The Atlas Sand Table

Group 4-39 Final Project for MTE100/MTE121

Project By:
Kabir Raval *(21011754)*
Ethan Catz *(21009328)*
Griffin Wilson *(21015542)*
Oliver Chamoun *(21004452)*

Date of Submission: Dec 6th, 2022

Summary

The Atlas Sand Table was an amazing project where the group gained substantial experience in woodworking, robot building, programming, integrating mechanical and software systems, and designing a unique idea to completion.

The project was an overall success with an amazing demonstration day performance and the group having achieved their goals. This report is a thorough breakdown of the entire project from ideation, research, synthesis, design, fabrication, testing, and completion. The group has come a long way in terms of their experience and learnings from this project and this report will highlight some of the most notable milestones, setbacks, failures, and successes of the journey to completion.

Acknowledgements

The group would like to acknowledge Ryan Consell for all his great guidance and assistance throughout the project. He functioned as our mentor and made sure our goals were in line with the available time frame.

The group would additionally like to acknowledge the advice and guidance of Andrew Beltaos in regard to the mathematical aspect of the program. Although our program never went as advanced as using some of the math provided, he was a valuable resource for any of our mathematical queries.

The group would also like to acknowledge Mohammed Nassar for his guidance and advice on some of the software aspects of this project and additionally for his kind works and motivation during demonstration day.

Table of Contents

List of Figures

Introduction

The main need identified was the need for an intriguing, dynamic, and engineering-oriented centerpiece for the POETS engineering lounge. This centerpiece had to be something that started conversations, something that people could gravitate around, and include an engineering aspect.

The Atlas Sand Table, also simply called “The Sand Table", is a piece of dynamic kinetic sand art that is meant to act as a centerpiece for a student study space on campus, namely the POETS engineering lounge. This was the answer to the problem.

A piece of off-the-shelf kinetic sand art would satisfy these criteria, however many options available on the market are priced anywhere from $1500 to $10000! [1] With the goal of producing something with similar capabilities in mind at a fraction of the cost, the group decided to embark on their journey to build the Atlas Sand Table.

Scope

The robot's overall design is centered around making it an intriguing and dynamic piece of furniture. It does this by using a ball bearing running through sand to create patterns using a magnet below the bed of a table. To create patterns the robot was able to move the magnet in an arc around the center of the table, and linearly towards or away from the center of the table. The final task the robot completed was to indicate the completion of a pattern by flashing the leds on the table.

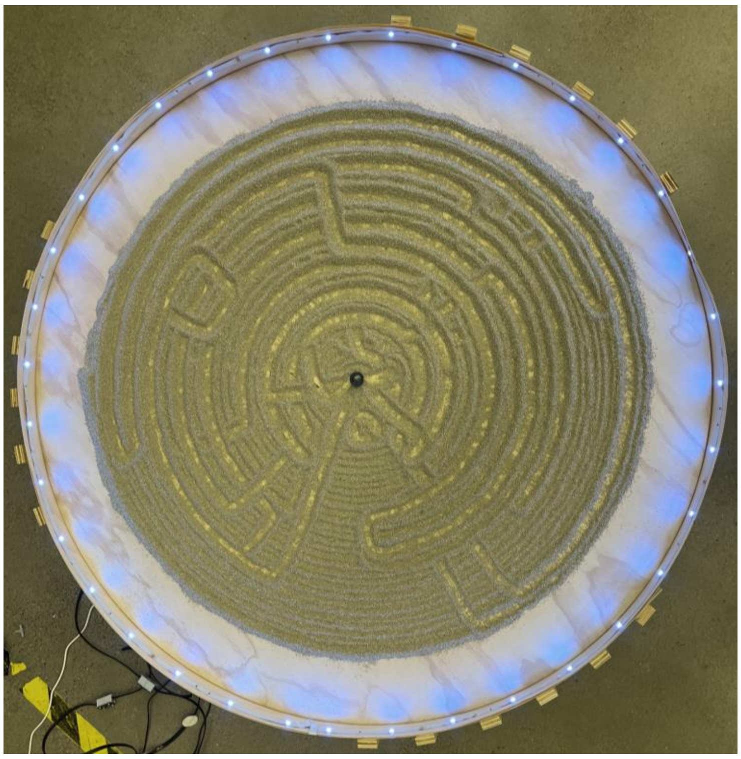


Figure 1: Example pattern created by the robot using randomly generated instructions.

To create these patterns the robot reads instructions from a text file which indicate whether to move in an arc or linearly from the center, and what position to move to, in degrees or cm respectively. The robot then converted the degree/centimeter position to an encoder value to move to based on where the robot was positioned at the time. To decide which pattern to draw, the robot relied on human input using the EV3 buttons. The robot also relied on a touch sensor for two purposes, it needed to be pressed for the robot to start reading instructions, and it was used as an emergency stop button for the robot.

The robot interacted with the environment by first accepting human input on which pattern to draw based on the press of an EV3 button. Afterwards it would move the position of a magnet using two motors, the first would run a chain of gears to pivot the arm of the robot, while the second motor would turn a belt to move the magnet linearly from the center of the robot. At the end of each pattern the robot used a third motor to turn the table's LEDs on and off multiple times to give a visual indication of completion. The robot also accepted human input in the event of failure via the touch sensor emergency stop.

The robot recognized a pattern is complete when it ran out of lines of instructions to read. However, this would not end the program as the robot was designed to run infinitely until it needed to be stopped. To indicate when the program should end the robot had an extra parameter added to its while loops to check for the touch sensor being pressed and would check when each loop ends if the sensor were pressed to determine if it would keep running or stop the program entirely by breaking the main loop.

The scope of the robot stayed consistent throughout the design and build process of the robot. One notable change was that the original intention for the third motor was to indicate the completion of a pattern with an audible cue such as knocking on the table, but this was changed to turning the LEDs on and off as an audible cue was determined to be too disruptive.

Constraints and Criteria:

Constraints

The main constraint of the robot was that due to the nature of it being a table. It would have to run indefinitely and hence the robot would need to be placed near a power outlet so that it can be constantly charging.

Originally when prototyping the robot another constraint was how many turns the robot could make before its wires got tangled. Since the EV3 brick is stationary and the second motor is rotating, one of the cables would eventually get too twisted. The code and files had to accommodate for this constraint.

Criteria

The first criterion of the robot was that the robot needed to be able to draw, at the very least, simple patterns that require only the individual movements of the belt and pivot mechanisms. More complicated movements, such as non-radial straight lines, could then be implemented later using the tandem movements of the belt and pivot.

The second important criterion is that the robot needed to be as quiet as possible. This was important as the main idea was to create a dynamic piece of furniture, specifically to be placed in the POETS lounge, and as POETS is a study space for many, the robot’s movements should not be a disruption.

The final criterion is that the robot had to be able to run consistently for prolonged periods of time. This was required because for complicated patterns the robot would have to follow hundreds of instructions accurately to create the pattern. As well, the robot was designed to do several patterns in a row and a single gear skip or disconnection could ruin an entire pattern.

Mechanical Design Breakdown

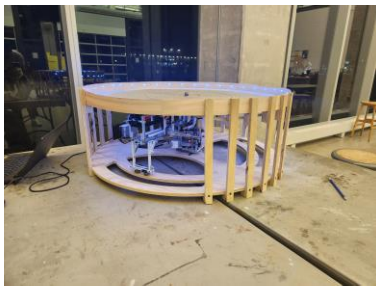


Figure 2: Final Assembly

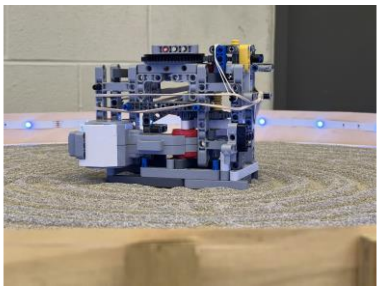


Figure 3: Base Assembly (Pivot)

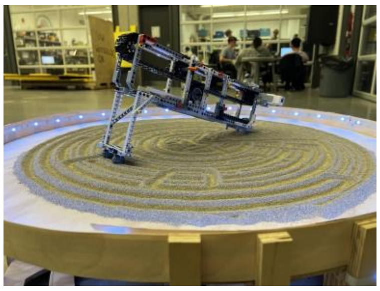


Figure 4: Arm Assembly (Belt)

Design of the Table

The final design of the table, the picture above in Figure 2, was the result of an interactive design process. Ideas and vision were adapted to accommodate material cost and availability, ease of manufacturing, demonstration requirements, and efficient construction.

The overall design consists of two 3ft diameter plywood circles, separated and secured using a total of eighteen plywood legs. All the wooden parts for the table, excluding the veneer, were cut from a single sheet of plywood to reduce wastage and maximize efficiency.

The upper disc is simply a plywood circle that was hand sanded to a smooth finish. The upper disc forms the floor of the "sand tray" and will be the surface on which the ball rolls atop the sand, hence why smoothness is important. The veneer border has a dual purpose. Firstly, it keeps the sand from falling out off the edge. Secondly, the veneer gets sandwiched between the legs and the upper disc, allowing the upper disc to remain in place without the use of any fasteners (which was important during the testing phase since the tabletop would often need to be removed to gain access to the robot mounted inside.

The lower disc is more complex than the upper disc. As seen in Figure 2, it is a plywood disc with a pattern cut into it. This pattern serves three key purposes. Firstly, it saves weight which was helpful when the table had to be carried around campus. Secondly, the pattern makes the table more visually appealing. Since the robot was preferred to be visible, the lower disc would also be visible, therefore a pattern cut into that disc makes it better to look at. Finally, the pattern that is cut underneath the table serves an important purpose, cable management. The cables from the belt and pivot assembly run underneath the table, through the large cutout, and then up the small cutout, to the EV3 brain. This allows the robot to move freely without having to cross over a series of cables.

