Tartan's Gambit

Authors: Juan Cortes, Luis Ortega, Lillie Widmayer: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of creating a fully remote chess playing experience. Users are able to interact with either the custom board system, or the web app, and have game status updates be received by a remote opponent. The board setup automatically moves pieces to update the game status. Games can be played from the start, or from custom scenarios for practice. Provides a more covid-friendly and customizable chess playing experience than previous solutions.**

***Index Terms*—3D Printing, Ajax, Arduino, CAD, Chess, CNC, Computer Vision, Django, Laser-cutting, Sockets, Raspberry Pi, Remote Interaction**

1 INTRODUCTION

Life during a pandemic can quickly become boring, and although online renditions of popular board games, and even novel versions of such games, have become ubiquitous and instantly accessible, online game play just isnt the same as playing in real life. The Tartan's Gambit is a physical and digital chess application intended to provide the user with an authentic physical chess playing experience without having to be in physical proximity to their opponent, since chess requires players to be closer than 6 feet for an in-person experience. Tartan's Gambit consists of a web server controlled robotic gantry system that moves pieces on a physical board, and a computer vision system that detects movements made by an in-person player. The Tartan's Gambit is designed to help quell the quarantine boredom by immersing the user in an authentic physical chess playing experience without having to leave their home.

To ensure an enjoyable user experience, the Tartan's Gambit must have a response time of 30 seconds or less between a player making a move on the virtual board and that move being reflected on the physical board and vice versa. Once the computer detects a move it is sent through web sockets to the web server allowing for quick state updates. The gantry system must not interfere with surrounding chess pieces when it is moving a piece and must be able to pick up and place any piece on any tile of the board. The gantry raises pieces it is moving above other pieces ensuring no interaction with unintended pieces. After each move the gantry returns to a resting position that is out of the way of the in-person user. The web server must track whose turn it is and notate the game state ensuring that the users only focus on playing the game. The computer vision must be able to detect contrasting color pieces to avoid any confusion for the in-person player. Lastly the system should also be aesthetically pleasing and provide an entertaining experience for the user. The system aims to achieve this by having an above table gantry that does a sequence of streamlines movements when it interacts with pieces.

2 DESIGN REQUIREMENTS

The first requirement for Tartan's Gambit is that virtual moves are reflected on the physical board within 30 seconds. This requires the web server to update its board state database, the Raspberry Pi to transmit the move to the Arduino, and the Arduino to execute the commands on the gantry within the time frame.

The second major requirement is that physical moves are updated on the web server also within 30 seconds. This requires the Raspberry Pi to detect the physical moves using computer vision, send the move to the webserver and have the webserver update the user interface.

For the gantry we are aiming for 99% move accuracy on any piece movement or capture; we aim for 99% because it is possible that there are alignment errors on pieces, or mechanical issues. The tests consist of the gantry moving each kind of piece across the corners of the board and the four center squares. Captures of pieces will be tested in the same manner.

For computer vision we are aiming for 95% accuracy as there are limitations to the processing speed and timing with the Raspberry Pi. The computer vision tests consist of moving each kind of pieces in their normal movements across black squares and white squares to ensure that the computer vision is resilient to piece color contrast and lightning changes.

For the web server testing involves communication between the server and the Raspberry Pi. We aim for 100% accuracy in receiving and sending of moves between Pi and server. This is tested through sending quick sequences of moves and tracking how many were correctly transferred.

To test speed, we are playing moves from recorded games of chess and timing how long moves take to accomplish for the gantry and detect for the computer vision. We aim to have physical and virtual player turns take less than 30 seconds to complete and update within their respective states.

3 ARCHITECTURE OVERVIEW

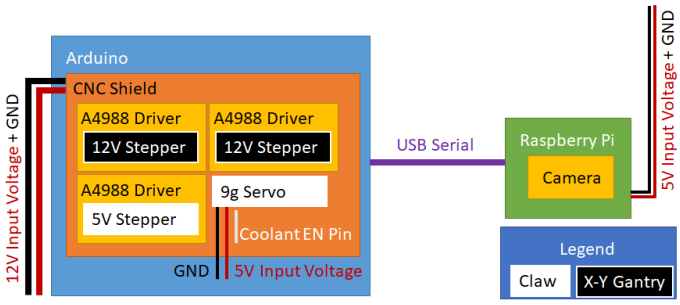


Figure 1: Hardware block diagram

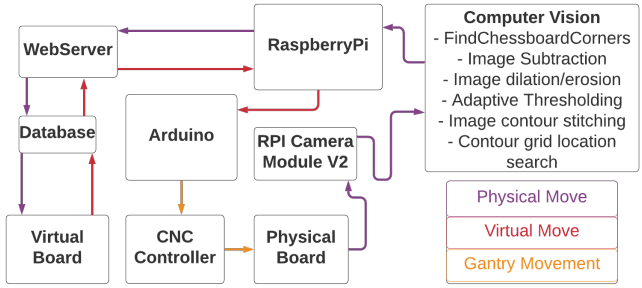


Figure 2: Software flow diagram

The Tartan's Gambit system can be separated into 3 primary components. The first is the physical gantry system that will be interacting with the chess board and pieces. The second is the web client interface that will allow a remote user to control the gantry and play a remote game of chess with the opponent who has the physical board. The third is the computer vision that monitors the physical board state and updates the virtual board after the physical player makes a move.

3.1 Gantry

As shown in Figure 1 the gantry is composed of 2 12V stepper motors for X-Y movement, a 5V stepper motor for Z movement and a 9g servo to move the claw, all connected to the Arduino CNC shield. The stepper motors use A4988 motor drivers. The servo is connected to an empty pin on the CNC shield because servo control is not native to the shield. The placement of the motors in the system is shown below.

