Department of Mechanical and Mechatronics Engineering

Connect 4 Robot

A Report Prepared For:
The University of Waterloo
Course Code ME101

Prepared By:
Kayden Thomas (-), Rein Keppo (-)
Charis Ho (-), Grace Dice (-)

150 University Ave W.
Waterloo, Ontario, N2N 2N2

July 30, 2024

Table of Contents

List of Figures

List of Tables

Summary

The Connect 4 robot is made of Lego EV3 components PLA 3D print and acrylic laser cut parts. The robot can play the board game Connect 4 against a person by dropping pieces into the board and keeping track of both it and the player's moves to recognize a win. The design that was made prioritizes speed and simplicity. The dropper system consists of a piece storage container with a rack and pinion pushing the pieces up out of the container. The raised piece is pushed by a crank and piston into a funnel over the board to turn the piece upright. The system is moved on a base with wheels that drive linearly to the correct column. The robot prompts the human player to input their move so that it can store each move in a 2D array and check that array for a 4-piece connection to determine if a win is there. The robot was tested for each function that was programmed. The motor distances were fine-tuned by testing using the game board and pieces to relate the code with the physical game components. Once each function was tested and worked as expected, the whole code came together to be able to run test games. The testing of the robot revealed flaws in its design. The dropper being placed on a moving base is a prominent problem because of the inconsistency caused by the Lego motors and wheels. A recommendation to avoid this problem is to make a track that is connected to the board and therefore fixed to the board. To make playing the robot more enjoyable, the ability to edit past moves would be good to be able to fix an accident. Larger text on the display would be helpful for the player reading the messages that are output.

1.0 Needs Analysis

Connect 4 is a classic game that requires two players. To play this exciting game individually requires mechanical and software design. The desired solution must satisfy various functional, non-functional, and constraint requirements related to gameplay and game rules.

1.1 Background

Connect 4 is a classic two-player connection game where the players take turns dropping coloured discs into a vertical grid. The game's objective is to be the first to form a horizontal, vertical, or diagonal line of four of one's discs.

The primary objective of this project is to create an innovative way for a traditional multiplayer game to be played by a single player. This involves designing and implementing a robotic system capable of autonomously playing Connect 4 against a human opponent.

This project was chosen for various reasons. Firstly, it is a challenging project intended to develop both mechanical and software design skills. It was anticipated that the various winning combinations and the vertical board setup would create some design and software challenges. The game Connect 4 is universal and enjoyed by people of all ages. The Connect 4 robot will allow people to enjoy the game even if they are unable to find a second person to play with them. The system should be able to replace the second human player, allowing Connect 4 to become a single-player game.

1.2 Need Statement and Engineering Specification

Listed below are the various functional requirements, non-functional requirements, and constraints that guide the design of this device.

Functional Requirements:

- Place Pieces in Gameboard: the system must be capable of accurately placing pieces into the gameboard's columns based on play input.
- Recognize Winning Combinations: the system should detect winning combinations to determine the end of the game (four pieces in a row horizontally, vertically, or diagonally).
- Detect Piece Colour and Placement: the system needs to ensure that pieces are placed correctly and that their colours are accurate.

Non-Functional Requirements:

- Speed: the system should operate within a reasonable time frame to ensure smooth gameplay. It should respond promptly to player moves and make decisions quickly.
- Easy to Use: Design should be easy to use and intuitive to play with.

Constraint Requirements:

- Cost: the entire project must be completed within a budget of 100$, including all materials and components.

The following is a need statement that outlines the primary functional requirement and constraint requirement of this device. A need exists to develop a robotic system capable of autonomously placing Connect 4 pieces into the gameboard while maintaining a total project cost under $100.

All the specified requirements were placed into an engineering design specification, as shown below.

Table 1: Engineering Design Specification.

No.	Characteristic	Relation	Value	Units	Verification Method	Comments
1	Place pieces in gameboard	>	90	%	Test	Observe game and tally how many successful pieces are placed.
2	Recognize winning combinations	=	100	%	Test	Play 10 games and see if winning combination is correctly recognized each time.
3	Detect that it is stocked with pieces of the correct colour	>	80	%	Test	Place 10 incorrect pieces into the robot and tally how many are recognized as incorrect.
5	Speed	<	25	seconds	Test	Time how long it takes to make moves, average results.
6	Easy to use	>	80	%	Test	Take a survey to have people rate the ease of use from 1 to 10, average results.
7	Cost	<	100	$	Analysis	Add up receipts to find total cost.

2.0 Conceptual Design

There are endless different ways to achieve the goal of a Connect 4-playing robot. While ideating possible concepts for the robot, the functional, non-functional, and constraint requirements need to be considered. The robot needs to be able to place connect 4 pieces, analyze human moves, and reload the supply of robot pieces. To come up with solutions to these functions, a morphological chart can be used. Figure 1 shows the morphological chart used for the Connect 4 robot.

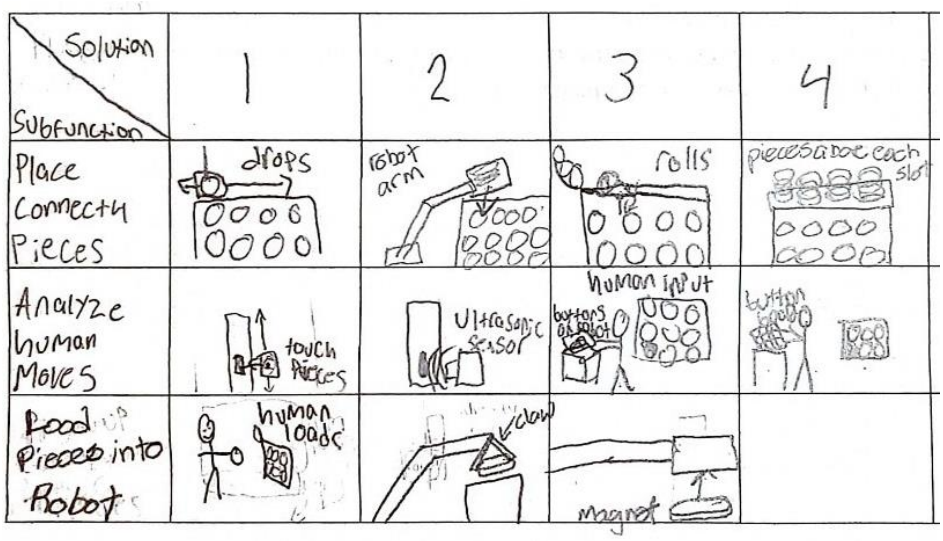

Figure 1: Morphological chart for Connect 4 robot.

2.1 Concept #1

Figure 2 displays the first concept for the robot. A dropper is chosen to place a piece into the board, with a stack of pieces loaded behind the dropper. The dropper would move across the board using motors and drop a piece into the column the robot wants to place in. A button board using human input is chosen for the method to analyze human moves. Pieces would be reloaded into the robot.

Figure 2: Concept Design 1.

2.2 Concept #2

Figure 3 has concept 2 for the conceptual design of the robot. Pieces would be stacked above each column to be released into the board when the robot wants to place it in the respective column. A touch sensor would check each possible place the human could have placed to find out where they placed to analyze the human moves. An arm with a magnet would pick up the pieces that would have magnetic material attached to pick up pieces and reload the robot's stock.

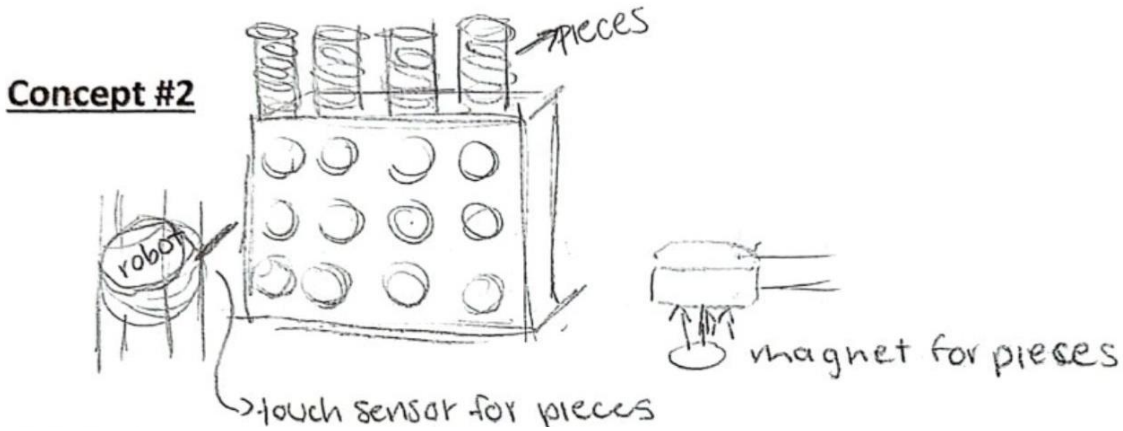

Figure 3: Concept Design 2.

