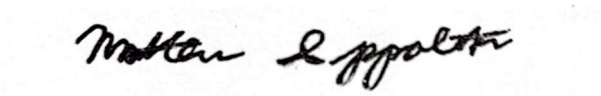
E.M.N.E.M Checkers Bot

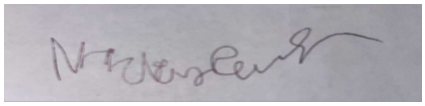
Emmanuel Erhabor, Matthew Ippolito , Matthew Hankins and Nicholas Palmer

12/13/2022

Capstone Design ECE 4440 / ECE4991

Signatures





Statement of Work:

Emmanuel Erhabor:

As a part of this team, my primary responsibility lay within the realm of Hardware design and construction. I took the lead on the majority of the design blocks for our PCB. I used KiCad [1] to create schematics for the motor drivers, solenoid and electromagnet drivers, as well as the LEDs and buttons for I/O. In this process, I conducted in depth searches for parts that met the constraints we placed on each component. Apart from circuit design, I created all of the wiring necessary for our Checkers playing robot. I arranged and organized everything concerning the presentation of the wires running to and from our PCB so that it was neat and professional.

I fixed a lot of the hardware issues we came across with connectivity testing and cross referencing the schematic. I had to desolder our solenoid driver and 5V regulator because they had issues as well as alter the design of the LED drivers to get them to work. I also spent considerable time selecting and purchasing the buttons and LEDs that would make up our user interface. It took a lot of discussing as a team to determine which buttons were necessary and which were not.

Matthew Ippolito:

My primary responsibilities within the team were to design and construct the gantry, as well as write the code for the piece imaging and a FSM for the Raspberry Pi [2] side of the code. I created a 3D CAD model in Solidworks[22] using OpenBuilds parts to design the gantry. I also selected an electromagnet and solenoid to use for picking up the checkers pieces and created models of them to be inserted into the assembly. I also designed the pieces, which required several iterations until acceptable pieces were created. Slicing and printing any parts which were not being ordered was also my responsibility as gantry designer.

For the piece imaging, I wrote functions to interface with the camera, identify and select groups of pixels which were likely part of a piece, and then to cluster these groups into pieces. More functions were created to turn these pieces into a boardstate that the bot could understand. I also created the FSM for the Raspberry Pi [2]. The job of the FSM was to ping the MSP430 [3] for updates on button presses and status of piece movements, then calculate a desired board state by either generating a move from the bot or undoing an error move, and then calculate moves that the MSP430 [3] could understand to create that boardstate.

Nicky Palmer:

My primary responsibilities for this project were in the design of our PCB, as well as the embedded code to control the peripheral devices. I created the top-level schematic of our PCB and organized how each of our sub systems would interact and connect with each other and the MSP430 [3]. Additionally, I also created schematics for our 5V and 3.3V regulators. Once the schematic was done, I contributed to laying out, routing, and having our board pass FreeDFM's checks. Once we had the board returned to use, I spent a lot of time debugging and working on validating the PCB. I wrote code to test all of the peripheral devices and their respective drivers.

On the second revision, I helped to diagnose and address an issue with our solenoid driver circuit.

Additionally, I worked on creating the embedded code to control many of the peripheral devices such as the motors, solenoid, and electromagnet. The code I wrote allowed us to adjust the duty cycle on our solenoid, tell the motors how many steps to move and in which direction, as well as turning the electromagnet on and off.

Matthew Hankins:

My primary responsibilities for this project were the algorithm for generating moves, the UART communication between the Raspberry Pi [2] and MSP430 [3], interfacing with the buttons and LEDs, and the overall MSP-side finite state machine (FSM). Originally, I wanted to use an open-source Python implementation. However, I found it to be too slow and inefficient. I ended up writing the bulk of a C++ [4] implementation from scratch but basing the evaluation functions off the original Python implementation. I made adjustments to the evaluation functions to ensure only integer math was used. In the end, this switch yielded a ~4,000 times increase in speed, allowing the bot to look up to 10 moves into the future in the time the Python implementation was looking 3 moves into the future. Additionally, I wrote a graphical utility for generating board states to aid in testing the bot.

On the MSP430 [3] side, I wrote an overall FSM to sequence the order of solenoid, electromagnet, LED, and motor actions. I also implemented the UART code on the Raspberry Pi [2] and MSP430 [3]. I wrote the logic specific to each LED and button. The LEDs needed to properly respond to indicate: major faults, that the motors will begin to move soon, when the human player is able to capture and when it is waiting for user input.

Table of Contents

Table of Figures

Table of Tables

Abstract

The checkers robot is a motor controlled gantry which can set up and play a physical game of checkers with a human opponent. It picks up and moves checkers pieces in reaction to its opponent's moves to play a legal and complete game of checkers. The robot uses a camera to locate pieces, and is able to identify invalid or illegal moves and undo them.

Background

This project was selected because it presents interesting and varied challenges. We implemented a computer vision system to detect the game state, integrated a checkers AI, built the physical mechanism and actuated it. Additionally, the deliverable is satisfying and fun to demo.

In reviewing similar projects, we found a variety of checkers playing robots. However, since the design space is expansive, there are many places for deviation. A checkers robot from a New Zealand university was incapable of distinguishing between enemy kings and men and made random legal moves [5]. We implemented a complete piece detection regimen, capable of detecting men and kings from either color and integrating a much more effective checkers engine. A paper out of MTI University focuses on the game engine at the cost of utilizing an expensive pre-made robotic arm and GPU usage [6]. A paper from UMass Lowell utilizes a more complex, two-armed robot and 3D cameras but seems to lack robustness, failing on 10% of attempted moves and requiring an operator to make corrections to the board when necessary [7]. In summary, our project is distinct from those that we researched in its planned robustness, cost benefit from avoiding pre-made robotic arms, and running in an embedded environment.

We drew from a wide variety of coursework. Introduction to Engineering (ENGR 1624) for mechanical design and CAD. Embedded Computing and Robotics I, II (ECE 3501, ECE 3502) for motor control and general embedded programming experience. For the Raspberry Pi [2], we will program in Python, drawing on Introduction to Programming (CS 1110). Machine Learning and Image Analysis (ECE 4501) and Algorithms (CS 4102) for image analysis. FUN I, II, III (ECE 2630, 2660, 3750) for PCB design and LED driving.

Physical Constraints

Design Constraints

With the design of our project there were a few constraints that we had to account for. Several came up with the design for our board. Due to the ongoing parts shortage, we had to design our PCB around the parts available. The motor and solenoid drivers we used were the ones that the department had in stock. Additionally, due to limited supply we were unable to get

multiple copies of our 5V switching regulator so for the second revision of our board we had to desolder and resolder the regulator from the first board to the second. In order to house our board, we purchased a NEMA rated box. The box has dimensions of 11.6”x9.6”x4.5” which means that our PCB needs to be small enough to be able to fit inside of the box.

In addition to the physical constraints, we had constraints on our software. We wanted the robot to be capable of making a move in under 15 seconds. This was broken into 5 seconds to run the image analysis and move generation and 10 seconds for the MSP430 [3] to execute the move. Both the image analysis and move generation are computationally intensive. We were originally using Python implementations for both of these but had to switch both over to C++ [4] in order to meet our original specifications.

Cost Constraints

There were a few cost constraints that we had to consider. Our biggest cost constraint was the price of a Raspberry Pi [2]. If we bought one directly from the distributor the price would be 35$. Unfortunately, due to chip shortages and supply chain issues you can’t purchase a Raspberry Pi [2] directly from the manufacturer unless you are willing to wait a year. On the second-hand market the price is almost 5 times greater at 150$ each. Fortunately for us, one of our group members happened to have a spare Raspberry Pi [2] that we could use so we did not have to purchase one second hand.

Another cost constraint that we encountered was our 5V regulator. Each 5V regulator was expensive, coming in at $17 each so we made the decision to buy only a single regulator for both of our boards. The idea was to desolder the regulator from the first board and resolder it to the second. We were able to successfully perform this, but while we were testing the board, one of the low voltage lines got shorted to 12V and destroyed both an MSP430 [3] and the 5V switching regulator. Due to not having an extra regulator we were forced to use the power supply for the Raspberry Pi [2] to power all of our 5V components.

Tools Employed

In order to complete this project, we used a number of different tools. When designing our PCB, we used KiCad [1], an open source, free, electronic design automation (EDA) tool. This helped us to create the schematic, as well as layout and route our circuit board. Creating our PCB was a collaborative process. In order to have multiple people work on the same board we used the online tool CADLab [8]. CADLab [8] is a git-based tool, very similar to GitHub, but it is primarily used for the parallelization of PCB [9] design instead of software. It was very useful being able to have multiple people have access to the newest changes seamlessly.

While we had CADLab [8] for sharing board design, we also used GitHub for sharing and working on code at the same time. It allowed multiple people to work on our code base seamlessly. Our real-time embedded system was developed using C in Texas Instruments Code Composer Studio IDE. We are able to run C code on the MSP430 [3] that we are using for

deterministic processes. In order to deploy our code onto the MSP we used Code Composer Studio. This allowed us to more efficiently flash and debug our code and allow us to go step by step in the code's execution. The image and game code were developed using C++ [4]. This allowed us to develop fast and efficient code to run our checkers algorithm. Our image processing code is based upon the OpenCV [10] C++ [4] library. OpenCV [10] allowed us to handle images better, as well as align the checkers game board and perform matrix operations. CMake was required with using the OpenCV [10] library in order to properly build and deploy our code onto the Raspberry Pi [2],

With a lot of the software and tools we used there was a learning curve to it. None of us had every used KiCad [1] before so we had to spend time familiarizing our self with the user interface, learning all of the keyboard shortcuts, and understanding what tools KiCad [1] had at its disposal for us to use. Additionally, not everyone was familiar with Git and CADLab [8] so we had to practice using git in order to prevent merge conflicts. Additionally, although CADLab [8] is very helpful for having multiple people work on the same PCB. Merge conflicts happen when multiple people edit the same file, which results in work being lost.

Societal Impact Constraints

Environmental Impact

The impact from our project comes from where we source our materials and whether we do so responsibility and sustainability as well as how we dispose of our electronics waste. There is a significant impact to the environment from PCB manufacturing. A large number of heavy metal ions that are used in PCB manufacturing such as copper and tin can leak into the water supply. These elements are harmful to plants and animals that live in the watershed [12].The most important thing we did to lessen our environmental impact is to recycle components and to reuse anything that we could. We reused the [11] Stepper motors, Raspberry Pi [2] and power supply from other projects, lessening our environmental impact. Additionally, Our device has a substantial energy consumption. Our robot needs 10A of current at 12V in order to function or 120W.

Sustainability

Construction of the robot used a fair amount of plastic and non-recyclable materials. Many of the components were 3D printed and made of plastic. Most of the metal structural components are commonly found and are easily recyclable for other projects or for scrap metal. It is necessary to have printed circuit boards to control all of the peripherals and PCBs have some issues with sustainability. They use rare earth elements that are bad for the environment and copper that if not recycled properly, could potentially leak into the environment. We did however use lead-free solder in order to minimize the amount of lead waste.

Health and Safety

Our primary safety concern was how the gantry handles human interference. We needed to make sure that when the gantry was moving pieces, it did not injure or hurt a person who might be in the way of movement. We included flashing lights to warn users of when the gantry was in motion. Additionally, another safety concern that we had is that we are going to be dealing with 120V, 60Hz wall power and we want to be sure that we deal with it carefully, so no one is at risk of getting shocked. We put our PCB inside of a NEMA rated enclosure to protect the surroundings from the high voltage source. Finally, we had to take into account the size of the checkers pieces as it might present a choking hazard for children. We did this by making the pieces big enough for the children not to swallow.

Ethical, Social, and Economic Concerns

There are no critical ethical issues pertaining to our device. The only ethical issues that we must consider is who is the end user for our product and to make sure that they are planning on using the product ethically. We would achieve this by including a note to potential customers detailing that the device is meant for recreational use within a certain age range (10+), so our intentions in creating the product are clear. Due to the cost of creating this device it would be feasible for schools or organizations to purchase but it wouldn't be realistic as a tool to help economically disadvantaged families. The cost of production is too high to justify purchasing it to play checkers when there are much cheaper alternatives.

External Considerations

External Standards

There are several standards that we are going to have to follow to ensure that our project is safe and conforms to best practices. The first standard that we're going to address is the Consumer Product Safety Commission of the U.S government [13]. It says that if an object fits into a cylinder 2.25" high and 1.25" wide, then it is classified as a choking hazard and must be classified for 3 years of age and over. We must abide by this as our checkers pieces are small enough to be considered choking hazards.

We must also follow the official tournament rules of checkers. This is because if we are making a checkers robot, in order to play checkers properly, it has to abide by international guidelines [14] so that it won't be confused by colloquial house rules.

Additionally, since there are going to be electronic PCBs in our design, we have to abide by NEMA design standards for PCB enclosures [15]. We are going to be abiding by NEMA type

1 standard since the PCBs are going to be primarily indoors and need to be shielded from dust and dirt.

There are also ANSI standards [16] for robot manufacturers that we must follow. This standard provides guidelines for robot manufacturers to ensure safety for operators. This will help us to create safe use conditions and to minimize potential risk while using the robot.

When designing and laying out our PCB we needed to keep IPC standards in mind in order to ensure that it would be able to be manufactured properly. Particularly standard IPC 2221B. This standard is about PCB trace spacing and clearance [17]. We were able to set design rules in our eCAD software that allowed us to abide by these industry standards.

Intellectual Property Issues

Our project has the potential to be patented. There are not a lot of patents related to a checkers playing robot, and the patents that are related are only loosely related about different sub-components of the system. One patent that is similar to our design is patent number 6257578, ABC checkers and other checkers games [18]. This patent describes patenting a checkers board, but instead of being 64 squares comes in either 100 or 144 squares[18]. The caveat is that the extra squares contain geographical or historical facts. While this patent patents a variation of checkers, we don't need to concern ourselves with conflicting claims since we use a standard checkers board which is not patented and do not use historical facts in our game at all. Specifically, their first claim describes a 12 by 12 board of alternating color squares with the letters A, B, and C imprinted onto it [18]. We use a different board so our claim will not conflict with this patent.