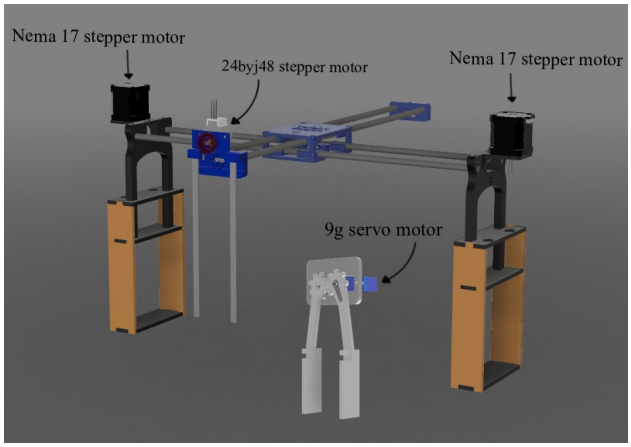


Figure 3: Placement of motors

The gantry control uses Arduino libraries to create commands that can be used to move the gantry in different ways. The main commands are moveX, moveY, lowerClaw, raiseClaw, openClaw, closeClaw. These commands are written using the stepper control libraries and serve as building blocks that create move sequences. The move sequences handle picking up a piece, dropping a piece on another square, and capturing pieces. The moveX and moveY commands control the 12V Nema stepper motors which in turn move the timing belt which moves the gripper. Below is a photo depicting how the timing belt, drawn in purple, is connected to the gantry system.

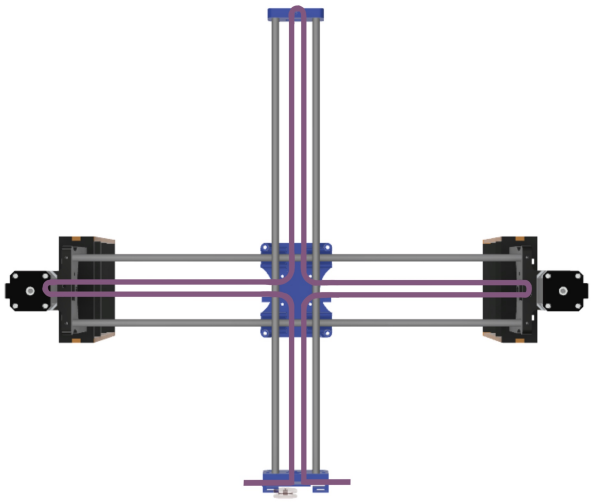


Figure 4: Connection between timing belt and gantry

3.2 Web Client

The web interface is deployed locally via Django and is simple and intuitive to facilitate seamless gameplay. The remote player uses the website to make their moves, and this website is also running on the Raspberry Pi. The Raspberry Pi web page communicates with the virtual player's web page via AJAX to visually update the board state on

the virtual player's side. The web player's page communicates with the Raspberry Pi via sockets to send moves to the gantry which reflects the virtual move on the physical board. Socket is a python library that establishes a wireless connection between a host and a port. In our case the host is the IP address of the device running the web server.

3.3 Computer Vision

As shown in Figure 1, the computer vision lies on the Raspberry Pi and serves as a constant monitoring system of the physical board, able to construct updates for the web server. The computer vision uses the OpenCV library in order to perform all computer vision operations.

The OpenCV monitors the physical board and calculates which squares are involved in any move made by the physical player; the steps and algorithms taken to accomplish this are covered in detail later. The OpenCV also lies at the center of the Raspberry Pi's functionality, including the interfaces to and from the web server and the Arduino.

The interfacing to the web server and Arduino requires the use of three libraries, requests, serial, and sockets. The requests library is used to send HTTP POST requests to the web server with any new moves made by the physical player. The serial library then serves to communicate with the Arduino by forwarding any moves received from the web server, and then awaiting an "ACK" signal along the same serial connection. Lastly, the sockets library is used to receive moves from the web server, which are then forwarded along to the Arduino via the serial library as described above.

4 DESIGN TRADE STUDIES

4.1 Piece Detection

A large part of our design relied on being able to accurately detect the moves made by a physical player. With this came some important decisions on which design paths would be chosen to best optimize user experience. Here we describe one of the more significant decisions that had to be made for each section of the OpenCV code.

4.1.1 Chess Square Boundary Detection

Being able to detect the squares involved in any movement is extremely important to creating a system which allows a seamless game between the physical and remote player. Having this user requirement in place meant the system needed to use the most accurate algorithm it could to determine the boundaries of the squares, however this brought up a tradeoff decision when considering the speed of such algorithms.

Originally we had hoped to use a HoughLines algorithm, which provided quick line detection. An initial round of testing showed that this algorithm could only properly detect up to around 65% of the boundaries present on the board, and was highly sensitive to any changes in lighting. Several weeks were spent trying to modify the usage of this algorithm by tuning its hyperparameters, and preprocessing the board image to create more clear boundaries. This was not proving to help accuracy, and thus we chose to move away from this algorithm and find a more suitable one.

With a fair amount of trial and error on different algorithms, and time spent searching the OpenCv docs, a new algorithm, findChessboardCorners, was found. The algorithm was able to return an ordered array with image coordinates corresponding to the corners of the chessboard squares. Some post processing on the returned array was required in order to determine the outer edges of the squares. After running the algorithm and calculating the remaining coordinates, it was found to be significantly slower than the HoughLines algorithm.

Since accuracy was not something which could be justifiably compromised on, the findChessboardCorners algorithm had to be chosen. In order to reduce calculation time, the board corners would be saved to an array at initialization time, and used for the remainder of the game. We had initially wanted to run the HoughLines algorithm at each loop iteration of the OpenCV program, as it was quick enough that it did not impede on the user experience, and allowed the game to continue with accurate detection even if the board was moved. By the time the boundary detection algorithm was switched, the board spacers had already been implemented, which would keep the board in place. This caused the one time calculation of the boundaries to be justifiable as an approach which would lead to correct results.

4.2 Piece Movement

We primarily considered two options for our piece movement, a robotic arm, and a gantry system.

4.2.1 Robotic arm

We considered using a three-degree of freedom robotic arm because of range of motion and flexibility. We went away from this approach because of the complexities fine control of a robotic arm entails.

4.2.2 Gantry system

We decided to use a 3-axis gantry system to move our pieces. We chose a gantry because it lends itself to be controlled easier than the robotic arm, while still being aesthetically pleasing, and entertaining for a user to watch in action. We determined that given proper stepper motor control we could achieve the speed goals for the system.. In addition, we wanted to base our movement on systems that use CNC controls to move gantries for fine control as this translates well since moving pieces on a chessboard is a grid based system.. We chose to use an X-Y gantry system with stepper motors to move around the board and a gripper for our Z movement of picking up and placing down

pieces. Many of the 3D printed components were adapted from a drawing bot[1] and modified for the system needs.