2.3 Concept #3

Figure 4: Concept Design 3 is the third conceptual design for the Connect 4 robot. A claw would be used to pick up pieces and place them into the board. This design would handle placing pieces and loading pieces. Like design 2 the robot would check for possible human moves but instead of using the touch sensor the ultrasonic sensor would be used instead.

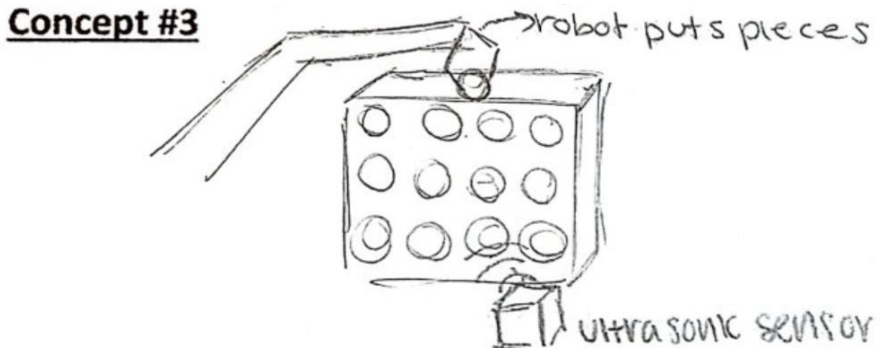

Figure 4: Concept Design 3.

2.4 Decision-Making Matrix

Table 2 presents the decision matrix for the three conceptual designs. In this matrix, design 1 is set as the datum to compare to. The first criterion to analyze is the cost to make the design. The next is the time it takes the robot to be able to do a move or the speed of the robot. Then it is the simplicity of the design or how difficult it would be to design. After that, the safety of the design is considered. The last criterion is how autonomous the robot is. Both concepts 2 and 3 are more autonomous than the datum design 1. However, to be more autonomous, the robot would need to check every possible option of the humans' moves, which would make it far slower than just having the person input their moves. Both designs 1 and 2 are more complex than design 1 because they involve an arm to pick up Connect 4 pieces. Picking up and placing the pieces would be very complex compared to the dropper design in Concept 1. It is because of the time limit that project concept 1 is the best design for the Connect 4 playing robot. It is the fastest and most simple design.

Table 2: Decision-Making Matrix.

Criteria \ Concepts	1 (datum)	2	3
Cost	0	0	0
Speed	0	-1	-1
Simplicity	0	-1	-1
Safety	0	0	0
Autonomous	0	1	1
Total	0	-1	-1

3.0 Mechanical Design and Implementation

The Connect 4 robot's mechanical design involves various mechanical components, to handle game pieces and interact with the game board.

3.1 Piece Handling System

The piece handling system is made up of three different sub-systems including how the pieces are stored, how to raise the pieces, and how to place pieces into the game board.

3.1.1 Dropper Piece Storage

The dropper is a crucial component within the robot to store pieces and allow for smooth dispensing. The initial CAD model, as seen in Figure 5, was created and stuck through as it worked well during the initial testing stages. The two-part system consisting of a vertical storage unit that

held approximately 11 game pieces at a single time and a funnel-like top that directs the pieces into the game board was 3D printed from PLA as seen in Figure 6.

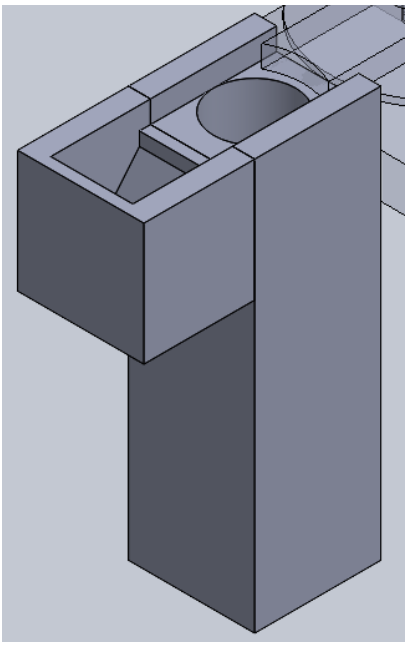

Figure 5: Dropper Initial CAD.

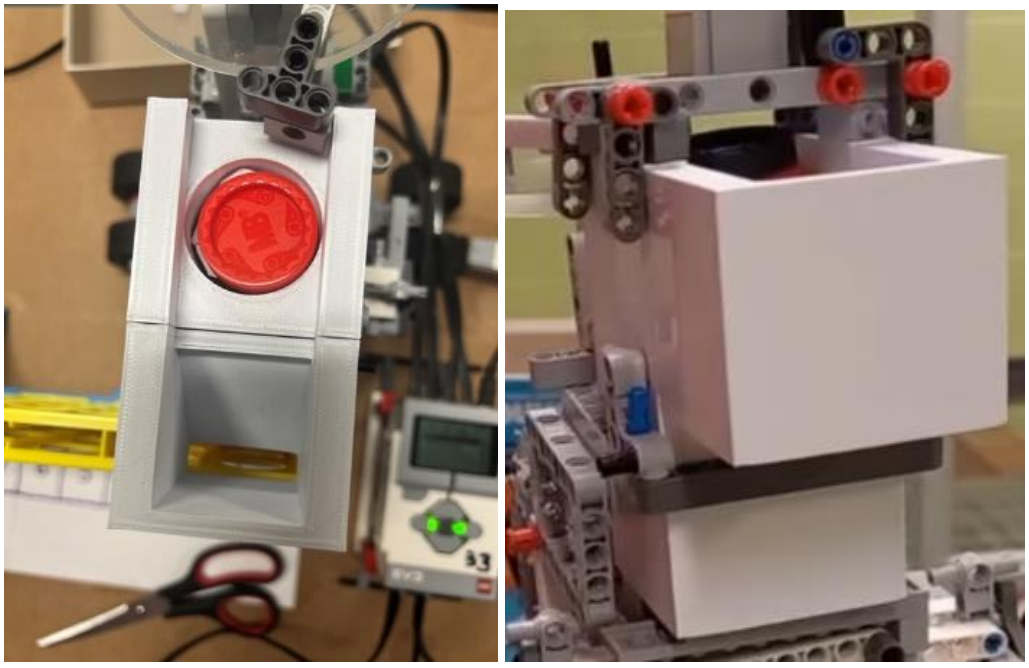

Figure 6: Dropper final design.

3.1.2 Gear Rack and Pinion Mechanism

The rack and pinion mechanism is used to shift the game pieces up the dropper allowing them to be later pushed by the crank and piston system into the chute of the dropper. As seen in Figure 7, the mechanism consists of one leading gear and one following gear and the rack is made from ¼" acrylic laser cut.

Figure 7: Gear rack and pinion within robot.

3.1.3 Crank and Piston System

The crank and piston system is responsible for pushing the game pieces that have been raised from the rack and pinion mechanism into the chute of the dropper to be placed into the game board. The initial iteration was created in CAD, as seen in Figure 8. However, while testing, it was found to be too loose and lacked precision, which is where the final design comes into play, as seen in Figure 9. It is made from one ¼" acrylic laser cut crank and some Lego pieces to make the piston. Through the momentum of the piston pushing the game piece into the chut by the crank's motion, the game piece can slide into the game board with ease.

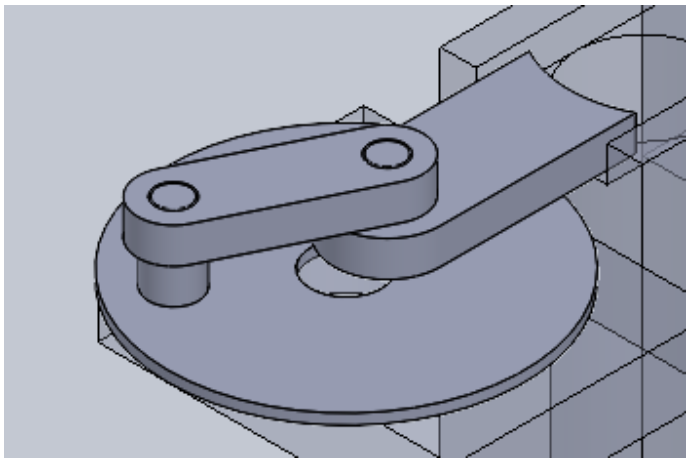

Figure 8: Crank and Piston CAD Design.