As for the legs of the table, they are all 1"x14" rectangles, all cut from plywood. The lower part of each leg has two 1"x1" squares glued onto it. This is to provide a "shelf" on which the lower disc rests. Two inches from the top of the legs there are more supporting pieces glued on to create a similar shelf for the top disc to sit on. The legs are secured to the lower disc with wood screws but only held my friction to the top disc, as discussed previously.

The Table Design Process

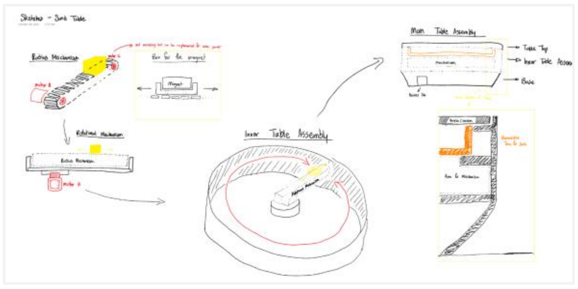


Figure 5: Initial Sketches of Design Ideas

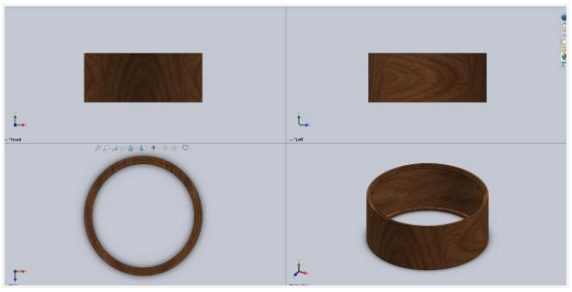


Figure 6: Initial CAD of a possible outer cover

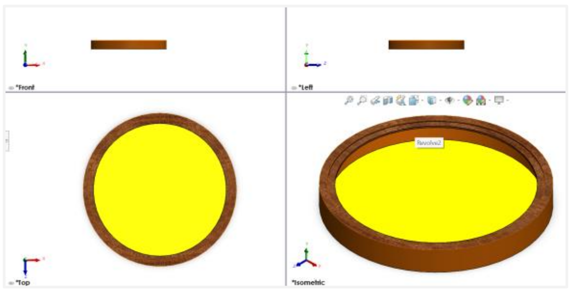


Figure 7: Initial CAD of the Sand Tray

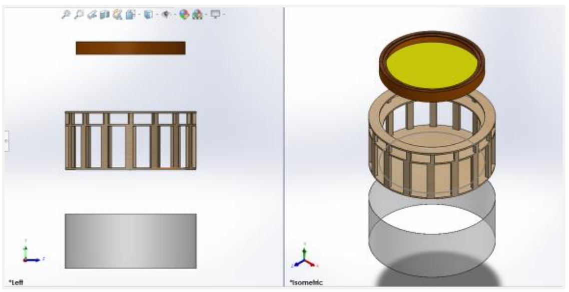


Figure 8: Initial CAD of the table assembly

Design #1

As alluded to above, the final design of the table was not the first design. In fact, the table is a completely redesigned version of the initial idea. Figure 5 shows some initial sketches that were created and submitted. The initial idea breaks down as follows. The table would consist of three main parts, the sand tray, frame, and outer cover/shell.

The sand tray would be a small circular tray that would hold the sand and would be removable from the table. This would provide easy access to the robot underneath for testing and let us efficiently clean up the sand for storage, not to mention that it would also guarantee that no sand fell onto the robot.

The frame would consist of three plywood discs, two of which would be rings. As pictured in Figure 8, the frame would have a large solid disc at the bottom, with tall legs going around the circumference, onto which a ring would be placed. This first ring would function as the shelf onto which the sand tray would rest. Then, more supporting legs, shorter this time would be glued onto the circumference of the first ring, onto which the upper ring would attach. This ring would have the same inner diameter as the sand tray, so the tray would create a snug fit when inside. Finally, on top of the entire assembly would be a sheet of acrylic or glass to function as the tabletop.

The outer cover/shell was the part for which there were the most options. Figure 8 depicts a clear cover that would be a thin sheet of acrylic bent around the frame. Figure 6 shows a wooden outer cover, which could be achieved with veneer sheets, providing a much more furniture-like finish. Other ideas that came up in discussion were canvas, vinyl sticker sheeting, and even leaving the cover off for a more raw/industrial aesthetic.

The original idea was that the table would consist of the sand tray, the frame, and an outer cover. However, this design needed revision very soon after.

Problems with Design #1

The first realization was a very general one, the table was too costly. It would require an unreasonable amount plywood and lumber. Cutting the five plywood discs that would be needed in total could not be done without wasting a lot of wood, which would increase the overall cost. This is because it was not feasible to cut the legs out of plywood since they would need to carry a substantial load, meaning it would require additional lumber such as two-by-fours. Finally, sourcing the amount and size of acrylic that was needed would cost anywhere from $200 to $400 dollars.

Secondly, the table was extremely substantial and would be difficult to construct. It would need to do at least five cycles of gluing to build the frame and sand tray, which would take too much time and create a large mess. Moreover, the design idea for the sand tray would not work. The outer wall of the tray was meant to be made from veneer, which would be too weak to secure the upper ring of the sand tray. This would mean that there would need to be a more substantial sand tray, with thicker walls. Not only would this be heavier, but it would also cost us more money and take more time to construct.

The third and final realization was that the table was overengineered. The goal with this project was to create a great-looking piece of furniture that would act as a piece of kinetic sand art. If too much time was spent on the table and not on the robotic components, the table would be useless as there would be no robot.

It was then established that a simpler design which was strong and easy to assemble. So, the strengths of the original were used to design a new table to meet the new criteria.

Design #2

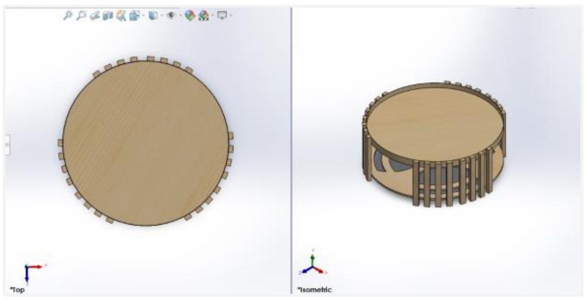


Figure 9: CAD of the new design.

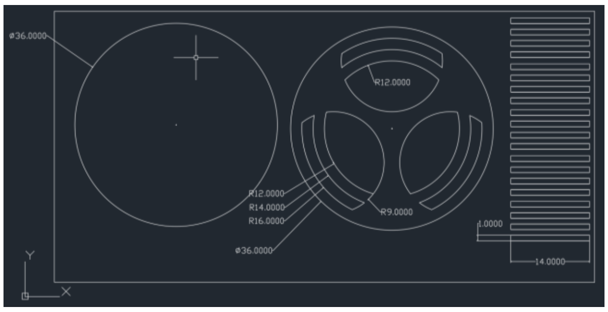


Figure 10: AutoCAD of the parts on a plywood sheet.

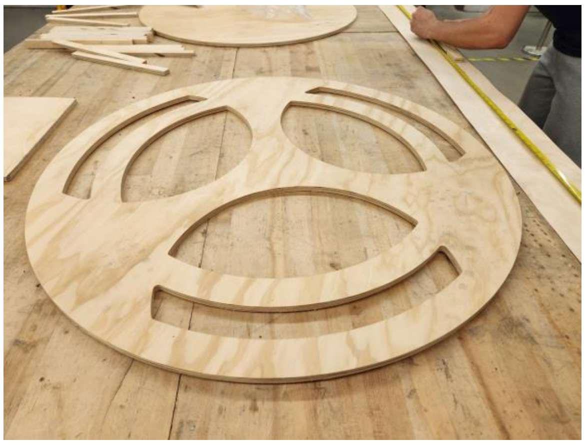


Figure 11: The lower disc of the table.

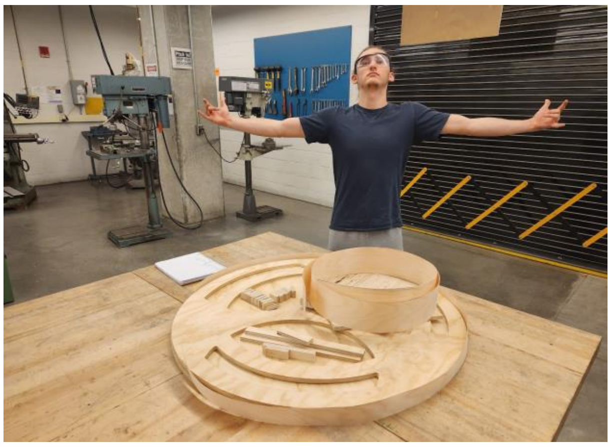


Figure 12: All the table's parts fit into this pile.

The new design (CAD shown in figure 9) was the answer to all of the problems of Idea 1.

- Significantly lower part count, at only two large discs instead of five.
- Decreased weight meant that the eighteen legs could be cut from plywood instead of having to buy two by fours or other such lumber.
- For the purposes of the demonstration the glass or acrylic tabletop was not a necessity and so priorities were shifted to focus on aspects that were essential to the demonstration.
- Easier and faster construction. The whole table took only 4 days to build, totaling about twenty-four man-hours worth of work.
- Upper tray design was updated to no longer need an upper ring, meaning the fragile veneer would not have to support any significant load.
- Overall design was simplified to make the entire build process smoother so that there could also focus on the robot and coding.

