30	RMCF080 5FT1K24	RMCF080 5FT1K24C T-ND		19,174	1	$0.10	$0.10
31	CRCW080 53K16FKE A	541-3.16K CCT-ND		7,874	1	$0.10	$0.10
32	RNCP0805 FTD1K50	RNCP0805 FTD1K50C T-ND		147,544	1	$0.10	$0.10
33	RMCF080 5FT28K0	RMCF080 5FT28K0C T-ND		87,898	1	$0.10	$0.10

34	RMCF080 5FT19K1	RMCF080 5FT19K1C T-ND		105,827	1	$0.10	$0.10
35	RMCF080 5FT100K	RMCF080 5FT100KC T-ND		4,776,317	1	$0.10	$0.10
36	RMCF080 5JT10K0	RMCF080 5JT10K0C T-ND		7,385,822	2	$0.10	$0.20
37	LM2586S-ADJ/NOPB		926-LM25 86S-ADJ/N OPB	1,013	1	$10.00	$10.00
38	TPS54334 DRCR		595-TPS54 334DRCR	8,115	2	$2.72	$5.44
						Total Costs	$57.85

Appendix E: Second PCB parts order