Robot Project

EV3EAL

Group #20 - Stream 8

Cheran #21013444

Savyo #21006924

Prabhjot #21005282

Sachin #20999003

MTE 100 AND MTE 121

Tuesday, December 6th, 2022

Summary

This report outlines the mechanical and software design process as the group created a card-dealing robot, nicknamed EV3eal. Furthermore, it thoroughly discusses each mechanical progress update and the software's functions. Finally, this report goes over one's recommendations if the group were to attempt this project again.

Acknowledgments

TA help

Online images for clamp system

Contents

Introduction....5
Scope....5
Constraints and Criteria....7
Requirements....7
Constraints....7
Valuable and Invaluable Constraints and Requirements....7
Mechanical Design and Implementation....8
Overall description of our mechanical design....8
Updates and Changes to the Design....8
Progress 1 (November 8th)....8
Progress 2 (November 10th – 12th)....10
Progress 3 (November 14th)....12
Progress 4 (November 16th):....14
Progress 5 (November 18th):....17
Progress 6 (November 21st):....19
Progress 7 (November 22nd):....20
Progress 8 (November 23rd):....21
Software Design and Implementation....22

Description of the software....22
Task list....22
Functions....23
Trade-offs....30
Choice of variable type....30
Testing....31
Problems....31
Verification....32
Final Constraints List....32
Project Plan....33
Conclusion....34
Recommendations....34
Mechanical Design:....34
Software **Recommendations:**....34
Back Matter....35
Appendix A....35

List of Figures

Figure 1: Concept sketches....9
Figure 2: Build from first day....9
Figure 3: Concept sketch for actual dealer....11

Figure 4: First dealer model....12
Figure 5: Card holding mechanism....13
Figure 6: Top view of initial build....13
Figure 7: Clamp system and ramp included....15
Figure 8: Touch Sensor included....16
Figure 9: Solidworks design for clearance....17
Figure 10: Clearance installed on robot....18
Figure 11: Reworked design....19
Figure 12: Final Clamp system....20
Figure 13: Final Dealer System....21
Figure 14: Final Build....22
Figure 15: Flowchart for Test Function....23
Figure 16: Flowchart Color sensor....24
Figure 17: Rotating and deal flow chart....25
Figure 18:....26
Figure 19:....27
Figure 20:....28
Figure 21:....28
Figure 25:....31
Figure 26:....32

Introduction

In any casual card game, the role of the dealer is usually something people would prefer to avoid. The aim for this project is to create a card dealing robot, which automates the dealing process while preventing cheating and human error. Throughout this report, the team will discuss their process for EV3eal, a robot card dealing system!

Scope

The robot's main functionality is to hold a deck of cards, receive data from the user about the number of players playing the game and the number of cards needed to be dealt. Then, start dealing the number of cards needed to the number of players with such speed and precision comparable to a human.

1. A general procedure is as follows:

2. Human puts the cards into the card holder, the clamp fastens onto the deck of cards
3. Code is run, the robot moves, rotating, forward, and backwards, before starting the actual dealing procedure. This is to inform the user the robot is preparing to deal and is completely functional.
4. User inputs the number of players.
5. Robot drives towards center until it reaches red dot
6. Robot begins to rotate, dealing cards to each player
7. Robot stops dealing and returns with the remainder of cards, dropping off any excess
8. Robot escapes the playing area to allow the players to begin playing!

The robot receives 5 inputs:

- Colour Sensor
- Gyro
- Touch sensor
- Motor encoders
- Push buttons (player interface

The colour sensor is used to bring the robot from anywhere on the table to the center where it will be able to deal with the cards. The robot will keep on driving until it stops detecting the color and then it will start dealing procedure.

The touch sensor is used as an emergency mechanism, if the robot is not performing its intended task or if a card is stuck, the user must push the touch sensor to stop the robot. This also serves as a buffer to account for the wide variety of unexpected events.

The gyro sensor is used to rotate the robot ‘x’ degrees to be able to deal with any player no matter where they are sitting. For example, given four players, the robot will rotate 90 degrees and deal to each player.

The motor encoder will be used to measure the distance of how far the robot travels to the center so that it will be able to drive to its original position, once it is done dealing with the cards needed to be dealt. Finally, the push buttons are used to receive user input regarding the number of players playing for the given and the number of cards being dealt in total.

- The robot will use 3 motors. Two motors are used for mobility. Since the robot will be moving and turning based on the user's inputs and other sensors, two motors are required so that the robot can functionally rotate and drive. The third motor is used for the dealing system. Based on the mechanical design the motor will turn and push the cards out one at a time.

- The robot will recognize when all the tasks are complete, when all the cards specified and inputted by the user are distributed evenly between all the players. If any error occurs during the process of dealing, the user can press the touch sensor to emergency shutdown the robot. If not, the regular shutdown process will occur once the robot reaches its original position.

Changes from previous designs:

The group made various changes since the original design, particularly, it was underestimated how much time the build process would take. This led us to lower than scope under the guidance of the TA’s.

- The first change to the scope was limiting it. The group had set the goals remarkably high at first, wanting to shuffle cards and then deal. After receiving feedback from Professor Consell and the informal report, the group followed the advice to limit this to strictly dealing.

- Secondly, the group elected to limit the scope by switching the idea from a forklift to a garbage dump-type system. After the formal presentation where the group discussed more thoroughly the idea of a forklift

- Lastly, an additional change to the scope from the earlier stages was removing the addition of the ultrasonic sensor. The original use for the ultrasonic sensor was to detect where the players were sitting so that the robot could distribute the cards evenly when it detected a player. However, with further testing and building, it was concluded that the ultrasonic sensor's use was redundant, as the inputs already included were able to complete the ultrasonic sensor's job without its addition. The colour sensor can bring the robot the center and then the gyro will turn the robot (360/number of player) degrees and the motor encoders will drive the robot to its original position, so it as concluded that addition of ultrasonic sensor was unnecessary so therefore it was removed from the final build.

Constraints and Criteria

Requirements

- Fairly fast → comparable to a human, the group does not want the robot to be a downgrade to the average human dealer
    - Although it is exceedingly difficult to match the robot to a human in terms of a very dexterous job such as dealing, the requirement is that the robot is not a significant downgrade from a human.
- Precision/Adaptability → Equal spaced cards
    - With an unreliable gyro it is difficult to get complete accuracy, but the group wants there to be distinct enough piles so that the user can identify who's pile is who's.

Constraints

- Lego pieces cannot handle thin paper well as there are varying decks, and the group wants to ensure only 1 card is handed each round. One of the changes the group made was switching decks to find the best quality deck that provides the most consistent results.
    - Insert video of us struggling with it dealing with cards poorly
- The ultrasonic was originally a constraint to the build due to lack of its precision however with further testing and progression in the build the ultrasonic was removed

Valuable and Invaluable Constraints and Requirements

- The constraints and criteria that helped us guide in the project design was precision: the focus of the design was to deal cards fast but mainly precisely because lack of precision defeats the purpose of the robot, as dealing cards at random does not solve any problems.
- The constraints not valuable in implementing the project design were the type of materials. This project was extremely limited with materials that were given to us

initially. The project was fully constructed with Legos and one 3D printed part. Lego parts are not able to handle thin paper very well. The parts also tend to not be able to resist much weight, so part breaking was another issue.

Mechanical Design and Implementation

Overall description of the mechanical design

The major mechanical goal was to compete with a regular human while having reasonable speed and consistency.

The robot had many major mechanical design changes throughout the design process, which led to the final design. Throughout the designing process group faced many difficult challenges in mechanical design, as Lego is not the best material to engineer complex projects, the parts did not often perform in the intended way that they were supposed to.

Although some parts were difficult to assemble and kept falling apart, making the mechanical process challenging. The group had difficulty keeping a straight dealing platform but through continuous testing and design changes the group was able to design a functional and aesthetically pleasing final design.

Updates and Changes to the Design

Progress 1 (November 8th)

Chassis Design:
The group started the Initial stages of the design with some concept sketches of how the robot will potentially look like. The sketches included potential motor and sensor placements on the robot (Figure 1). As the group finalized the ideas and decided on a sketch. The group began the construction stage of the robot robot.

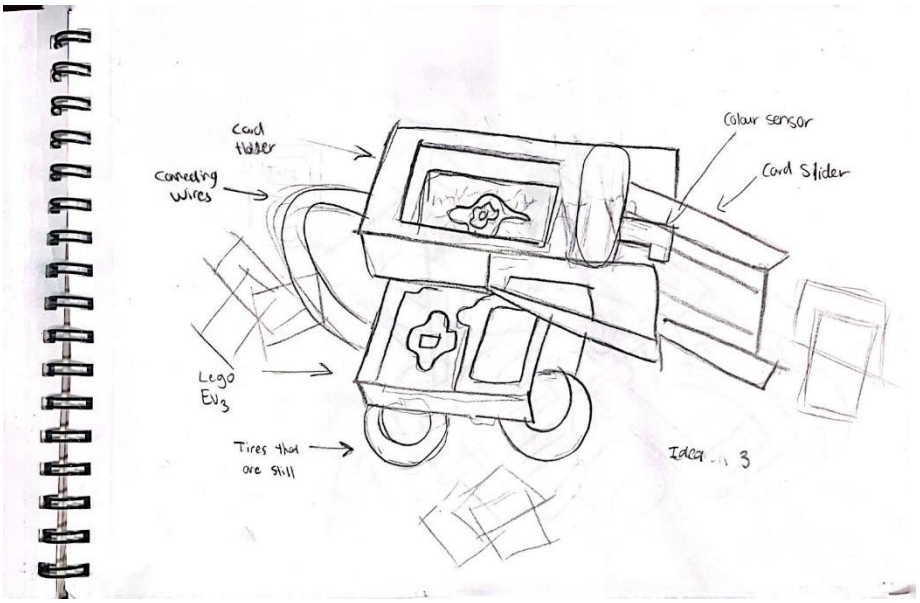


FIGURE 1: CONCEPT SKETCHES

The group began the chassis assembly through connecting a rod with two wheels to a motor. This was the start of the card dealing compartment. The motor would turn on causing the wheels to spin and through friction the wheels would pull on the cards to deal.