Build Process

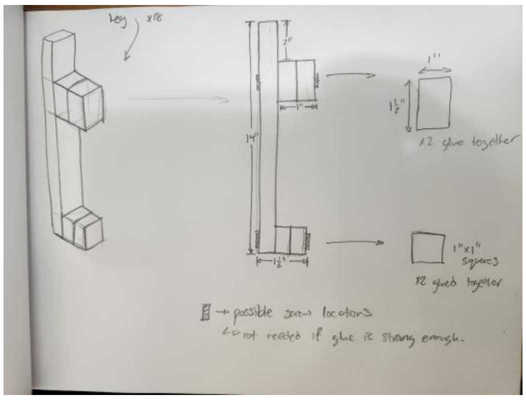


Figure 13: Leg assembly sketch.

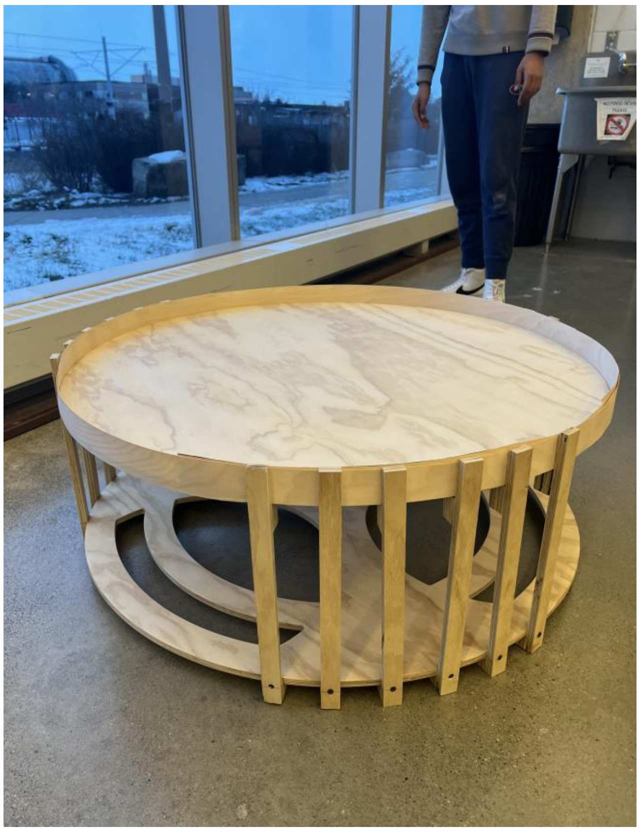


Figure 14: Completed Build with the legs glued and screwed into place and the veneer fitted into the top.

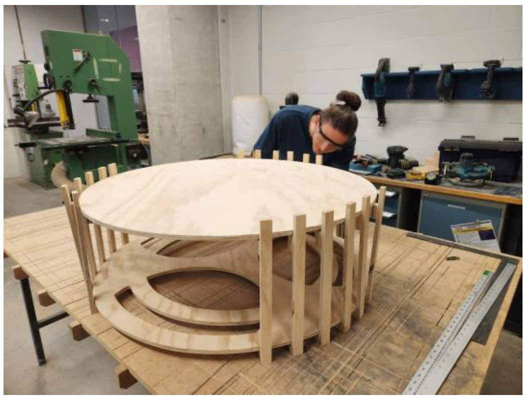


Figure 15: Test fitting the legs

1. All parts (top disc, bottom disc, long beam of the legs) were cut on the CNC router.
2. All supports (1”x1” squares and 1”x1½” rectangle supports shown in figure 13) and veneer sheeting were cut to size.
3. Legs were assembled into their final structure (figure 13) and test fitted (figure 14).
4. Legs glued and screwed into the bottom disc. Top disc friction fitted into with the veneer going around to fill the gap and ensure a secure and snug fit.

The Robot Design Process

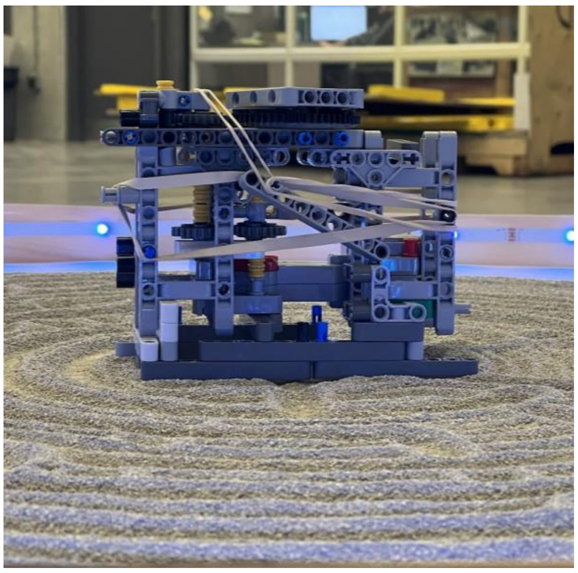


Figure 16: Base Pivot Assembly

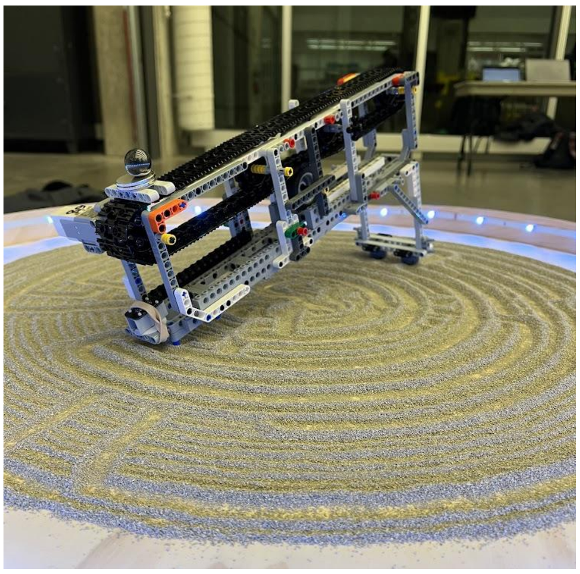


Figure 17: Arm with supporting wheel structure

The robot had 2 main sub-assemblies, the base (pivot) and the arm (belt). Each sub assembly serves its own purpose.

The base (figure 16) contains within a large Lego motor, which is responsible for moving the main arm (figure 17) in a circular motion. This motor is set up on a gear train that reduces the gear ratio from 1:1 to 7.5:1. This gives us a huge torque advantage on the arm, allowing the large motor to rotate more consistently the nearly 16” long arm. The high gear ratio also helps us reduce the backlash of the system. Since the arm is so long any backlash in the motor is amplified at the arm, reducing accuracy. The high gear ratio works to help counteract some of the effects of this backlash. The base is also the robot’s one and only mounting point on the table and it is thus built to be as stable and rigid as possible. The large central gear on the base also allows us to run a cable through the rotational center of the robot, which means the robot arm could turn indefinitely without the fear of cables tangling (in theory).

Onto the central gear the mounts the arm assembly. This assembly rotates around in a circle, and it is the “radius” of the table. The arm assembly has a Lego conveyor belt, driven by a medium motor, which can move to any point on the radius, all the way from the center to the outermost edge. On top of the belt there is a small tray which carries the magnet. The combined motions of the pivot and belt can move the magnet to any point underneath the table’s surface. Since the robot must fit inside the table it can only operate in an area smaller than the table’s diameter. In this case, the robot can cover the circle that is 28” in diameter, centered in the center of the table. The arm assembly also includes the supporting wheels, which provide support for the end of the arm. The wheels are Lego ball bearings and they roll on the lower wooden disc of the table.

Like the table the robot also went through an iterative design process to be realized into its final form.

Prototype 1.0 - Goals: Visualization/Motor Placement/Measurement

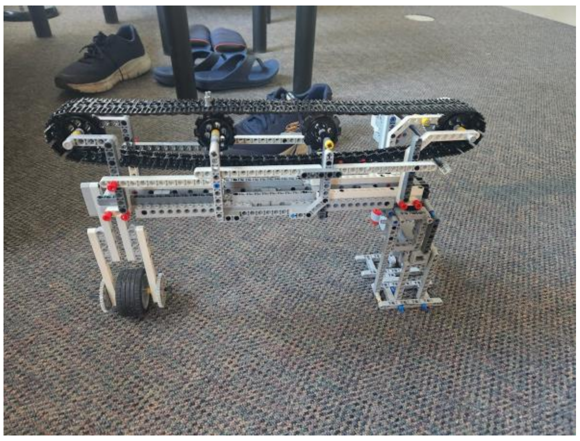


Figure 18: Prototype 1.0

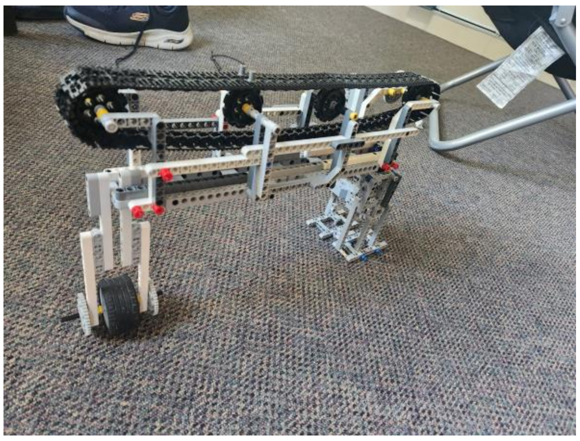


Figure 19: Prototype 1.0 (Better view of the base)

The first prototype was not very functional, although it was a very solid starting point. This prototype gave us an idea of the size and shape of the robot which would inform the CAD design for the table it would sit inside of. Moreover, several lessons were learned. For one wheel on the arm was too wobbly. The base would have to be improved as well, since it had to be more substantial to support the arm and later mount to the table. The track was also very unstable because it had no underside support or lateral support. This was important because the robot would eventually have to accurately carry a magnet on top of it, which requires both kinds of support. Finally, the main drive gear was too small to accommodate the high gear ratios that were needed.