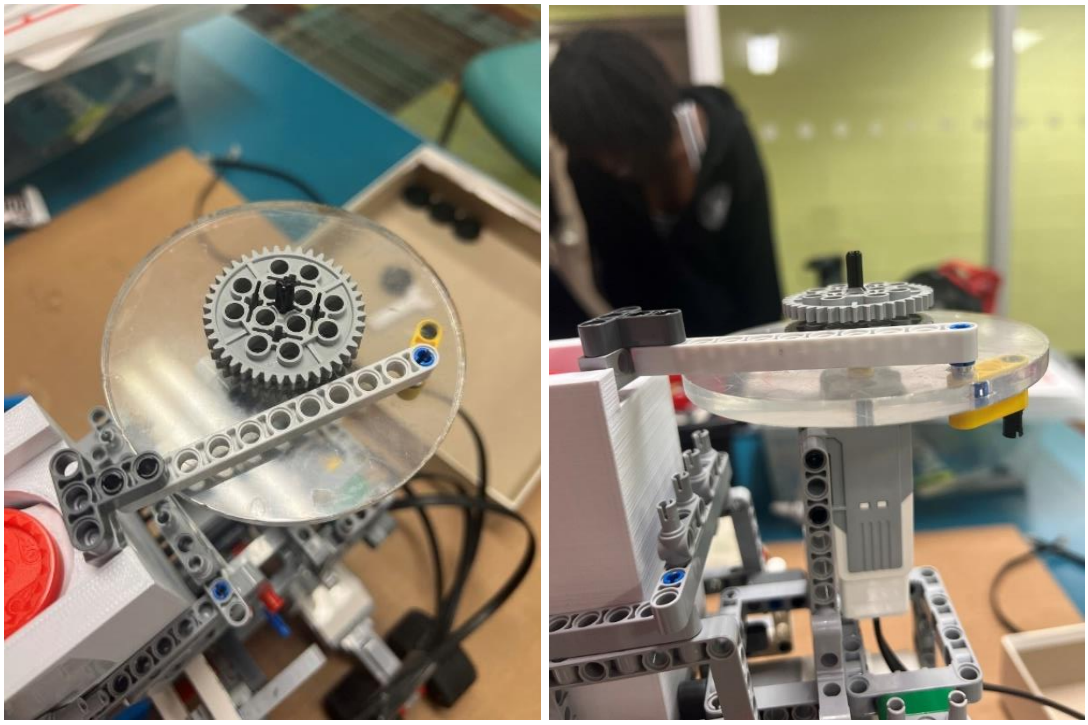

Figure 9: Final crank and piston iteration.

3.2 Sensor Involved in Movement System

To ensure proper gameplay, a touch sensor to bring the robot home and a colour sensor to detect correctly coloured game pieces are being played is important.

3.2.1 Home Base

To ensure that the robot's movements are precise, after each move, it must head back to home base. As seen in Figure 10, a touch sensor is placed at the end of the game board to signal where the robot's home is located.

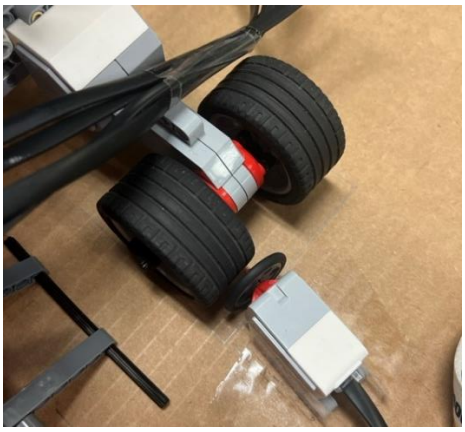

Figure 10: Touch sensor at home base.

3.2.2 Colour Detection

One of the main functions of the robot is to recognize piece colour. To determine this, a colour sensor is used, as seen in Figure 11. It is also used to detect whether the dropper is empty, which explains why the top of the rack is coloured white in Figure 7.

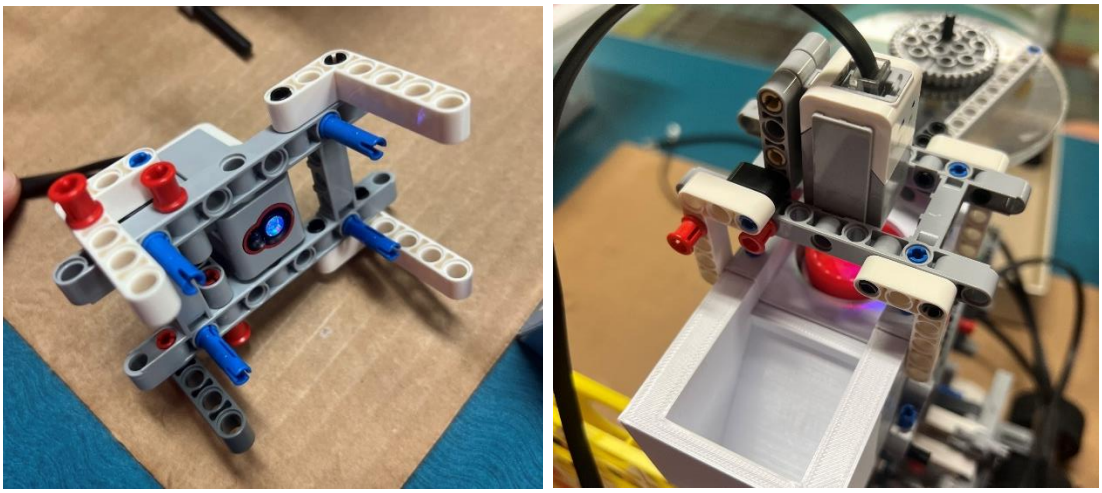

Figure 11: Colour sensor and placement on robot.

3.3 Assembly and Testing

Each system was assembled and tested individually before being put all together, to ensure proper operation and accuracy. Once the simulation on CAD worked for the piece handling system, Figure 12, it was then assembled and tested in real-time, Figure 13.

Adjustments were made based on feedback from sensors and visual cues to ensure precision in game piece placement and movement is achieved. The final mechanical assembly of the robot can be seen in Figure 14.

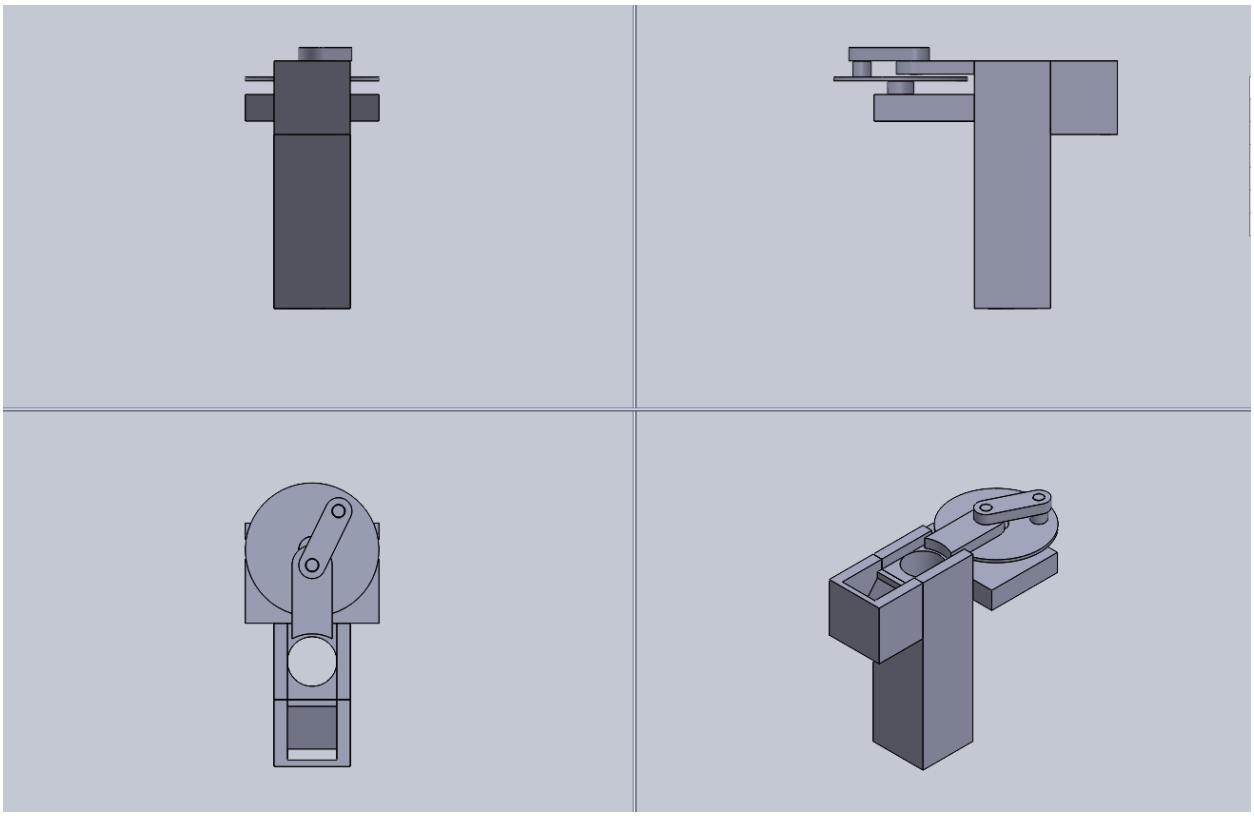

Figure 12: Full CAD of piece handling system.

Figure 13: Functioning design of piece handling system.