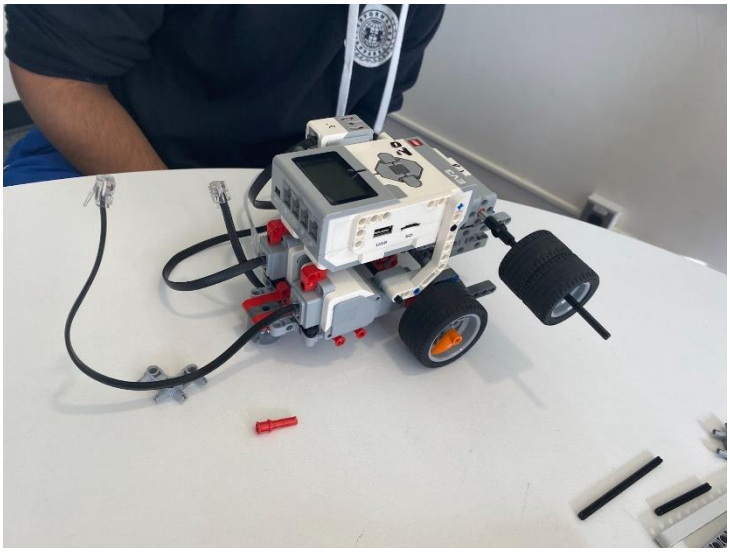


FIGURE 2: BUILD FROM FIRST DAY

Motor Drive Design
While looking at the different design ideas, the initial thoughts were to keep the robot stationary meaning that the robot would only require one motor for the dealing process.

Sensor attachment design
Since this was the beginning stages of the design a lot of the sensor positions on the robot were still undecided. One of the main design ideas, however, was to use the colour sensor to measure when a card was handed out or not. The use of gyro and touch was required for the build, but their placement was still undecided.

Overall assembly
To conclude the overall assembly for the first progress update, the group was able to construct a potential design for the card dealing process (Figure 2) by connecting a rod to a rotating motor which would cause it to spin. The group added a wheel to the spinning rod so that the friction will be able to pull individual cards and deal them.

Progress 2 (November 10th – 12th)

Chassis Design

Recorded some measurements and calculations to develop a surrounding basis fit perfectly to hold the deck of cards. As displayed in (figure 3) group developed the initial steps to a working dealing system for the robot.

Motor Drive Design

Before the initial meeting with the teaching assistants, the group was discussing the potential of adding two additional motors to the current design to have a potential clamp and forklift system for shuffling and dropping the cards after the robot was done dealing.

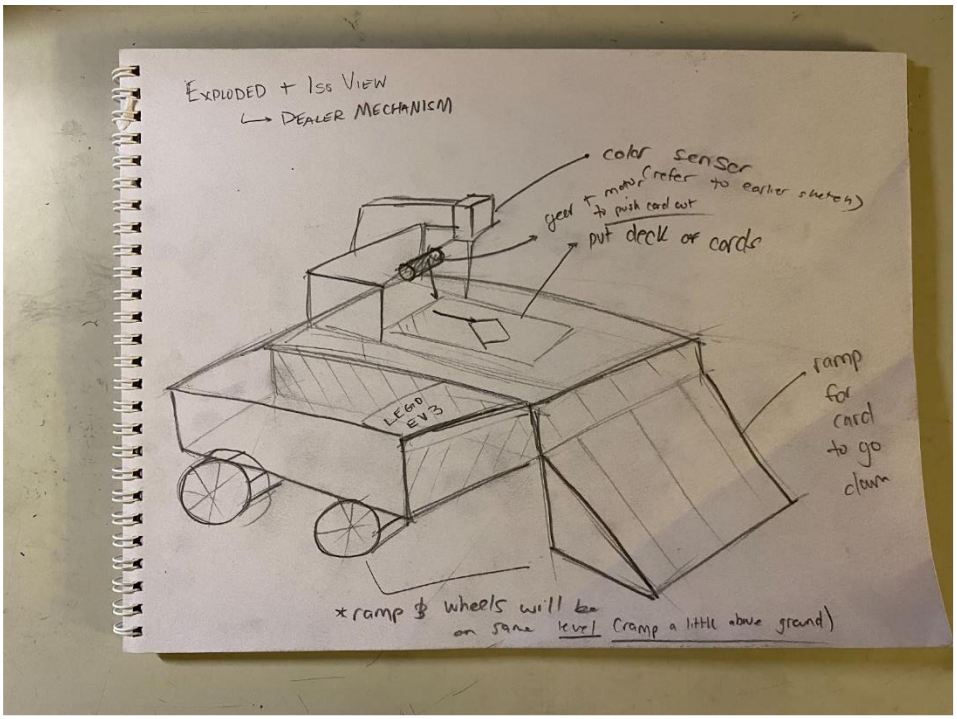


FIGURE 3: CONCEPT SKETCH FOR ACTUAL DEALER

Sensor Attachment Design

The main change in this progress update was changing the function of the colour sensor. Instead of using the colour sensor to check if a card was dealt with or not. The colour sensor was used as a guide for the robot to stop somewhere near the center of the table. The group realized that through software and mechanical components they would be able to control the dealing of individual cards

Overall Assembly

To conclude this progress update, the overall assembly of the dealing system was starting to take shape. The group developed a working rotating rod to deal cards and a surrounding basis to hold the deck. These two initial progress reports consisted mostly of brainstorming, sketching, calculating and design and less on the overall assembly. This was done because any good build requires these preliminary steps to have good design and functionality.

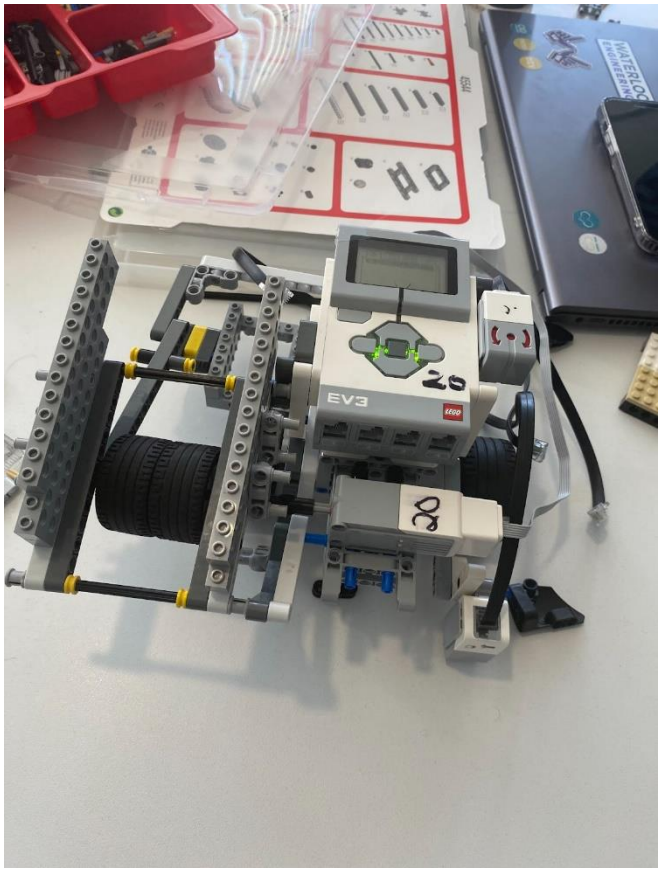


FIGURE 4: FIRST DEALER MODEL

Progress 3 (November 14th)

Chassis Design:
A major update to the chassis occurred when the group began to design the system to support the cards. Seeing it was needed to find Lego to not only tightly clamp onto the sides of the cards, the group also had the goal of making it adjustable for various card sizes. However, this secondary goal was not accomplished in the preliminary chassis design. When designing the chassis to hold the cards, the group needed long enough Lego pieces to fit the length of a card, and enough distance in between the center for two wheels to rotate and distribute the cards. Although this two-wheel design was later changed to just be a single wheel dealer as the group achieved more precision with this layout.

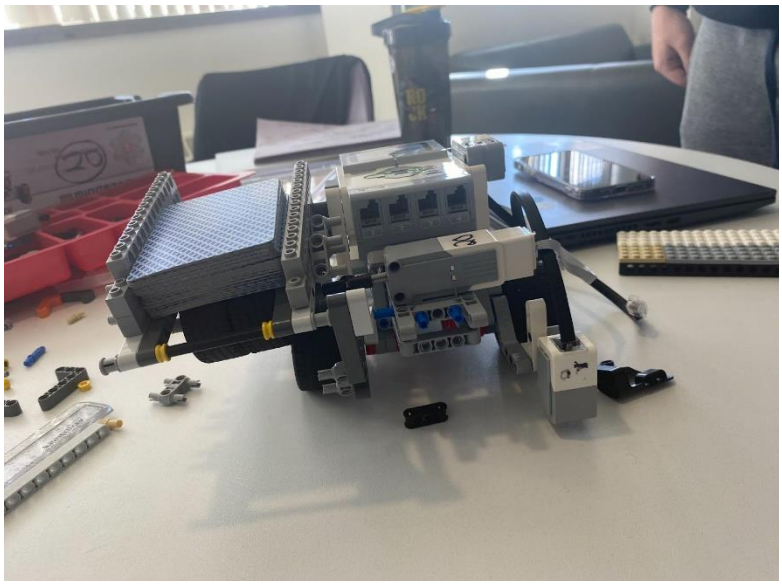


FIGURE 5: CARD HOLDING MECHANISM

Furthermore, the group had to build the chassis around the motor that powered the card dealer. This proved to be one of the most challenging parts of our mechanical build. This proved to be challenging as having a long shaft connected to the motor caused that shaft to be bent down drastically when connected to any sort of weight, this led to significant bending in our build that the group soon recognized would prevent the cards from even exiting the dealer.

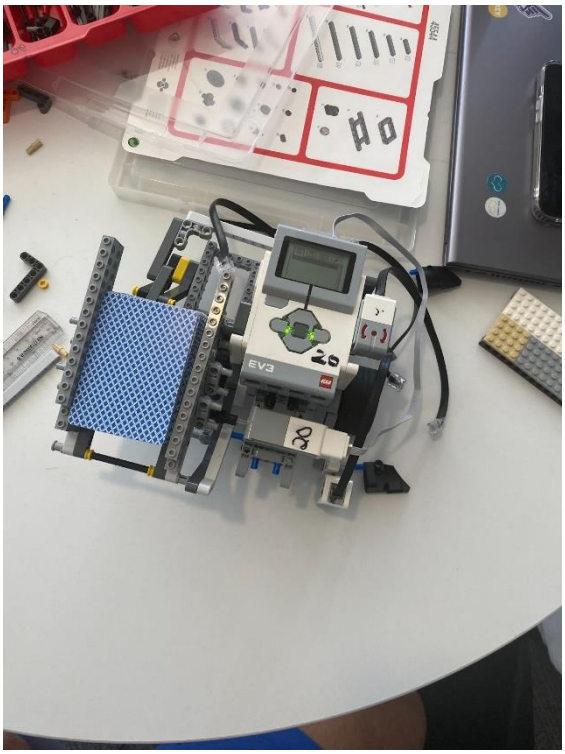


FIGURE 6: TOP VIEW OF INITIAL BUILD

Motor Drive Design:

Extending from the chassis, this was the first real test of whether the motor was able to shoot out cards. Since the group did not have any counterweight. The group expected the motor to rotate and push out a bulk of cards at one time. The group used the EV3_SystemTEST and pushed down an appropriate amount of pressure onto the cards to see if our wheel system was able to generate enough power to push the cards out. This preliminary test confirmed that our wheel sketches would be the basis for our later designs.