Prototype 2.0 - Goals: Sturdiness and Stability

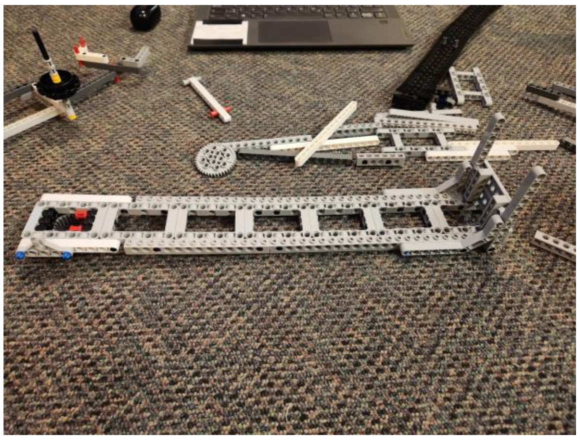


Figure 20: Substructure for the arm was redesigned to be built off of one rigid base that was reinforced throughout.

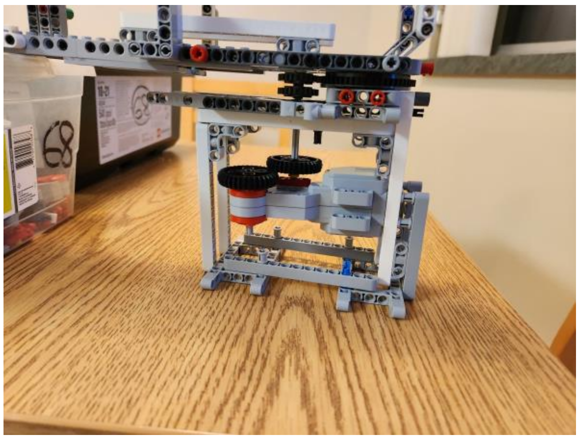


Figure 21: Larger 60 tooth main drive gear added to base to allow for a higher gear ratio to to added with less gears.

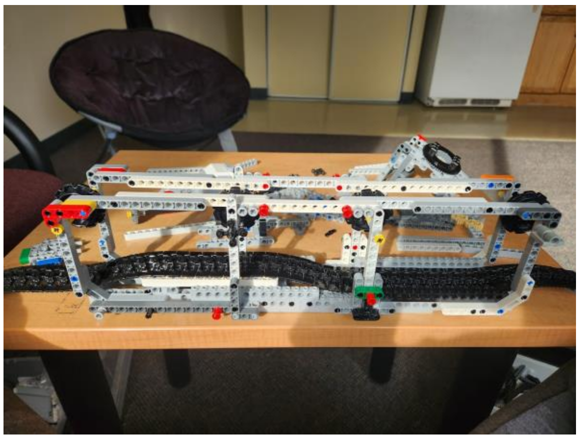


Figure 22: Vertical supporting arms were reinforced, and lateral and under-belt support was added.

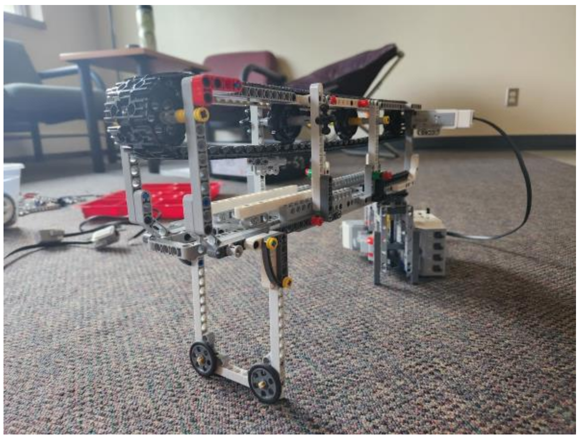


Figure 23: Wheel structure was updated to provide more stability for the arm.

In prototype 1.1, the base was redesigned and made more substantial. Updated includes better supporting legs, better mounting of the large motor, temporary mounting points for the EV3 brain, a larger sixty tooth main drive gear. The entire arm assembly was redesigned to include a better reinforced skeleton (figure 20). The arm was also given stiffer supporting arms and an upper structure that provided the belt with lateral and underside support. The single wheel was swapped out for two smaller ones, which was very stable, but the wheels ended up being too small and the wheel structure scrapped the ground.

At this stage it was important to run test code, but before that could be done the robot needed a new gear train for the pivot motor, since the current one was too sloppy. It also needed a taller set of wheels, along with test code to run.

Prototype 2.1 - Goals: Codable/Achieve Predictable Movement

Figure 24: Upgrading prototype 2.0 with new parts.

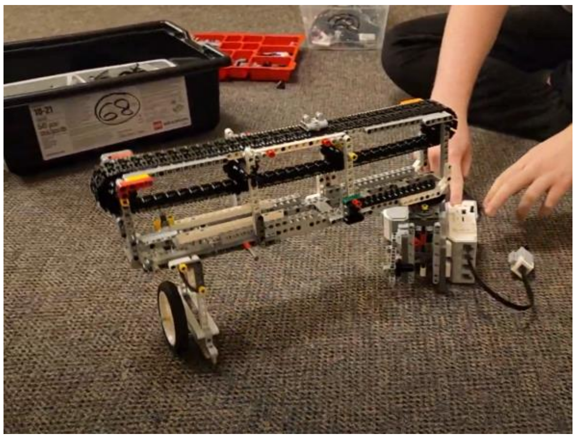


Figure 25: Robot was able to rotate on the new wheel.

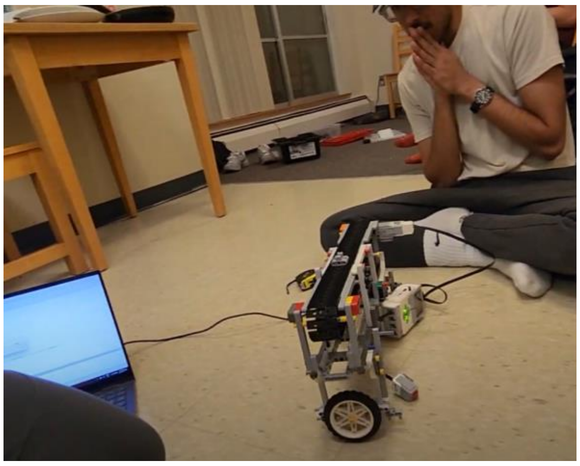


Figure 26: Testing basic code on the robot

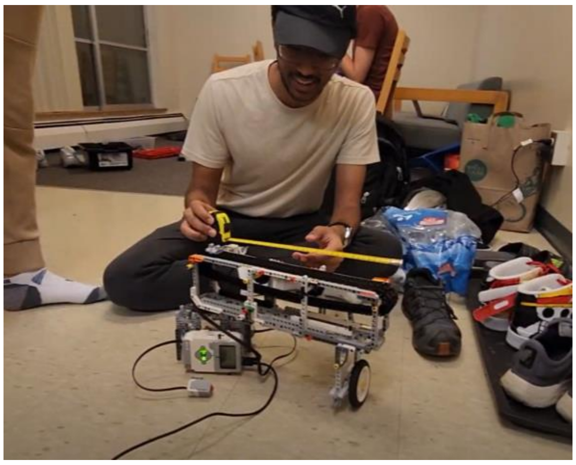


Figure 27: Measuring the movement in order to test if the gear ratio constants are correct

Prototype 2.1 was able to successfully rotate and move the belt by using the medium and large motors. After the new gear train was implemented the required gear ratio constant required to move the belt using an input given in centimeters was calculated. The constants were evaluated by measuring for accuracy (figure 27). There was also a new wheel added to the robot. Although one wheel was unstable, it was used temporarily so that code could be run but would be swapped later for a more stable alternative.

For prototype three only minor changes were made before the robot could be mounted inside the table. The wheels were updated to use ball bearings instead of rubber for less friction and noise. The touch sensor was also mounted somewhere more noticeable (next to EV3) since it was the system E-stop. The base design was also changed to be able to mount it in the table. Finally, a tensioning wheel was added to the belt so that it would stay securely on the gears and rails. These design upgrades were implemented into prototype 3 and this became the design that was carried through to completion with some minor tweaks (Figure 28 below showcases prototype 3).

Integrating the Robot and the Table

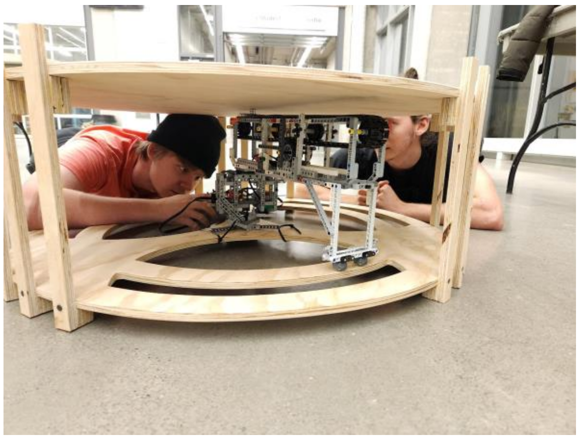


Figure 28: Prototype 3.0 being zip tied into the table for temporary testing (Note: New wheels and tensioner added).