We chose between below-board and above-board gantries. The below-board gantry hides the piece movement which improves aesthetics, but it does change how the pieces would move. To use a below-board gantry we would need to move the pieces through the board. Pieces would have to be moved around other pieces rather than lifting them up as well. We thought that this made the system unnecessarily difficult and chose to go with an over-board gantry instead. The over-head gantry allows us to meet our requirement of minimal interference with other pieces when moving a piece. We also thought that being able to see the gantry pick up, displace, and drop the pieces added to the user experience.

4.2.3 Gripper Design

We iterated through a few gripper designs before landing on the final version. We were originally planning to 3D print custom chess pieces with a small uniform circular base to make grabbing them easier. We planned to control this gripper with a servo and attach it to a rack and pinion mechanism for the z-axis motion. The circular base of the gripper in the following photo was designed to fit around the circular bases of the custom chess pieces pictured to the right of the gripper. However, we ultimately decided to use chess pieces from an off the shelf chess set to avoid the time and financial cost of 3D printing an entire chess set.

Figure 5: First gripper design (left) custom chess pieces (right)

Our next approach for the gripper was also a claw mechanism, but instead of grabbing chess pieces from the bottom, this gripper would grip the sides of the chess pieces. As shown in the image below, one of the arms of the gripper had a micro servo attachment embedded in it which would allow a servo to open and close the gripper once the gripper was mounted on an acrylic plate. This design of the gripper was close to our final iteration. The primary issue was the gear shape of this gripper had too loose of a fit, so the servo was unable to drive the arms together tight enough to securely grab a chess piece.

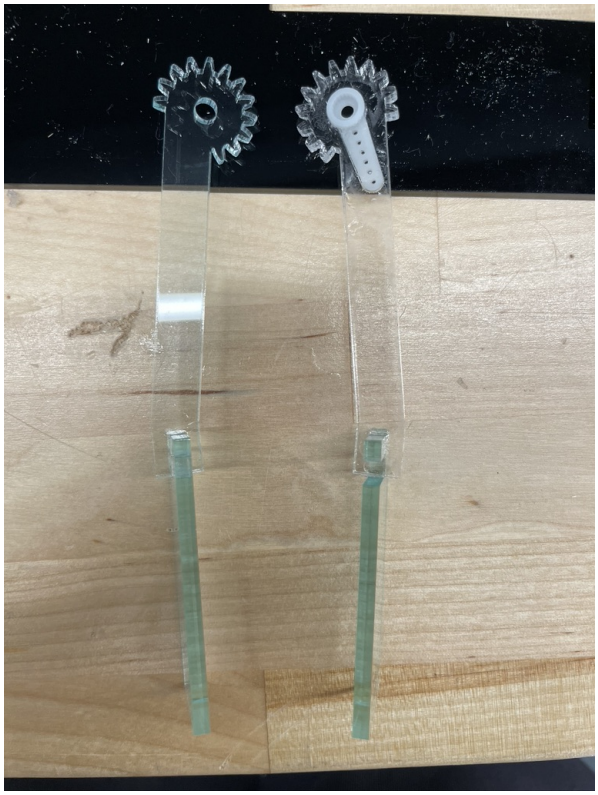


Figure 6: Old gripper design

Finally, we arrived at our final iteration of the gripper. For this design we chose chunkier, rounder gear teeth and decided to have fewer teeth. This design choice allowed the gripper arms to fit together very snuggly and enabled the servo to close it tightly enough to firmly grab chess pieces. Below is a CAD rendering of the gripper model that shows the gripper arms mounted on the mounting plate and how the servo attaches to the gripper.

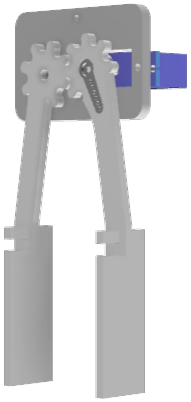


Figure 7: CAD of final gripper design

Attached to the rectangular plates at the bottom of the gripper arms were pieces of a crutch armpit pad. This rubbery material had just the right amount of friction to hold a chess piece without any slipping. By this point in the design and rapid prototyping process we determined that

the best way to facilitate the z-motion of the gripper was through a pulley system. The following picture is the final iteration of the gripper and shows it attached to the servo and clear acrylic mounting plate. The grey portions at the bottom of the gripper are the rubber pieces mentioned previously. Shown attached to the top of the mounting plate is a piece of yarn which was wound around the attachment for the small stepper motor.

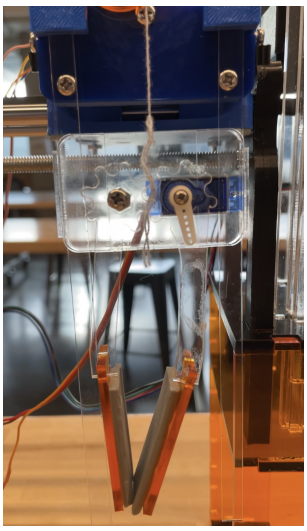
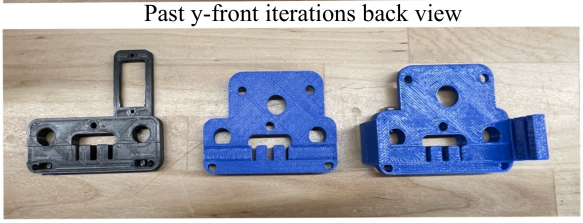


Figure 8: Final gripper

The y-front piece of the gantry went through many iterations as shown below. The first iteration, in black, was when we were going to try using a servo for the z-motion, the second iteration, in the middle, features a place for the small stepper motor but no place for guide rails. We discovered that guide rails were necessary to stabilize the z-motion of the gripper. The right most iteration shows extrusions for guide rails to be inserted. We were initially going to have the gripper go between 2 guide rails on each side.

Past y-front iterations back view



Past y-front iterations front view

Figure 9: Iterations of the y-front piece

We ultimately chose to instead have one guide rail on each side which would be inserted through the mounting plate of the gripper, shown in Figure 8. Below is a rendering of the CAD model for the final y-front with the guide rails attached. The y-front is a 3D printed part and the guide rails are laser cut acrylic.