Overall assembly:
In summary our overall assembly saw its first major change in the incorporation of actual cards. This was significant as it allows us to measure a perfect fit for the cards, as well as measure enough clearance for the wheels to contact the base of the cards and push them out accordingly.

Progress 4 (November 16th):

Chassis Design:
After designing the fundamental chassis, the group had to begin thinking about how the cards would be distributed and how they would counteract the force of the wheel pushing the cards forward, in addition to preventing too many cards from being at once. This is where it was elected to include a counterweight and rubber band system. Referring to earlier sketches, the group also designed and implemented it to direct the cards to the respective player.

The idea behind the counterweight was to have consistent weight being applied to the cards, like that of how the group used the hands in the previous example so just enough pressure would be applied so only 1 card could be shot out at a time. group soon found out that using a wheel with a rubber band was not precise enough for one card to come out at a time and a complete dealer redesign was needed (see progress 7). Moreover, the rubber band connected to the base design (that was directly connected to the robot) allowed for a tension force that would tighten the wheel back onto the cards.

As previously mentioned, the ramp was designed to direct the cards to the player. The group quickly noticed, however, that this piece would be better off 3D-printed as it would allow for a cleaner path for the player, or ideally, no ramp at all, and the card is shot directly to the player.

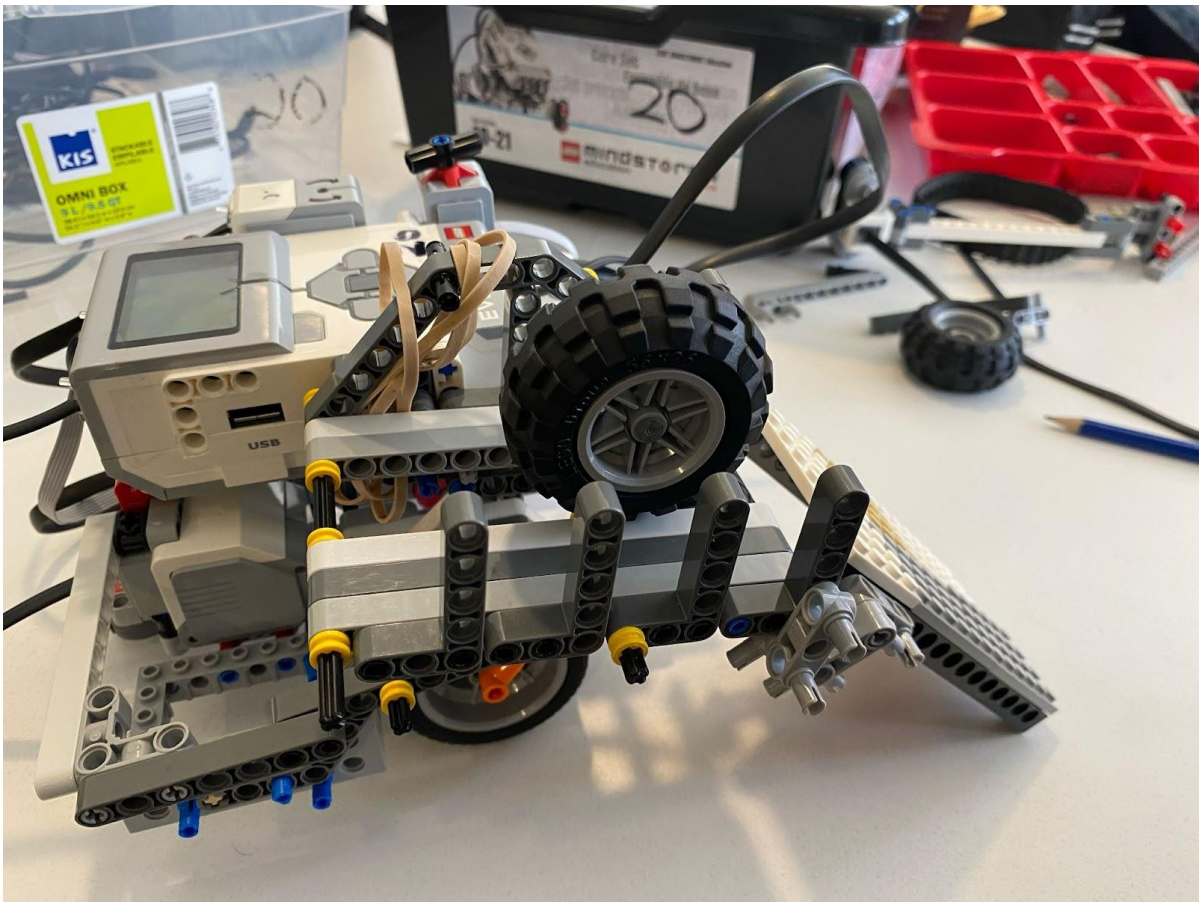


FIGURE 7: CLAMP SYSTEM AND RAMP INCLUDED

The main concerns of this updated build were stability and the clamp system, the group decided to continue with this design until the software was complete. It was recognized that this unstable design coupled with a clamp that was not tense enough to pressure each card would lead to problems down the line.

Sensor Attachment Design:

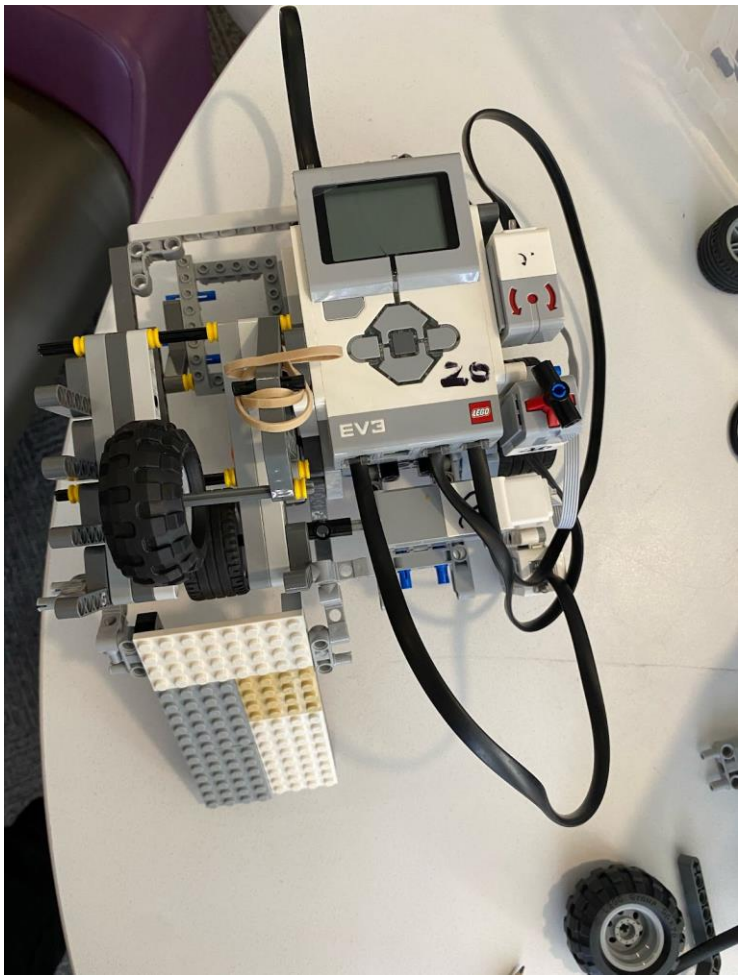


FIGURE 8: TOUCH SENSOR INCLUDED

As the group began to approach the software testing phase, the group wanted to incorporate a testing procedure. Specifically, the group included the touch sensor as an emergency exit button. This button allowed us to exit the code without any concern, this was used to prevent any damage to the cards, motor, and overall design, particularly in the initial stages of testing.

The touch sensor can be seen in front of the gyro sensor, on the right side of the robot.

Overall Assembly:

Two major changes can be seen in the overall assembly. Firstly, the group decided to switch to a one-wheel design for the motor connecting to the cards. This one-wheel design as seen below, allowed us to help with the previous problem of having too much weight on the shooter side of the robot. Moreover, one wheel still provided us with enough power to push out the cards. Finally, as mentioned in the previous progress update, the axle connected to the power sagged down significantly, in turn reducing the weight helped with this.

This provided a temporary solution to the sagging issue, however, the group soon noticed that the Lego pieces would ‘weaken’ over time and thus require us to incorporate more of a counterweight system and be a much stronger basis to hold our shooter. The group considered placing our shooting device on top of the Lego robot, however, this would prevent us from incorporating our user interface to put in the number of players and cards, so the group decided to make work with the currently unstable side dealer.

Secondly, it was realized that a clearance design was required to be able to fit a single card at a time. After running the System EV3 test, although the motor power was not accurate to what the group would later use (a lot higher) the group was able to see that with the current clamp system, it was pushing the whole deck instead of a single card at once, thus the group decided to complete our WATiMake training and 3D print a piece that provides just enough clearance to fit 1 card under it.

Progress 5 (November 18th):

Chassis Design:

After creating the initial design, the group began to realize that it would rely heavily on having an appropriate clearance system. However, the group did not have the appropriate Lego to design the piece so that only a card would fit through! Thus, the group decided it was time to get 3D modelling. After using calipers to dimension the piece, the group went to Solid works, referencing online dimensions for the hard to dimension hole.