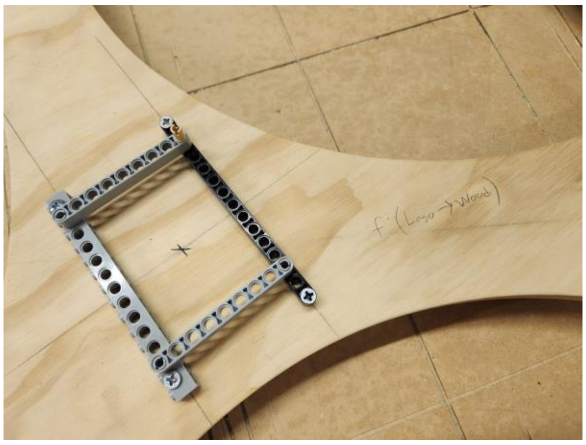


Figure 29: Mounting Points for the robot screwed into the table and aligned with the center.

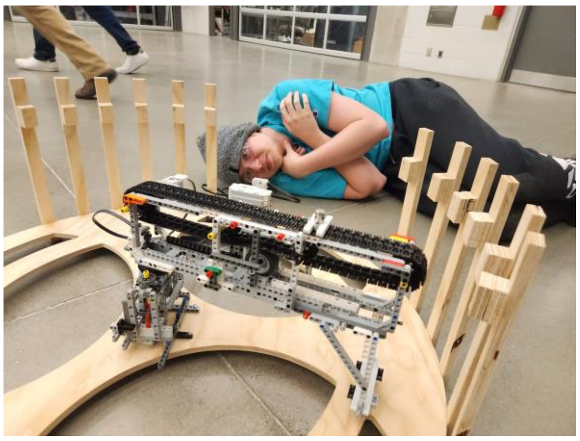


Figure 30: EV3 displays debugging data that can be read while the robot is in motion to see if code is running correctly

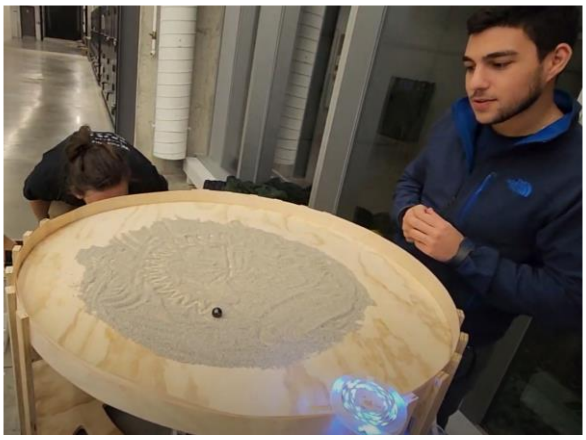


Figure 31: Testing movement in sand by creating basic gear patterns on a thin layer.

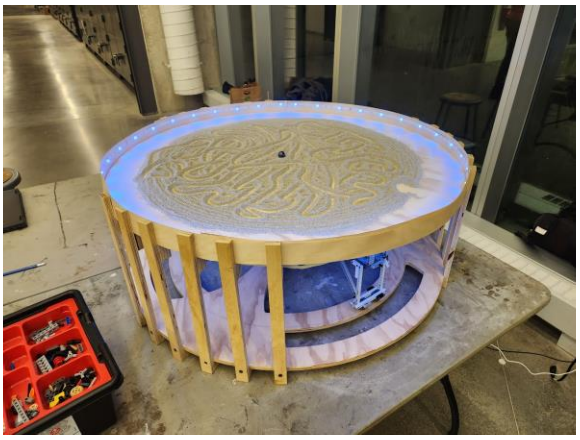


Figure 32: LEDs fully attached to both the underside and around the surface of the table to have ambient lighting.

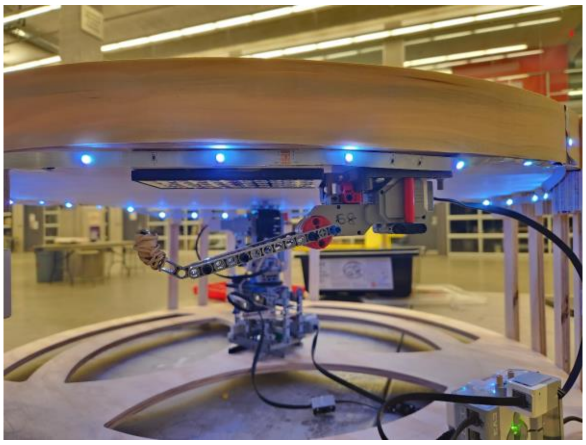


Figure 33: Second large motor used to flash the LEDs by pressing on the on/off button of the LED remote.

Integration was the hardest part of the whole project. Getting the robot, table, and code to all work together as one unit was a challenge. However, the system was working after a few days

The robot was originally mounted in place using zip ties (figure 28) so that the lining up of the wheels could be tested. The code was also assessed with no sand just to see if the magnet was strong enough. The robot had to be held in place since the zip ties were not very strong and it was found out that the magnet was in fact not strong enough to attract the ball bearing through the plywood. After this initial test new magnets were ordered, and a new mounting method was implemented.

The new mounting idea was to take advantage of the holes in the Lego pieces and screw into them, directly through to the wooden base (figure 29). The pieces were made sure to align so that when the robot was mounted, the table's center would line up with the rotational center of the robot. This mount worked beautifully and secured the robot in place very well.

Once mounted the robot could be assessed more thoroughly by running different patterns over and over to test endurance. The accuracy was also evaluated by running more complex patterns like the gear in figure 31. Throughout the testing process, the ev3 brick was secured to the side of the table in an accessible position along with the touch sensor. Cables were routed for the motors and made sure none of them would tangle by using extenders and zip ties where needed. The LED light strips were also attached as shown in figure 33 and their corresponding remote control and motor, which was mounted similarly to the base of the robot.

The entire integration process took about 3 days and testing was going well until a critical error appeared, gear skipping. Due to the high torque ratio of the pivot gear train the frame of the base would flex and cause the gears to come out of alignment, sometimes so much so that they would pop off. Despite multiple efforts, the base could not be made rigid enough to withstand the torque of the gears in its current state, a major redesign was not on the table. This meant that the gear train needed to be reinforced heavily. The gear train ended up being redesigned a total of six times in one day of testing.

This redesign required six major iterations to the main gear system leading to the completed product.

1. Problem: The first gear system had a free gear in multiple locations, the most important being the transition gear between the vertical shaft and the pivot/concentric gear for the conveyor arm.
    a. Solution: This was solved by adding an idler gear attached to the main frame of the cage and the gear driving the pivot.
2. Problem: After the first adjustment redirected the torque into a gear closer to the motor on the lower section of the mechanism. This gear is responsible for driving the axle that transfers power to the pivot gear.
    a. Solution: Adding a bar that prevents major flexing in the driving axle allowing gears to have full grip. An Idle gear was also added to the lower mechanism.
3. Problem: The now reinforced gear train near the motor allowed even more force to the pivot gear causing the gear launch out of the system instead of bending the axle.
    a. Solution: Made an extension cage surrounding the upper gear mechanism that kept the gears fastened to the gear cage.

4. Problem: Further skipping issues and inaccuracy caused by backlash and over-torquing of the system to the point of bending the entire frame.
    a. Multiple fixes: Disassembly and reassembly of the motor placement and gear system moved inside the gear cage rather than running along the bottom of the cage with very tight tolerances. The motor was now mounted with every possible peg in use as well as an unattached bracing bar on the outer side of the motor.
5. Problem: Main drive axle was made of two axles with a middle adapter between them causing shifting of the axle vertically.
    a. Solution: Since the gear train was internal the distance between motor and upper gear train was reduced allowing for one longer axle to cover the distance. To prevent further movement the axle was covered with collars completely bracing any visible component of the axle.
6. Problem: General looseness and twisting of the main frame:
    a. Solution: Rubber bands were attached to the upper cage and a large band was wrapped around the outer cage.

Overall, despite the arduous trials of integration hell the group were able to make it out with a completed sand table, which could create and destroy patterns at their command.

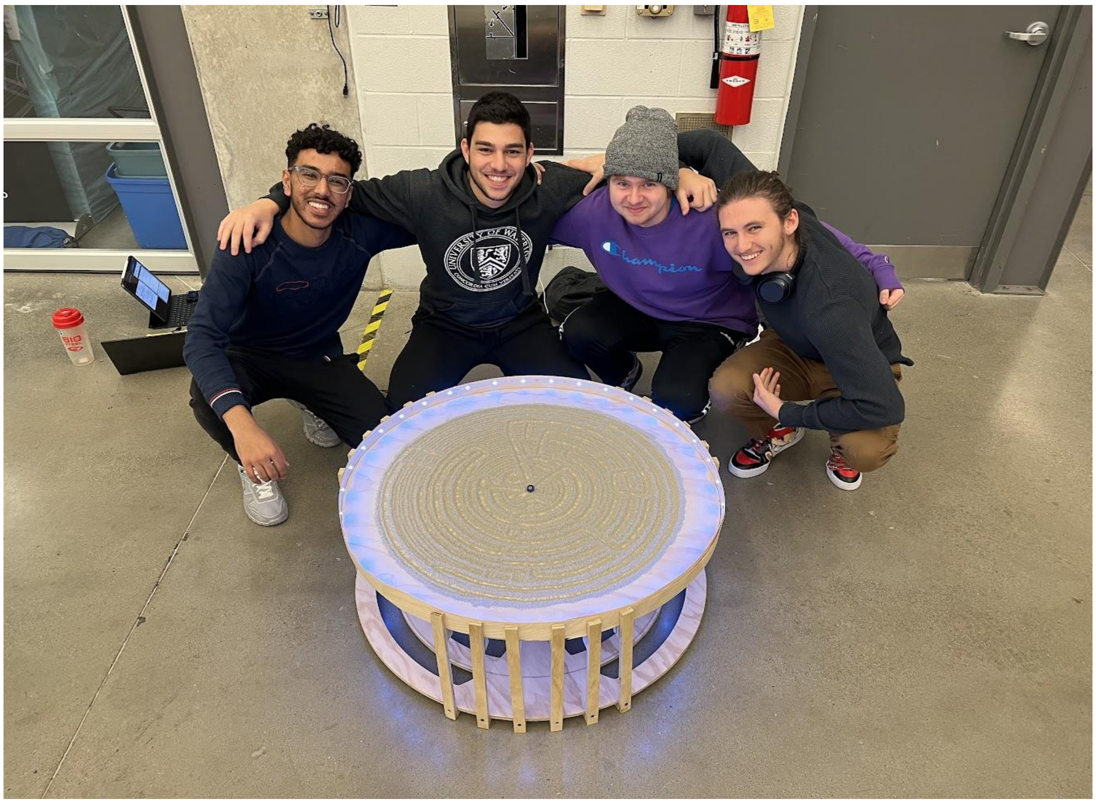


Figure 34: The final product.