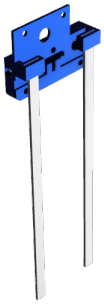


Figure 10: The final y-front component

4.3 Web Client

For our web client we are using HTML, CSS, and Django as they are what we have worked with in other courses. We originally planned to host the web server using Amazon Web Services' EC2 in order to allow the Raspberry Pi to establish a connection and to facilitate remote testing. However, as we were developing we determined that a more efficient method was using the database to store the current move and send this information between the web server and Raspberry Pi using AJAX and GET/POST requests. Using this approach made deploying the web server on AWS unnecessary. The rendering of the chess board and pieces was achieved through integration of chessboard.js[2] which is an open source API we chose to use in an effort to avoid wasting time reinventing the wheel. Additionally, we utilized chess.js[3] to achieve move validation and game state detection. We initially planned to use Django Channels to facilitate communication between the Raspberry Pi and the web client, but we quickly realized that this added unnecessary complexity to the web server development. We also were considering adding various notifications to the user interface such as if the connection to the Raspberry Pi on the physical board was stable, if a move was invalid, etc. Ultimately, we decided to keep the user interface simple in an effort to maximize usability and allow more time to be dedicated to the gantry controls and computer vision algorithms. Below is a series of images of the user interface that depicts each user making 2 moves.

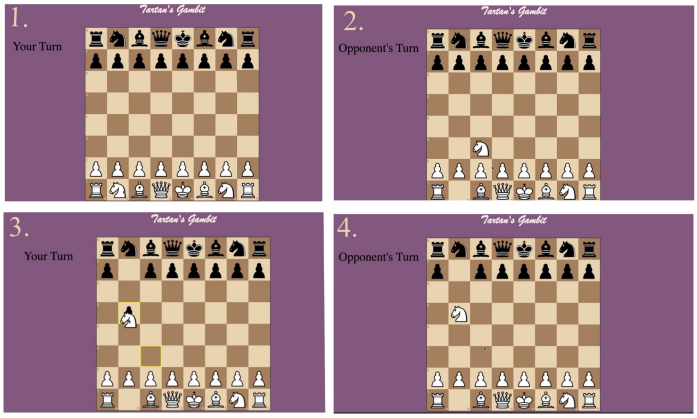


Figure 11: Series of screen grabs from the web client

4.4 Hardware

We chose an Arduino UNO to take advantage of CNC shield controls for our gantry.

We also needed a main hardware element that could interface with the Arduino and had enough computation for the OpenCV and web server hosting.

We considered using the Jetson Nano because of its superior GPU, however our computer vision does not require us to analyze many frames continuously, so we would not be using it to full capabilities. We decided instead to use a Raspberry Pi with a Linux subsystem. The Raspberry Pi also has camera modules that connect through MIPI serial interfacing that we could take advantage of. We found the Raspberry Pi to sufficiently fit our needs for the system.

The control mechanism of the gantry changed from using GRBL and g code which are commonly used in CNC machines and 3D printers to custom software. We decided to make our own software because the GRBL system, while useful for CNC machine control, has many elements that over complicate simple movements. Instead we created our own control platform. We ended up with two platforms, a testing platform that allowed for manual control of the gantry, and the deployment platform that took specific commands to do movement sequences. Because we were writing all the command sequencing and input parsing we could design the system to fit our needs. Commands were simple to interpret and movement sequence became modular. We did not sacrifice any precision for changing control methodologies, instead we reduced the complexity of commands and move sequencing.

5 SYSTEM DESCRIPTION

5.1 Software

The bulk of our software is broken up into two categories, the OpenCV, and the web client. While other components such as the Raspberry Pi and the Arduino will also have important software running on them, these are best described as components of the hardware subsystems.

5.1.1 OpenCV

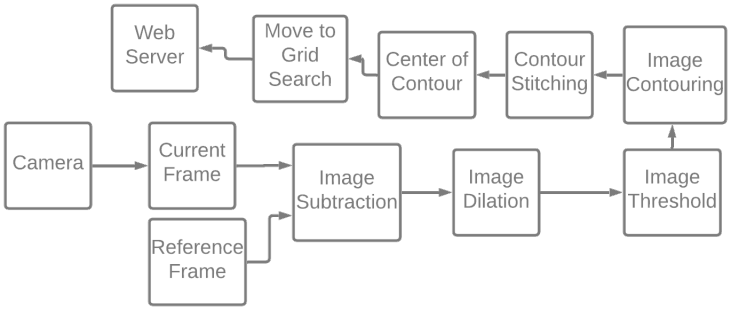


Figure 12: Flowchart for the OpenCV move recognition

Our OpenCV algorithm will be running on a Raspberry Pi 4 with a RPi Camera Module V2 looking down on the board. This setup will be primarily focused on detecting piece movements completed by the physical player, and calculating which squares were involved in the move.

Upon initialization, the OpenCV creates a 9x9 array of coordinates corresponding to the corners of the squares. This is crafted first by running the findChessboardCorners function of the OpenCV library, and then manually processing beyond that. The manual processing begins taking the average distance between coordinates in both the x and y direction. These differences are then added or subtracted to the outermost coordinates in order to get the coordinates for the outer corners.

The CV detection will be running on a loop to show the most up to date board status. When the OpenCV detects no active movement in frame, it calculates an image difference to determine if a move has been made. When a move is made, a signal to the web server, updating the game state, will be sent via an HTTP POST request. The occurrence of a move is done by taking the image difference, from the new and old frame, performing a morphological opening on the resulting image. At this point, the image is thresholded so that all areas involved in the move have high contrast from the rest of the image. The thresholded areas are then separated into contours, which are stitched together when found to be within 20 pixels from each other. The center of contours is then calculated on these newly stitched contours, and taken as the coordinates of movement. These coordinates are then translated into square coordinates based on the grid determined during initialization, and then these square coordinates form the move to be sent to the web server.

Additionally, the OpenCV is listening for signals from the server, on an open socket, to alert it of a remote move which occurred and needs to be updated. In this case, the algorithm will take the board coordinates received from the

server, and forward this information to the Arduino, where the CNC controller will finish the move.

When interfacing with the Arduino, the OpenCV halts all calculations. This is done by awaiting an "ACK" to return from the Arduino once the move has been completed. Upon receiving the "ACK" the image used as a baseline or reference in move detection image subtraction is updated, such that it includes the new move. At this point the OpenCV algorithm restarts the operation of its loop, and then continues processing.