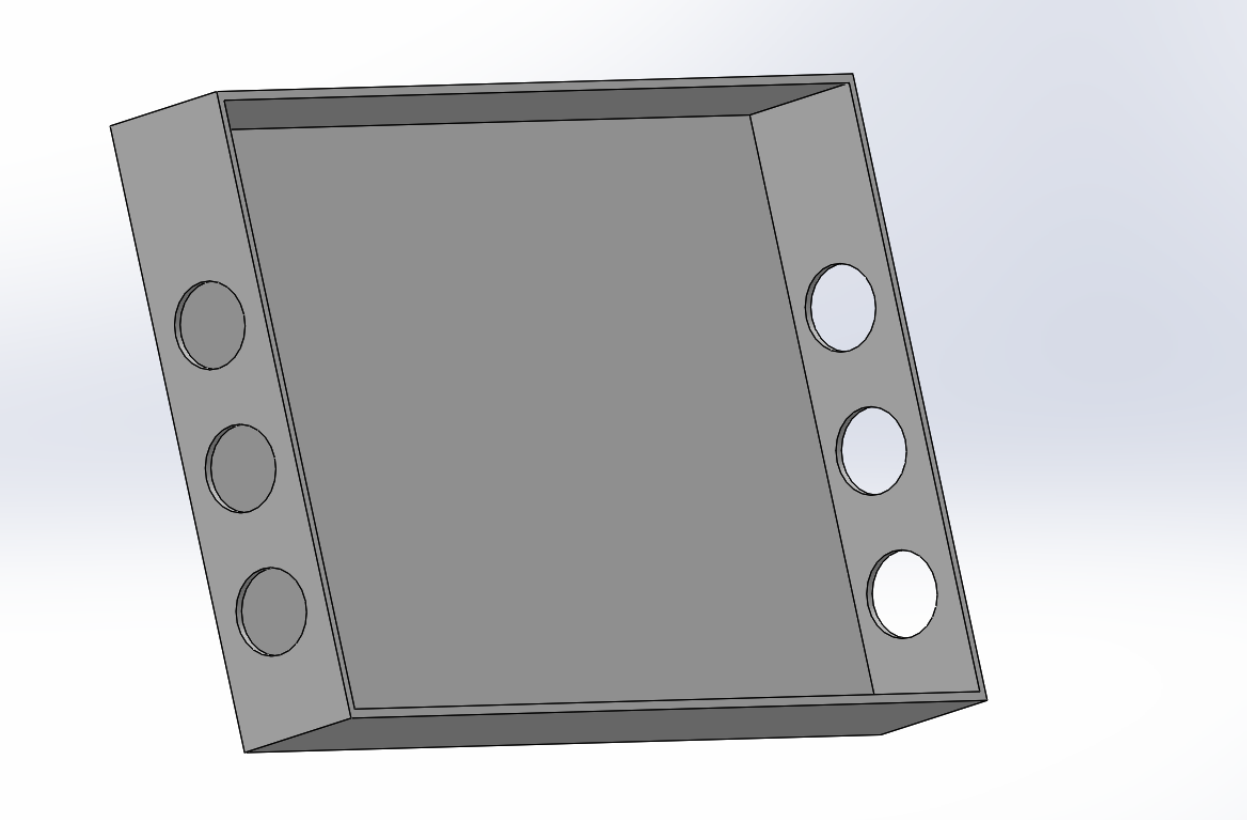


FIGURE 9: SOLIDWORKS DESIGN FOR CLEARANCE

When the group went to print, to print, the piece was changed to not be hollow, as the printer could not design something so thin. Below the 3D print can be seen installed to the build, although at first confident that this would ensure only 1 card is pushed out at a time, it led to the jamming of cards with the dealer, as the clearance was not perfect. Thus, as seen later in our mechanical design portion, the group repurposed the print to help serve as a part of the ramp.

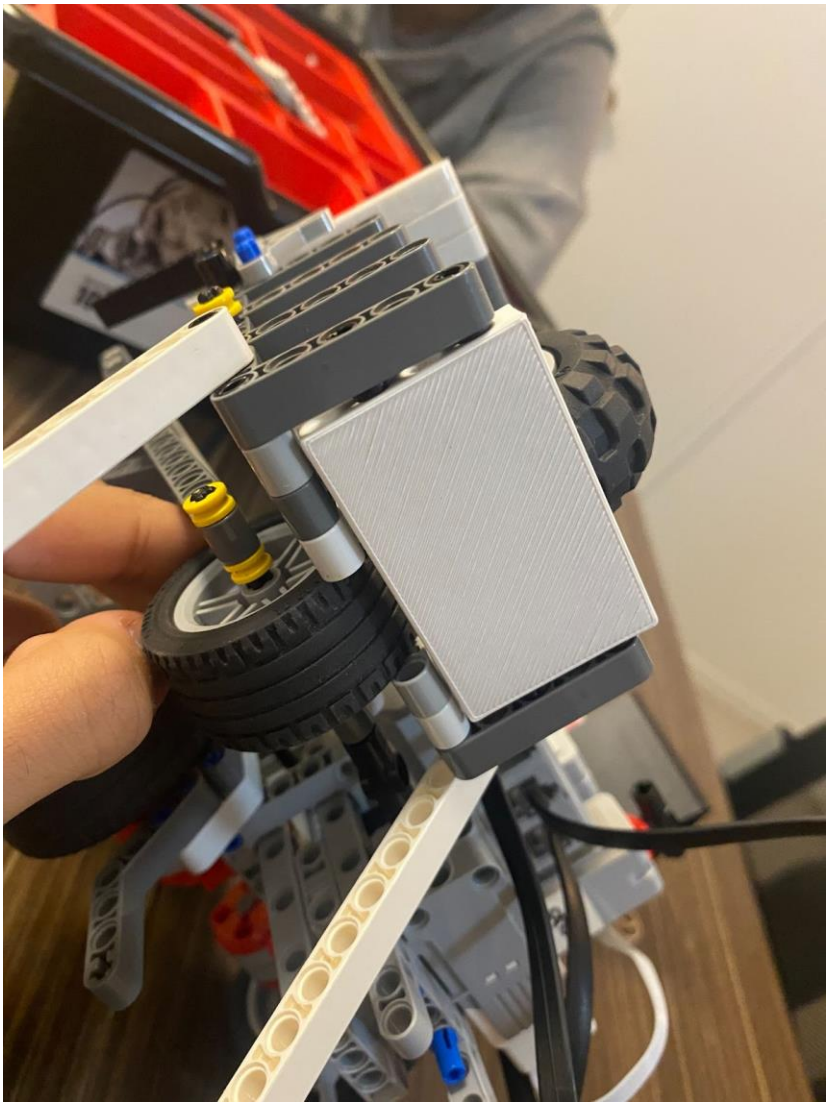


FIGURE 10: CLEARANCE INSTALLED ON ROBOT

Overall Assembly:

At this point the initial design was complete to the point the group had sketched it to be, however, the group soon learned that this design was not capable of dealing one card at a time consistently, requiring us to redesign the individual parts of our assembly. The redesign of the clamp mechanism is discussed in the next progress update.

Progress 6 (November 21st):

Chassis Design

This was the most major step in the chassis design in our whole project. As the group discussed in progress 5 our initial chassis design was done, and it followed our sketches and previous planning. However, it possessed one major flaw, due to all the weight on one side of the robot, the dealing side was drastically sagging to one side and the robot was not as flat as the group wanted it to be. Another flaw was with the way the group assembled our clamp system, our clamp proved to be problematic because as the group started running tests, it was very inconsistent in the way it dealt. It was either putting too much force on the card where the whole deck was being dealt at once, or no force and the whole deck was staying still and not being dealt. These two flaws combined made the robot initially use less because the problem the group was trying to fix was to automate the job of a dealer, but the group could not solve that problem because the robot was either not dealing at all or dealing very inconsistently.

Even though the due date was swiftly approaching, the group decided that together if everyone worked hard together, the group would be able to resign the whole chassis of the robot. The robot disassembled the whole previous design and began brainstorming new ideas. Since the group became more experienced throughout this project, the group quickly redesigned a new and better clamp that included rubber bands. The group also decided that it was need to build a counterweight on the opposite side of the card dealer to reduce sagging.

FIGURE 11: REWORKED DESIGN

Overall Assembly

Overall, this was the most stressful, yet most effective and successful progress in our project. the group quickly realized that when working with Legos it is very difficult to have a working design as the group intended it to be. In the most challenging time in our robot assembly, the group was

able to come close together as a group and work the best they have ever worked. The group was able to successfully redesign a new clamp design and finalize all our ideas before the group got to the final build.

Progress 7 (November 22nd):

Motor Drive Design:

Having realized that a major problem of our previous dealer was the axle connected to the motor bending under the weight of the dealing device, the group decided to move the motor behind the wheel instead of beside the wheel. Furthermore, the group used a similar design to before when it came to building a base for our motor, but now, as both the motor and wheel were facing the same direction, the axle did not bend under the weight. The group accomplished this through using gears to convert the motor's forward motion back onto the tire allowing it to also rotate forward [2]. This gear design can be seen below.

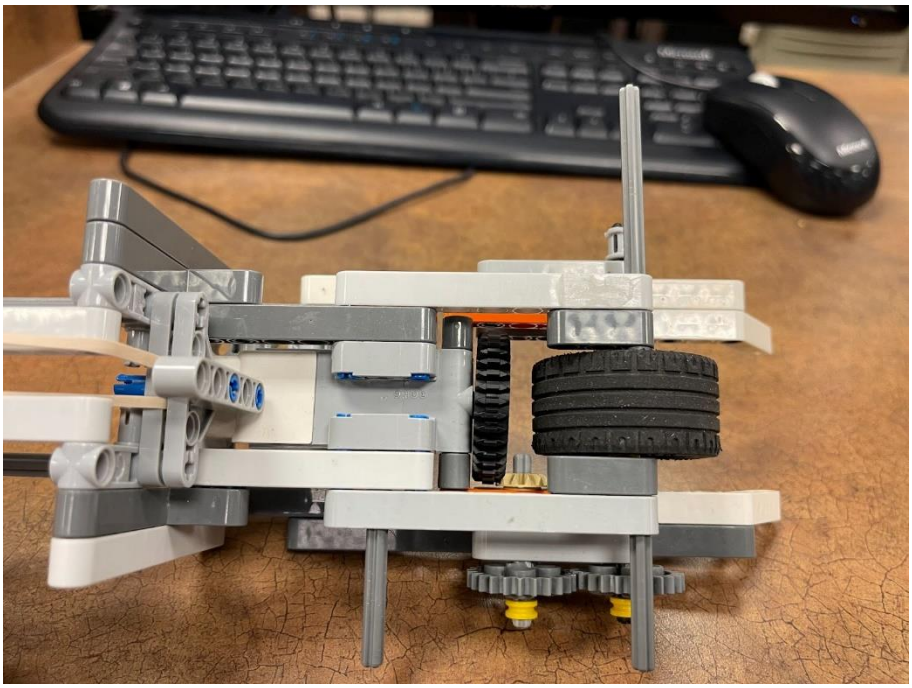


FIGURE 12: FINAL CLAMP SYSTEM

Overall Assembly:

At this stage, the group started testing the dealer's functionality without any additional software (i.e., did not input many players or have the robot rotate).