Software Implementation

Overview

At a high level the software was broken into three main tasks: startup/pattern selection, drawing, and shutdown. A complete copy of the code has been included in appendix A.

1. The startup and pattern selection, here the robot waited for an ev3 button to be pressed and opened a file corresponding to which button was pressed. The robot afterwards waited for the press of the touch sensor to begin reading instructions.
2. After pattern selection the robot then needed to read the instructions written inside the text file that was selected, the layout of instructions will be explained more in depth later. The robot afterwards needed to move either the pivot or the belt of the robot to specific positions based on the instructions it read and repeat this for each instruction in the file.
3. The Robot lastly needed to be able to shut down instantly when the touch sensor was pressed during movement or while waiting for a pattern to be selected.

Task List

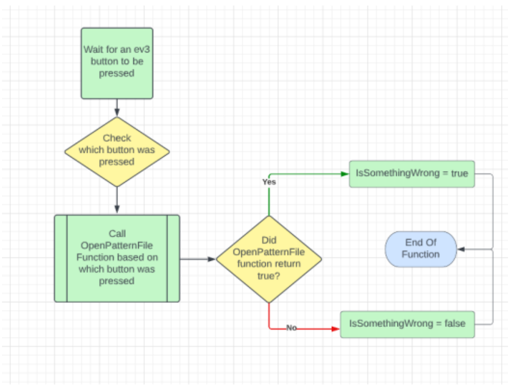
The following is a checklist that was used to assess each aspect of the robot when demoing, the same checklist was used when testing the functionality of the robot as well.

1. Starting with the magnet centered, run the code.
2. Successfully select a pattern.
3. Robot waits for the touch sensor to be pressed.
4. Robot reads file instructions and performs movements for Pokéball or gear patterns.
5. Robot resets to pattern select.
6. Robot begins drawing the clear pattern when selected just like the first pattern.
7. Robot emergency stops on touch sensor press.
8. Robot also exits the program if the touch sensor Is pressed during pattern selection.
9. During all movement’s robot displays information about movements on screen.

The main difference in this checklist from past ones is that the robot needed to display information about its movements on the screen as knowing what the robot was reading and trying to move always was extremely useful in debugging. As well, information such as what instruction out of how many the robot is reading or the time elapsed when drawing a pattern can be interesting to see and can give users an idea of how long until a pattern is completed.

Functions

SelectPattern - Griffin Wilson



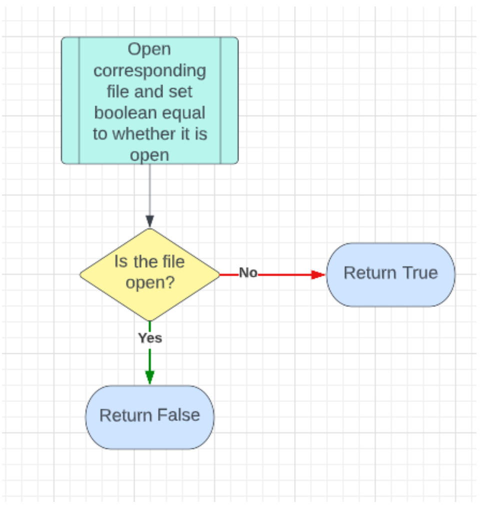
Wait for an ev3 button to be pressed
Check which button was pressed
Call OpenPatternFile Function based on which button was pressed
Did OpenPatternFile function return true?
Yes
No
IsSomethingWrong = true
IsSomethingWrong = false
End Of Function

Figure 35: Flowchart for SelectPattern function

The SelectPattern function's purpose is to manage opening the correct pattern instructions file based on which ev3 button was pressed.

The function begins by waiting for an ev3 button to be pressed by the user. It then uses a series of if/else statements to determine which button was pressed. Depending on which button was pressed PatternSelect calls an OpenPatternFile function. The OpenPatternFile function will return a Boolean based on whether the file opened properly, and it then sets the variable IsSomethingWrong to that value. The PatternSelect function receives the IsSomethingWrong variable as a parameter, and returns a Boolean based on if something went wrong.

OpenPatternFileX - Griffin Wilson



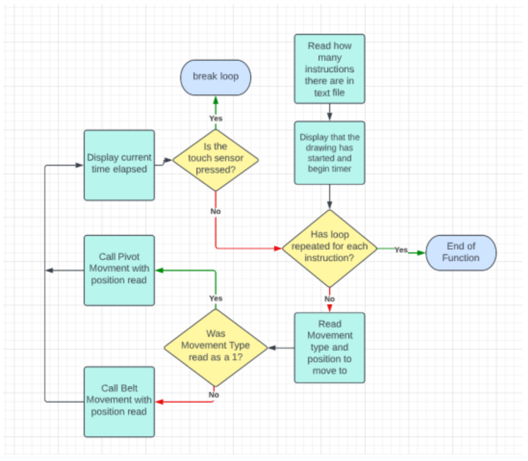
Open corresponding file and set boolean equal to whether it is open
Is the file open?
No
Return True
Yes
Return False

Figure 36: Flowchart for OpenPatternFileX function

OpenPatternFileX is a collection of similar functions which are called in SelectPattern depending on which ev3 button was pressed. Each OpenPatternFileX function's purpose is to open the file corresponding to itself and check that it opened properly.

The function returns false if the file is open as true and false here indicates whether something in the program has gone wrong.

ReadInstructions() - Griffin Wilson



Read how many instructions there are in text file
Display that the drawing has started and begin timer
Has loop repeated for each instruction?
Yes
End of Function
No
Read Movement type and position to move to
Was Movement Type read as a 1?
Yes
Call Pivot Movment with position read
No
Call Belt Movement with position read
Display current time elapsed
Is the touch sensor pressed?
Yes
break loop
No

Figure 37: Flowchart for ReadInstructions() function

The purpose of the ReadInstructions function is to read the data from the previously opened text file and call the proper movement functions based on what it reads. It also displays information useful in debugging on the ev3 screen as instructions are read and performed. It receives the current position of the belt and pivot as well as the variable to indicate if something has gone wrong as parameters.

The ReadInstructions function starts by reading how many instructions there are in the previously opened text file. After it enters a for loop for a number of iterations equal to the number of instructions it just read. It then for each instruction reads a one or a zero to indicate if the belt or pivot should be moved and a position for whichever is moving to move to. After reading an instruction the function calls the appropriate function to move the belt or pivot of the robot and displays the time elapsed drawing after each movement. Lastly the ReadInstructions function checks whether the movement function was ended by the touch sensor and will end the function by breaking the for loop if that is the case.

MoveBelt() - Oliver Chamoun

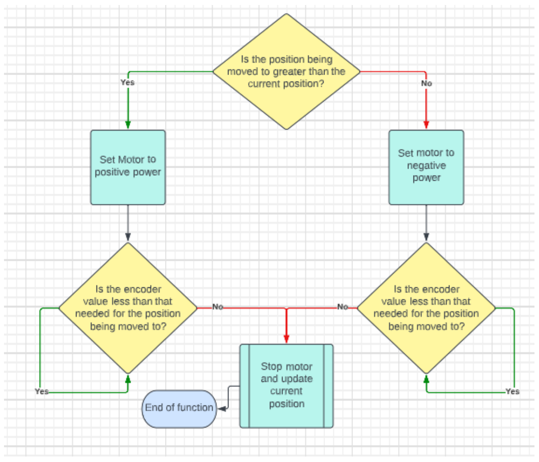

Is the position being moved to greater than the current position?
Yes
No
Set Motor to positive power
Set motor to negative power
Is the encoder value less than that needed for the position being moved to?
No
No
Is the encoder value less than that needed for the position being moved to?
Stop motor and update current position
Yes
Yes
End of function

Figure 38: Flowchart for MoveBelt function

The MoveBelt function's purpose was to move the magnet on the robot to a specific distance away from the center of the table. It received the current position of the belt as well as the position to move to as parameters. The parameter for the current position is passed by reference.

The MoveBelt function starts by comparing the position being moved to and the current position to determine in which direction the motor needs to run. It then sets the belt motor to a positive or negative power depending on which way it needs to turn, and it waits until the belt just reaches the correct encoder reading before stopping the motors. Lastly it stops the motor and updates the CurrentPosition variable based on the current encoder value.

MovePivot() - Oliver Chamoun

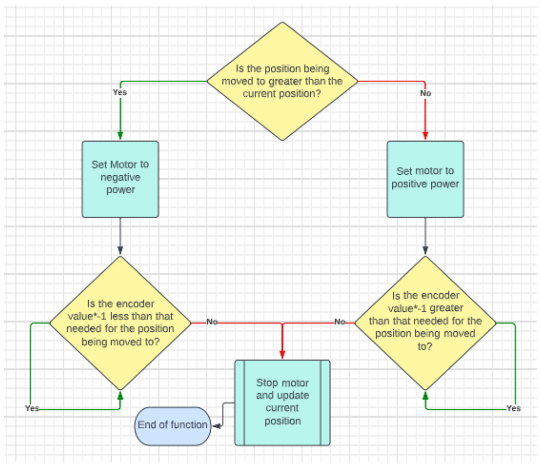

Is the position being moved to greater than the current position?
Yes
No
Set Motor to negative power
Set motor to positive power
Is the encoder value*-1 less than that needed for the position being moved to?
No
No
Is the encoder value*-1 greater than that needed for the position being moved to?
Stop motor and update current position
Yes
Yes
End of function

Figure 39: Flowchart for MovePivot function

The MovePivot function's purpose is to pivot the robot to a specified position in degrees. It receives the current position of the pivot and the position to move to as parameters.