Another patent with potential conflicts is patent number 4398720, Robot computer chess game [19]. This patent is for a robotic chess arm, and magnetized chess pieces. The robotic chess arm can function as a chess opponent and perform all of the same actions that a human player would be able to make. While this patent and our own invention both have similar spirits of acting as a robotic opponent for a board game, they are implemented in different ways. Specifically, claim 8 describes the system used for picking up and moving chess pieces as "a plurality of spring biased arms and a cam means for opening the arms against the spring pressure." [19]. Our implementation uses a gantry and a solenoid to pick up and move pieces, while this patent relies on a robotic arm to move pieces. Additionally, our piece recognition system is completely different. We use a camera to find out where clusters of different colors are to recognize pieces while the chess robot uses magnetic pieces to actuate a switch in order to determine if there is a chess piece on a specific tile according to claim 19 [19]. This is much different than our method of piece recognition so there are no conflicts there.

Finally, the first patent that we took a look at was patent number 20180215590, Belt Drive Dual Robot Gantry [20]. This patent describes using a gantry to position one or more robots above a base platform. This patent is used primarily by Boeing for assisting in fuselage assembly. While the intended application of the two designs is different, they share a lot of

similarities in the implementation. The biggest similarity between the two is the fact that both use a gantry to maneuver around an area. Claim 1 of this patent claims that this patent is an apparatus for moving robots consisting of a base platform, robot, and gantry [20]. While having similar purposes there are differences between the two designs. The main difference is the fact that our robot is the gantry, we aren't using a gantry to move a robot around. Another source of concern however for a patent conflict is claim 8, that the apparatus is driven by belts for positioning of the robots [20]. This could potentially conflict with our patent since both of our designs use belts to control our gantry, but based on the design, the patent itself doesn't focus on the use of drive belts so there shouldn't be a conflict in the patent.

In conclusion, our design should be patentable, if we heavily constrain it. We would not be able to patent using a gantry to move board game tiles, but if we constrain it to our specific design for how we designed our gantry, motor configuration and solenoid piece moving apparatus we would be able to secure a patent for our design. There is a lack of patents for robots using a gantry to pick up and move checkers pieces against a human opponent so our design shouldn't conflict with any similar patents. It is important to note that said patent would be incredibly specific, and as such although it would be possible to receive a patent, it would likely not be very useful.

Technical Description

Brief:

The checkers robot is a motor controlled gantry which can set up and play a physical game of checkers with a human opponent. The pieces contain metal washers, allowing them to be picked up by an electromagnet. A vision system using a camera is placed above the board in order to view the game state. A Raspberry Pi [2] controls the camera and calculates moves, while an MSP430 [3] controls the motors, buttons, and LEDs.

The gantry uses Open Builds [21] hardware in a rectangular xy fashion to move a solenoid to within 1 mm anywhere on the checkers board. Solidworks [22] will be used to design and build the gantry. The solenoid will have an electromagnet attached, allowing it to pick up the individual pieces, which will have washers inside of them.

Motors, buttons, status LEDs are controlled by a MSP430 [3] microcontroller. It communicates with the Raspberry Pi [2], which tells it what pieces to move and their locations. It then translates that info into the IO to control the gantry, and communicates back that it has completed the task. We used CCS (Code Composer Studio) [23] to program the MSP430 [3] and debug. The status LEDs indicate what the robot is currently doing, and also serve as a safety indicator to tell the user whether they should be interacting with the board. The buttons can tell the robot to set up a new game, make a move, align the camera to the board, or home the gantry.

The robot examines the board state with a camera controlled by a Raspberry Pi [2]. The Raspberry Pi [2] takes an image, then checks kernels of pixels and see if they match any colors of the checkers pieces. It then finds clusters of these kernels and finds the average coordinate in the image which it can then convert to a location in real space. The vision system can identify

pieces and their location on the board to within 5 mm and was written using C++ [4] using the Open CV [10] library.

We used a minimax algorithm to generate competent moves for the bot to play. To make the minimax algorithm more efficient, alpha-beta pruning was utilized. This algorithm was run on the Raspberry Pi [2].

Technical Details

Gantry:

The gantry design creates an xy axis by using three OpenBuilds [21] gantry carts which run an aluminum rail. Elliptical nuts on the carts allow them to be properly tensioned onto the rail. Connected to the gantry is an arch used to hold the camera above the board. The camera is fixed to the gantry during play, which allows the camera to remain in alignment during the game even if the table is bumped. The brackets to hold the Stepper motors[11] and idler pulleys were 3D printed. The models were created by taking the CAD models of OpenBuilds 3D parts and modifying them to be more resilient by adding a border. During construction, the design changed slightly to include a place to include the buttons for user input as well as changing the arch to accommodate aluminum extrusion that was used from a previous project. The Initial CAD design and final construction of the robot can be seen in Figure 1.

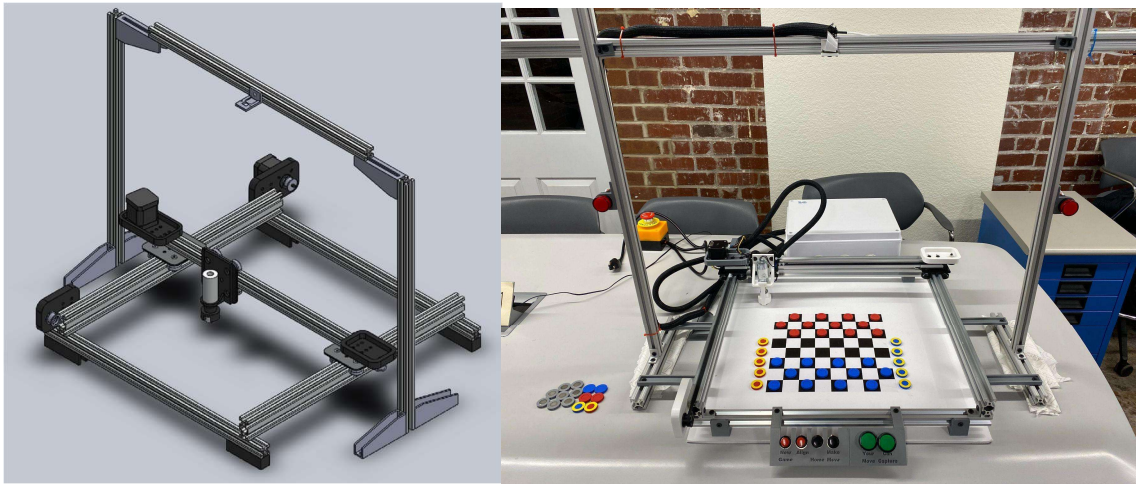


Figure 1: Original Design of Gantry (Left) and Final Construction of Gantry (Right)

Image Recognition:

The image recognition software used Open CV [10] to locate the checkers board within the image. The algorithm locates the interior corners of the checkers board and returns a list of 49 coordinates of the corners in a specific order. A sample visualization that Open CV [10] provides can be seen of the alignment in Figure 2. The pi uses these corners to calculate the average width and height of a square, as well as the edges of the board. All information about the state of the board can then be stored as the left, right, top, and bottom edges as well as the average square height and width.

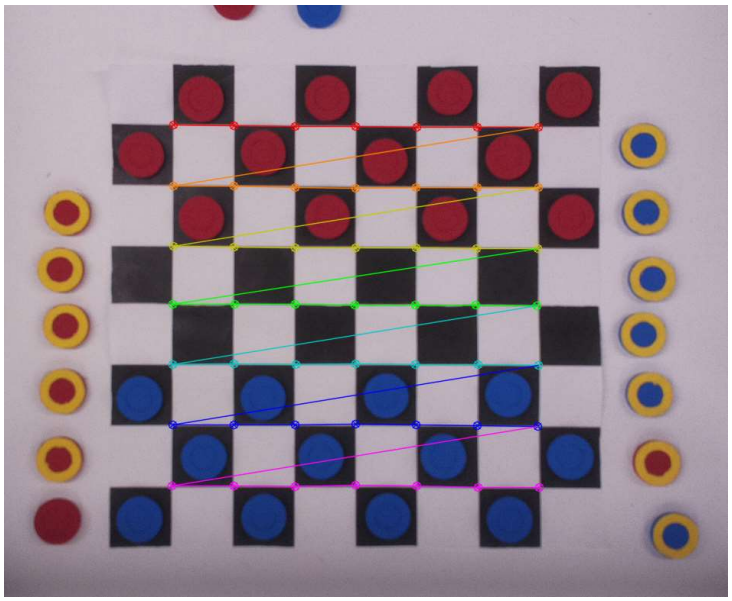


Figure 2: Visualization of Open CV [10] Checkers Board Finder

The algorithm to locate the checkers pieces uses the color of the pixels in the image to determine whether it is a piece. The camera takes a 1080i image, which is then split into its color channels. The color channels are then combined linearly to create a score for each desired color: blue, red, and yellow. The linear combinations for each score are listed below, and are computed using matrix operations in the code to increase speed.

Red Score = 2 * Red Channel - 1 * Green Channel - 1 * Blue Channel
Blue Score = 2 * Blue Channel - 1 * Red Channel - 1 * Green Channel
Yellow Score = 1 * Red Channel + 1 * Green Channel - 2 * Blue Channel

Pixels are then grouped together in kernels of four pixels by four pixels, and the average score for each color is calculated. Each score is then compared to a cutoff value. If the score is equal to or above the cutoff value, then a point is added to a list of points where the location of the point is in the center of the kernel and the color is the first score for which it exceeded the cutoff value. It is important to check threshold values in the order of yellow, then blue, then red, as it is possible for yellow to appear as red occasionally. The list of points can then be clustered using an agglomerative clustering method. Points within a certain distance are grouped together, and are then classified as too small, too big, red, blue, and whether they are a king based on the number of points of each color in the cluster. All clusters which are calculated as valid pieces are then classified into several groups: red pieces on board, red pieces off board, blue pieces on board, blue pieces off board, and error pieces. Pieces on the board are matched to squares on the board. Error pieces are defined as pieces on the board which are not in any particular square. A visualization of the piece recognition process can be seen in Figure 3.

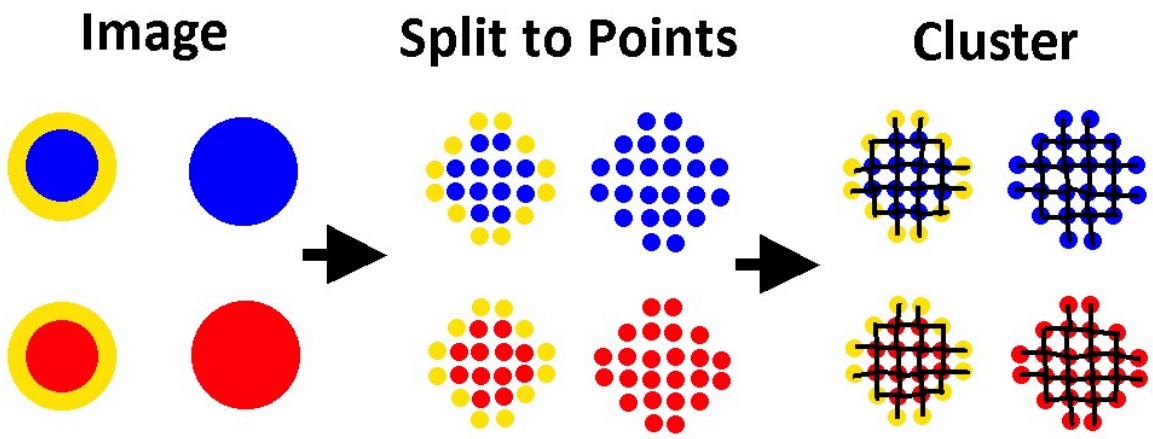


Figure 3: Visualization of Piece Recognition Process

Raspberry Pi [2] FSM:

The FSM for the Raspberry Pi is responsible for managing the piece imaging, boardstate, getting moves from the bot, and generating the appropriate moves to send to the MSP430 [3]. It must also handle communication to and from the MSP430 [3] in general. The FSM contains only two states, and does all calculations during potential state transitions. The two states are waiting for player input and sending moves. While waiting for player input, the FSM will ping the MSP430 [3] every 300ms, and ask if any buttons have been pushed. The MSP430 will respond with any buttons pushed. If any have been pushed, the FSM will attempt to perform the action associated with the button. If the action results in moves being generated, the FSM will switch states to send moves. If the action generates moves but a major fault such as not enough pieces have been located on the board, then the FSM will tell the MSP430[3] that a major fault has occurred and stay in the "wait for player" state. While the FSM is in the send moves state, it will send a move, then ping until the MSP430[3] sends a flag back indicating that it has completed the move, at which point it will send the next move. Once the list is empty, the FSM will send a flag telling the MSP430 [3] to wait at home. A general block diagram for the FSM can be seen below in Figure 4.