5.1.2 Web Client

The web client is the piece that brings the whole project together as this is what initiates gantry movement. The web client is deployed from a laptop. Two instances of the webpage are run, one on the Raspberry Pi and the other on a laptop. The laptop instance is what the virtual player interacts with while the Raspberry Pi instance is how commands are sent to the Arduino for gantry movement. Communication between the laptop and Raspberry Pi is facilitated by POST requests and a socket connection. The socket connection is how the laptop instance sends moves made by the virtual player to the Raspberry Pi; this connection is established in the CV code because the CV algorithms need to know a physical move is being executed so they can halt operation. The Raspberry Pi instance uses POST requests to update the move stored in the database, and the web page polls the database for changes every 5 seconds to ensure that the physical board state and virtual board state are synchronized. Below is a flowchart depicting the communication between the Pi and laptop instances. The Pi instance is denoted as the physical player and the laptop instance is denoted as the virtual player.

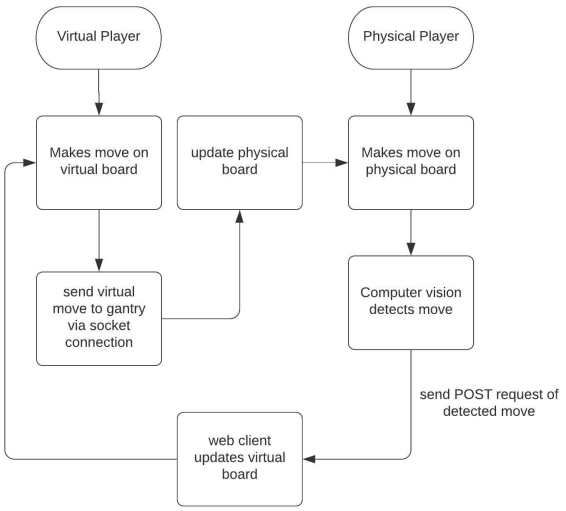


Figure 13: Flowchart for the web server

5.2 Hardware

5.2.1 Arduino

The Arduino will be placed in an acrylic housing located beside one of the tall gantry supports as shown in figure 5. The grey box in the right half of the image represents the Arduino. Also contained in this box and connected to the Arduino are the CNC shield and stepper motor drivers. The stepper motors will each be wired to their own stepper motor driver which are connected to the CNC shield. The Raspberry Pi will send data to the Arduino in an encoding string. This data will then be used to drive the stepper motors to the desired position. The micro servo will be attached to the grabber and wired to the Arduino. Using the state changes recognized through CV, 2 locations will be sent to the Arduino, the piece location and piece destination. When the grabber arrives at the first location, the piece location, the micro servo will be triggered to activate the gripper motion to retrieve the desired piece. Next, the grabber will move to the destination location to place the piece. The data will be received from the Raspberry Pi at a baud rate of 9600. The stepper motors will receive power from the Arduino which will be plugged into a wall outlet using a 12V 2A power adapter.

The libraries used for the gantry control are AccelStepper which is a library used to interface with stepper motors, and MultiStepper which is a library that allows simultaneous running of multiple stepper motors, which is necessary to achieve movement in just the X and Y directions in the gantry. To move in +X both motors must run the same number of steps in the same direction. To move in +Y both motors must move the same number of steps in opposite directions.

Moves are received by the Arduino from the Raspberry Pi through serial at the 9600 baud rate. The command that is transferred is in the form of "$< A1 - B1 >$" where the first letter and number refers to the location a piece is coming from and the second letter number pair corresponds to the square the piece is going to. The move is parsed into these two square locations. Then the letters are subtracted from one another, likewise for the numbers. This gives the relative changes in X and Y positioning on the board. The first move is also subtracted from the home location of H8 giving the X and Y position relative to home.

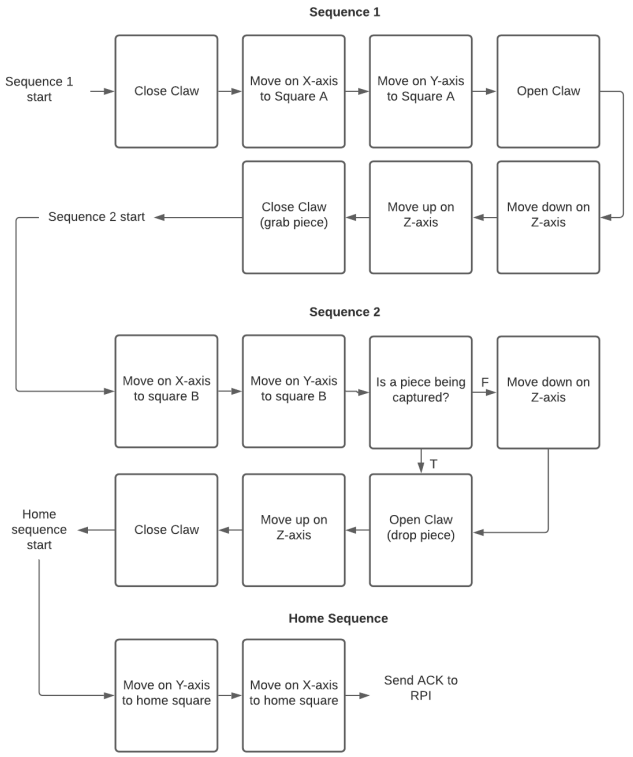


Figure 14: Gantry control flow diagram for piece movements

The relative home position and positions between squares are used as inputs to the gantries movement sequences. On startup the gantry will move from its resting position of 0,0 to the home location of H8. Once a command has been parsed and the relative differences are determined the gantry goes through two major sequences. The first sequence closes the claw, moves the gantry on x and then y, lowers the claw, grabs a piece, and raises the piece. The second sequence moves on x and y to the second square location, lowers the claw, drops the piece, raises the claw, and then goes back to home.

In the case of a capture the command is split into two moves. The first move is the square that is captured to the dispose square I8. The second move is the piece that is captured moving to the captured square. For example if a piece on B3 is capturing C4 then the input command is "$< C4 - XX - B3 - C4 >$".

Once a move is completed the arduino sends an "ACK" signal to the Raspberry Pi to signal that a move is complete and begins listening for another command.