The MovePivot function works identically to the MoveBelt function. The biggest difference between the two is their motors are set in opposite directions based on the comparison of the current position and the position being moved to, this is because the gear system on the robot reverses the direction that the motor turns because of there being an even number of gears. The other notable difference is that in the code they use different constants to convert distance to encoder values. Otherwise, the logic of both functions are identical.

Rave() - Ethan Catz

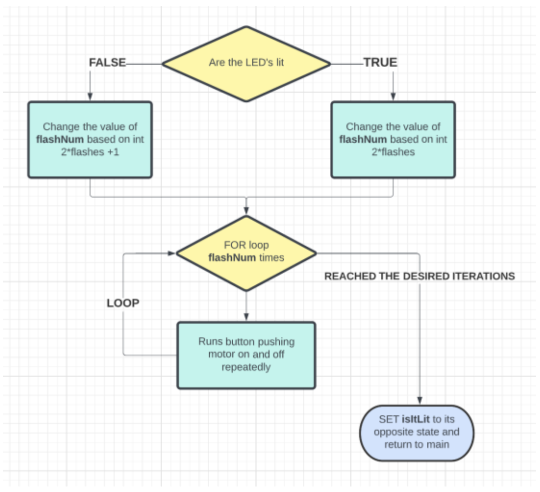

Are the LED's lit
FALSE
TRUE
Change the value of flashNum based on int 2*flashes +1
Change the value of flashNum based on int 2*flashes
FOR loop flashNum times
LOOP
REACHED THE DESIRED ITERATIONS
Runs button pushing motor on and off repeatedly
SET isItLit to its opposite state and return to main

Figure 40: Flowchart for rave function

This function is called when any design is completed on the table's surface. It drives the third motor to press a remote that controls the LEDs surrounding the whole table.

The function is void and receives a boolean storing if the lights of the table are on, and an int indicating how many flashes are desired for the light show.

Based on the isItLit variable, the function will set a flash number variable to the number of off-on cycles needed to flash the LEDs the desired number of times.

Once the program is completed the status of the LEDs is updated and returns to the main function.

Clear Table Instructions Writer - Kabir Raval (C++)

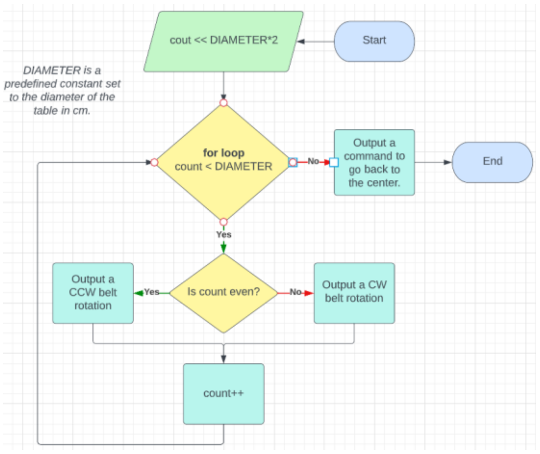

Start
cout << DIAMETER*2
DIAMETER is a predefined constant set to the diameter of the table in cm.
for loop
count < DIAMETER
No
Output a command to go back to the center.
End
Yes
Is count even?
Yes
Output a CCW belt rotation
No
Output a CW belt rotation
count++

Figure 41: Flowchart for Clear File Writer Function

The purpose of this function was to write an instruction file for the clear pattern more efficiently than by hand.

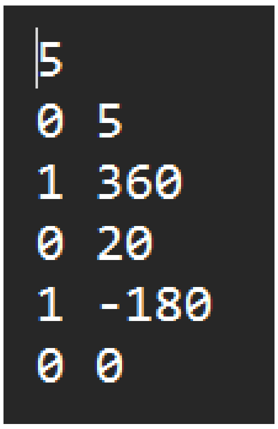
In this case, the file that this code outputs are the clear file, which instructs the robot arm to move out the edge of the table, move around the circle at that location and then move inward 1cm and repeat the circles. The robot does this thirty-six times, all the way to the center.

The function runs in a for loop and outputs a CW movement if the instruction number is odd and a CCW movement if the number is even. Since the robot cannot rotate more than 4-5 full rotations, the clear file makes sure that the robot only has to spin in one direction a maximum of 2 times.

Data Storage

The most important way data was stored in the robot was the way in which instructions for the robot were stored to be read. It was decided that instructions should be stored in a text file and read using fileIO. Storing data in a text file was more efficient as many patterns required over one hundred instructions so calling a function repeatedly with different values would not have been efficient, nor would populating an array. As well, a text file of instructions made it easy to write other programs to output larger repeating patterns or random patterns, examples of these programs are found in appendices B and C.

The text file was laid out as shown in figure 42, with the first line containing an integer representing the number of instructions in the file and each line after containing a one or a zero to represent which type of movement to make followed by a position to move to.



```
5
0 5
1 360
0 20
1 -180
0 0
```

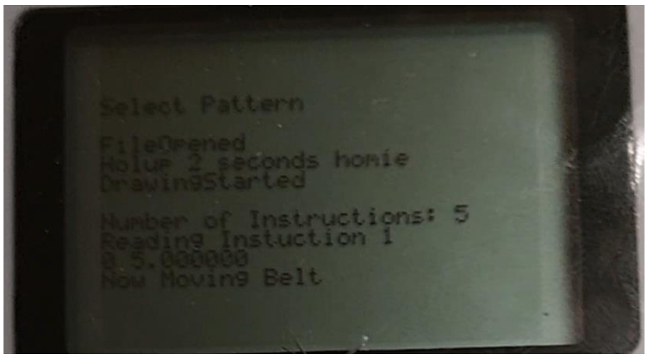
Figure 42: Example text file of instructions for the robot

The other data that was kept track of and modified was the positions of the belt and pivot. The belt and pivot positions were stored as floating points in task main and passed by reference to all functions that needed to modify them. The position variables were updated after every robot movement to account for if the robot over or undershoots the position it is instructed to move to.

Testing Procedure

To test the robot’s performance, different patterns were set to be drawn on the robot, and the robot's movements were tracked and compared to what the robot was instructed to using debug data on the ev3 brick’s screen and marks on the table.

The robot was coded to display many pieces of information on the ev3 screen to aid in debugging as it runs. Information it displayed included showing when a file had been opened, showing when a drawing was started, and what instructions it had most recently read from the instructions file. An image showing some of the information the robot displayed as it ran is below in figure 43.



```
Select Pattern

FileOpened
Holup 2 seconds homie
DrawingStarted

Number of Instructions: 5
Reading Instuction 1
0 5.000000
Now Moving Belt
```

Figure 43: EV3 screen with debug information

This debug information was useful at one point for example when the robot was not moving the belt when it should have based on the instructions given. The information on the debug screen was able to communicate that the robot was reaching the code where it should be moving the belt, and it confirmed that the robot was reading the instructions correctly.

The other way the code was tested was by marking on the table the points the robot should move to for a short and simple pattern, and then comparing where the robot moved to those points. This was useful in measuring the extent gear backlash affected the pivot when changing directions as well as confirmed that the constants that were calculated to convert encoder values to a distance, and vice-versa, were accurate. Later in testing these marks were replaced by actually being able to see the designs the robot made in the sand and comparing them to what they should look like.

One significant problem these testing methods helped overcome was when at one point the robot would not move the belt inwards to a point that was not zero. Using the debug screen, it was determined that the code was calling the proper function to move the belt in the instances it was not working, as well as confirmed the robot was reading the correct instructions. Knowing the instructions were being read properly and the proper function was being called it became a lot easier to determine that the reason the robot was not moving inwards was due to a constant dividing integer improperly and a result evaluating to 0.

Verification

The group knew they had met their constraints when the table was able to be operated near an outlet for long periods of time and without fear of a power cable coming loose. Also, the constraint of having a limited amount of rotations for the robot was dealt with using smarter software, which would never push the robot beyond 4-5 full rotations, and using extension cable.

Project Plan

Work on the robot was split between the group as follows.

- All group members worked equally on the planning, prototyping, and building of the robot's mechanical system.
- Griffin and Oliver primarily contributed to the robot's code, with Ethan and Kabir still writing the functions they were required to write.
- Kabir and Ethan primarily contributed to the construction of the table for the robot.
- All group members worked equally on tuning the robot in all its aspects and integrating the various elements of the project together.

The following is the original schedule that was set to complete all tasks.