While testing this the group realized that they had to make some major design decisions and prioritize the dealer's ability to deal one card at a time as consistently as possible. The design below, frequently dealt in clumps, pushing out 2-4 cards at once, and oftentimes got jammed against the 3D printed clearance. After realizing a re-work was needed, it was brainstormed what to do as the clamp system was failing. The group decided on sticking with the rubber bands for tension but instead use a different piece, after looking online for clamp designs, the group went with two 'L 'connectors (pictured below), as it had a flatter surface that would be flush with each individual card, in addition to using our rubber bands in a crisscross configuration for more tension. Finally, the group replaced the deck of cards used to eliminate any other potential causes for error.

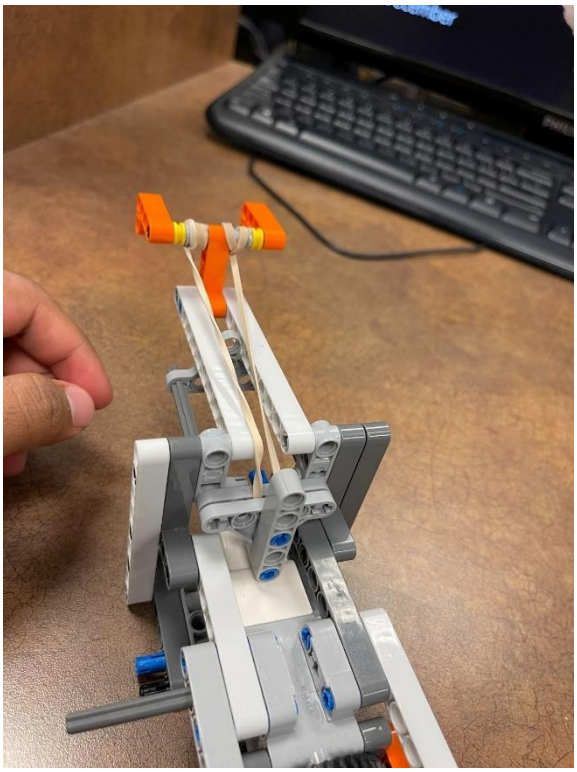


FIGURE 13: FINAL DEALER SYSTEM

This new system pointed us in the right direction, it was a lot more consistent at dealing one card at a time. The next stage was perfecting the wait time and motor power, so it can deal only one card every time!

Progress 8 (November 23rd):

Chassis Design

The group perfected our software timings. It was noticed the new design was performing up to par. It was meeting expectations during testing, except for some inconsistencies. The group noticed that if there was an error early in the dealing process, the error kept on getting worse as more turns occurred. The group understood that gyro sensors will not be as accurate as they were

wanted to be. Our next best option was to add more counterweight on the opposite side of the robot, which would make it more stable and improve the turns slightly

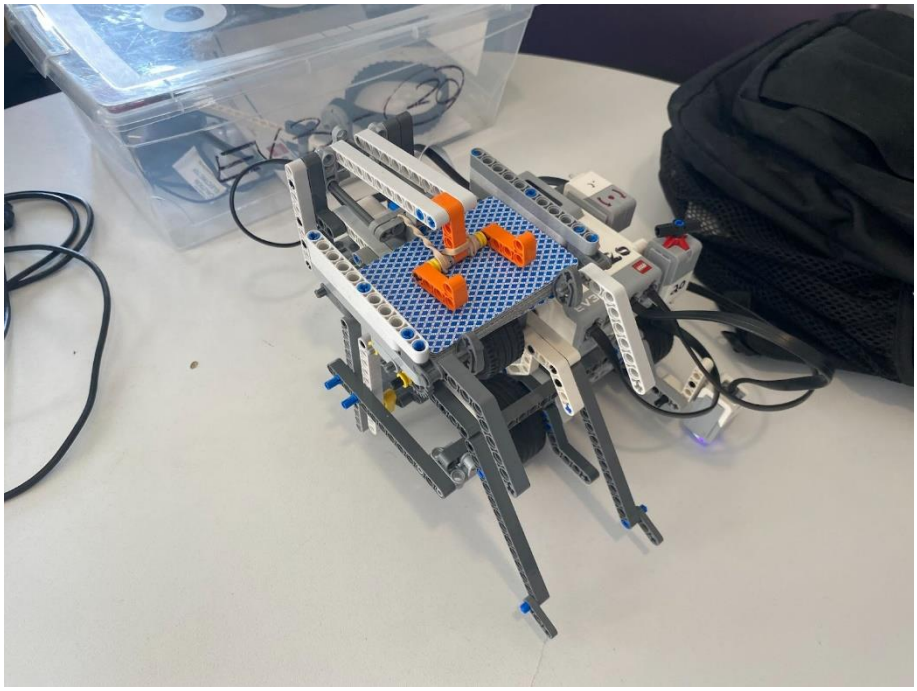


FIGURE 14: FINAL BUILD

Overall Assembly

The overall assembly of the robot consisted of a new clamp system as well as a counterweight on the opposite side. After perfecting the mechanical, a few software tests completed our final robot project. The group had difficulty in consistently dealing single cards; the issue was later cleared with some adjusting of the timers.

Software Design and Implementation

Description of the software

Task list

The tasks required from our software to perform during the demo are as follows.

- Robot starts up
- Robot does test motion (moves forward, backward, spins)

- User enters number of players
- User enters number of cards
- Robot goes to colored tape at table centre
- Deals to the respective number of players
- Robot exits game playing area after dealing
- Robot dumps out remaining cards
- Robot displays number of cards dealt
- Touch sensor to be used as an emergency exit

Throughout the entirety of the project most of the requirements of the robot did not change. However, the task of the robot adjusting itself using an ultrasonic sensor was taken out because it did not improve the accuracy of the dealing by a notable degree.

Functions

Test_ Function (parameters: none, return type void)

The test function was written by Sachin. This function performs a test motion in the beginning to ensure that the gyro and motors are working correctly while carrying a deck of cards.

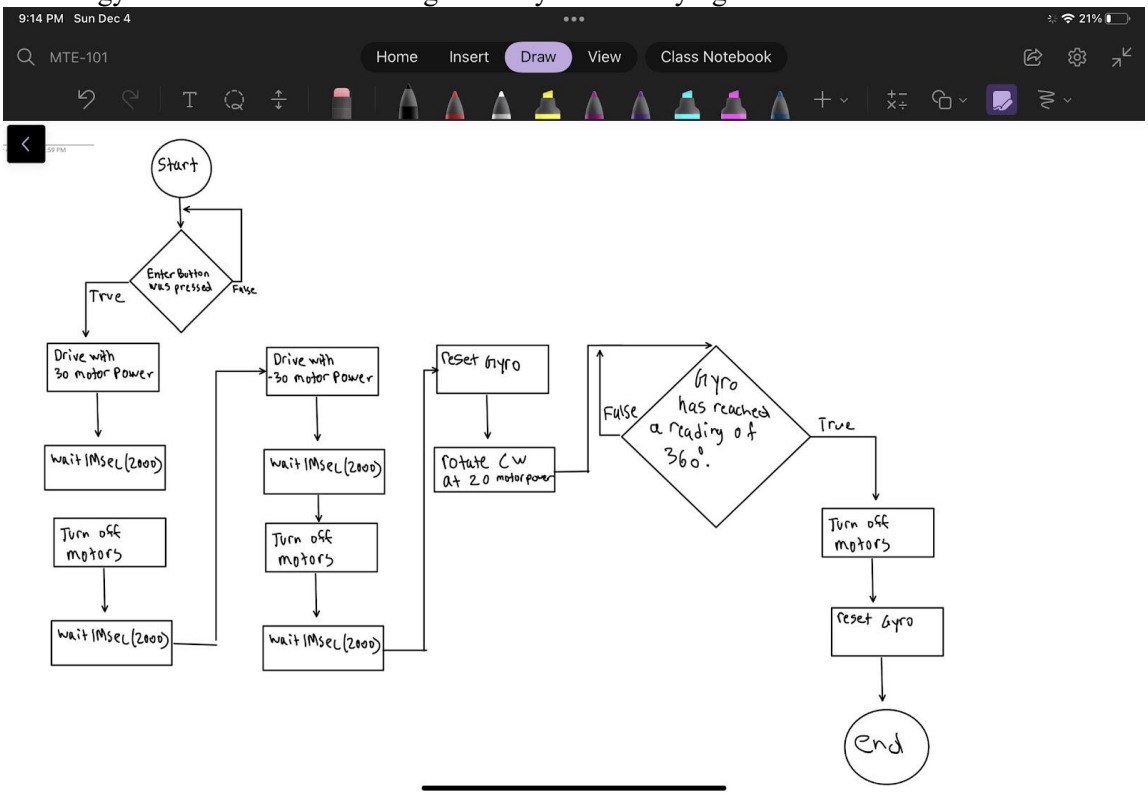


FIGURE 15: FLOWCHART FOR TEST FUNCTION

Drive_ to _Colour (parameters: int colour, return type float)

This function was written by Savyo and takes an input of a number associated with a colour. It then turns the motors on and drives the robot forward until the colour sensor senses the inputted colour as seen in the flow chart.