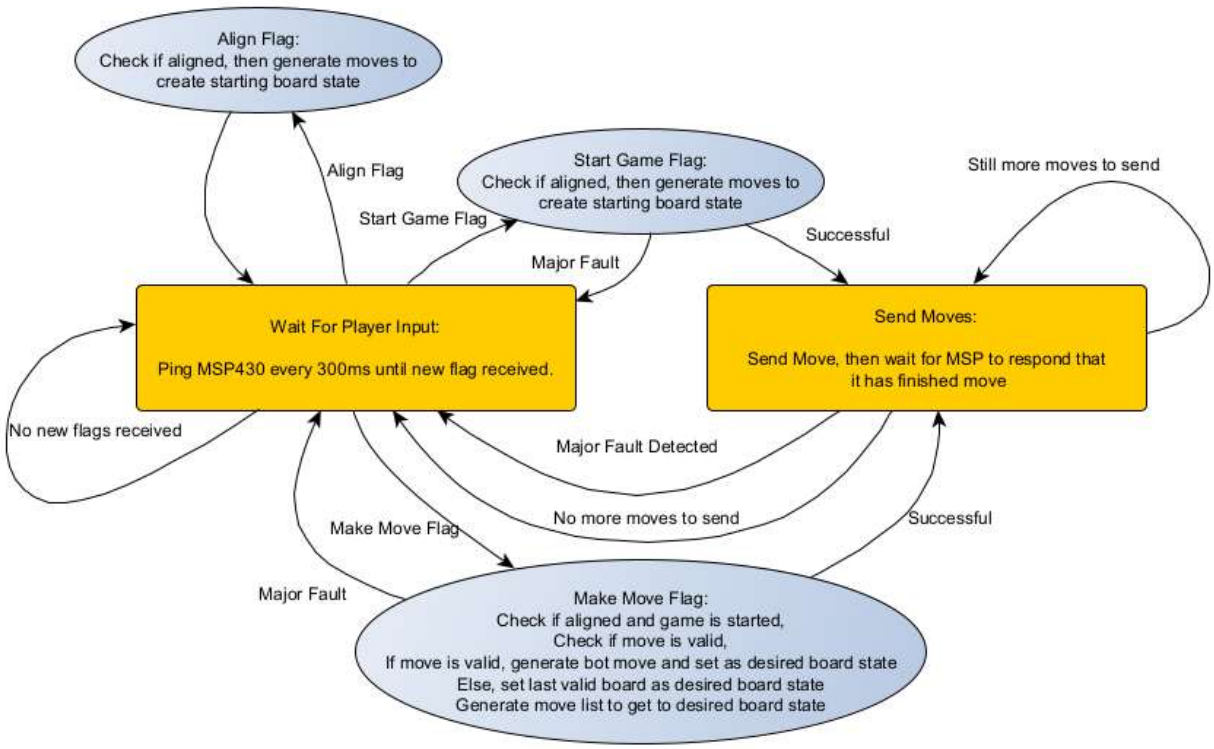


Figure 4: Raspberry Pi FSM Block Diagram

Whenever the FSM generates moves for the gantry, it does so by comparing the current boardstate to a desired board state. This allows the same function to be used for setting up the initial board, making a bot move, and correcting an erroneous player move. The strategy for the algorithm begins with removing all pieces on the board which are not in any specific square. These are also called error pieces, and must be removed first because they can be inside multiple squares, and therefore impact other moves. Next, all incorrect squares are classified as a square that is incorrect and should end up empty, and squares that are incorrect that should end up filled. Squares that are incorrect that should end up empty are then matched with any squares that should end up filled. The algorithm generates moves from a square that should end up empty to a square that should end up filled for every matched pair that was found. If the square that should be filled has a piece in it already, then a move is generated to remove the incorrect piece off of the board. After that, moves are generated to empty out squares which are incorrect and are filled with pieces. Finally, moves are generated to fill the remaining incorrect squares with pieces from off of the board. If at any point the algorithm attempts to find an empty spot off of the board or find a type of piece from off of the board and is unable to find it, then a major fault occurs, and no moves are sent. A flowchart describing the algorithm can be found in Figure 5.

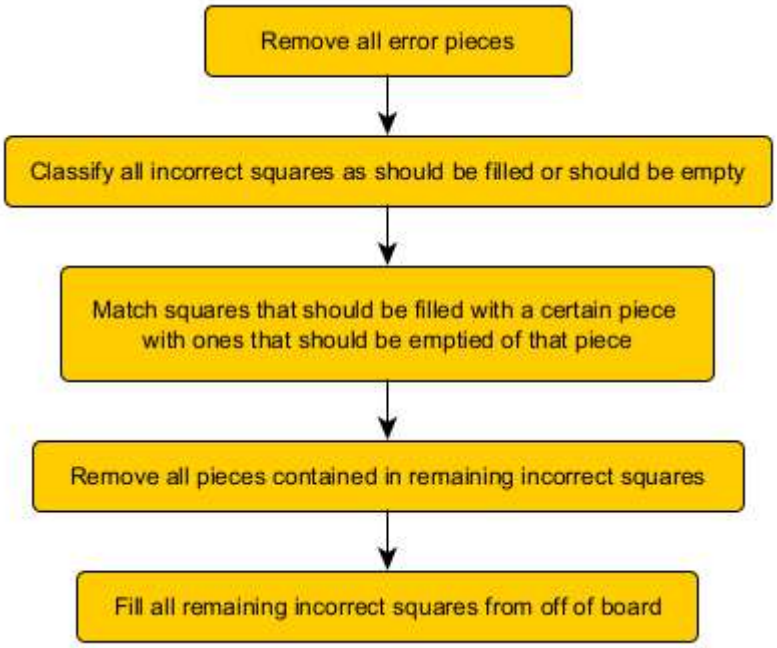


Figure 5: Flow Chart for Physical Move Generation Algorithm

Move Generation:

For move generation, a minimax algorithm was utilized. A minimax algorithm is an algorithm for playing zero-sum games, where the bot checks every possible move, every possible response, and so on to a particular depth. To analyze the boards it reaches, it must use an evaluation function. The evaluation function takes in board states as inputs and outputs an integer representing how favorable the board is for the bot. While the bot attempts to maximize this evaluation function, it assumes that the opponent is attempting to minimize with respect to the evaluation function. This allows it to predict the worst-case scenario moves that the opponent can play and operate accordingly. A common addition to minimax is alpha-beta pruning. Alpha-beta pruning cuts the minimax algorithm off from needlessly evaluating moves that are certain to not impact the bots decision.

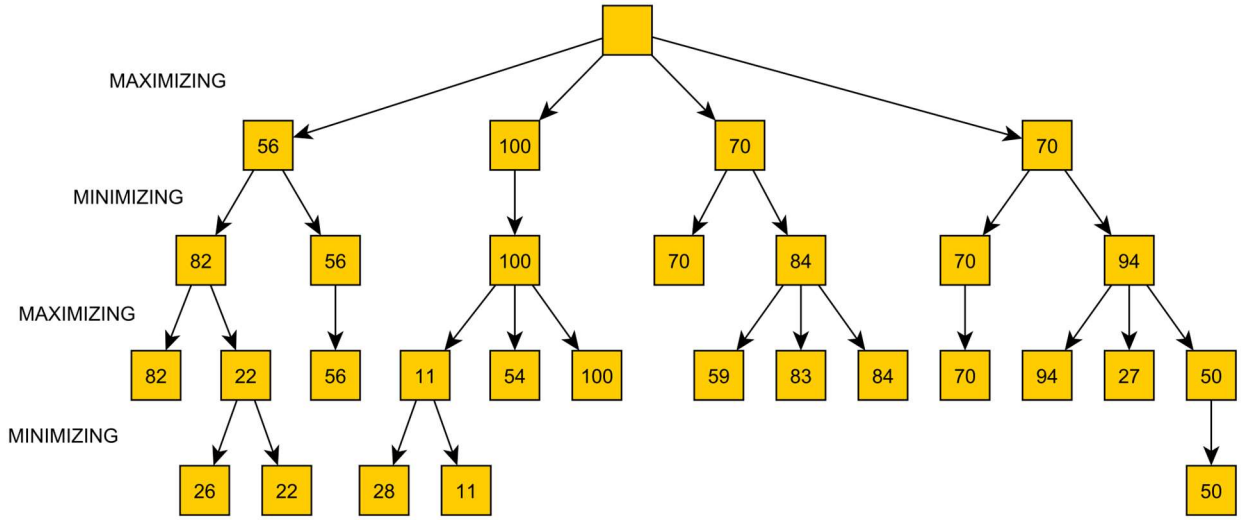


Figure 6: Example Game Tree

The above figure serves as an example of how a minimax algorithm works. The algorithm iterates until it reaches end nodes. These end nodes are due to the algorithm reaching its max depth or there being no available moves from that state (e.g., game over). In this example, the end nodes all have their evaluations that would be generated by the evaluation function. Those evaluations are then propagated towards the root via the assumption that the bot will choose the maximum number available and the opponent will choose the minimum. This allows the bot to see that the second branch is the ideal move.

Movement Sequencing:

In order for the robot to make a move, it must execute the following sequence:

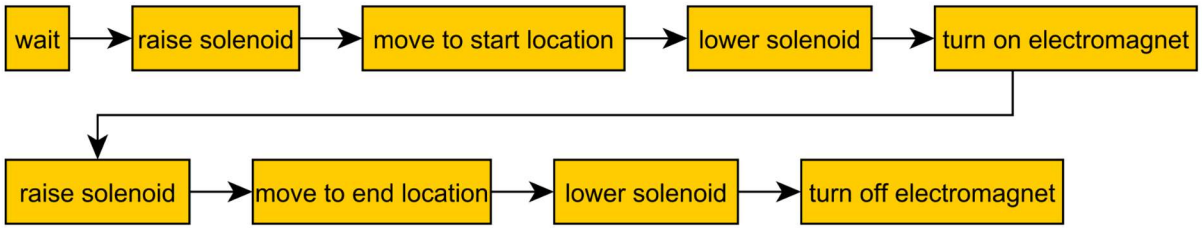


Figure 7: FSM Sketch

To implement this behavior, we used a finite state machine (FSM). The FSM ended up being considerably more complex than this. We ended up adding delays between certain transitions and allowing it to return to “wait” early (before turning on the electromagnet), so we can use the same FSM when we want to direct the arm back to its home position.

In parallel, to the FSM handling the actuation of the various peripherals, we also have 4 LEDs and 4 buttons. Two of the buttons are warning LEDs that flashed before and throughout the gantry executing a move. This behavior is loosely coupled to the move FSM. The toggle behavior begins when either the “play move” or “new game” buttons are pressed. The warning LEDs turn off once the FSM completes its move. There is also a “can capture” LED. Whether or

not the human player can capture is communicated by the Raspberry Pi [2] to the MSP430 [3]. The LED remains lit until the Raspberry Pi [2] is no longer indicating that the human player is able to capture. The remaining LED is the “waiting for input” LED. Waiting for input is typically turned off when the player presses a button and turns back on once the corresponding action has finished. The two remaining buttons are the “align” and “home” buttons. “Align” is pressed once at startup to tell the Raspberry Pi [2] to take a picture, find the board, and align its coordinate system. The “home” button is never pressed. It prompts the arm to return to its home position. This was useful in development but is not used by the player as the robot automatically homes when necessary.

Peripheral Code:

There are 4 different peripheral devices that the board needs to be able to control. Two [10] Stepper motors, a solenoid, and an electromagnet. We wrote embedded software so that our MSP430 [3] would be able to control the position and functionality of these peripheral devices to a high degree of accuracy.

The code for our electromagnet is very simple. Since our electromagnet drive is just a mosfet, all we have to do to turn the on or off the electromagnet is to toggle a pin connected to the gate of the mosfet.

In order to control our solenoid we have to adjust the duty cycle of the solenoid driver. If we want the solenoid to be on and pulled up we can’t set the duty cycle to 100% because if we did then the solenoid would get too hot and burn up. So in order to properly control it we need a way to easily control the solenoids duty cycle. The way we do this is to use a hardware interrupt timer. We set up one of the Timers on the MSP430 [3] to be configured in Up Mode and the output to be in Output Mode Reset/Set. As shown in the figure below, when set in reset/set mode, an interrupt toggles a pin so when the timer counter is between 0 and the value stored in the TAxCCR1 register the pin is set to 1, and while the timer counter is between TAxCCR1 and TAxCCR0 register values the output of the pin is 0. Using this, we can change the value stored in the TAxCCR1 register to be a fraction of the TAxCCR0 register in order to change the duty cycle of the solenoid.

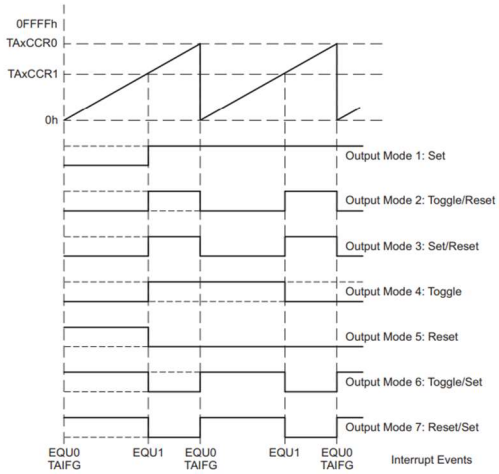


Figure 8: Diagram of MSP430 [3] Timer Output Modes [25]

We used two NEMA17 [11] Stepper motors in order to control the position of the gantry. These two motors functioned identically, being controlled using copies of the same motor drivers and being run by identical code. Since there are [11] Stepper motors, every single time they receive a rising edge signal they move a single step. By counting the number of steps it is away from home, we are able to determine the position of the motors and thereby allowing us to tell it how far to move to pick up and place pieces. In order to move both motors we use two separate timers, one for each motor. These timers then toggle the motor step pins, so every two timer interrupt cycles the motors move a step. In order to not lose a step on the motor drivers, we accelerate and decelerate the motors as they move to their position. The way we do this is every single step the motors take we decrease the period of the timers that they run off of. By decreasing the timer period, we increase the acceleration by shortening the amount of time it takes for the motor step pin to toggle. We do this by setting a fixed acceleration and deceleration period, and during the period the CCR0 register that sets the motor step period is decreased or increased corresponding to acceleration or deceleration. A graph of what this acceleration would look like is slow below.

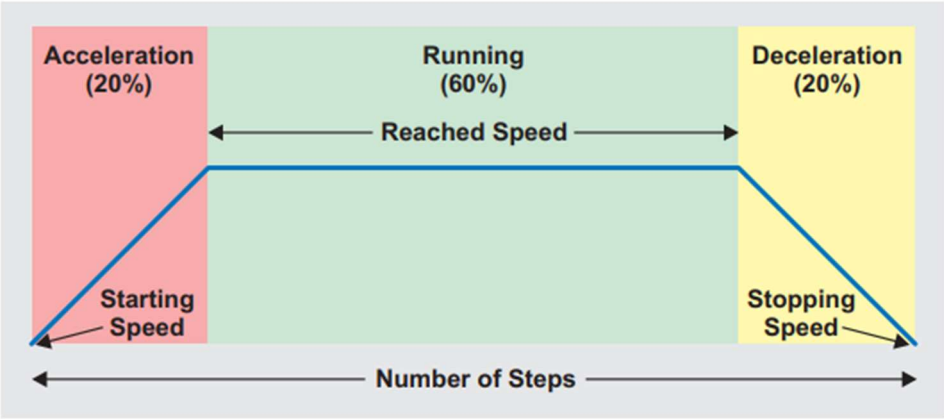


Figure 9: Graph of [10] Stepper Motor Acceleration [26]

In order to determine the number of steps that the motors have to move. We calibrated the motors and cameras in order to determine a transfer function of image coordinates to the number of steps. There was not much distortion from the camera so we were able to create a linear step function to convert image coordinates to steps. We fixed the image coordinates to be (1000,1000) at the corner of the board. From there we were able to create a linear transfer function from image coordinates to steps by calculating how many steps it took to get from (1000,1000) to home, as well as the number of steps it took from the opposite corner to home.