5.2.2 Raspberry Pi

The Raspberry Pi is placed in a housing located on top of the stilt that holds the camera and gantry. It is running the standard Raspbian OS, a Linux distribution made for Raspberry Pi. The Raspberry Pi is powered via a usb-c connection and can be powered from a laptop or directly to a 12V outlet. The Pi transfers data to the Arduino through the Arduino's USB Serial connection at a baud rate of 9600. This is sufficient enough for us to transfer the move commands to the Arduino in the following format "$< src - dst >$". This communication requires the import of one library, the "serial" library.

The system camera is a Raspberry Pi Camera Module V2 that is connected directly to the Pi through its ribbon cable. The camera and Pi communicate through a MIPI serial interface protocol. The camera is accessed directly through the OpenCV library. This camera provides an 8mp resolution, with a pixel size of $1.12\mu mx1.12\mu m$. This resolution provides us with more than sufficient accuracy for detecting moves and square boundaries.

The Pi will be connected to the internet through Wi-Fi. It will output new moves to the web server via HTTP POST requests, in the format "$< square1 - square2 >$". Information received comes from the web server through an open socket. This means that communication with the server requires two imported libraries, "requests" and "sockets".

5.2.3 Gantry

Mounted on the gantry will be the GT2 timing belt, 20 teeth 5mm bore timing pulleys, 500mm M8 linear rods, LM8UU linear bearings, micro servo and grabbing mechanism, and Nema 17 stepper motors. The timing belt and pulleys, controlled by the CNC shield on the Arduino, will drive the grabber to the appropriate locations. The linear rods and linear bearings will allow for smooth x-y movement. An acrylic mounting stand for the Raspberry Pi and Pi camera module is placed in the orange/black acrylic stands. The following diagram shows 3 CAD models, the upper left is the assembled gantry all together without the gripper, the lower left is an isolation of the 3D printed components, and the upper right is an isolation of the laser cut components. The gripper, which is entirely made from laser cut acrylic, is shown in Figure 3.

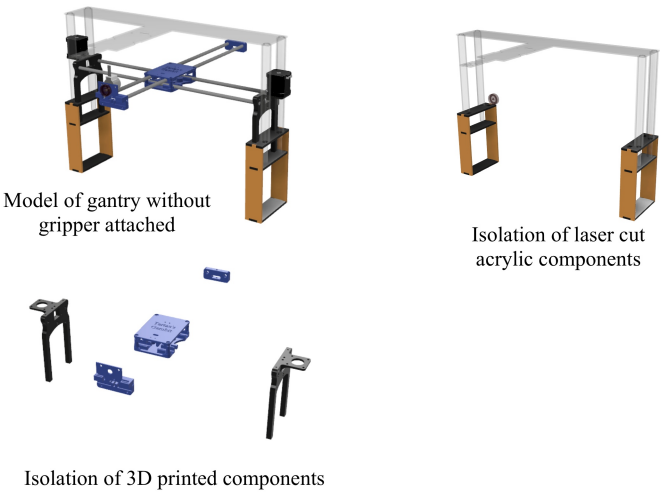
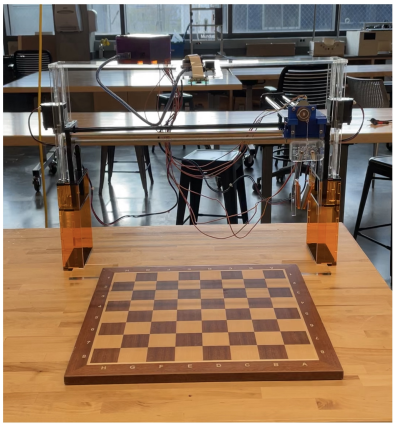
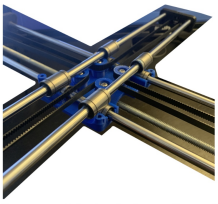


Figure 15: Isolation of different parts within the CAD

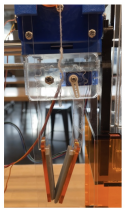
The black/orange acrylic stands were necessary to achieve the height above the board needed for the Pi camera module to maintain a full view of the board and to achieve a large enough z-axis range to allow the gripper to pick pieces up and move them over the board without interfering with other pieces. The following photo is our gantry system setup in the real world.



View of assembled gantry



Close up of timing belt configuration



Close up of gripper mounted on gantry

Figure 16: Photos of the final gantry

The purple acrylic box placed on top of the camera mount houses the Arduino fitted with the CNC shield and includes holes in its side for cable management. The blue cable in the picture is the serial USB connection between the Arduino and Raspberry Pi.

6 TEST & VALIDATION

6.1 Gantry

Gantry movement was tested on pawns, knights, bishops, rooks, queens, and kings. Each piece was moved to and from opposing corners, and to the four center squares of the board. Each movement was tested three times to ensure that the gantry was consistently achieving moves. The tests were conducted continuously to ensure that the moves could be complicated in games that went for 40 plus moves. In our tests we found that out of 216 movements, there were only 3 errors that occurred, giving us a 98.6% success rate. In two cases the knights were rotated at 45 degrees which caused issues for the claw. The other error case occurred when a rook was grabbed too low which caused the piece to slip when the gantry began to do far movements.

Capturing pieces was tested in the same manner as described above. The difference being that before the piece moved from square A to square B, there was a piece to move from square B to the disposal location. In these tests we found 12 errors out of 432 moves, giving us a 97% success rate. We saw more errors when testing captures due to the human player placing pieces off center or in off angle orientations. There were 6 knight movement/capturing, 4 with pawn movement/capturing, and 2 with rook movement/capturing.

Movements were timed during these trials. The average time to move a piece to another square was around 33 seconds. The average time to capture a piece was around 68 seconds. Capturing a piece takes much longer due to having to move two pieces to different squares. The single move time was close to our goal, however the capturing sequence is much higher than we wanted. Since captures are less frequent than normal moves we determined that having moves and captures take this long was sufficient for our needs.