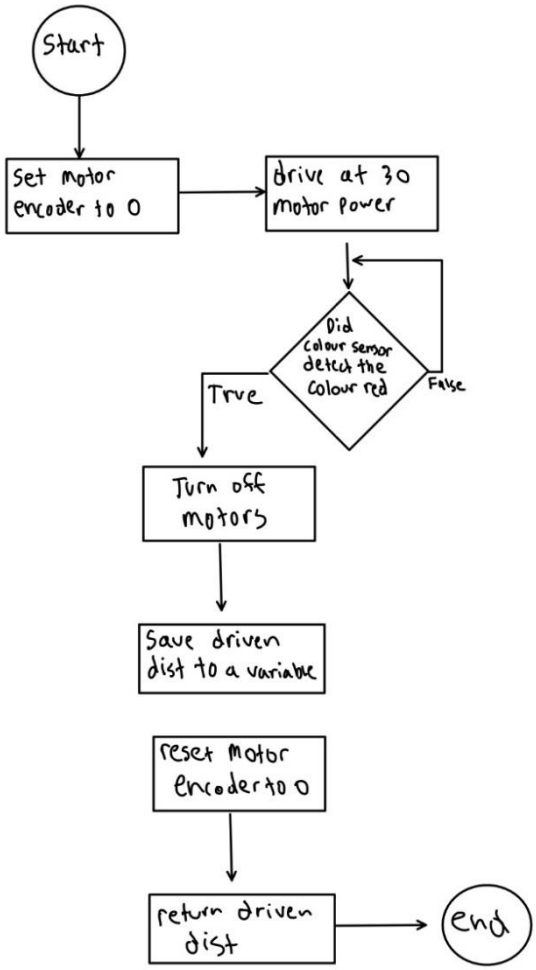


FIGURE 16: FLOWCHART COLOR SENSOR

Rotate _dealer (parameters: int players, return type void):

This function was written by Cheran and takes an input of the number of players in the game and rotates the robot using the gyro sensor. To calculate the angle to rotate a trivial function is called. An example of how the angle is calculated is as follows. If there were 10 players, the function would call the trivial function and divide one rotation (360 degrees) by 10 to rotate the robot by 36 degrees. Note that it rotates clockwise by setting one motor power to a positive value and the other to a negative value. The chronological order of how the function works can be seen in the following flow chart.

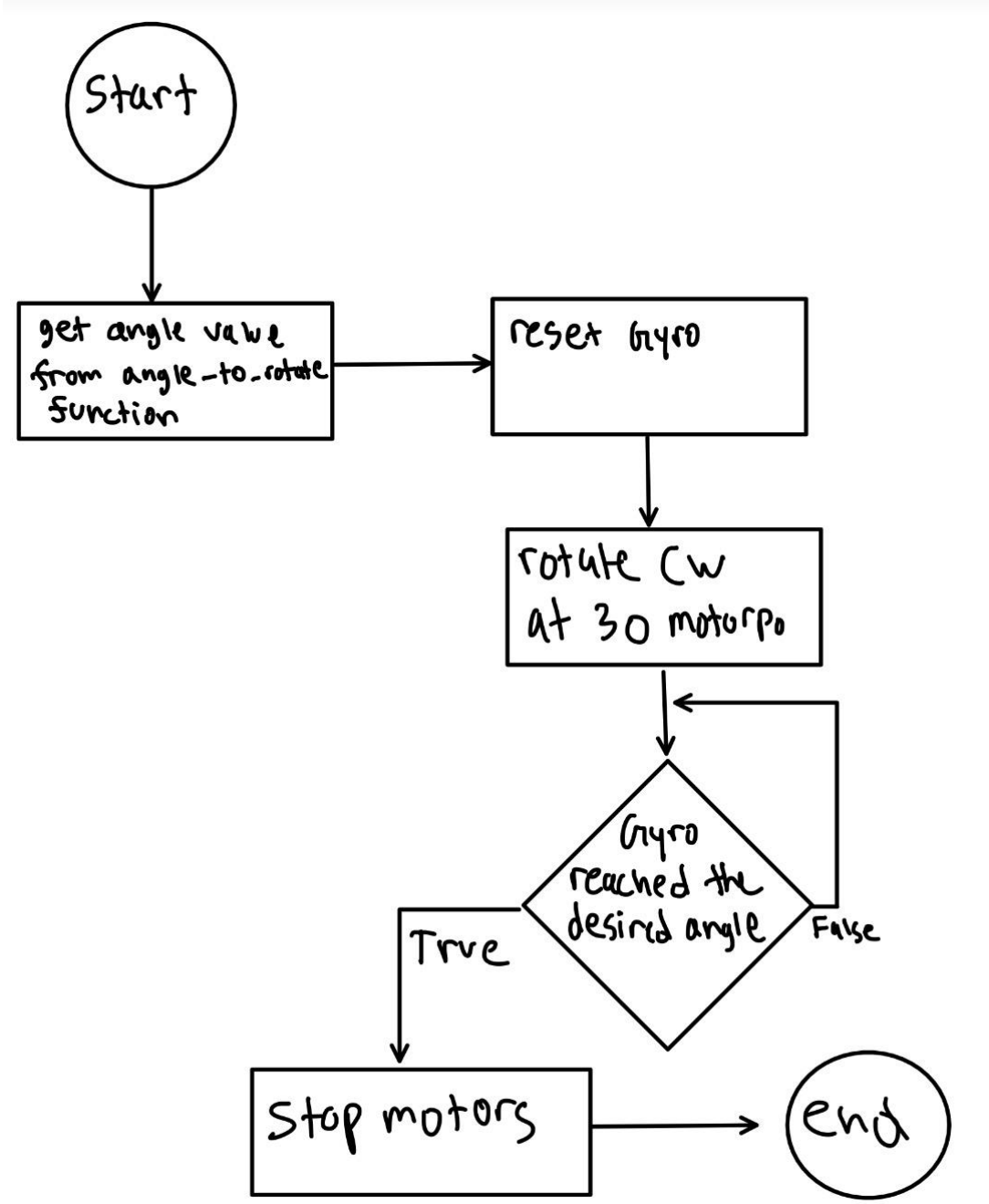


FIGURE 17: ROTATING AND DEAL FLOW CHART

Number_of_players(parameters: none, return type int)

This function was written by Prabhjot and is used to acquire user input to determine the number of players in the card game. The function requires the user to use the left and right button to increment or decrement

the number of players. Each time the right button is clicked the function will increment the counter by 1, when the enter button is clicked the function will return the counter and exit the function.

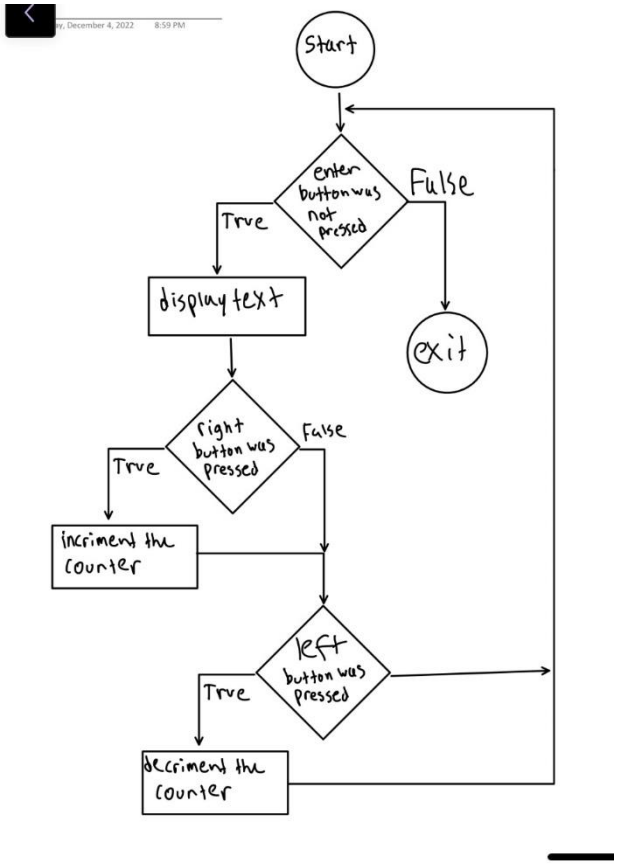


FIGURE 18:

Input_error(parameters: none, return type int)

This function was written by Savyo and is used to error check the user input of the number of cards. It uses a while loop and within the loop calls the number_of_players functions. Once it gets the input from the user it checks if it meets the condition of less than 12 and greater than 1 player was inputted. If not it will display that there is an invalid input and iterate through the while loop again. Once a valid input is given the while loop will break and will return the number of players.

Number_of_cards(parameters: none, return type int):

This function was written by Sachin and is used to acquire user input to determine the number of cards per person in the card game. The function requires the user to use the left and right button to increment or decrement the number of cards. Each time the right button is clicked the function will increment the counter by 1, when the enter button is clicked the function will return the counter and exit the function.

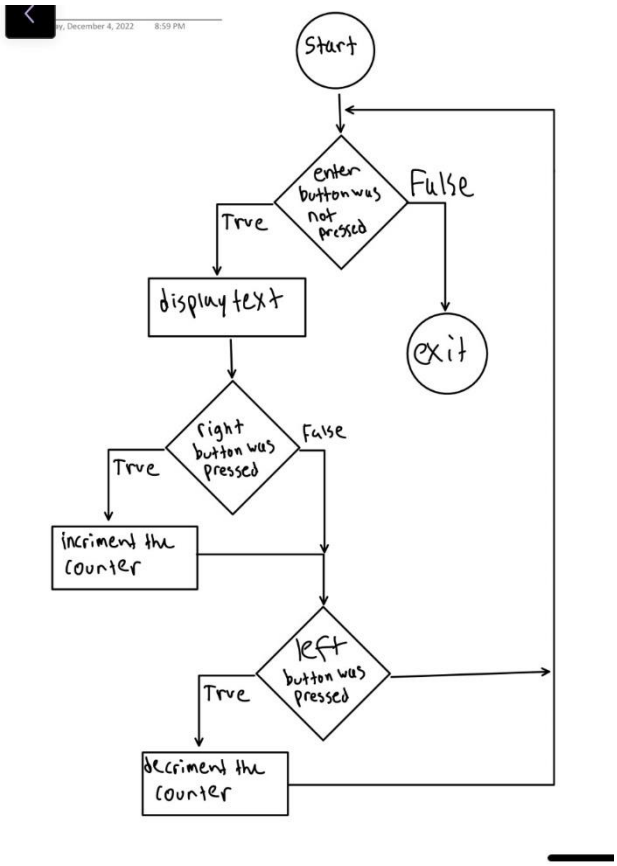


FIGURE 19:

Cards_error(parameters: int players, return type int):

This function was written by Cheran and is used to error check the number of cards inputted. It takes in a parameter of the number of players. It then uses a while loop to ensure that the total number of cards does not exceed 52. For example, if there are 10 players, within the while loop it will run the number_of_cards function and then multiply that number of cards per person by 10. If 8 cards were inputted it will multiply it so that 80 cards are needed total. However, 80 cards exceeds the maximum, in this case an error message will be outputted on the EV3 display and iterate through the while loop again. Once a valid input is given the function will return the amount of cards per person.