Schematic

Top Level Hierarchical Design

For the top level schematic, we aimed to keep the design as simple as possible and very organized so we can distinguish between blocks and where they are meant to connect. We primarily used wire labels to connect between subblocks to avoid a rat's nest of untraceable connections. The primary components of our custom PCB can be shown below. The major subsections include Power, Motor Drivers, Servo(Solenoid) Driver, Electromagnet Driver, I/O and the MSP430 [3] and Raspberry Pi [2] headers.

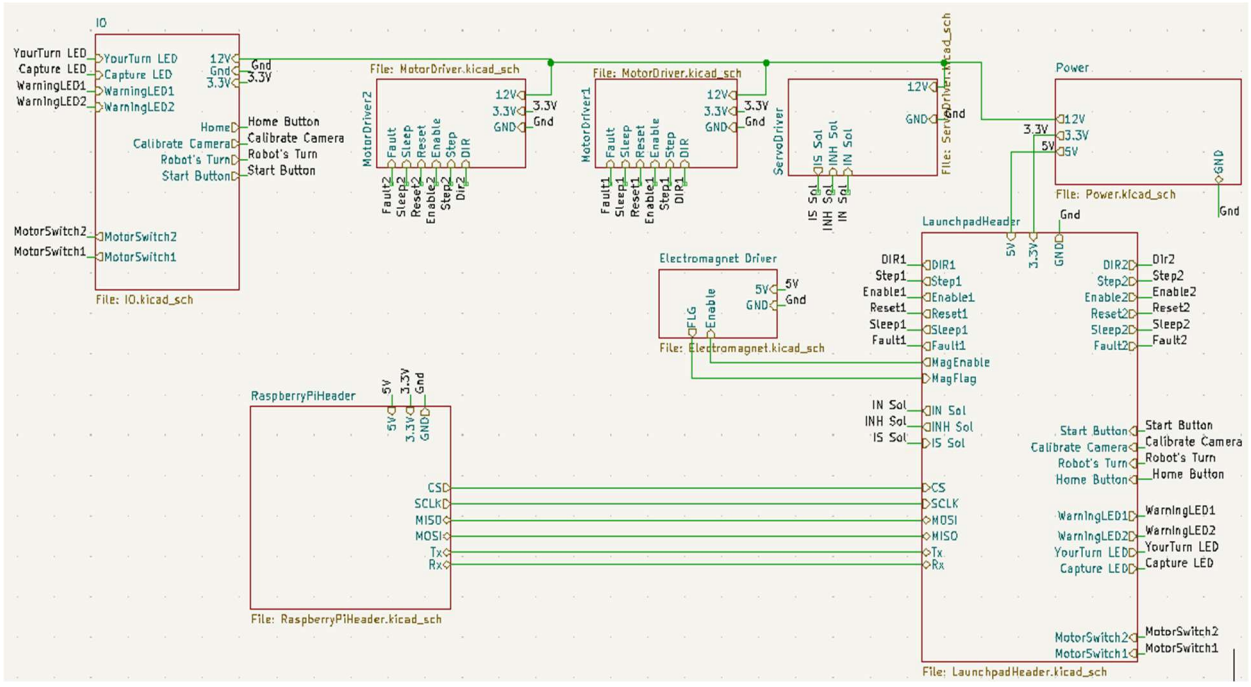


Figure 10: Top Level PCB Schematic

Motor Drivers

We used two motors in our project design, one for the x axis and another for the x axis. The integrated circuit we used to control them were the DRV8824[29], a [11] Stepper motor driver, which was suggested and provided by Professor Powell. For the majority of the schematic, we replicated the typical application circuit provided in the datasheet, only adjusting

the 400m ohm resistors for two 100 ohm resistors in parallel to make purchasing the materials easier. The [11] Stepper motors have 2 lines per winding so the driver outputs to a 4 pin phoenix connector.

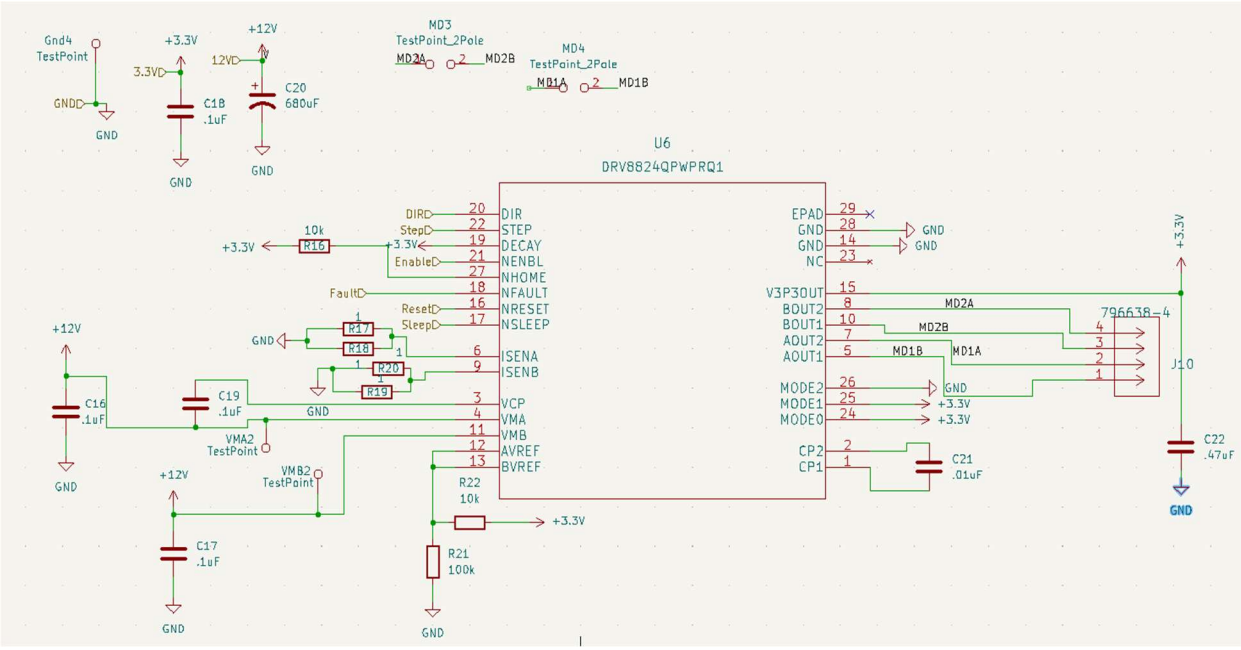


Figure 11: Motor Driver Schematic

Solenoid Driver

To drive the solenoid responsible for picking up and dropping pieces, we used a BTN8962 IC [30], designed for DC motors and suggested to us by Professor Powell. For this circuit we primarily followed the guidelines found in the component datasheet, only adding a Schottky diode in parallel with the load to prevent any flyback voltage to damage our chip. The solenoid driver outputs to a 2-pin phoenix connector.

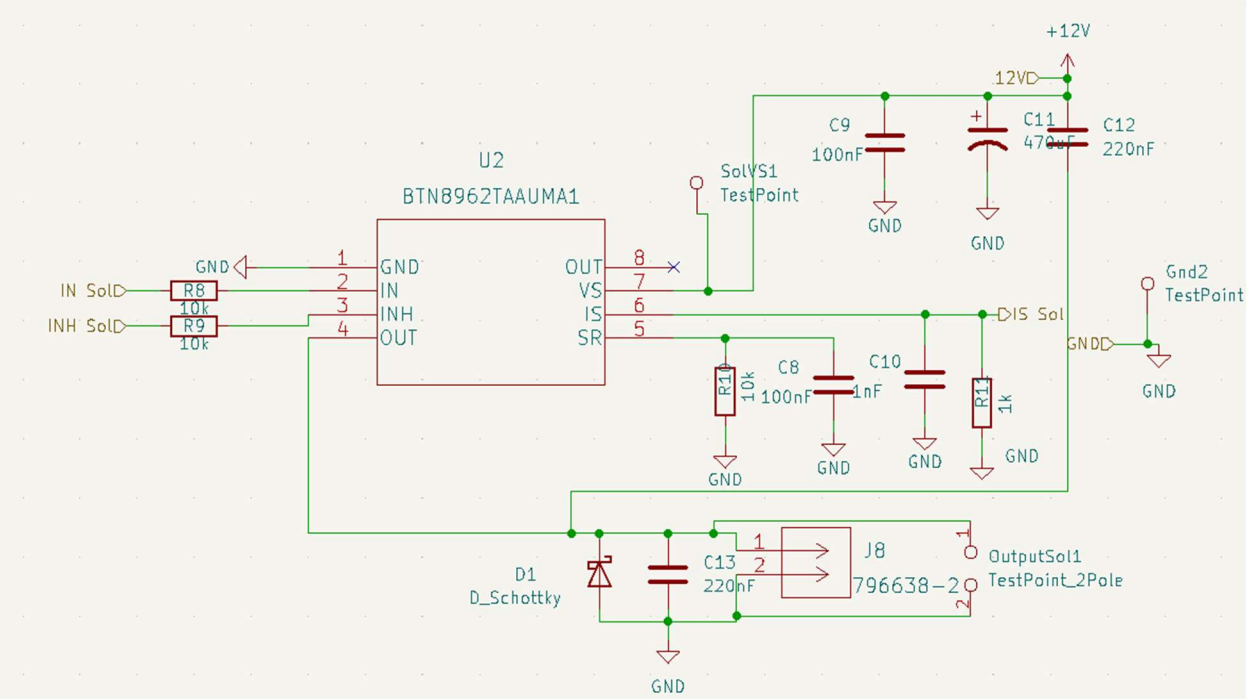


Figure 12: Solenoid Driver Schematic

Electromagnet Driver

The electromagnet driver was the simplest of all our circuits. We used the R524N00 integrated circuit to drive the magnet attached to our solenoid that is directly interacting with the checkers pieces. We considered designing the circuit entirely ourselves using a MOSFET, but we found a cheap Low Side switch IC on Digikey that had thermal protection so we opted to use that instead. The circuit in Figure 13 below is identical to the suggested layout in the datasheet, with an added Schottky diode in parallel with the load to prevent flyback. The electromagnet driver outputs to a 2 pin phoenix connector.

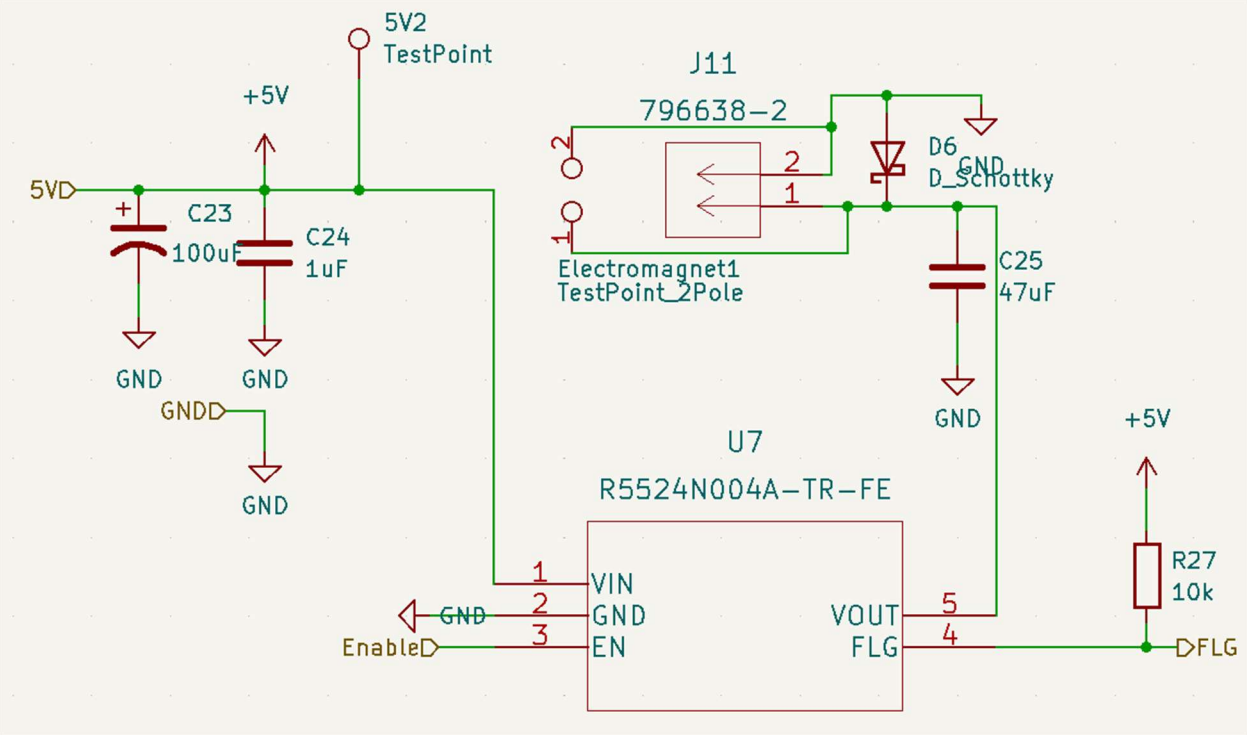


Figure 13: Electromagnet Driver Schematic

Power Regulators

Our project required 3 different voltage levels, 12V for the motor drivers, solenoid, and LEDs, 5V for the electromagnet, MSP430 [3], and Raspberry Pi [2] and 3.3V for the buttons. We used a 12V 30A transformer as a 2-pin phoenix connector input to the board. To step down to 5V and 3.3V, we used a LTI1374CR Switching regulator rated for 4.5A, and a AZ1117CR linear regulator rated for 800mA. We included status LEDs to indicate that the respective voltages were reading correctly. For added protection, we used a circuit design for reverse polarity using a S1451DY mosfet and zener diode as shown in Figure 14. It was set up to turn off if the current flowed in the wrong direction due to a short or some unforeseen circumstance.