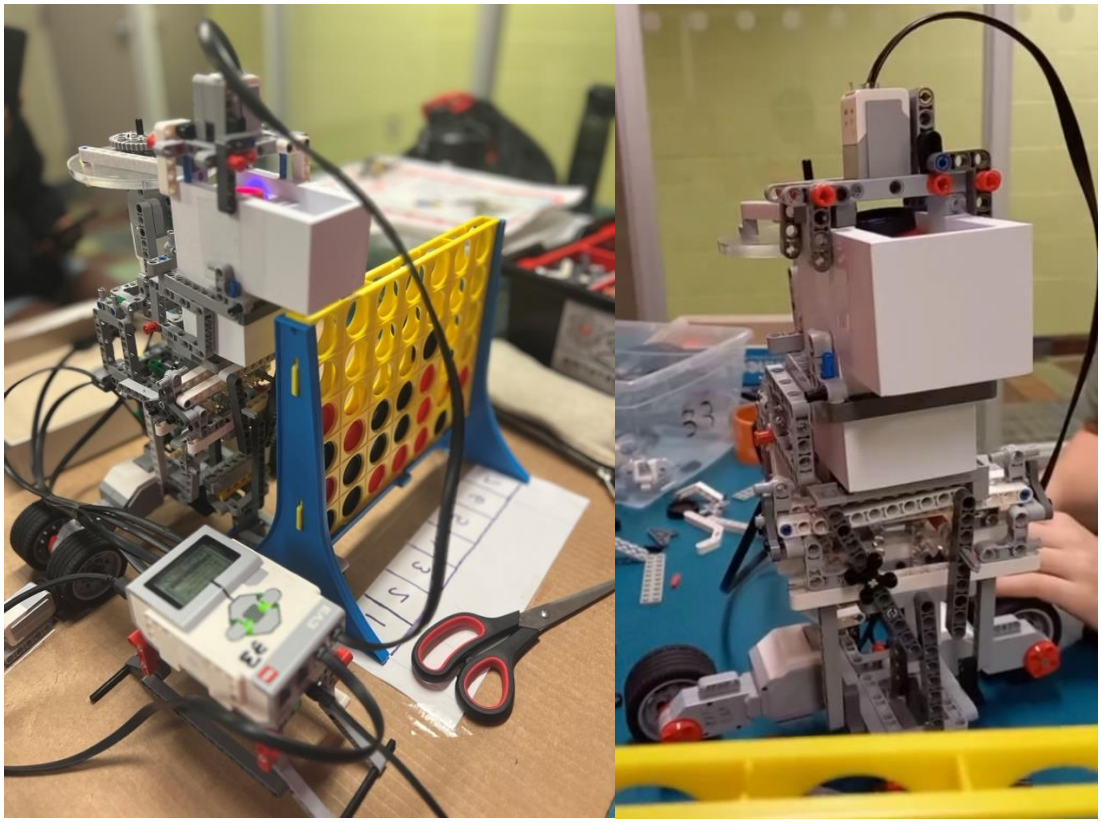

Figure 14: Full mechanical assembly of robot.

4.0 Verification of Design

In this section, each line of the engineering design specification was analysed to ensure the completed prototype conformed to all requirements.

4.1 Line-by-Line Verification of Engineering Design Specification

Line 1 of the engineering specification states that the design must successfully place pieces in the gameboard more than 90% of the time. To verify this requirement, several Connect 4 games were played, and the number of successful moves was compared to the number of total moves as shown in the table below. As shown in the table, 39 out of 42-piece placements were successful. This is approximately 93% of piece placements, which meets the requirement. It is worth noting that small amounts of adjustments were required when getting pieces into the board, such as lightly tapping the robot so that the piece correctly aligns with the slot. This was counted as a successful piece placement as the amount of human involvement was minimal. This issue is further addressed in the recommendations section of the conclusion.

Table 3: Number of attempted piece placements compared to number of successful piece placements.

Attempted Piece Placements	Successful Piece Placements
42	39

Line 2 of the engineering specification states that the device must successfully recognize winning combinations 100% of the time. To verify this requirement, more than 10 games were played throughout the testing process and during the design symposium. The device was successfully able to identify winning combinations 100% of the time; however, there were a few cases of human error where the human incorrectly entered their move, causing the robot to miss a winning combination due to the error.

Line 3 of the engineering specification states that the device must successfully recognize incorrect pieces—black pieces instead of red—at least 80% of the time. During testing, more than 15 incorrect pieces were placed into the robot, and incorrect pieces were identified 100% of the time.

Line 4 of the engineering specification states that a design must take less than 25 seconds on average to make its move. The code of the robot includes a timer to display how long it took for the robot to get from the home position to the correct piece placement position and place a piece

in the board. To be more conservative, a stopwatch was also used to time how long the robot takes to get from the start position to the correct position to place a piece and successfully place a piece in the board. Times gathered by the robot were between 1 and 3 seconds, while data from stopwatch timing is shown below in the table. The longest time was still under 6 seconds, so this requirement was successfully completed.

Table 4: Time for the robot to go from the home position to having successfully placed a piece.

Time for Robot to Move from Home and Place a Piece in Gameboard (s)
3.44
5.48
4.20
4.90
4.10
3.94
3.43
7.31
6.04
3.65
3.71
4.84
4.14
3.05
4.98
4.47
5.34
4.81
5.32
4.95
3.04

Line 5 of the engineering specification states that the device must be easy to use, with an average rating of at least 8 out of 10 based on a survey. Various classmates were asked to rate the device on a scale of 1 to 10, with 1 being hard to use, and 10 being easy to use. Data from the survey is shown below. The resulting average rating is approximately 8.15 out of 10. The biggest feedback received from classmates relating to ease of use was that the text on the display was quite small,

so it was difficult to read the various instructions. This issue is discussed further in the recommendations section of the conclusion.

Table 5: Ease of use rating of the device gathered from ME101 classmates.

Ease of Use
9
7
6
10
9
7
9
6
9
10
9
8
7

Line 6 of the engineering specification states that the total project cost must be under $100. All expenses from the project were tracked. The expenses were $20.45 worth of 3D prints and $3 of cardboard. The total cost, $23.45, is well under the constraint of $100.

4.2 Overall Verification of Design

In conclusion, all specified requirements outlined in the engineering design specification were successfully met by the design. Therefore, the design works successfully as designed. All further improvements and recommendations are discussed in the recommendations section of the conclusion.

5.0 Software Design and Implementation

This section provides an overview of the software design and implementation strategies used for the Connect 4 robot. The software design for our game involves multiple interconnected modules that handle various tasks such as user input, game logic, and robot movements. The overall operation can be visualized through flowcharts illustrating the actions the robot executes. The full source code implementing these functions is provided in Appendix B.

5.1 Function List

Table 6: Complete list of functions created in RobotC.

Function	Parameters	Return	Person
moveRobot	int robotMove	potentialColumn	Charis
buttonPress	n/a	n/a	Rein
goPlatform, goPiston, goRack	int motorPower	n/a	Charis
movePlatform, moveRack, movePiston,	int direction	n/a	Charis
firstMove	n/a	move	Kayden
generateRobotMove	n/a	n/a	Kayden
getHumanCol	n/a	col-1	Kayden
moveToHome	n/a	n/a	Charis
returnRack	n/a	n/a	Rein
correctPiece	n/a	n/a	Rein
updateMove	int column, int player	n/a	Grace
winningCheck	n/a	1(robot), 0(player), or 3(neither)	Grace

5.2 Function Breakdown

This section delves deeper into each specific function, explaining its purpose and how it contributes to the overall game.

5.2.1 Calibration

The calibration of the software includes sensor configuration, initialization of the game board, and the needed constants. Additionally, the functions: goPlatform, goPiston, and goRack are all used to input motor power to their respective parts.

5.2.2 Correct Colour Piece

This verifies that the game pieces are of the correct colour before they are placed on the board, preventing errors during gameplay.

5.2.3 Get Column

This captures the column number selected by the user. It receives the value by the player pressing up or down to get the correct column value they place the piece in until they press enter.

5.2.4 Generate Robot Move

Using the random function, it will return a column for the robot to place a piece in as long as the column is not already full.

5.2.5 Move platform, rack, piston, piece

These 4 functions manage the physical actions required to move and place the game pieces. Each will move their respective area of the robot.

5.2.6 Winning Check

Examining the game board, this function will return whether the player, robot, or neither has won. It first analyzes the horizontal combinations, then the vertical, and finally the diagonal, including forward slashes and back slashes.

5.2.7 First Move

This takes the input of the user whether they would like to play first, or the robot. 1 is used for the robot to go first and -1 for the player.

5.2.8 Move Home

This commands the robot to return to its starting position (home) and will stop once it hits the touch sensor. This runs after the robot has successfully completed placing a game piece.

5.2.9 Return Rack

It will reset the rack to its initial position which is it completely lowered so that more game pieces can be placed into it.

5.2.10 Move Robot

This function moves the robot for its specific turn to place its game piece. Due to the inaccuracies of the motor encoders, if a column exceeds 4, the plate will go an extra bit to compensate for that. After that, it checks for the correct game piece, moves the rack into position, and activates the piston to push it out. Finally, the time is displayed for the design specification requirements to calculate the move time and speed.

5.2.11 Main

The main function utilizes all of these previous functions together, acting as a central hub for the written code, as seen in Figure 15. Initially, it configures the sensors, and after the game begins with the firstMove() function, it determines which player to play first.