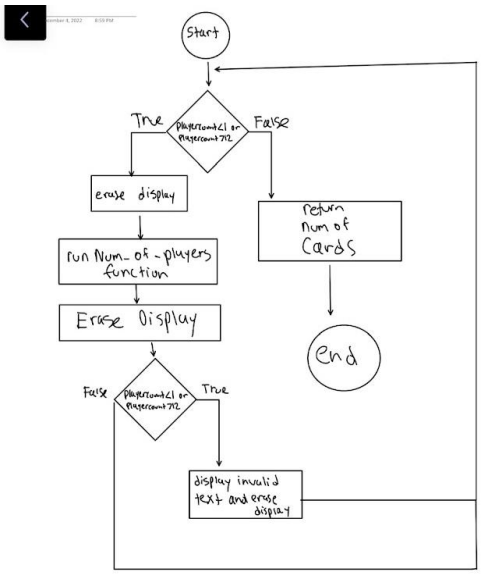


FIGURE 20:

Dispense_cards(parameters: int players, return type void):
This function was written by Prabhjot and is used to dispense 1 round of cards. It uses a for loop for the number of players. Within 1 iteration of the for loop it calls on the rotate_dealer function to rotate the dealer to a person. It then turns on motor to dispense a card. It breaks the for loop once each person gets one card.

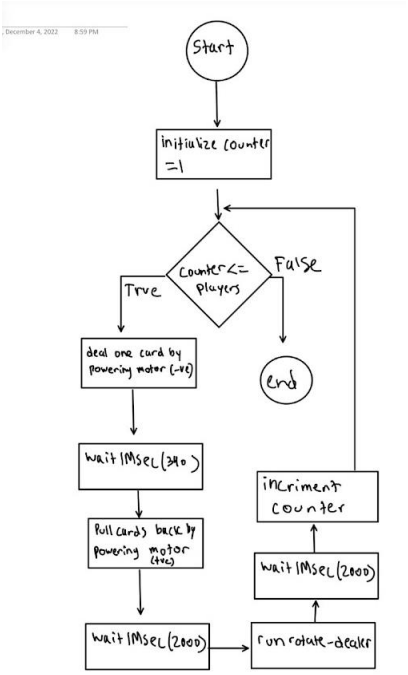
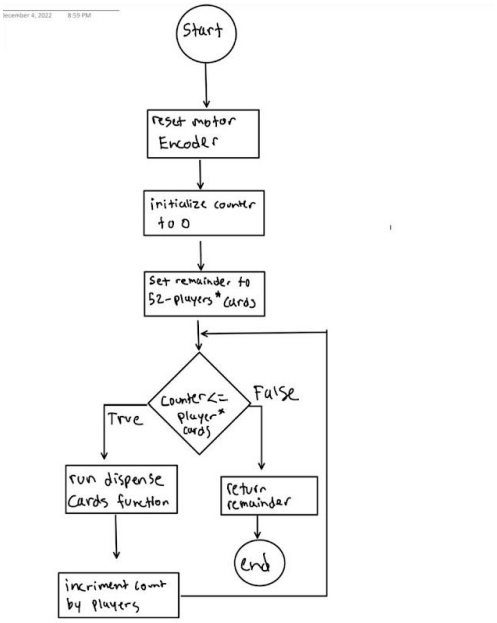


FIGURE 21:

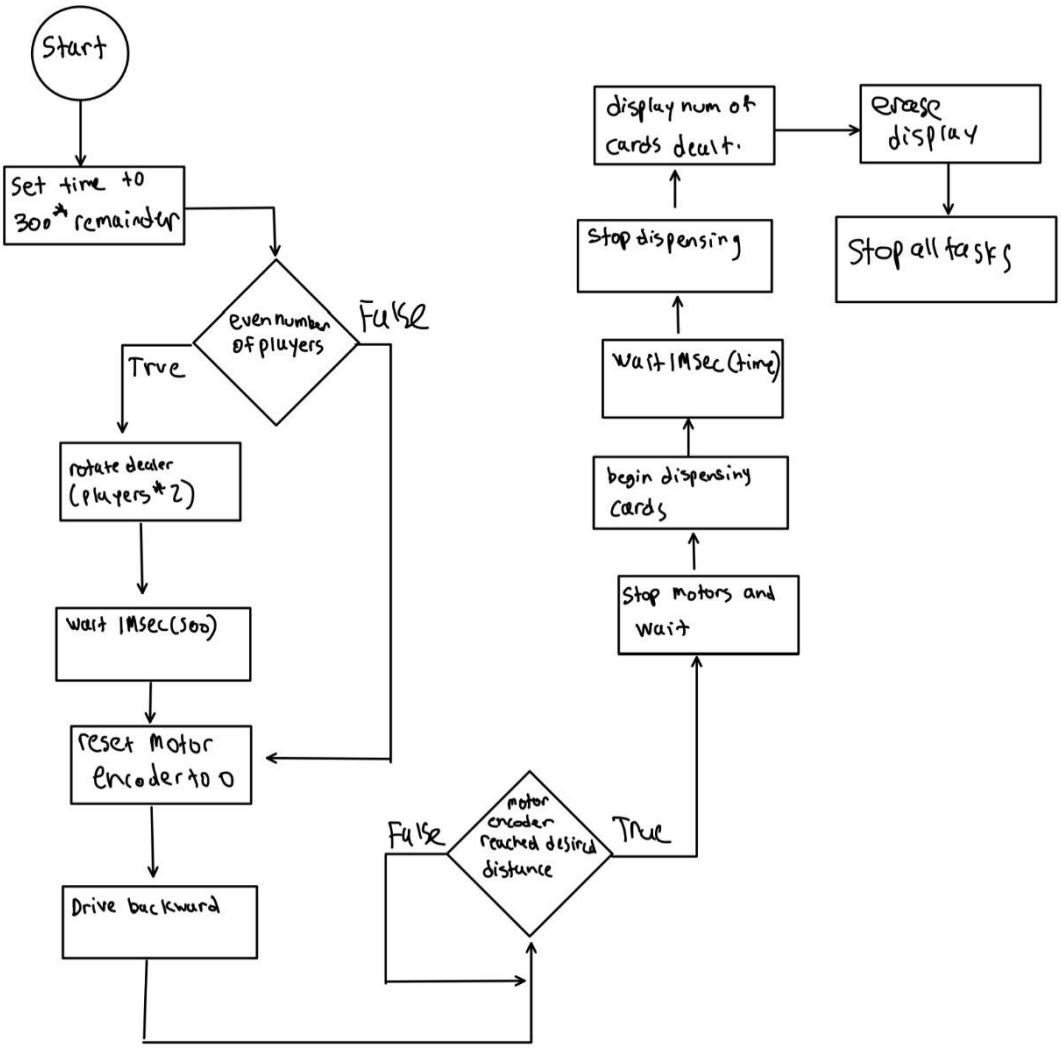
Remainder Dispense(parameters: int players and cards, return type int)

This function was written by Savyo and is used to dispense an equal amount of cards to each player. First it uses the modulus operator to ascertain the amount of remaining cards. For example, with 10 players 52%10 will return 2. Which entails that 50/52 cards should be dealt. Then a while loop is used where it breaks if the number of cards dealt becomes greater than or equal to the maximum amount set beforehand. Within this while loop the dispense_cards function is used. To keep track of how many cards were dealt a counter is used that increments by the number of players each iteration. It increments by the number of players as when the dispense_cards function is called it will deal one card per person. Finally, the function returns the remainder. This can be seen in the flowchart below.



Exit_code (parameters: int players, float distance, int remainder, return type void)

This function was written by Sachin and performs a set of tasks to end the function of the robot. The robot reverses out of the playing field (using the inputted distance), deals the remaining card to the side to act as a bank of cards, displays the number of cards dealt and turns off [3]. This can be seen in the flowchart.



Trade-offs

The choice of resetting the Gyro after each rotation was an action that resulted in a trade-off. Doing so, made it so that the team did not need to implement the incrementing rotation code. However, it did result in a reoccurring error of a few degrees. Meaning, after many rotations, the robot was off course by a significant angle. Another trade-off resulted due to the choice of not to implement the ultrasonic sensor. This forced the user to direct the robot towards a player rather than enabling it to start from any point on the table's perimeter.

Choice of variable type

A vast majority of the variables used in the team's code were in the form of either floats or integers. The code required integers to count objects like cards as they are single entities and cannot be in the form of fractions. Integers were also used in all the for loops in the form of counters. The use of floats can primarily be found using the distance variables. These variables had recorded the value from the motor encoder which was then converted to centimeters using pi. This value will not be an exact integer meaning that it will include a fractional segment as well.

Testing

In terms of testing our main code, the group decided to take a similar approach to our MTE-121 tutorials where the group would test out functions one at a time. The group initially tested the test motion function to ensure all the motors and sensors were set up properly as well as to check if our robot was driving and rotating properly. In doing so, it also tested out primary functions which included drive and drive all. Next, the group began testing our drive_to_colour function. The group expected this function to stop the robot once it had detected a certain colour. The group had to test this function to ensure the robot did not drive off a table when the program was run. This function would enable the robot to stop at the Centre of a table, denoted by the colour. After doing so, werobot's ability to deal a single card at a time by running the dispense card's function. This test was run to ensure that the cards were being dealt one at a time, as otherwise it would defeat the purpose of our robot. The testing purposes had a set number of rotations and cards that the robot would deal with. This function required the most amount of testing and configuring as the timings would need to be changed each day. The group expected this function to have the robot dispense a single card at a time while also rotating a certain angle repeatedly with a certain level of accuracy.

Problems

There were a few problems that occurred with our software. The most prominent issues include the user input double counting and implementing the emergency exit button.

The card dealer requires a user to use the right and left button to increment or decrement the number of players and cards that were needed. At first, the function only waited for the button to be clicked. Due to the speed at which the loops were being iterated through the counter would not increase/ decrease by 1. Before the user could lift their finger it would count it as 2 or even 3 clicks. To overcome this issue, added a condition to the function such that it would not go to the next iteration until the button was released. A while loop was used so that the function would not progress while the button was being pressed as seen in the image below.

```
if(getButtonPress(buttonRight))
{
  pressCount+=1;

  wait1Msec(5);
  while(getButtonPress(buttonRight))
  { }
}
```

FIGURE 22:

The other issue was the implementation of the emergency exit button. The robot uses the touch sensor to end the program if something were to go wrong. At first the program would not terminate with the touch sensor while the code was within a loop. For example, while the robot was executing the drive_to_colour function it would not stop while it was performing the loop to drive till the colour was sensed. It was also not very practical to add a lot of if statements to check if the touch sensor was clicked. To overcome this, the group used a separate task to perform multitasking within the code. Alongside the main task a separate task was executed in tangent to the main that continuously checked for the condition of the touch sensor being pressed.

```
task EmergencyStop()
{
  while(SensorValue[S1]!=1)
  {}
  stopAllTasks();
}
```

FIGURE 23:

Verification

Final Constraints List

Below was our final criteria list, and how the group met each criteria during the demo.