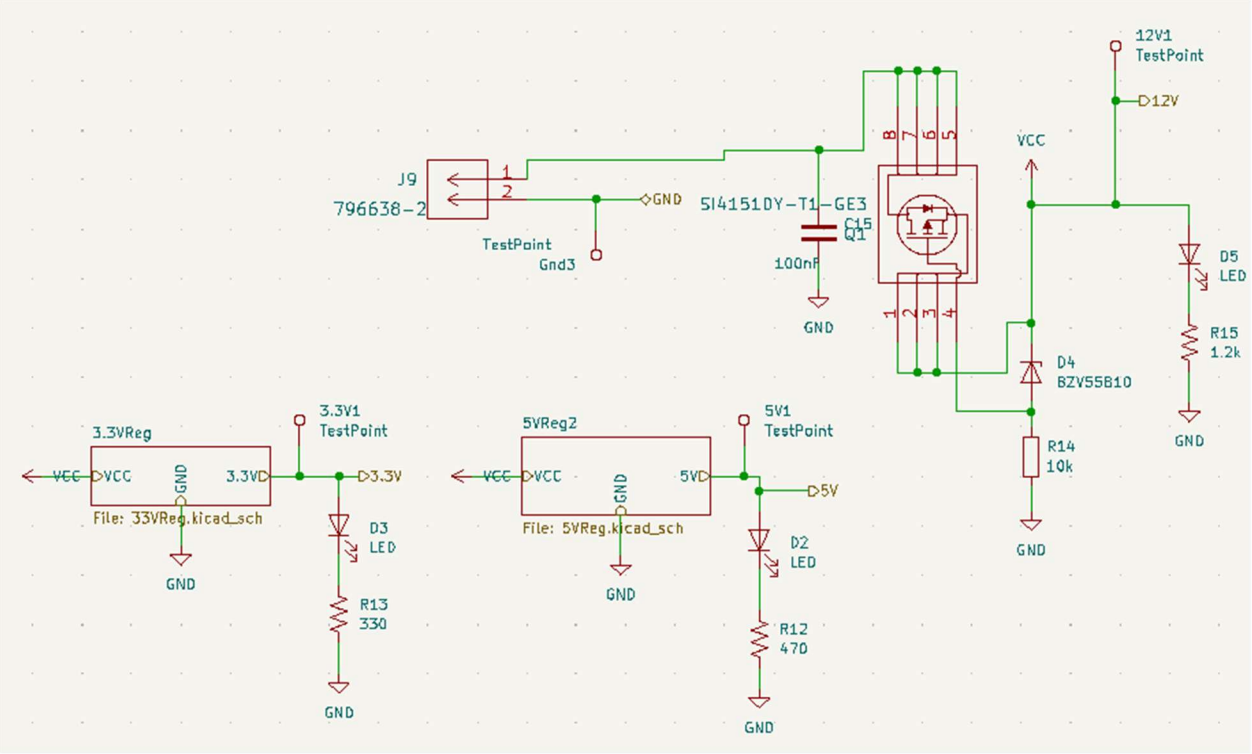


Figure 14: Power Regulation Schematic

I/O Hierarchal

Our project consisted of 10 different I/O peripherals. We allotted four buttons and 2 LEDs for the User Interface panel, 2 motor limit switches, and 2 warning LEDs. The warning LEDs flash red whenever the gantry is in motion, letting the human player know that it isn't safe to put their hand inside of the robot. The limit switches alert the MSP430 [3] that the gantry cart has reached the home position and should stop moving. The buttons on the User interface panel help with gameplay. The New game button tells the robot to reset the board to starting positions, the home button forces the gantry cart to the "home" position in the back position of the xy coordinate plane, the align button tells the robot that the camera is ready to be aligned, and the Your move button alerts the robot that it is its turn to play. The LEDs on the User Interface panel alert the player when it is their turn to play and when they need to capture a piece. To protect our MSP from electrostatic discharges, we used a 4 input SMDA12C-4 ESD chip embedded with zener diodes.

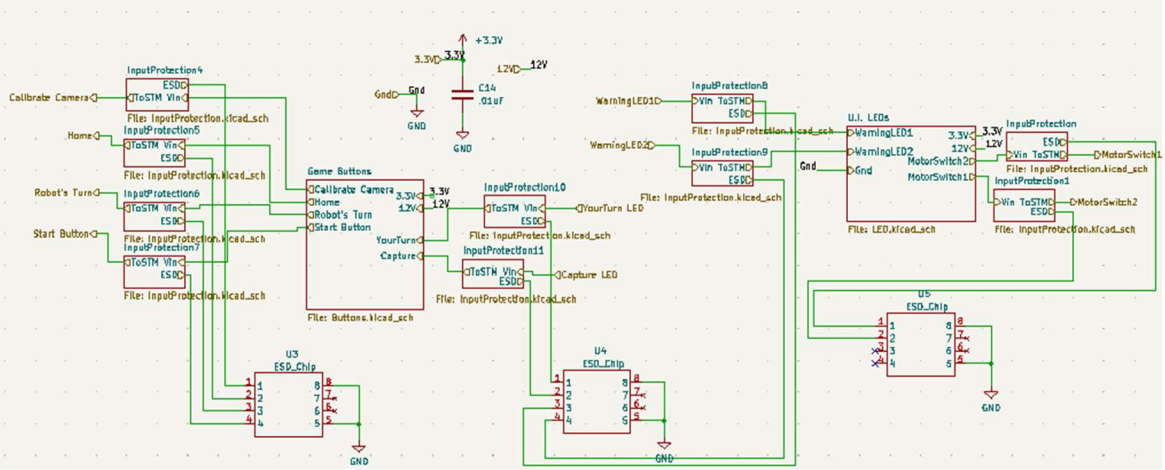


Figure 15: I/O Top Level Schematic

User Interface Panel

To drive the LEDs of our project, we used 2SK1828TE85LF MOSFETs as low side switches with the pull up voltage being 12V and 10mA running through them. The Buttons were connected identically to the limit switches, with pull up resistors connecting them to 3.3V and lines going to ground. For space, convenience and practicality, we used a single 12 input ribbon connector for the entire U.I panel since very little current needs to flow and we wanted to limit excess space on the board used by our many phoenix connectors.

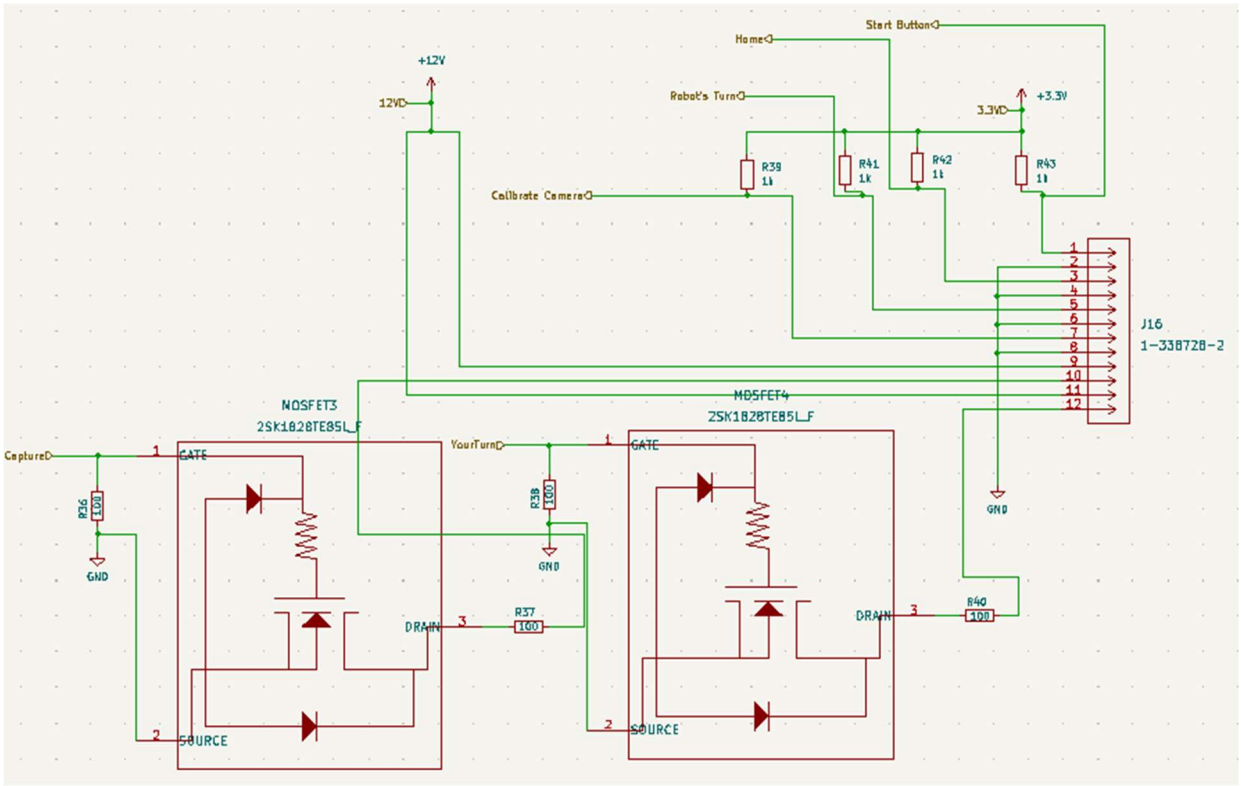


Figure 16: User Interface Panel Schematic

MSP430 [3] and Raspberry Pi [2] Headers

The header rows for the MSP430 [3] and Raspberry Pi [2] were taken from online to ensure the spacing and pitch matched exactly with our mounting components. We include pins so we can choose whether or not to connect the 5V supply to the Pi and MSP430 [3] using a removable shunt. We determined which pins had specific input requirements on the MSP and wired those first. For everything else, we connected them in groups of which driver circuit they were associated with as much as possible.

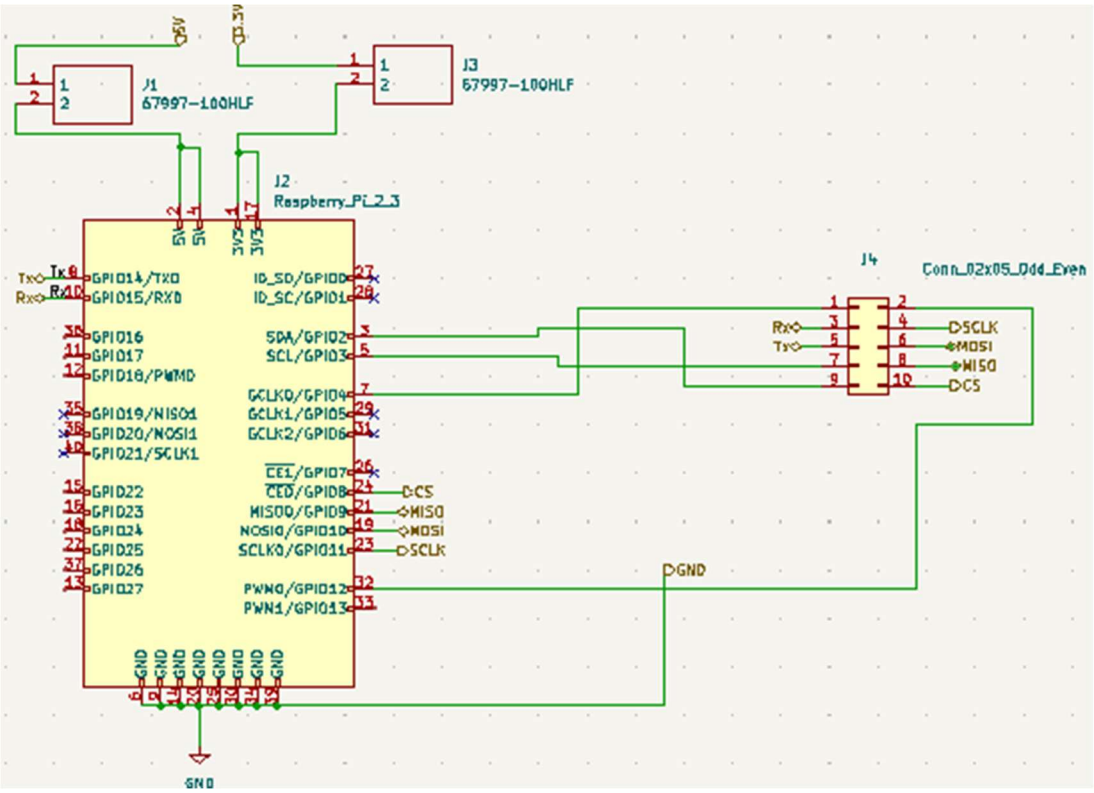


Figure 17: Raspberry Pi [2] Headers

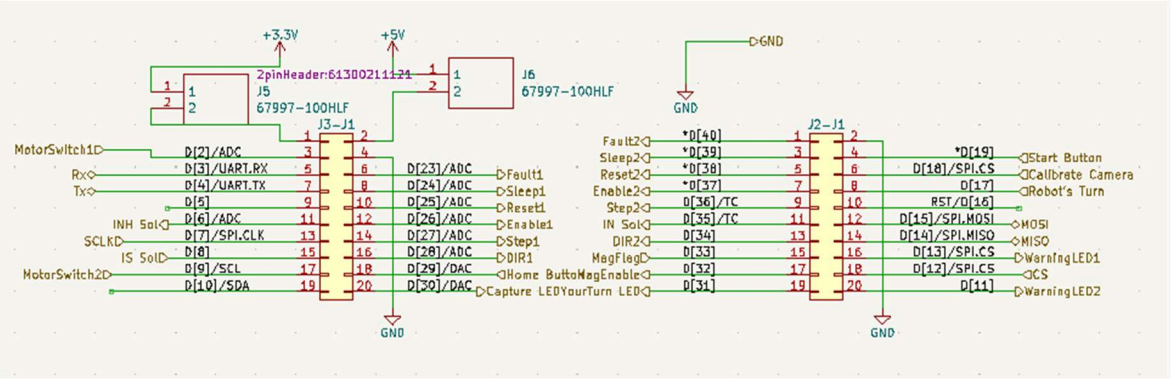


Figure 18: MSP430 [3] Header

Board Layout

To construct our board layout we prioritized ease of access to components over space optimization. In the first revision of our board, we used the maximum size allowed for our project (10” by 6”). After going back to the drawing board for revision two, we were able to reduce the length of our board by over an inch, resulting in the final board below (8.6” by 6.1”). To trivialize population and testing of the board, we organized the PCB into general blocks.