6.2 Web Client

The web client was first tested by playing a full game solely on the web page to ensure that the user could move pieces properly and the game state was detected appropriately. The next step of testing was sending POST requests via a terminal command from a separate laptop to ensure that the board state could properly update on the web page when a POST request from a separate device was received. We then tested the socket connection by setting up one device to receive information packets and the other device, the laptop the virtual player was making moves on, to send information packets. To test this we would make moves on the web application and print the data received to the terminal to ensure the correct move was being received. After both methods of communication were thoroughly tested we integrated them with the computer vision code and website code. To test functionality after integrating we would make a move on the web client and see if the gantry performed the correct move. After a virtual move was made, we would make a move on the virtual board and see if the website updated with the correct move.

6.3 OpenCV

The testing of the OpenCV involved validation of the square coordinates, and the move calculations.

Beginning with validating the coordinates at the corners of the squares, we sought accuracy across many runs. By overlaying the coordinates on a live feed of the board, we were able to determine if the coordinates were accurately calculated. An earlier version of the code led to accurate calculations only about 80% of the time. It was determined this was due to the fact that the orientation of the board returned by findChessboardCorners was not always consistent across runs, this was a simple fix by checking the orientation before continuing calculations on the grid. This led to consistent and accurate results across runs. The other main point of validation was the move calculation. Early iterations of the code were exhibiting low accuracy with approximately 50% accuracy, as moves were further from the center of the board, the chances of them being miscalculated rose significantly. Additionally, shadows cast on the board would cause the contour centers to be calculated in

adjacent squares, as the shadows would be included in the contour. Fixing this involved adjusting hyper parameters used in all the image processing functions, as well as processing the board images prior to taking their differences. This was as simple as dilating the board, performing a median blur, and then taking the absolute difference between the board image, and the one that was dilated then blurred.

7 PROJECT MANAGEMENT

7.1 Schedule

Figure 17 features our most up to date Gantt chart. Our most recent updates include changes to progress statuses, and many task date changes, with many being pushed back to beyond the time for their proposed stage.

The chart in the image is broken into three sections, blue representing planning and research, green representing the design and implementation, and orange representing finalization. The column of green cells represents our progress with each task, and as the green indicates, we have finished all portions of our project.

The main tasks that required delays were the building of the web server, along with the integration of subsystems. Determining the interfaces for the individual subsystems, and then building and polishing them simply took more time than we had anticipated.

The final stage was mostly reserved for full system integration, robustness testing, and documentation. This has mostly remained the case, however extra time had to be spent those last weeks to account for the extra tasks carried over from the second stage.

7.2 Team Member Responsibilities

Our work was split into a few different categories, the webserver, the gantry controls, the OpenCV, and the design of the gantry.

The web server was headed by Lillie. This involved creating the server and frontend, managing the database, and working with the interfaces to the Raspberry Pi. Juan helped with creation of the web server, as well as research into different implementation strategies. Luis assisted with testing of the frontend, and the interfaces to the Raspberry Pi.

Gantry controls were led by Juan. The main roles here involved configuring the Arduino to control the gantry, calibrating the gantry movement to the squares, calibrating movement for grabbing pieces, and reading instructions from the Raspberry Pi. Luis' contributions to this subsystem involved assisting in code reviews when debugging, as well as aiding in the calibration of the z-axis and grabber mechanism.

The software environment was managed by Luis. This involved calibrating the OpenCV environment to detect board boundaries, and centers of piece movement, outputting the squares involved in a move, and interfacing with the web server and the Arduino. Juan assisted in research into different image processing methods.

The design of the full gantry system was done by Lillie. This involved creating models for 3D printing and laser cutting, as well as modifying designs to account for issues which arose during the implementation of the design. Juan and Luis contributed by assisting in the construction with the pieces made by Lillie.

7.3 Budget

Figure 18 below, shows our Bill of Materials. We are well under budget because many of the electrical components we already had and most of the acrylic we used for the laser cut parts was from the scrap pile in TechSpark. In the budget table, a quantity of 0 indicates that either we used found scrap material or already had the item.

7.4 Risk Management

One of the largest risk points we found was the material on the gripper design. With our initial design utilizing foam pads on the arms of the grippers, we noted poor grip strength along with a wide radius which could interfere with other pieces. This involved a search for other materials which could solve these issues, this is where we landed on the high friction rubber. This is an example of our openness to new materials and concepts throughout the implementation in order to address issues which could arise.

Our schedule also had risk management built in, with slack time incorporated into our plan. This will allow for any falling behind, or unforeseen errors. Additionally our task break up is rather fluid, in that individual tasks could be switched to a team member that was currently taking on less work.

Lastly we have our budget. Our spent budget does include a few extra components of those we believe may fail throughout our design process. Our main concern was with the cost of fabrication. Our initial plan of 3D printing many components seemed to be too costly with campus resources available to us charging high rates per gram of filament. We had researched other sources to save costs, but in the end we were able to still print using campus resources and stay well within budget.

Overall our risk management has been to account for unforeseen circumstances to arise, and thus avoid being delayed in our process later on.

8 ETHICAL ISSUES

The largest ethical issue that can arise with this project is security related. As with any internet connected device, there lies an inherent risk of the device being accessed by a malicious agent. Pairing this risk with the inclusion of the camera, there becomes a risk of remote and malicious camera access. The steps which can be taken to reduce this

risk include disabling ssh access to the Raspberry Pi, and authenticating the communication between the web server and the Raspberry Pi, ensuring that external communication comes from a trusted source.

9 RELATED WORK

The best case of related work we were able to find are the boards made by SquareOff. They offer a smart board, allowing users to play against other players remotely using either the board or an app, or play against AI. Our design should prove to be simpler in that the SquareOff boards use magnets to move the pieces around, which can lead to complicated circuitry and mechanics. Additionally, since the SquareOff boards have all the components embedded within, the game must be set up very precisely, and the cost is relatively high due to requiring the custom set. Our approach, given reasonable production methods, should run at a lower cost, as it can be used simply as an addition to one's current set. Additionally, the inclusion of computer vision in our design allows for the game to be set up in various ways, granting more freedom than the SquareOff board provides.[4]

Another similar device would be one which was described in the International Journal of Engineering Research & Technology, Volume 6 Issue 9 from September 2017, titled Design and Implementation of a Wireless Remote Chess Playing Physical Platform. This takes a very similar approach to the SquareOff board, however here we see how the design was implemented. Using magnets on the pieces, and a hall sensor to detect piece location, an under the board gantry system controls an electromagnet in order to move pieces. While this leads to a similarly simple design as ours, it suffers some of the same downfalls as SquareOff, namely that the cost must include a custom chess set. Additionally, hall sensors are very sensitive, and the reliability of this system may not be the best, especially with so many magnets close together on the board. The complexity needed to accurately calculate piece location proved to be an issue for the designers, as Arduinos were too limited in memory, and Raspberry Pis were too limited in GPIO pins.