The core of the main program is a loop that continues to run until the game concludes, making sure it alternates between the user and the robot playing. Inside of this, it calls getHumanCol() to capture the player's move and generateRobotMove() to get the robot's move.

Each move involves more in-depth steps. For example, updateMove() is called to update the game board. While also checking correctPiece() to ensure the right colour game piece is being placed. The robot's movements are controlled through movePlatform(), moveRack(), movePiston(), and moveRobot(), which handle the physical actions of the robot.

After each move, the winningCheck() function is invoked to determine if the latest move has created a win for either player. If that condition is met, the loop will stop; if not, the game will continue, and the function backToHome() will be called for the robot to go back to its origin.

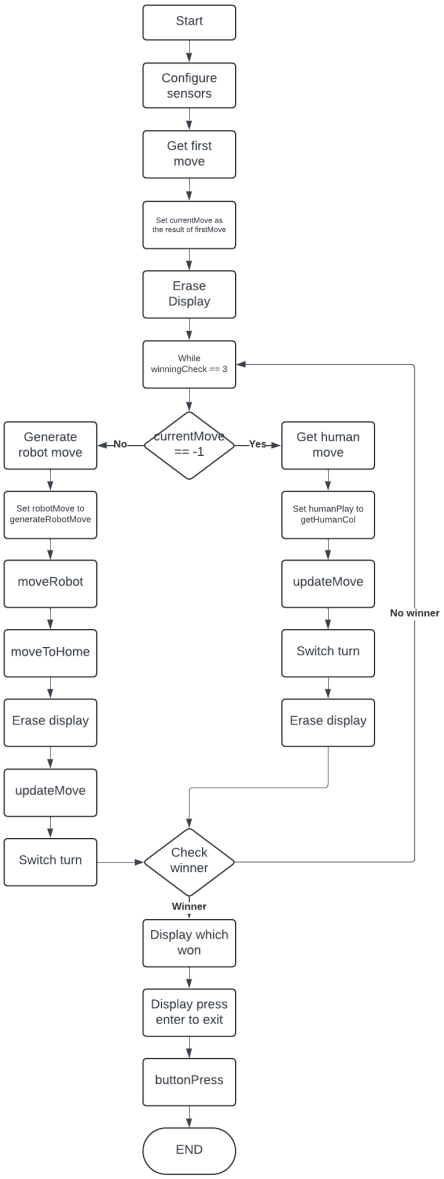

Figure 15: Flowchart displaying main program.

5.3 Storing Data

A 6 by 7 array was used for the game board to model the real-life Connect 4 game. In addition to this, the below function was used to update and store the moves played by the robot and user. It takes in the column and the user. It iterates through the rows of the specified column, from bottom

to top, until it finds an empty spot and stores that index. Depending on whether the player is putting it in or the robot, it will store 1 or 0, respectively.

5.4 Software Design Decisions

The first decision was to create a modular design. Since the functions were all distinctly divided, they could be tested separately to ensure proper output. This also allowed for easier development and debugging and allowed for modifications to be made later without issue.

Since using outside libraries to create a robot that could accurately respond to user moves was not something that was necessarily doable with robots and user knowledge, A random function was used to place the pieces in a random column. The decision makes the game playable even though it lacks the complexity of an actual robot like one would play on a computer.

The 6x7 array was used to store the moves because it could properly model the real-life Connect 4 gameboard. This made storing the moves easy, as it is a direct digital model of what is happening on the physical board in real-time. With this, the user input of the played column was found to be a good idea for knowing where the piece was played. Without multiple sensors in the board slots at all times, this was the direction that was used instead. Though it requires a bit of human input, it makes for a more intuitive project design.

5.5 Testing

In this section, the strategies used to test each function are described. After each function is tested, main was also tested in the same fashion by playing Connect 4 games and making adjustments as necessary.

5.5.1 Correct Colour Piece

This function was tested by placing various pieces underneath the colour sensor at varying distances to test whether incorrect pieces were accurately detected. This testing was later refined to include having the colour sensor overtop of the piece storage, to ensure the pieces and the top of the rack would be accurately detected during gameplay.

5.5.2 Get Column

This function was tested by isolating the robot and attempting to input various columns using the robot buttons to ensure that only columns between 1 and 7 could be entered.

5.5.3 Generate Robot Move

This function was tested by generating various robot moves and displaying them on the robot screen to ensure that only numbers between 0 and 6 could be generated. Then, an array was hard coded to have various combinations of full columns to ensure that the outputted move is prevented from including columns that are full of pieces.

5.5.4 Move platform, rack, piston, piece

These functions were tested by hooking up individual motors to the robot, and then ensuring the mechanisms moved smoothly. Then, pieces were placed into the robot, and the functions were tested again to ensure each function moved the robot/piece the correct distance.

5.5.5 Winning Check

This function was tested by hardcoding various winning combinations to ensure the function accurately outputted that the game was won, and who it was won by. This included winning combinations on columns, rows, forward diagonals, and backward diagonals. This was done in C++, and then translated to RobotC.

5.5.6 First Move

This function was tested the same way as get column, by isolating the robot and ensuring that the human would be able to input either up or down to either go first or let the robot go first.

5.5.7 Move Home

This function was tested by placing the robot at varying distances away from the touch sensor to ensure it drove all the way back and stopped as soon as it hit the sensor.

5.5.8 Return Rack

This function was tested by verifying that the time allowed for the rack to lower was enough for it to move all the way down without straining the motors excessively. This was then tested in combination with the correct colour piece function to verify that the rack would lower if an empty rack were detected.

5.5.9 Move Robot

This function was tested by placing pieces in the robot and then generating random moves repeatedly to ensure the robot could move pieces to any column consistently.

5.5.10 Main

The main program was tested by attempting to play various Connect 4 games with either the human or the robot going first, incorrect pieces placed in the rack, and achieving different winning combinations to ensure all functions and gameplay worked as intended.

5.6 Problems

The testing done for the functions of the robot revealed some issues with the design. One problem was the rack width in the rack and pinion not being a perfect fit with the Lego. When trying to build and test the Lego, the rack was found to be in between two holes in the Lego. So, the pinions would not be tight to the rack to drive it correctly. The rack and pinion also needed to be restrained at some other parts lower than where the pinions were attached. From testing, it was found that the rack would move out of position when driven up and down. Another problem was the piece not falling into the board. Testing out the full dropping system, the piece would get stuck at the top of the board instead of falling into the board. The reason for this is the dropper not being perfectly over the board and instead being a little forward or behind the top. The distance was caused by the robot's motors having a small amount of play, which caused the dropper to move away from the board. The last problem encountered is the crank and piston not being ridged. When the crank and piston were assembled, the Lego fasteners were loose, so the connections were not strong enough to consistently push out the pieces.

6.0 Project Management

In this section, the work breakdown and scheduling will be discussed as creating and coding the robot required many different parts to be allocated.

6.1 Breakdown Structure

To complete the project, it was important to know how many subsections needed to be done. A work breakdown structure was completed, as seen in Figure 16. Tasks were split based on group member’s strengths.

Table 7: Designated tasks for project according to project breakdown.

Group Member	Tasks
Charis	- Designed game pieces and board on CAD. - Laser cut crank and piston.
Grace	- Sketching and drawing of concepts.
Kayden	- Designed piece handling system. - Laser cut rack and pinion.
Rein	- CAD end of game arm → was no longer used. - 3D printed CAD parts.

Note: All members took on a part of coding, which can be found within Section 5: Functions List.

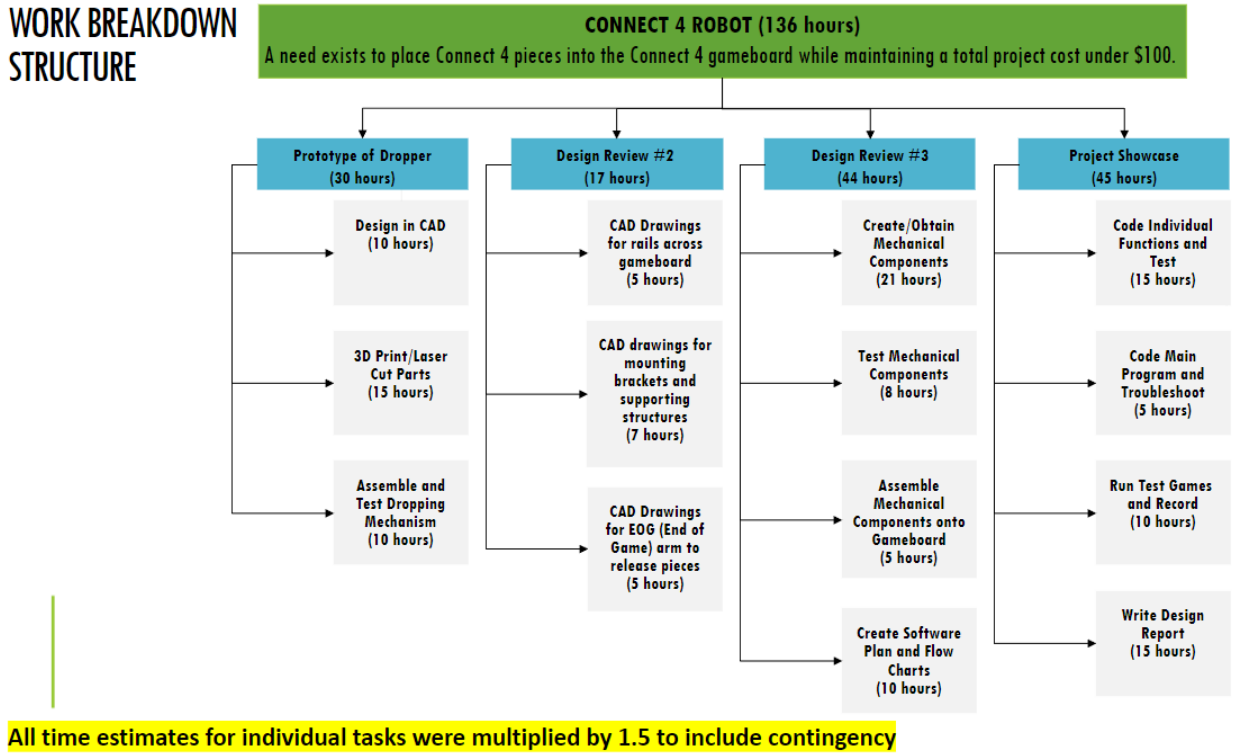

Figure 16: Connect 4 robot work breakdown structure.