We put the Raspberry Pi [2] and MSP430 [3] headers near the top of the board so that they could hang off the edge and the motor driver circuits as close to the MSP as possible to reduce the length and complexity of the I/O traces as seen on the right side of Figure 19. Beneath that, we placed our two magnet driver circuits, since their respective wires had to go to the same area of the gantry as the motors. At the bottom of the layout we put the components for the User Interface panel, the warning LEDs and the motor switches. These subblocks have similar design and simple functionality so they were the optimal section to put furthest from the MSP430 [3].

The Input power and 5V and 3.3V regulators were positioned on the left side of the board to separate them from the subblocks they were driving. Each voltage level (12V, 5V and 3.3V) were accompanied by a status LED that indicated the regulators were working as intended. We have four 2mm drill holes for mounting the PCB in the NEMA enclosure.

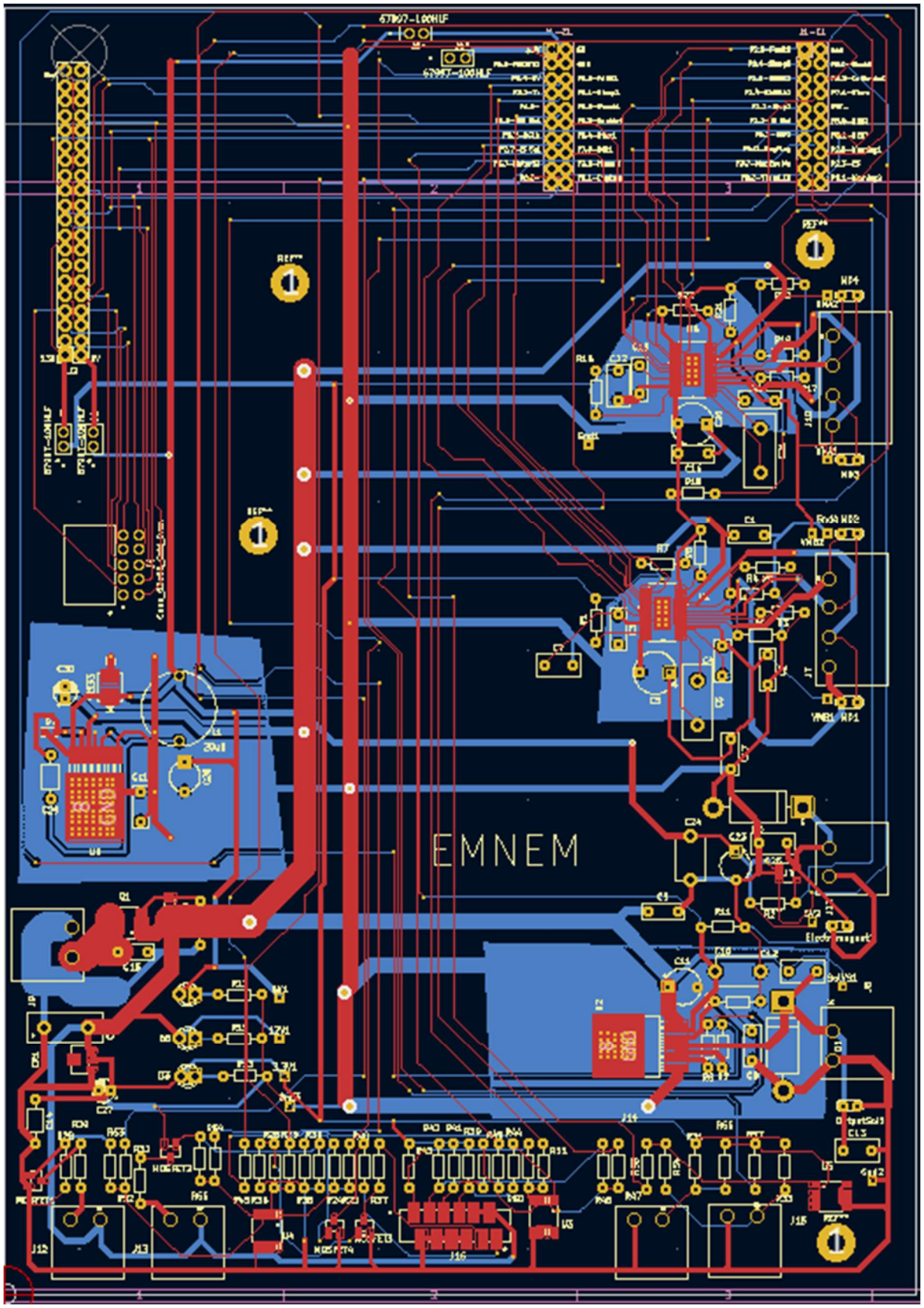


Figure 19: PCB Revision 2 Layout

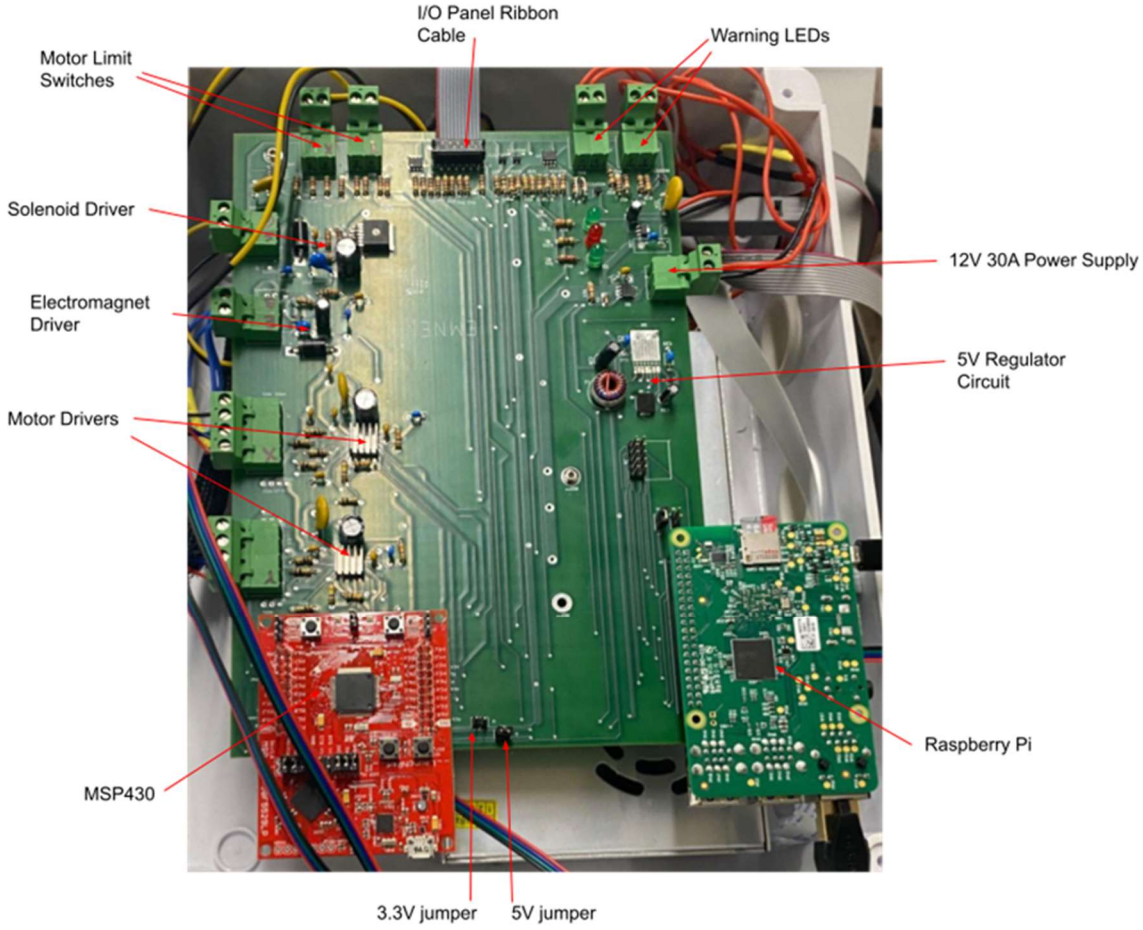


Figure 20: Finished PCB

Major Components Used

1. NEMA17 [11] Stepper Motors(12V, 3.4A) - The [11] Stepper motors we employed have two phases, each consuming 1.7A while in operation. These motors were the most consistent current drawing devices and had to have significant heat sinking to prevent thermal breakdown. We used one for each axis (x and y) and they were fastened into 3D printed gantry carts.
2. 527-1029-ND Intermittent Solenoid(12V, 3.4A)[27] - The solenoid was positioned to vertically drop and raise the armature, assisting the electromagnet in picking up pieces and keeping it airborne as the motors move across the game board.
3. 3872 2.5kg Electromagnet (5V, 220mA) [28]- The electromagnet was used to attract a ferromagnetic iron washer embedded in the game pieces.
4. MSP430F5529[B] Launchpad- The main control unit for the robot gantry. Loaded with C code, the MSP actuates the motors, magnets, LEDs and buttons to provide a real time response to user input.
5. Raspberry Pi [2] 3B - The de facto brain of the Checkers playing robot. The Raspberry Pi [2] is directly connected to the camera and processes images to determine board states

and runs a search algorithm Checkers A.I to determine the robot's next move and communicate it with the MSP430 [3].

6. 1195-Set (V-Slot Gantry Kit - 20mm) [C]- The Gantry cart that the solenoid and electromagnet are mounted on. It has holes for tension relief and carries the y axis motor wherever it needs to go to perform a move.
7. Pi Camera Module 2[2] - The camera is designed to take images to ensure board alignment and run alongside the Raspberry Pi [2] as the robot's computer vision mechanism.

Design Decisions and Tradeoffs

Hardware

There were many design choices that impacted the outcome of our PCB and entire hardware setup. Firstly, choosing to use a 12 pin ribbon connector as opposed to a phoenix connector used for everything else coming off the board saved us a lot of space. The 12 pin ribbon connector was for the User Interface panel that had four buttons and two LEDs on it that would've been more challenging to wire successfully if we used 20 awg wire like the majority of our other off board connection. The flatness of the ribbon cable allowed us to smoothly run in underneath the cardboard that our gantry sits on with no worry of it making the surface uneven. Another major hardware decision we made was to use [11] Stepper motors as opposed to using far less accurate DC motors. The [11] Stepper motors gave us the ability to pinpoint the position we want to move it to up to 1/1600 of the total circumference of the gear. This helped us create a smoother game experience with minimal placement failure (under 5%).

To simplify the actuation of picking up and dropping the checkers pieces, we opted to use a solenoid rather than a servo. The servo would've increased complication in our board layout and was unnecessary for the distance we needed it to move. Instead, we used pulse width modulation to raise the solenoid whenever it is in motion and release it using gravity whenever it's time to pick up a piece or be put at rest. The major tradeoff that we ran into with this is a loss of precision when it come to how hard the solenoid would drop because of gravity and more fine tuned calibration for the solenoid so it was strong enough to pick up a piece while not being so strong that it knocked the piece off after raising itself up.

To reduce the cost of hardware parts, we opted to 3D print all the brackets and other complimentary pieces that would typically be metal. The downside to this choice was the longevity that printed parts had in relation to its metal counterpart. This was an easier decision to make since we only needed it to be operational for a short period of time. If we wanted to commercially reproduce our checkers robot, we would have to reconsider this greatly.

Software

The image recognition code had to balance resolution with computation time. The code could come up with more points, and thus identify pieces within the image at a higher resolution, however, the number of points it would have to cluster would increase quickly. The high compute time would cause a constant delay. Because the player is unlikely to notice an increase in piece recognition accuracy as long as moves were consistently successful, the code generated

points using 4x4 squares of pixels, effectively using a resolution of 480x272 pixels. This resolution was accurate enough for us to meet our goal in terms of piece movement success rate while also reducing compute time to about 1 second.

Originally, we had to contend with the tradeoff between strength of the move generation algorithm and time to generate a move. We were able to bypass this by switching from Python to C++. The algorithm is undefeated in over 20 games, while generating a move in a fraction of a second. Additionally, the core of the move generation needs to be very fast as it is executed millions of times per move.