- Completes test motions well
  - It did what it was programmed to do. Drove forward, backward, and spun.

- Detects red and stops
  - The color sensor stopped precisely at the red tape the group had laid out for it.

- Invalid inputs for number of players are detected and user is informed (i.e., 25 players)
  - After inputting an impractical number of players, our robot told the user this is impossible.

- Deals in general direction to player (distinct piles)

    - Although the gyro is inconsistent, our robot was able to deal in 3 distinct piles so that the players were able to tell that a specific pile was theirs.

- Deals to correct number of players
    - During the demo the group put 3 players in, and the robot rotated and dealt to the 3 spots.

- Escapes area cleanly
    - Our program used the number of players to calculate an angle where cards would not be so it could escape. This was done cleanly during the demo as no dealt cards were moved/hit.

- Dumps remainder of cards out
    - After escaping it dumped out all excess cards.

- Displays correct number of cards dealt
    - The robot displayed 12 cards on the screen (3 players, 4 cards).

Project Plan

One of our main priorities was distributing work evenly, not only so the group could accomplish a major task but also, so everyone felt involved. Nevertheless, the group did have members work on certain parts more than others.

- The group all created our EGAD project concept sketches and as a group chose the one, the group wanted to go with.
- Prabhjot and Cheran focussed on the early build, while Sachin and Savyo worked on the slideshows
- The all coded at least 2 functions.
- During testing the group alternated keeping the robot and trying to improve each part.
- Nearing the final build, the group alternated 2 people working on the bot, and 2 people writing the reflection.

The group would have preferred to have everyone work equally on the mechanical part, however, the group felt that having all 4 of us working on the build at once only limited our productivity.

Conclusion

Overall, as a group everyone was happy with the outcome of the robot considering all the challenges faced, given that the Lego material is not able to handle thin paper very well, as well as inaccurate sensors with addition of time constraints and worrying about other classes, the group was able to come out with functional robot, which solves a problem of automating a dealer's job. Throughout the process the group faced many challenges such as difficulty of distributing one card at a time, but in the end through thorough testing of the software and constant changes in the mechanical design the group was able to come out with a final product.

Some highlights of our mechanical design were adjusting the Lego material to distribute a single card, this was done with the help of a new clamp system with the assistance of rubber bands, gears, and a thin clearance just enough to let a single card through

A highlight in our code was the emergency exit function. In any machine there is always a chance for error, therefore, the group developed a code which takes input form the touch sensor. If the touch sensor was pressed, then the robot would automatically stop. This is important in case if a card gets stuck or if the robot deviates from its intended path.

Recommendations

Mechanical Design:

In the end the robot successfully completed the verifications list. Given the Lego material the robot performed to its best abilities. If there were any changes the group could do to make the robot perform better, it would be to change the material and get better sensors. If the group used Tetrix robotics kit instead of Lego in some parts of our robot, our robot would have been more stable causing less chance for error and sagging. The incorporation of new and better sensors would have guaranteed our robot almost perfect results since the only thing hindering our build was the errors in the gyro in every turn

Software **Recommendations:**

The software design for this project worked flawlessly. It was the mechanical aspect that held the project back. One change that would have helped the software was starting to test in the earlier stages of the design to account for any potential issues that may have arisen. The group also thinks that it would have been a better idea to ensure that the frame was more stable as the group found that the initial build did not hold up as it was transported.

Back Matter

References

[1] C. Bartneck, “Lego brick dimensions and Measurements,” *Christoph Bartneck, Ph.D.*, 15-Jan-2020. [Online]. Available: https://www.bartneck.de/2019/04/21/lego-brick-dimensions-and-measurements/. [Accessed: 06-Dec-2022].

[2] “Simple machines – principle models: Gears,” *LEGO® Education*. [Online]. Available: https://education.lego.com/en-us/lessons/sm/gears#contemplate. [Accessed: 06-Dec-2022].

[3] *Introduction to programming: VEX IQ*. [Online]. Available: http://cmra.rec.ri.cmu.edu/products/teaching_robotc_vexiq/. [Accessed: 06-Dec-2022].

Appendix A

```
void configureAllSensors()
{
		SensorType[S1] = sensorEV3_Touch;
		SensorType[S2] = sensorEV3_Ultrasonic;
		SensorType[S3] = sensorEV3_Color;
		wait1Msec(50);
		SensorMode[S3] = modeEV3Color_Color;
		wait1Msec(50);
		SensorType[S4] = sensorEV3_Gyro;
		wait1Msec(50);
		SensorMode[S4] = modeEV3Gyro_Calibration;
		wait1Msec(100);
		SensorMode[S4] = modeEV3Gyro_RateAndAngle;
		wait1Msec(50);
}

task EmergencyStop()
{
		while(SensorValue[S1]!=1)
```

```
                {}
                stopAllTasks();
}

void drive(int motor_power)//powers both drive motors with the same power
{
                motor[motorA] = motor[motorD] = motor_power;
}

void driveall(int motor_power_A, int motor_power_D, int
motor_power_c)//powers both motors independently
{
                motor[motorA] = motor_power_A;
                motor[motorD] = motor_power_D;
                motor[motorC] = motor_power_c;
}

void Test_Function()
{
                while(!getButtonPress(buttonEnter))
                {}
                drive(30);
                wait1Msec(2000);
                drive(0);
                wait1Msec(1000);
                drive(-30);
                wait1Msec(2000);
                drive(0);

                resetGyro(S4);
                driveall(-20,20,0);
                while (abs(getGyroDegrees(S4))<360)
                {}
                drive(0);
```

```
		resetGyro(S4);
}

int Number_of_players()
{
		int pressCount = 0;
		while(!getButtonPress(buttonEnter))
		{
				displayBigTextLine(2,"Enter # of ");
				displayBigTextLine(6,"players: %d",pressCount);
				if(getButtonPress(buttonRight))
				{
						pressCount+=1;

						wait1Msec(5);

while(getButtonPress(buttonRight))
						{		}
				}
				if(getButtonPress(buttonLeft))
				{
						pressCount-=1;

						wait1Msec(5);

while(getButtonPress(buttonLeft))
						{		}
				}
		}
		return pressCount;
}

int input_error()
{
		int player_count = 0;
```

```
		while(player_count < 1 || player_count > 12)
		{
			eraseDisplay();
			player_count = Number_of_players();

			wait1Msec(500);
			eraseDisplay();

			if(player_count < 1 || player_count > 12)
			{
				displayBigTextLine(2,"invalid input");
				wait1Msec(3000);
				eraseDisplay();
			}
		}

		return player_count;
}

int Number_of_cards()
{
		int pressCount = 0;
		while(!getButtonPress(buttonEnter))
		{
			displayBigTextLine(2,"Enter # of");
			displayBigTextLine(6, "cards: %d",pressCount);
			if(getButtonPress(buttonRight))
			{
				pressCount+=1;

				wait1Msec(5);

while(getButtonPress(buttonRight))
```

```
                                        {               }
                        }
                        if(getButtonPress(buttonLeft))
                        {
                                        pressCount-=1;

                                        wait1Msec(5);

while(getButtonPress(buttonLeft))
                                        {               }
                        }
        }
        return pressCount;
}
int cards_error(int players)
{
        int cards_count = 0;

        while(cards_count < 1 || cards_count*players > 52)
        {
                        eraseDisplay();
                        cards_count = Number_of_cards();

                        wait1Msec(500);
                        eraseDisplay();

                        if(cards_count < 1 || cards_count*players > 52)
                        {
                                        displayBigTextLine(2,"invalid
input");
                                        wait1Msec(3000);
                                        eraseDisplay();
                        }
        }
```

```
                return cards_count;
}


float drive_to_colour(int colour)
{
                nMotorEncoder(motorA)=0;
                float distance = 0;
                drive(30);

                while(SensorValue[S3] != colour)
                {}

                drive(0);
                distance = nMotorEncoder(motorA);
                nMotorEncoder(motorA) = 0;
                return distance;

}

float angle_to_rotate(int players)
{
                return 360.0/players;
}

void rotatedealer(int players)
{
                float angle = angle_to_rotate(players);
                resetGyro(S4);
                driveall(-30,30,0);
                while (abs(getGyroDegrees(S4))<abs(angle))
                {}
                drive(0);
}
```

```
void dispense_cards(int players)
{
		for(int count = 1; count<=players; count++)
		{
				driveall(0,0,-22);
				wait1Msec(360);
				driveall(0,0,15);
				wait1Msec(800);
				rotatedealer(players);
				wait1Msec(2000);
		}
}

int RemainderDispense(int players, int cards)
{
		nMotorEncoder(motorA)=0;
		int remainder = 52-players*cards;
		int count = 0;
		while(count<(players*cards))
		{
				dispense_cards(players);
				count+=players;
		}

		return remainder;

}

void Exit_Code (int players, float distance, int remainder )

{
		int time = 300*remainder;
```

```
   if (players %2 == 0) //means even number of players
   {

      rotatedealer(players*2);

      wait1Msec(500);

      nMotorEncoder(motorA) = 0;
      drive (-25);
      while (abs(nMotorEncoder[motorA])<distance)
      {}
      drive (0);
      wait1Msec (500);
      driveall (0,0,-15);
      wait1Msec(time);
      driveall (0,0,0);



      displayBigTextLine(2, "%d cards were dealt", 52-     r
emainder);

      wait1Msec(10000);
   }

   else
   {

                  nMotorEncoder(motorA) = 0;
                  drive (-25);
                  while (abs(nMotorEncoder[motorA])<distance)
                  {}
                  drive (0);
                  wait1Msec (500);
                  driveall (0,0,-15);
```

```
                                wait1Msec(time);
                                driveall (0,0,0);


                                displayBigTextLine(2, "%d cards", 52-
remainder);
                                displayBigTextLine(5, "were dealt");
                                wait1Msec(10000);
                }

                eraseDisplay();
                stopAllTasks();


}




task main()
{
                startTask(EmergencyStop);
                configureAllSensors();

                Test_Function();

                int player_count = input_error();
                wait1Msec(1000);
                int cards_count = cards_error(player_count);

                wait1Msec(1000);

                float radius = drive_to_colour((int)colorRed);
```

```
        wait1Msec(2000);

        int remainder = RemainderDispense(player_count,cards_count);
        eraseDisplay();

        Exit_Code(player_count,radius,remainder);




}
```