6.2 Project Schedule

The original project planning schedule can be soon in Figure 17. Due to the postponement of the project start date caused by midterms, all tasks were done in shorter amount of time, as seen in Figure 18.

PROJECT: CONNECT 4 ROBOT

Project start date: 6/22/2024

Milestone description	Duration (hours)	Start	Finish
Prototype Dropper	30	2024-06-22	2024-07-02
Design in CAD	10	2024-06-22	2024-06-25
3D Print/Laser cut components	15	2024-06-24	2024-06-29
Assemble and test	10	2024-06-28	2024-06-30
Design Review #2	17	2024-06-23	2024-07-04
CAD drawings for rails across gameboard	5	2024-06-23	2024-06-30
CAD drawings for mounting brackets and supporting structures	7	2024-06-25	2024-07-01
CAD Drawings for EOG arm	5	2024-06-27	2024-07-02
Design Review #3	44	2024-07-04	2024-07-16
Create/Obtain Mechanical Components	21	2024-07-04	2024-07-10
Test Mechanical Components	8	2024-07-10	2024-07-13
Assemble Mechanical Components onto Gameboard	5	2024-07-13	2024-07-14
Create Software Plan and Flow Charts	10	2024-07-13	2024-07-16
Project Showcase	30	2024-07-05	2024-07-25
Code Individual Functions and Test	15	2024-07-05	2024-07-09
Code Main Program and Troubleshoot	5	2024-07-09	2024-07-16
Run Test Games and Record	10	2024-07-16	2024-07-20
Write Design Report	15	2024-07-20	2024-07-25

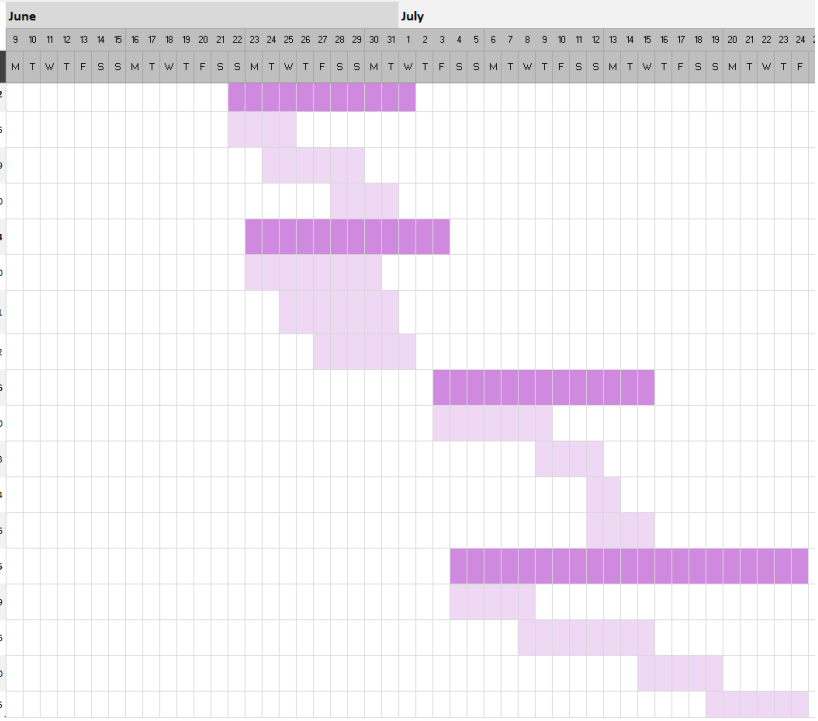

Figure 17: Original project planning schedule.

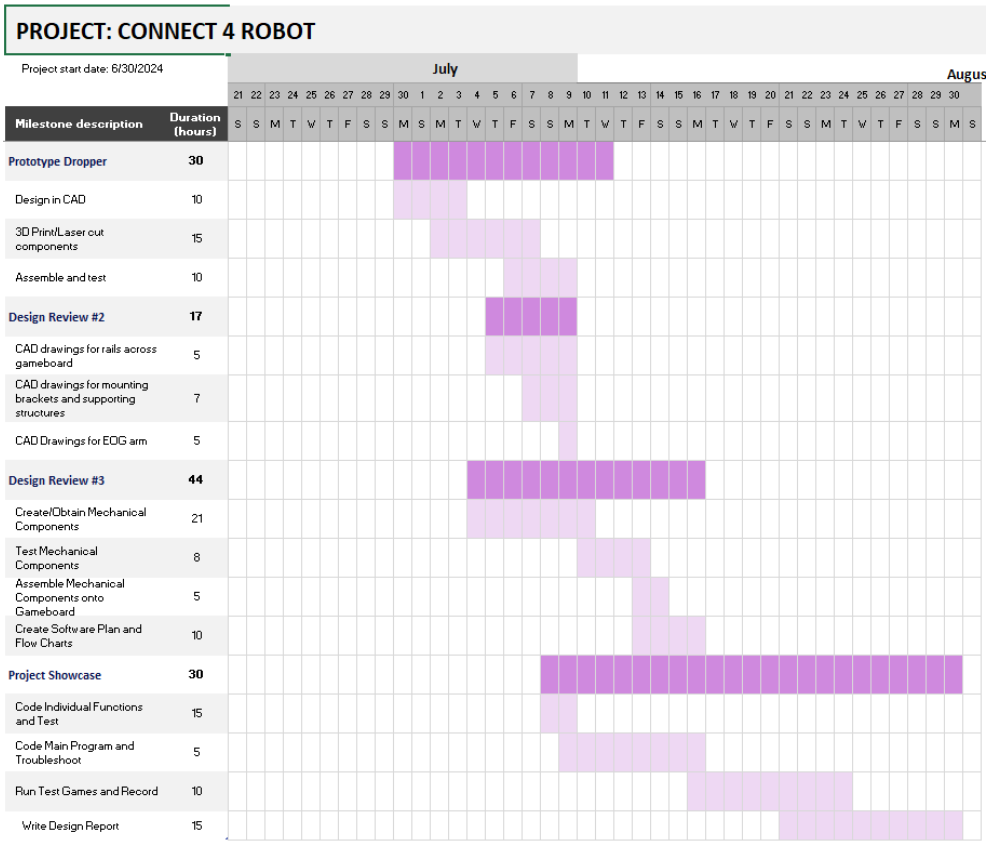

Figure 18: Final Project Schedule.

6.3 Project Plan vs. Reality

The main difference between the project plan and the actual schedule was the timeline. The project experienced delays that pushed back the overall schedule. Despite these setbacks, all tasks were ultimately completed successfully. The initial project plan laid out a clear timeline for each phase of development, including design, implementation, testing, and final adjustments. However, with classes and other challenges, what happened in reality happened later than what was put down on paper.

7.0 Conclusions

The goal of the robot project is to create a robot using the design principles and programming skills from what was learned in ME 101. The problem solved by the robot is to play Connect 4 against a person. The robot needs to place pieces and recognize when a player gets 4 pieces in a row. Three different designs were considered. Ultimately, the design that became the end product has a piece

dropper that gets reloaded by the human and lets the human input their move. This design is the fastest and simplest, although it is the least autonomous. To drop the pieces, the robot raises its pieces out of the piece storage using a rack and pinion to get pushed out. The pieces are pushed out using a crank and piston system. The pieces, after being pushed, fall into a funnel to be dropped into the Connect 4 board. After each move, the robot returns to its home marked with a touch sensor. A colour sensor is mounted over top of the piece storage to ensure the robot is placing the correct piece and that there are pieces left in the storage. Each mechanism was tested separately and tested while put together. From an analysis of the engineering design specification, it was found that each requirement was met. The cost of the robot was below $100 at $23.45. The robot was rated as easy to use, it made its move in a short amount of time and could successfully place pieces. The robot could also detect wins with 100% accuracy and identify incorrect pieces in the storage 100% of the time. The code consisted of 16 functions for all the robots' processes and movements. The code configures the sensors, turns on the motors, checks the colour of the piece is correct, and allows players to input their moves and store moves in an array. The code also randomly generates the robot's move and moves the robot to the appropriate space to release a piece into the board. The robot's code will check if a player has won the game. A prompt is displayed if the player wants to move first or the robot. The code will return the robot back to its home, and return the rack to the lowest spot. An array was used to store the information about the game moves to know what moves are possible on the physical board and know if a win occurs. The robot was limited to playing moves randomly because inputting a library was not possible for the robot. This means the robot is unable to play intelligently. Each function was tested individually before using each function together in the main. After testing the robot’s functions, some problems with the design were discovered. The rack and pinion did not have consistent movement, the dropper was a little off relative to the board after driving the robot, and the crank and piston were not held strongly together.