Project Time Line

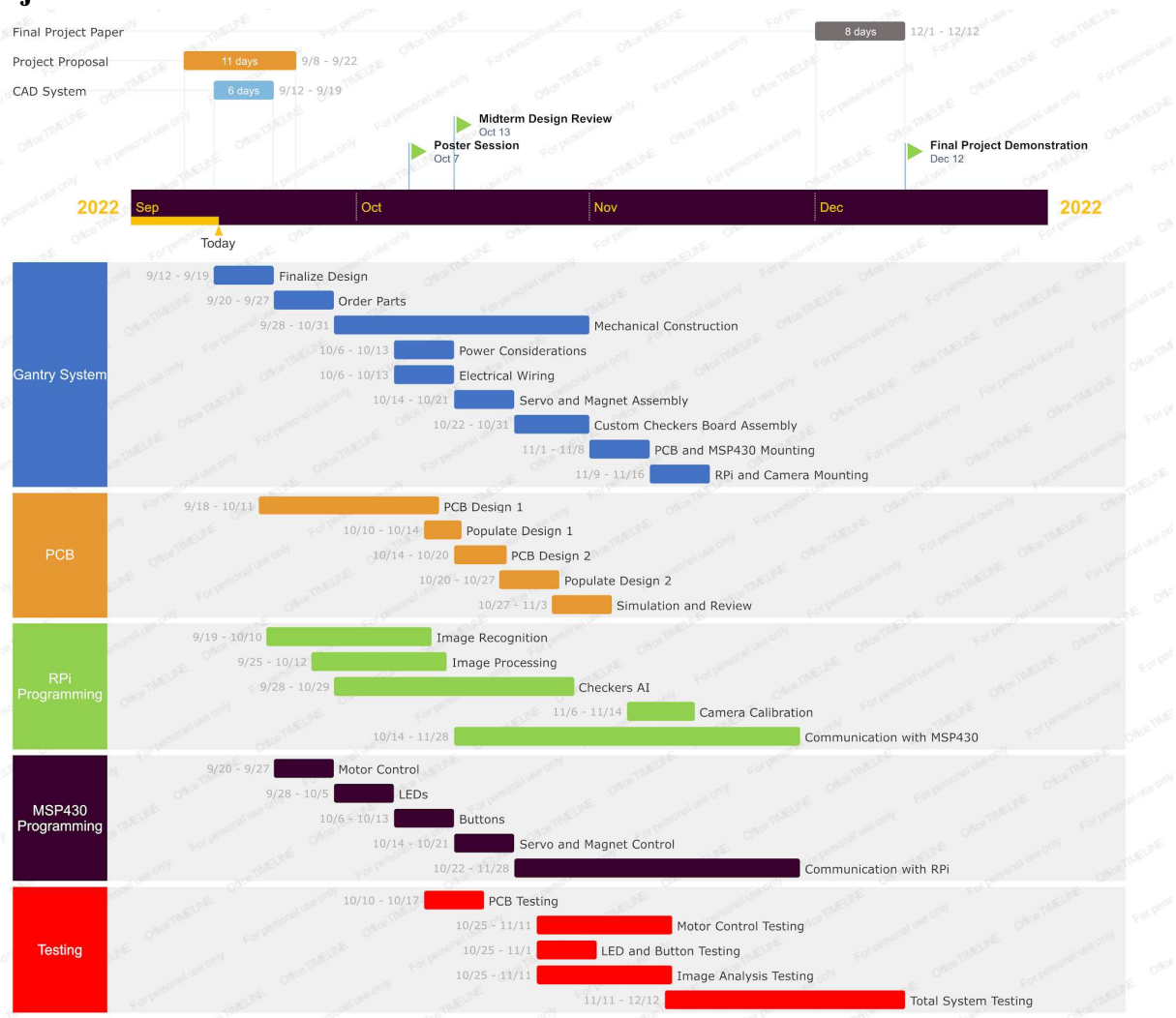


Figure 21: Proposed Gantt Chart

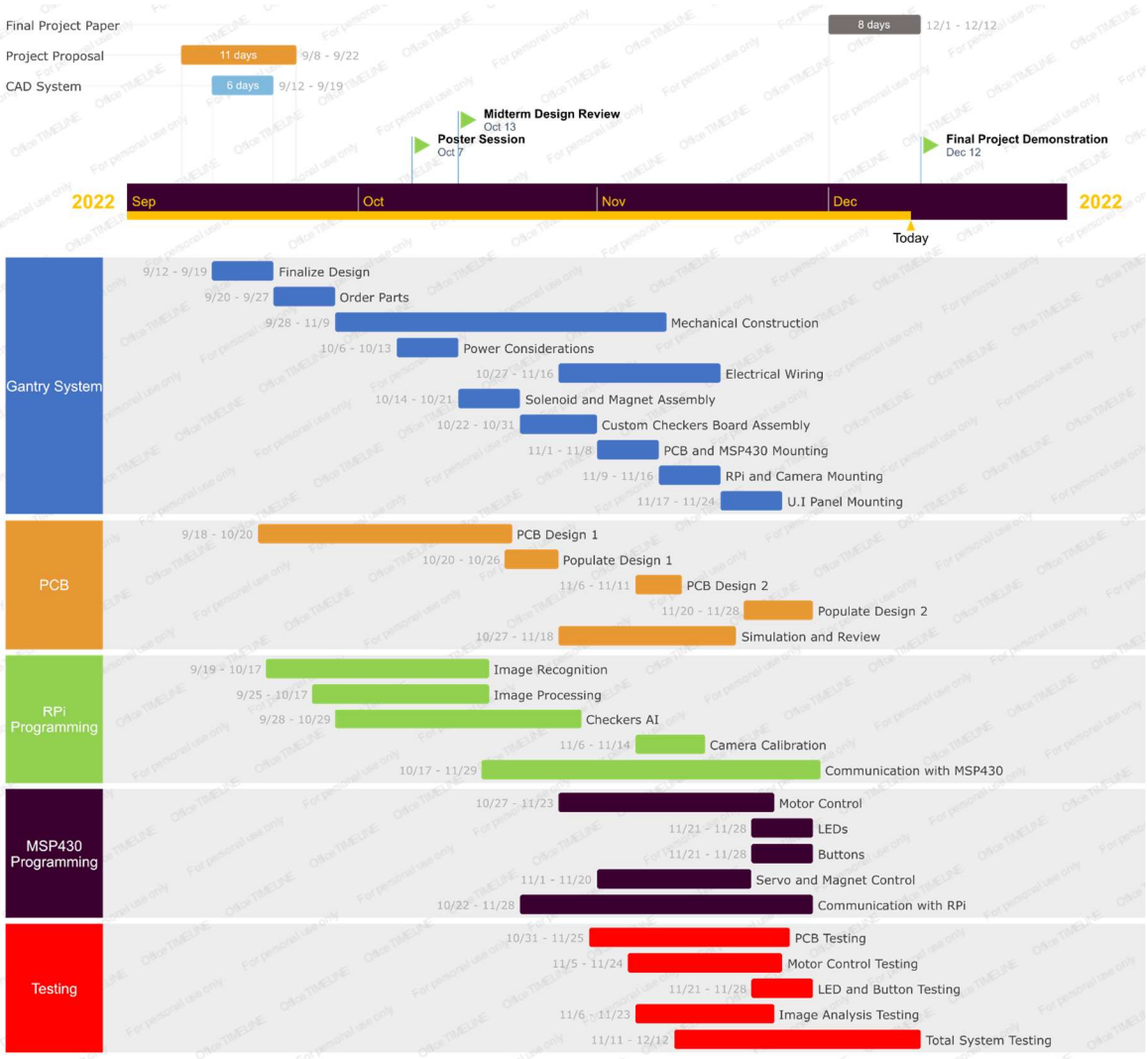


Figure 22: Final Gantt Chart

Serial Tasks

1. The PCB testing and redesign could only take place after the PCB initial design was fabricated and populated.
2. The total system testing hinged on the gantry construction, image analysis and PCB being complete. The gameplay couldn't be experimented with if those subsections still had major issues.

Parallel Tasks

1. The Imaging Code, Checkers AI, and Embedded Code(motor drivers, UART communication, electromagnet, etc.) could all be abstracted and written in parallel until the high level integration began.
2. The CAD for the robot construction as well as the wiring could be done in parallel without interference.

Responsibilities

1. Nicholas Palmer
    a. Primary - PCB Design
    b. Secondary - Embedded Software
2. Emmanuel Erhabor
    a. Primary - PCB Design and Testing
    b. Secondary - Robot Construction / Electrical Wiring
3. Matthew Hankins
    a. Primary - Embedded Software
    b. Secondary - Checkers AI / Piece Imaging
4. Matthew Ippolito
    a. Primary - Piece Imaging and Recognition
    b. Secondary - Robot Construction / CAD

Gantt Chart Adjustments

From the first day of class, we were told that we should have very pessimistic views on dates and times for completing tasks. The first major setback we encountered was a weeklong setback for the first PCB sendout. This delay helped us finish and refine the first rendition of our design so that we would have an operational revision in case our second PCB sendout didn't work. Pushing the design back also affected the first PCB test and population, pushing it back by an entire week.

Another key adjustment we had to make was concerning the timeline of the embedded code on the MSP430 [3]. We underestimated how long the Checkers AI and PCB would take to be finished so that Matt and Nicholas could move on to programming the actuation of the robot gantry. The MSP communication with the Raspberry Pi [2] took days of troubleshooting and dragged out for three days longer than anticipated. On the other hand, the programming of the LEDs and buttons only took a single day each for the initial testing. The programming for the solenoid and electromagnet took considerably less time than allotted for in the timeline, time that was used for the skeleton code and organization of the larger picture of the embedded system.

One key task that didn't get a role specifically allotted to it was the design of the system's Finite State Machines and their interaction with each other. This was partially captured in the communication between the Raspberry Pi [2] and MSP430 [3], but the logical functionality took considerable time and debugging to nail down.

Test Plan

1. Connectivity Test the board by cross referencing all components with the schematic and verifying proper continuity (Voltmeter). Ensure no trace is shorted to ground or to one of our voltage levels.

a. If there are any issues or faults, double check soldering for COLD solder joins or accidental short circuits. Desolder and Resolder accordingly.
b. Make sure that any long leads are cut short enough so they don't interfere with other parts of the PCB.
c. Ensure continuity between all ground test points.

2. Test Power
    a. Test the 12V power supply. LED should light up to verify power and the 12V test point should show the correct voltage (Voltmeter).
        - Verify that 15A is being supplied using a power resistor.
        - If the power supply doesn't work, check the wiring and confirm continuity. If the issue is the power supply, buy a new one. (Reverse Polarity MOSFET should provide added protection).
    b. Test the 5V power pin by checking the 5V test pin (Voltmeter). Add a power resistor connected to the 5V and gnd that draws 4A to check that the supply will power enough current to the electromagnet, PI, and MSP430 [3]. Verify that 5V is seen at the appropriate locations.
        - If there is no 5V supply, check that each component is connected to the board and check that it's connected to the same node as it is in the schematic. Check the inductor and other components of the 5V regulator including the forward voltage of the diodes for sensing and voltage.
        - If that fails then check the schematic to make sure that our schematic lines up with the datasheet.
            - Check Component values
        - If there is a Voltage but it isn't 5V, double check the 5V switching Regulator is connected properly. Verify that the capacitors and diodes aren't broken.
        - If there is 5V but not enough current then check the inductance of our inductor. That determines the amount of current. Ensure there is a good solder connection.
        - Check 5V shunt for Pi and MSP430 [3].
    c. Test the 3.3V power pin by checking the 3.3V pin (Voltmeter). Add a power resistor connected to 3.3V and GND to see if it can draw 0.5A of current.
        - If there is no 3.3V supply, check that each component is connected to the board as well as its orientation.
        - If that fails then check the schematic to make sure that our schematic lines up with the datasheet.
            - Check Component values
        - If there is 3.3V but not enough current check the data sheet for the linear regulator and swap it out with a different regulator.
        - Check 3.3V shunt for PI and MSP430 [3]

3. Connect Launchpad to header board using external power. Use Launchpad to turn on then off electromagnet.
    - If electromagnet stops working test electromagnet independent from driver
        - If electromagnet works then the issue is with the electromagnet driver. Check the output of the electromagnet driver and that it is outputting 5V.
            - If it is outputting 5V check the current by wiring a multimeter in series with the electromagnet.
                - If it produces too small of a current then get a new chip.
            - If there is no current check the part orientation and if the PCB matches the schematic and the driver datasheet.
        - If the electromagnet doesn't work then swap out the electromagnet. Then repeat
4. Test Solenoid using the Launchpad. Use the launchpad to turn on then off the solenoid.
    - If solenoid doesn't work then test it independently from the board. If it works off of the board then the issue is with the solenoid driver. Check the input of the solenoid driver and that it is receiving 12V.
        - If it is receiving 12V check the current by wiring a multimeter in series with the electromagnet.
            - If it produces too small of a current then check the component output values
        - If there is no current check the part orientation and if the PCB matches the schematic and the driver datasheet.
            - If everything still matches up check the launchpad solenoid code
    - If the solenoid doesn't work independently then swap out the solenoid. Then repeat
5. Test Motor Drivers using the Launchpad after Heatsink is attached. Use the launchpad to control direction and step of the motors. Repeat test for 2nd motor driver
    - If the motor doesn't work then test it independently from the board. If it works off of the board then the issue is with the motor driver. Check the input of the motor driver and that it is receiving 12V.
        - If it is receiving 12V check the current by wiring a multimeter in series with the driver.
            - If it produces too small of a current then check the component output values
        - If there is no current check the part orientation and if the PCB matches the schematic and the driver datasheet.
            - If everything matches up check the component values
                - If everything still matches up check the motor driver code
    - If the motor doesn't work independently then test the other motor driver. Then repeat

- Monitor Motor Drivers to ensure that they won't overheat and burn out during peak conditions(use software to run a loop of gantry movements to run the motors).

6. Test User Interface Control by verifying continuity on 12 POS Ribbon Connector. and Cable.
    a. Select and Deselect Buttons to verify signal integrity inputted to the MSP430 [3].
        i. If the Buttons don't work, repeat connectivity test to ensure proper orientation and continuity with regard to the schematic. If it still doesn't work, swap out button for a different one to see if the button is broken or the Ribbon Cable
    b. Verify U.I. LED output by testing for 12V signals and a max 20mA of current when connected to the PCB.
        i. If the LED doesn't work, repeat the connectivity test to ensure proper orientation and continuity with regard to the schematic. If it still doesn't work, swap it out for a different one to see if the LED is broken or the Ribbon Cable.
    c. Verify Warning LED output by testing for 12V signals and a max 20mA of current when connected to the PCB.
        i. If the LED doesn't work, repeat the connectivity test to ensure proper orientation and continuity with regard to the schematic. If it still doesn't work, swap it out for a different one to see if the LED or if there;s a problem with the schematic.
7. Once the external peripherals are tested and working then plug in the Raspberry Pi [2] and launchpad and try communication with them over UART and SPI.
    - If communication doesn't work check the voltage levels for the communication protocols/voltage on the communication lines.
        - If there is still nothing check voltage levels on boards independently
            - If it still doesn't work re-check the communication code
                - Else if both the code and board are correct replace launchpad and or Pi

8. Check Heat sinking of motor drivers, regulators and solenoid driver.
    a. Run motors and solenoid using test code and feel if the ICs get warm to the touch.
        i. If they are super warm, Add external heat sinks.
9. Ensure wiring between the gantry and NEMA enclosure is tight and without tension.
    a. Extend wires that don't have enough slack to reach enclosure
    b. Use wire sheathes to organize wiring.