Overall while our approach leads to a design less sleek than these two boards, we aim to offer more freedom, at lower cost, and will likely not face issues with memory storage.

10 SUMMARY

In the end we were able to successfully meet our design requirements, except for the speed of a move being carried out by the gantry. Fixing this would require a more appropriate motor to control the z-axis motion. Our current motor faced strong limitations on the speed, as it would quickly overheat beyond a safe operating point.

Given more time and resources there are a few things which could have improved performance. First there is the upgrade of the z-axis motor as described above. We also noted that the Raspberry Pi faced large limitations in its processing capabilities. The opportunity to host the OpenCV algorithm on a stronger processing unit would lead to faster processing times which would have allowed the algorithm to be flushed out to a point where the movements could be detected with higher accuracy and speed.

Another point of improvement would be improving the interface between the Raspberry Pi and the web server. While communication across this interface was stable, there would occasionally be latency issues from the Raspberry Pi. Some moves would take longer to send to the server, and some moves would be received but not processed for longer times than others. This was likely due to the Raspberry Pi completing its current operations before acknowledging the interface. A solution could be as simple as having the Raspberry Pi operating with interrupts to immediately read/write to the interface.

10.1 Lessons Learned

Important takeaways from this project would be to plan for the unexpected. When issues arose, we often had to delay our goals in order to address them. Had we not had slack time built into our schedule, this would have led to a backup of tasks that may not have been addressable.

Additionally, found materials can be both a blessing and a curse. The new rubber material for the gripper came from spare material and was exactly what was needed. However, the z-axis motor which led to a low cap on z-axis motion was also a found material. Having openness in your design for unplanned materials and concepts is a great way to address implementation issues which may arise.

11 REFERENCES

1 Henry Arnold and Jonathan K Drawing Bot Guide, www.thingiverse.com/thing:2349232

2 chessboard.js, www.chessboardjs.com

3 chess.js, github.com/jhlywa/chess.js

4 Square Off, Inc. www.squareoffnow.com

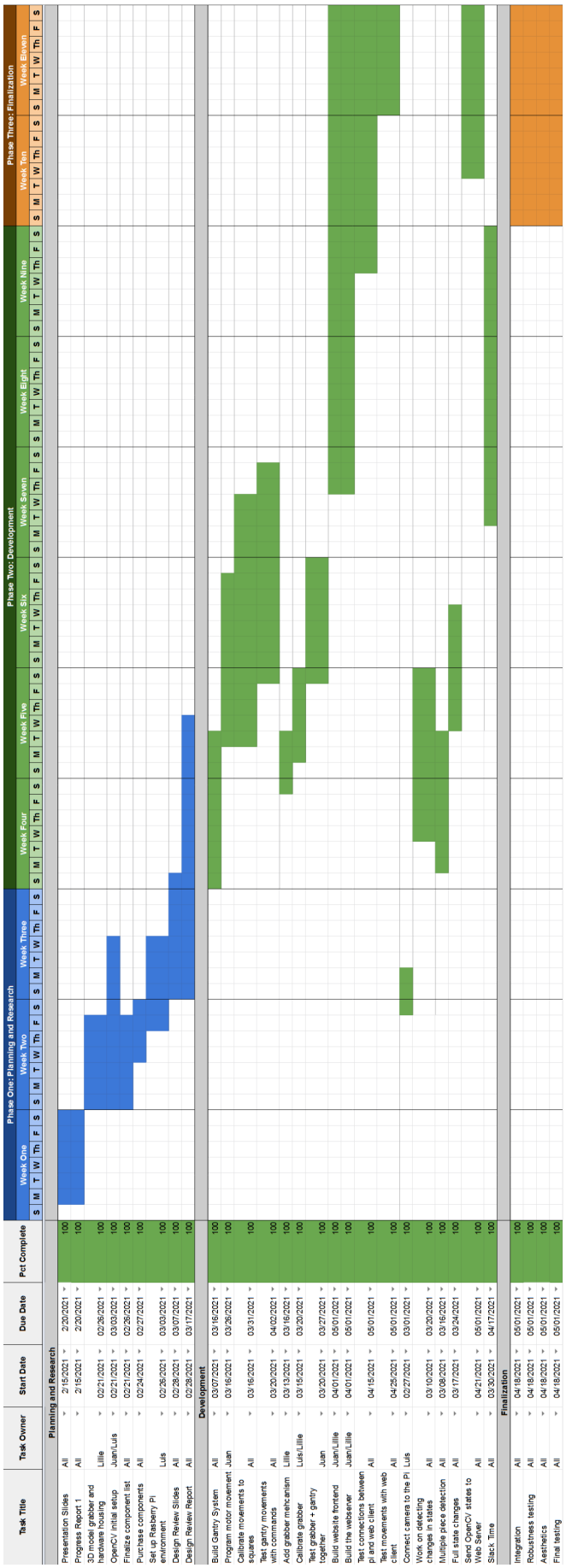


Figure 17: Final Gantt Chart

Item	Price	Quantity
Raspberry Pi	0	1
Arduino UNO	0	1
Arduino CNC shield	0	1
Nema 17 Stepper Motor	10.99	2
Linearing Bearing Rod M8 x 500 mm	19.89	2
624 Bearing	12.99	1
M4 Screw Kit	18.99	1
Timing belt and pulleys	15.99	1
M8 Nut Kit	14.99	1
Pi Camera module	25.49	1
chess board	49.99	1
lm8uu linear bearings	10.95	1
3D printed parts	150.58	1
12V 2A power adapter	0	1
1/4 inch acrylic	20	1
1/8 inch acrylic	0	1
24byj48 stepper motor	0	1
2mm acrylic	0	1
Gorilla Super Glue	9.99	1
Acrylic Glue	10.99	1
Armpit cushion from crutches	0	1
yarn	0	1
Total Cost:	402.71	

Figure 18: Bill of Materials