7.1 Recommendations

Although the current robot works to a reasonable degree, a couple of recommendations made to the mechanical design will allow for smoother gameplay. To achieve smoother piece dropping by allowing the piece to fall straight into the board without human interference, a track could be used instead of a base with wheels. A track would be able to relate to the board itself to ensure no

movement or variability in the dropper placement. Using the current design the dropper cart needs to be set up by the board. A track could always stay at the exact position relative to the board and guarantee a perfect drop. The player experience could be improved by allowing the player to edit a past move in case they accidentally input an incorrect move. One last recommendation is to make the text size larger for messages output to the display. This change would make the messages easier to read.

Appendix A: Flowcharts

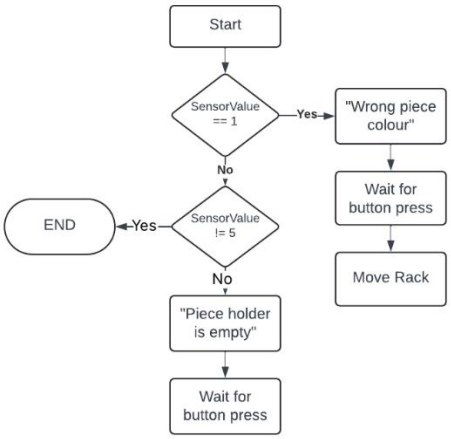

Figure 19: Flowchart to check correct color piece.

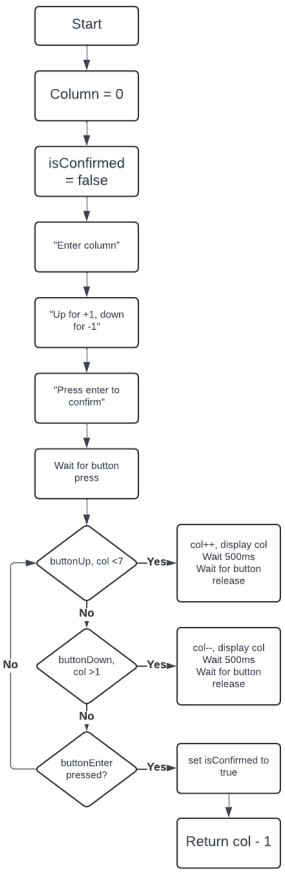

Figure 20: Process on how to get column from user.

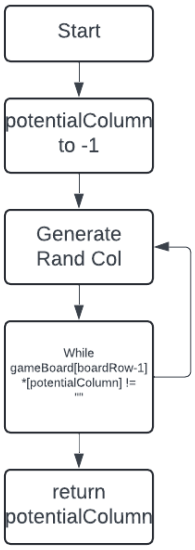

Figure 21: Flowchart describing how to generate robot's move.

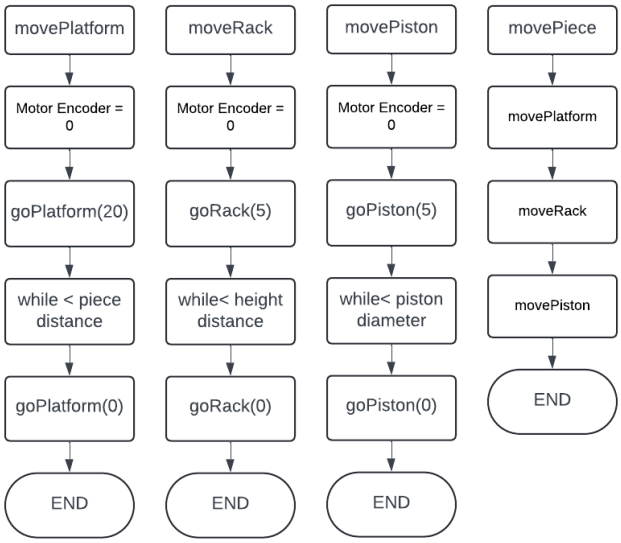

Figure 22: Processes to move the platform, rack, piston, and piece.

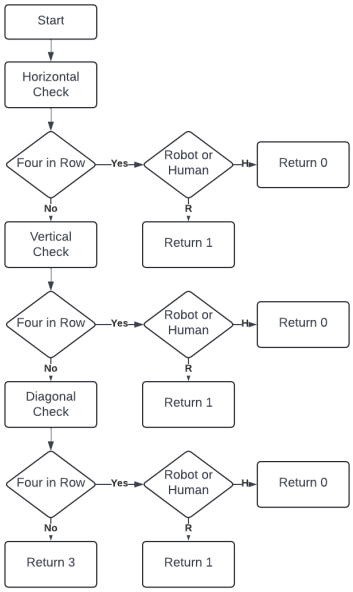

Figure 23: Flowchart to check for a winning combination.

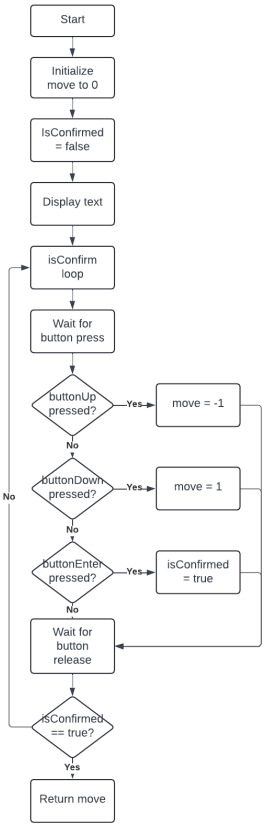

Figure 24: Process to decide if the user or robot goes first.

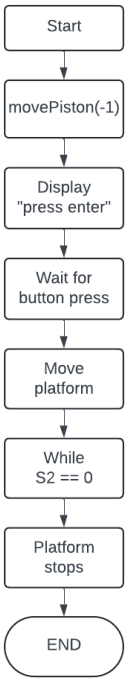

Figure 25: Process to move the robot back home.

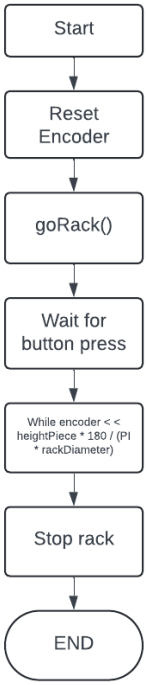

Figure 26: Flowchart showing rack return to starting position.

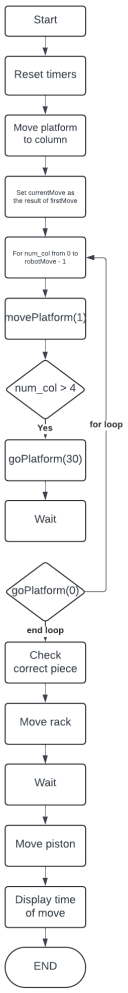

Figure 27: Move robot process.

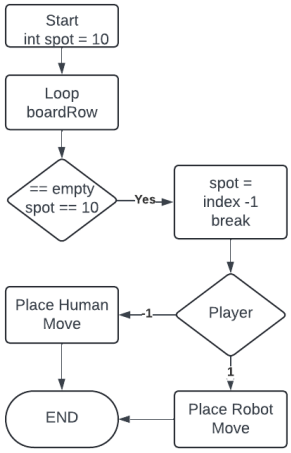

Figure 28: Process to store data.