Final Results

We were able to successfully complete the task we set out to do and were fully able to complete 100% of the intended functionality. Our robot worked as we designed it and we accomplished the goals we set out to meet. The first goal we met was the ability to control the position of the gantry. We were successfully able to do that when we controlled where we wanted to pick up and move pieces. We also wanted to be able to make a legal move given a board state as well as have an algorithm that makes logical moves and not random decisions. During the expo, our robot defeated 7 opponents in a row, proving that it can make smart decisions and make moves given checkers board states. Part of being able to play checkers and make moves is being able to detect pieces. Our piece detection allowed us to robustly detect where pieces are on the board and where they are in relation to the grid. This allowed us to both recognize both valid and invalid states since we know where the pieces are in relation to the board. The algorithm that determines the robots next move was written in C++ [4] in order to obtain better performance. This allowed the robot to decide its next move quickly and start moving within 5 seconds.

Additionally, when playing against the robot, if the user made an illegal move the robot is able to pick up and move the piece back to its original position, due to the illegal move that it occurred. While playing, the robot has a high degree of accuracy. It did not have trouble picking up and moving pieces, and when it did it was able to correct its mistake after the user pressed the "make move" button again. It was able to perform with over 95% accuracy. Finally, during the expo our robot played checkers against Professor Powell.

Costs

Table 1: Top Level Cost Breakdown

Subsystem (in budget)	**Major Components**	**Cost ($)**	**1000 unit Quanitities ($)**
Physical Gantry	Gantry Carts, Metal frame, screws, nuts, pulley kit	197.82	197.82
Piece Movement	Solenoid, Electromagnet	42.46	33.71
Onboard PCB Components	Regulators, ceramic capacitors, electromagnet driver,	103.72	18.16
Offboard PCB components	Phoenix connectors, ribbon connector	46.87	10.31
PCB Fabrication	2 layer PCB	66	10

PCB Population	SMD components	35	10
	Subtotal	491.87	280
Pre-existing Materials	**Major Components**	**Cost**	
Offboard PCB components	LEDs, buttons, wire, E-stop, wire sheath, ribbon cable	39.74	39.74
Offboard Components	wire sheath, NEMA enclosure, 3D printed components	60.99	60.99
Raspberry Pi [2]	Raspberry Pi 3B, Pi Camera	60	60
MSP430 [3]	MSP430 [3]	15.59	15.59
[10] Stepper Motors	NEMA17	23.98	23.98
Power Supply	12V 30A Power Supply	20	20
	Subtotal	220	220
	Total	711.87	500

Future Work

There are possible ways to expand on this project. While we were able to implement all of the functionality that we originally set out to. New ideas came up along the way. The first idea is to create a difficulty switch for the checkers algorithm. Either a knob or a switch that would allow the user to adjust the difficulty of the AI. Second would be to update the moving pieces software to recognize if the robot failed to move a piece. Occasionally, the robot fails to pick up pieces. Once the user presses the make move button again the robot will try to rectify its mistake but we could add extra functionality for the robot to check that it made all of the correct moves after its turn.

Additionally, in terms of movements a lot of the robots' movements are inefficient. It takes pieces off of the board only to put a different piece of the same color onto the board in a different location when resetting the game board. We could make more optimizations on the robots pathing to make its piece placement more efficient. The final improvement that we could've made would be to add voice lines to our robot to interact with the user. Such as the robot voicing words of encouragement or congratulating the user on making a nice move. This would add a more personalized experience.

References

[1] “KiCad Eda,” *Schematic Capture & PCB Design Software*. [Online]. Available: https://www.kicad.org/. [Accessed: 27-Sep-2022].

[2] Raspberry Pi, “Buy A raspberry pi 3 model B+,” *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/.

[3] “MSP430 Microcontrollers,” MSP430 microcontrollers product selection | TI.com. [Online]. Available: https://www.ti.com/microcontrollers-mcus-processors/microcontrollers/msp430-microcontrollers/products.html?utm_source=google&utm_medium=cpc&utm_campaign=epd-msp-null-58700007777668010_msp430_datasheet_rsa-cpc-pp-google-wwe&utm_content=msp430_datasheet&ds_k=msp430%2Bmicrocontroller&gclid=CjwKCAjw-L-ZBhB4EiwA76YzOQ_2xCi92C_VPO66VppERntq0I1y7KOMLyromgBdw4NA9GpU-FWUORoCtBMQAvD_BwE&gclsrc=aw.ds.

[4] “C ++” Stroustrup, Bjarne. (1995). The C++ programming language. Reading, Mass. : Addison-Wesley,

[5] D. Lewis and D. Bailey. A checkers playing robot. [Online]. Available: https://www.researchgate.net/publication/228969205_A_checkers_playing_robot

[6] M. Zribi and E. Sung, “An Autonomous Robot Playing the Board Game Checkers,” The International Journal of Electrical Engineering & Education, vol. 35, no. 1, pp. 14–28, Jan. 1998, doi: 10.1177/002072099803500102.

[7] D. J. Brooks, E. McCann, J. Allspaw, M. Medvedev, and H. A. Yanco, “Sense, plan, triple jump,” in 2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA), May 2015, pp. 1–6. doi: 10.1109/TePRA.2015.7219681.

[8] *Visual collaboration and Version Control Platform for YOUR PCB*. CADLab .io | Visual collaboration and version control platform for your PCB. (n.d.). Retrieved December 13, 2022, from https://cadlab.io/

[9] *An introduction to printed circuit boards*. Cadence. Retrieved December 13, 2022, from https://resources.pcb.cadence.com/blog/2019-printed-circuit-board-an-introduction-and-the-basics-of-printed-circuit-boards

[10] “OpenCV-python tutorials,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html.

[11] Spec, G. (n.d.). *NEMA 17 Stepper Motor*. NEMA17 Stepper Motor. Retrieved December 13, 2022, from https://www.globalspec.com/industrial-directory/nema_17_stepper_motor

[12] *Top 5 small and medium PCB design, manufacturing and Assembly Factory*. PS Electronics. (n.d.). Retrieved September 27, 2022, from https://www.quick-pcba.com/pcb-news/wastewater-treatment-of-pcb-fabrication-process.html#:~:text=The%20wastewater%20produced%20by%20PCB%20fabrication%20process%20also%20contains%20plenty,fluoride%2C%20phosphorus%20and%20other%20pollutants.

[13] “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-andChildrens-Products.

[14] GameColony.com. (n.d.). ACF standard laws of Checkers. [Online]. Available: http://www.chesslab.com/rules/CheckerComments3.html

[15] “NEMA 1 Enclosures,” Nema Enclosures. [Online]. Available: https://www.nemaenclosures.com/enclosure-ratings/nema-rated-enclosures/nema-1-enclosures.html.

[16] “ANSI/RIA R15.06-2012- The industrial robot safety standard” Industrial Safety & Hygiene news [Online]. Available: https://www.ishn.com/articles/107815-ansiria-r1506-2012--the-industrial-robot-safety-standard

[17] Rozenblat, L. (n.d.). *Printed circuit board (PCB) spacing / clearance vs. voltage*. IPC-2221B PCB Trace Spacing / Clearance by Voltage. Retrieved December 13, 2022, from https://www.smpspowersupply.com/ipc2221pcbclearance.html

[18] *ABC checkers and other checkers games*. Justia. Retrieved December 13, 2022, from https://patents.justia.com/patent/6257578

[19] California R & D Center. (n.d.). *Robot Computer Chess Game*. Justia. Retrieved December 13, 2022, from https://patents.justia.com/patent/4398720

[20] The Boeing Company. (n.d.). *Belt Drive Dual Robot Gantry*. Justia. Retrieved December 13, 2022, from https://patents.justia.com/patent/20180215590

[21] “OpenBuilds,” *OpenBuilds Part Store*. [Online]. Available: https://openbuildspartstore.com/.

[22] “My solidworks,” *MySolidWorks - Official SOLIDWORKS Community*. [Online]. Available: https://my.solidworks.com/. [Accessed: 27-Sep-2022].

[23] “CCSTUDIO,” *CCSTUDIO IDE, configuration, compiler or debugger | TI.com*. [Online]. Available: https://www.ti.com/tool/CCSTUDIO.

[24] Hsankesara. (n.d.). *HSANKESARA/draughts-ai: AI based Checkers Game-bot*. GitHub. Retrieved September 27, 2022, from https://github.com/Hsankesara/Draughts-AI

[25] UserManual.wiki. (2018, August 17). *MSP430X5XX and msp430x6xx family user's guide (rev. Q) users*. UserManual.wiki. Retrieved December 13, 2022, from https://usermanual.wiki/Document/UsersGuide.896291051

[26] Quinones, J. (n.d.). *Applying acceleration and deceleration profiles to Bipolar Stepper Motors*. Retrieved December 13, 2022, from https://www.ti.com/lit/pdf/slyt482

[27] Coul, P. (n.d.). *Model L-10 DC Solenoid 2110 Queensway Searcy, AR 72143 Tubular, Pull Type*.

[28] Adafruit. (n.d.). *5V Electromagnet - 2.5 Kg Holding Force - P20/15*. Retrieved December 13, 2022.

[29] *DRV8824QPWPRQ1 active*. DRV8824-Q1 | Buy TI Parts | TI.com. (n.d.). Retrieved December 13, 2022, from https://www.ti.com/product/DRV8824-Q1/part-details/DRV8824QPWPRQ1

[30] *BTN8962TAAUMA1: Digi-Key Electronics*. Digi. (n.d.). Retrieved December 13, 2022, from https://www.digikey.com/en/products/detail/infineon-technologies/BTN8962TAAUMA1/4772018

Costs

Table 2: Full Cost Breakdown

Index	Digikey Part #	Mouser Part #	Open Builds Part #	Qty in Stock	Qty Req'd	Per Unit Price	Cost
1			155-LP (V-Slot® 20x40 Linear Rail 500mm)	Yes	2	$ 6.99	$ 13.98
2			150-LP (V-Slot® 20x20 Linear Rail 500mm)	Yes	3	$ 5.49	$ 16.47
3			230 (Drop In Tee Nuts)	Yes	40	$ 0.39	$ 15.60
4			110-pack (Low Profile Screws M5 8mm (10 Pack))	Yes	4	$ 0.99	$ 3.96
5			1195-Set (V-Slot Gantry Kit - 20mm)	Yes	3	$ 31.99	$ 95.97
6			200 (GT2-2M Timing Pulley - 30 Tooth)	Yes	3	$ 6.99	$ 20.97
7			550 (Smooth Idler Pulley Kit)	Yes	3	$ 5.99	$ 17.97
8			545 (L Bracket)	Yes	10	$ 1.29	$ 12.90
9	527-1029-ND			103	1	$ 34.96	$ 34.96
10	1528-2688-ND			50	1	$ 7.50	$ 7.50
11		571-7966382		4305	20	$ 0.86	$ 8.24
12		571-7966342		9480	10	$ 1.55	$ 9.23
13		78-SI4151DY-T1-GE3		4906	2	$ 1.29	$ 2.58
14		571-7966344		4675	6	$ 3.05	$ 18.30
15	2129-R5524N004A-TR-FETR-ND			508	3	$ 0.91	$ 2.73
16		621-AZ1117CR-3.3TRG1		885	4	$ 0.44	$ 1.76
17	LT1374CR-5#PBF-ND			122	1	$ 13.20	$ 13.20
18	SW1982-ND			1695	2	$ 0.93	$ 1.86
19	A120906TR-ND			314	5	$ 1.70	$ 8.50
20	A123441-ND			51	2	$ 2.85	$ 5.70
21	UCLAMP3301HTR-ND			100000	20	$ 0.56	$ 11.22
22	2SK1828TE85LFTR-ND			11900	10	$ 0.28	$ 2.80
23	CMDSH-3 TR PBFREE			71000	2	$ 0.84	$ 1.68
24	1727-4247-2-ND			17000	3	$ 0.22	$ 0.66
25	SS33FSTR-ND			10399	3	$ 0.69	$ 2.07
26	1655-SB3150CT-ND			11788	5	$ 0.45	$ 2.25
27	445-173480-1-ND			13000	3	$ 0.28	$ 0.84
28	732-2672-ND			500	2	$ 1.11	$ 2.22
29	732-5315-ND			246000	20	$ 0.11	$ 2.26
30	BC1101TR-ND			894000	10	$ 0.16	$ 1.61
31	1255PH-ND			12500	10	$ 0.30	$ 3.04
32	445-173600-1-ND			10000	10	$ 0.22	$ 2.20
33	399-9472-1-ND			27000	10	$ 0.17	$ 1.70
34	445-173377-1-ND			22000	10	$ 0.21	$ 2.10
35	445-173579-1-ND			10000	3	$ 0.35	$ 1.05
36	445-173433-1-ND			1500	2	$ 1.04	$ 2.08
37	445-181284-1-ND			15000	2	$ 0.51	$ 1.02
38	283-5113-ND			200	1	$ 4.67	$ 4.67
39	732-5301-ND			2000	5	$ 1.40	$ 7.00
40	S6104-ND			1000	2	$ 2.29	$ 4.58
41	1655-SMDA12C-4TR-ND			5000	4	$ 0.89	$ 3.56
42	BC1101TR-ND			8888	20	$ 0.16	$ 3.22
43	A120906TR-ND			300	2	$ 1.70	$ 3.40
44	1655-SMDA12C-4TR-ND			7000	4	$ 0.89	$ 3.56
45	PCB Fabrication				2	$ 33.00	$ 66.00
46	PCB Population				1	$ 35.00	$ 35.00
	Pre-existing					**Subtotal**	$ 491.87
45	MSP-EXP430FR2355			129	1	$ 15.59	$ 15.59
46	NEMA17 Steppers				2	$ 11.99	$ 23.98
47	Raspberry Pi				1	$ 35.00	$ 35.00
48	Raspberry Pi Camera				1	$ 25.00	$ 25.00
49	Power Supply				1	$ 20.00	$ 20.00
50	Offboard Components				1	$ 60.99	$ 60.99
51	Offboard PCB Components				1	$ 39.74	$ 39.74
						Total	$ 712.17

Calculations

Stepper Motor Current Regulation

"The PWM chopping current is set by a comparator which compares the voltage across a current sense resistor connected to the xISEN terminals, multiplied by a factor of 5, with a reference voltage. The reference voltage is input from the xVREF terminals." [10]

$$I_{CHOP} = \frac{V_{REFX}}{5 * R_{ISENSE}} = \frac{3.3V}{5 * .5\Omega} = 1.32A(per\ winding)$$