Appendix B: Source Code

This appendix contains the complete source code used to run the Connect 4 robot's software.

```
//the gameboard

const int boardCol = 7;
const int boardRow = 6;
const float heightPiece = 0.9;
const float diameterPiece = 3.200;
const float wheelDiameter = 2.75;
const float rackDiameter = 1.5;
const float pistonDiameter = 60;
string gameBoard[boardRow][boardCol];

void configureAllSensors();
void buttonPress();
void goPlatform(int motPower);
void goRack(int motPower);
void goPiston(int motPower);
void movePlatform(int direction);
void moveRack(int direction);
void movePiston(int direction);
int generateRobotMove();
void moveRobot(int robotMove);
void moveToHome();
void returnRack();
void correctPiece();
int getHumanCol();
void updateMove(int column, int player);
int winningCheck();
int firstMove();


task main()
{
    configureAllSensors();
    int currentMove = firstMove();
    eraseDisplay();
    while (winningCheck() == 3)
    {
        if (currentMove == -1)
        {
            int humanPlay = getHumanCol();
            updateMove(humanPlay, currentMove);
            currentMove *= -1;
            eraseDisplay();
        }
        else
        {
            int robotMove = generateRobotMove();
            moveRobot(robotMove);
            moveToHome();
            eraseDisplay();
            updateMove(robotMove, currentMove);
            currentMove *= -1;
        }
    }
    if (winningCheck() == 0)
        displayString(5, "Human has won the game!");
    else
        displayString(5, "Robot has won the game!");
    displayString(7, "Press enter button to exit.");
    buttonPress();
}

```

```
62  void configureAllSensors()
63  {
64      SensorType[S2] = sensorEV3_Touch;
65      SensorType[S1] = sensorEV3_Color;
66      wait1Msec(50);
67      SensorMode[S1] = modeEV3Color_Color;
68      wait1Msec(50);
69  }
70
71  void buttonPress()
72  {
73      while(!getButtonPress(ENTER_BUTTON))
74      {}
75      while(getButtonPress(ANY_BUTTON))
76      {}
77  }
78
79  void goPlatform(int motPower)
80  {
81      motor[motorA] = motPower;
82      motor[motorD] = -motPower;
83  }
84
85  void goRack(int motPower)
86  {
87      motor[motorB] = motPower;
88  }
89
90  void goPiston(int motPower)
91  {
92      motor[motorC] = motPower;
93  }
94
```

```
95   void movePlatform(int direction)
96   {
97       nMotorEncoder[motorA] = 0;
98       goPlatform(20 * direction);
99       while(nMotorEncoder[motorA] < diameterPiece * 180 / (PI * wheelDiameter))
100      {}
101      goPlatform(0);
102  }
103
104  void moveRack(int direction)
105  {
106      nMotorEncoder[motorB] = 0;
107      goRack(5 * -direction);
108      while(abs(nMotorEncoder[motorB]) < heightPiece * 180 / (PI * rackDiameter))
109      {}
110      goRack(0);
111  }
112
113  void movePiston(int direction)
114  {
115      nMotorEncoder[motorC] = 0;
116      goPiston(10 * direction);
117      while(abs(nMotorEncoder[motorC]) < pistonDiameter)
118      {}
119      goPiston(0);
120  }
121
```

```
122  int generateRobotMove()
123  {
124      int potentialColumn = -1;
125      do {
126          potentialColumn = random(6);
127      } while (gameBoard[0][potentialColumn] != "");
128      displayString(1, "Robot Move: %d", potentialColumn + 1);
129      return potentialColumn;
130  }
131
132  void moveRobot(int robotMove)
133  {
134      time1[T1] = 0;
135      for (int num_col = 0 ; num_col < robotMove; num_col++)
136      {
137          movePlatform(1);
138          if (num_col > 4)
139          {
140              goPlatform(30);
141              wait1Msec(300);
142              goPlatform(0);
143          }
144      }
145      while (SensorValue[S1] != 5)
146          correctPiece();
147      moveRack(1);
148      wait1Msec(500);
149      movePiston(1);
150      displayString(10, "Time to complete move: %f", time1[T1]/1000.0);
151  }
152
153  void moveToHome()
154  {
155      movePiston(-1);
156      displayString(5, "Press Enter to confirm");
157      buttonPress();
158      goPlatform(-20);
159      while (SensorValue[S2] == 0)
160      {}
161      goPlatform(0);
162  }
163
164  void returnRack()
165  {
166      nMotorEncoder[motorB] = 0;
167      goRack(15);
168      wait1Msec(3000);
169      goRack(0);
170  }
171
```

```
172  void correctPiece()
173  {
174      if(SensorValue[S1] == 5)
175      {}
176      else if(SensorValue[S1] == 1)
177      {
178          displayString(5, "Wrong piece colour please");
179          displayString(6, "change, press ENTER BUTTON");
180          displayString(7, "   when issue is fixed");
181          buttonPress();
182          moveRack(1);
183      }
184      else
185      {
186          displayString(5, "Piece holder is empty please");
187          displayString(6, "refill, press ENTER BUTTON");
188          displayString(7, "    when issue is fixed");
189          returnRack();
190          buttonPress();
191          moveRack(1);
192      }
193  }
194
```

```
195  int getHumanCol()
196  {
197      int col = 0;
198      bool isConfirmed = false;
199      displayString(1, "Enter column");
200      displayString(2, "Up for +1, down for -1 column");
201      displayString(5, "Press enter to confirm");
202      do {
203          while (!getButtonPress(buttonUp) && !getButtonPress(buttonDown)
204                                           && !getButtonPress(buttonEnter))
205          {}
206          if (getButtonPress(buttonUp) && col < 7)
207              col++;
208          else if (getButtonPress(buttonDown) && col > 1)
209              col--;
210          else if (getButtonPress(buttonEnter))
211          {
212              isConfirmed = true; //cannot use else as program should do nothing if any other button is pressed
213              if (col == 0)
214                  col = 1;
215          }
216          while (getButtonPress(buttonUp) && getButtonPress(buttonDown)
217                                          && getButtonPress(buttonEnter))
218          {}
219          wait1Msec(500);
220          displayString(4, "Current column: %d", col);
221      } while (!isConfirmed);
222      return (col-1);
223  }
224
```

```
225   //will place the move that the player or robot played
226
227   void updateMove(int column, int player)//player is the -1 or 1
228   {
229       int spot = 10;
230       for (int i = boardRow; i > 0 ; i--)
231       {
232           if (gameBoard[i-1][column] == "" && spot == 10)
233           {
234               spot = i - 1;
235           }
236       }
237       if (player == 1)
238       {
239           gameBoard[spot][column] = "1"; //robot
240       }
241       else
242       {
243           gameBoard[spot][column] = "0"; //player
244       }
245   }
246
247   //check for win, returns 1 for robot win, 0 for player win, and return 3 for no win
248   int winningCheck()
249   {
250       //horizontal
251       for (int r = 0; r < boardRow; r++)
252       {
253           for (int c = 0; c < 4; c++)
254           {
255               if (gameBoard[r][c] == gameBoard[r][c + 1] && gameBoard[r][c] == gameBoard[r][c + 2] && gameBoard[r][c] == gameBoard[r][c + 3])
256               {
257                   if (gameBoard[r][c] == "1")
258                   {
259                       return 1;
260                   }
261                   if (gameBoard[r][c] == "0")
262                   {
263                       return 0;
264                   }
265               }
266           }
267       }
268
269       //vertical
270       for (int r = 0; r < 3; r++)
271       {
272           for (int c = 0; c < boardCol; c++)
273           {
274               if (gameBoard[r][c] == gameBoard[r + 1][c] && gameBoard[r][c] == gameBoard[r + 2][c] && gameBoard[r][c] == gameBoard[r + 3][c])
275               {
276                   if (gameBoard[r][c] == "1")
277                   {
278                       return 1;
279                   }
280                   if (gameBoard[r][c] == "0")
281                   {
282                       return 0;
283                   }
284               }
285           }
286       }
287
```

```
    //diagonal slash
    for (int r = 3; r < boardRow; r++)
    {
        for (int c = 0; c < 4; c++)
        {
            if (gameBoard[r][c] == gameBoard[r - 1][c + 1] && gameBoard[r][c] == gameBoard[r - 2][c + 2] && gameBoard[r][c] == gameBoard[r - 3][c + 3] )
            {
                if (gameBoard[r][c] == "1")
                {
                    return 1;
                }
                if (gameBoard[r][c] == "0")
                {
                    return 0;
                }
            }
        }
    }
    //diagonal backslash
    for (int r = 0; r < 3; r++)
    {
        for (int c = 0; c < 4; c++)
        {
            if (gameBoard[r][c] == gameBoard[r + 1][c + 1] && gameBoard[r][c] == gameBoard[r + 2][c + 2] && gameBoard[r][c] == gameBoard[r + 3][c + 3])
            {
                if (gameBoard[r][c] == "1")
                {
                    return 1;
                }
                if (gameBoard[r][c] == "0")
                {
                    return 0;
                }
            }
        }
    }
    return 3;
}

int firstMove()
{
    int move = 0;
    bool isConfirmed = false;
    displayString(1, "Who do you want to go first?");
    displayString(2, "Press up or down:");
    displayString(5, "Press enter to confirm");
    do {
        while (!getButtonPress(buttonUp) && !getButtonPress(buttonDown)
                    && !getButtonPress(buttonEnter))
        {}
        if (getButtonPress(buttonUp))
        {
            displayString(4, "Player wants to go first!");
            move = -1;
        }
        else if (getButtonPress(buttonDown))
        {
            displayString(4, "Robot wants to go first!");
            move = 1;
        }
        else if (getButtonPress(buttonEnter))
            isConfirmed = true; //cannot use else as program should do nothing if any other button is pressed
        while (getButtonPress(buttonUp) && getButtonPress(buttonDown)
                    && getButtonPress(buttonEnter))
        {}
        wait1Msec(500);
    } while (!isConfirmed);
    return move;
}
```