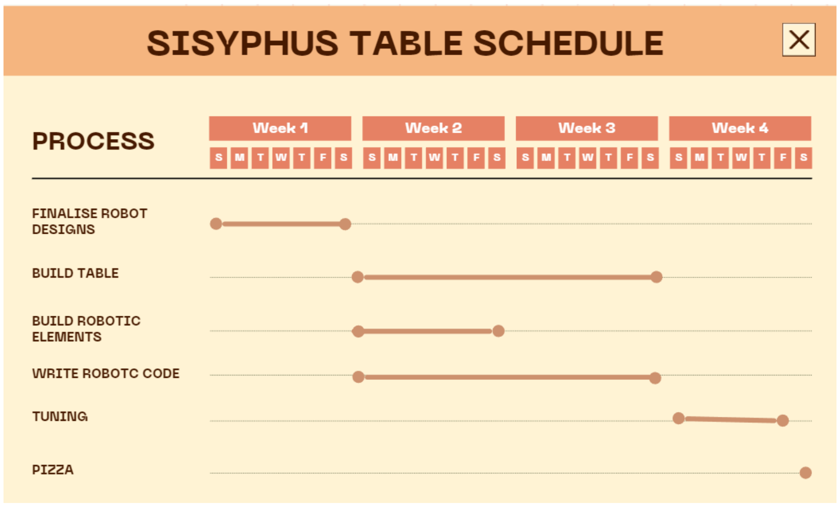


Figure 44: Original project schedule in the form of a gantt chart.

Overall, the original schedule was stuck to very closely. There were two main differences between the original schedule and when things were completed. First, the mechanical build of the robot was completed several days early compared to what was planned. Second, the RobotC code was completed two days late due to problems with integrating FileIO. However, the scheduled plan was stuck to very closely.

Conclusion

Overall, the robot was able to effectively solve the problem of the POETS lounge needing an intriguing and dynamic centerpiece. It solved the problem by being a table which passively drew interesting designs in sand under the tabletop. By doing so it becomes a conversation starter and something for people to gravitate around, as well it matches POETS by being a student-built work of engineering.

The robot drew patterns in the sand by moving a magnet under the sand bed of the table, which in turn would move a bearing through the sand and create lines. The robot moved the magnet by turning a belt which the magnet sat on to move it linearly along the radius of the circular table. The robot also pivoted the belt around the center of the table to move the magnet in circular patterns.

The original criteria for the robot were that it could draw patterns in sand using the individual movements of the belt and pivot, and that it needed to be relatively quiet so as to not be disruptive to the people around it. It met these criteria very well and the groundwork was laid for it to be able to draw more complex patterns with the belt and pivot moving in tandem. The main constraint of the robot was its constant need for power as it was designed to run infinitely until it needed to be shut off, this meant the robot needed to always sit near a power outlet. This constraint only limited where the table could be put in POETS and had insignificant effect on the robot’s actual design.

Recommendations

Table Design Changes

Substitute Veneer

One aspect of our design that was worse than expected was the veneer. It presented many problems including difficulty mounting LEDs and most importantly, keeping a tight fit around the edge of the table as well as constantly chipping. The replacement would consist of quarter circle ribs cut from plywood glued together in a brick-like fashion to create a ring that could be machined to have gaps for lighting and the support legs for the table.

Groove for LED

A section of the table surface should have a machined gap to allow a sealed edge to keep sand in from all sides while still allowing discrete implementation of the LED strips.

Leveling Studs

The process used to build the feet of the table worked but had some legs not bearing any weight since they hovered above the ground. This was caused by two factors; the bandsaw making inconsistent cuts, and the gluing process being offset when clamping the legs to the corresponding stud. Solving this problem would require clamping the legs into a brick and sanding them together as one object. Testing with a level once the bottom and top disks are mounted for inspection would also improve accuracy.

Sound Dampening

Adding a coat of paint below the sand as well as adding felt under the path of the conveyor would significantly improve noise pollution of the system.

Robot Design Changes

3D Printed Gears

In short, to reduce backlash and precisely adjust our gear ratio. This would be accomplished the best by getting rid of the current gear structure and replacing it with a single worm gear to the pivot gear with our desired ratio.

Double Motor Power

The group noticed that the central pivot tended to have a challenging time rotating the entire arm assembly and even with the gear train the motor was definitely being strained. Implementing a dual motor drive, as in having a driven gear be powered by two motors in tandem would help solve some of these issues. The power could be set to ⅔ on both motors to reduce the overall load on them both while still providing ample torque.

Protruding Legs to Mount Through the Table

The current mounting system relies on the set of Lego rails being screwed into the plywood. Although this design works well as is, the rails were still shaky in their mounts and this is due to the screws not being fully tightened, to avoid the risk of fracking the Lego. A practical alternative would be drill holes for the base and have Lego legs that protrude from the underside that could “slot” into the holes. A precisely drilled hole would allow for a strong friction fit. If that was not enough, then a sacrificial part could be glued into the hole using cyanoacrylate glue and then later removed with pliers or drilled out. These design changes would help better secure the base to the table.

Tighter Conveyor Belt Retention

Making the conveyor belt tighter and better held to the arm assembly would result in the magnet being more accurately able to control the ball bearing. This can be accomplished by implementing the following things. First, an additional rail system can be added on top of the belt, pinning it down on the sides of the belt to not interfere with the magnet. Another idea was to add a higher resolution adjustment system for the tensioner gear which would allow for finer control over the tension.

Software Design Changes

New Coordinate System

The current coordinate system by nature cannot move the conveyor and the radial arm in one motion meaning that coordinated movement between the two is physically impossible and non-radial shapes also impossible to draw. The system operates by moving the radial arm by a set number of degrees and the conveyor by a set number of centimeters (refer to Figure 13.). A new coordinate system would receive polar coordinates of the format, (distance, angle) allowing for simultaneous reading of length and angle opening possibilities for continuous synchronous movement.

Point to Point Structure`

Given two points from a file formatted with the new coordinate system proposed in the previous recommendation, using the following formulas polar coordinates can be converted to cartesian coordinates.

$$x = r * \cos(\theta)$$
$$y = r * \sin(\theta)$$

Figure 45: Math

Using these values, the equation of a line between two polar coordinates can be derived resulting in Figure 46.

$$Let\ point\ A\ and\ B\ be\ defined\ by\ the\ coordinates\ A(x_1, y_1),\ and\ B(x_2, y_2)$$

$$q = y_2 - y_1 \text{ , } p = x_2 - x_1 \text{ , } s = y_1 - \frac{qx_1}{p}$$
$$y = mx + b \quad \Longrightarrow \quad y = \frac{q}{p}x + s$$
$$y = \frac{y_2 - y_1}{x_2 - x_1}x + y_1 - \frac{(y_2 - y_1)x_1}{x_2 - x_1}$$

Figure 46: Math

The Motor Power Equation

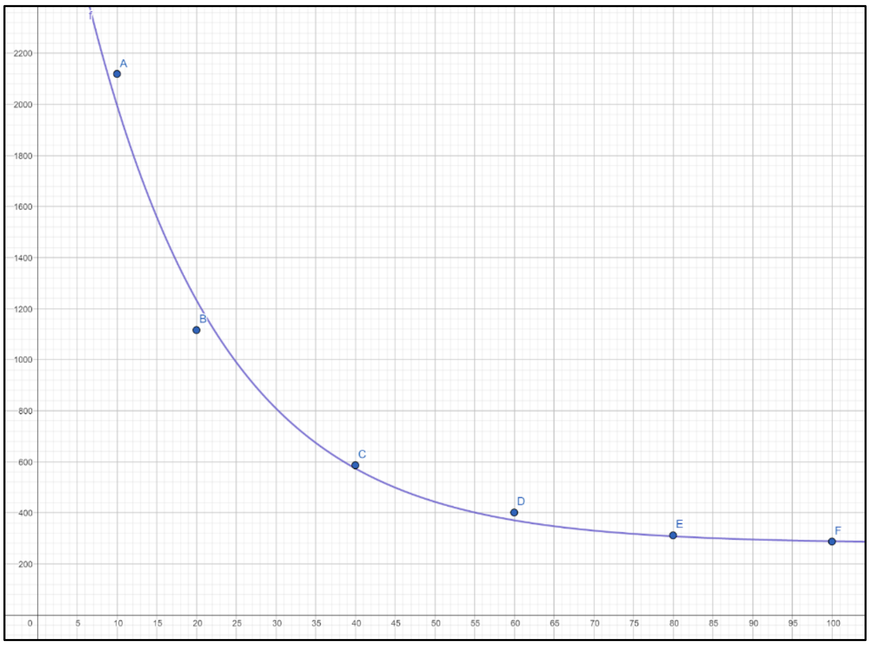


Figure 47: Graph of milliseconds/revolution vs. integer of motor power

Before implementing Figure 47, there first must be a relationship between the motor power integer provided to a motor and the speed needed to make a continuous movement with both motors. This is where the “motor power equation” comes into play. It was derived by testing the motor with assumed full power since our EV3 is built to always be plugged into a wall socket. After the data points were determined they were plotted on a graph displaying the apparent arrangement of an exponential decay. Figure 47 shows the line and data points collected. This equation is accurate within +/-40ms.

The Simulation Graph

This graph was created to assess whether the geometric quotations from Figures 45, and 46 would work relative to the table. The two extra lines are to represent what the arm of the robot would look like at each end of each path selected by the sliders which control the position of the points within the table.

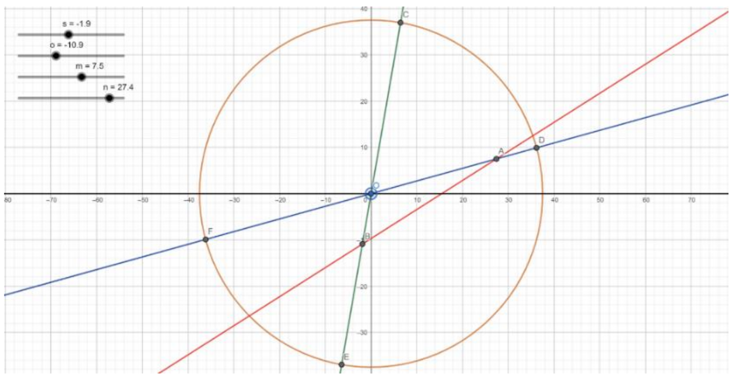


Figure 48: Simulation Graph

Potential Methods of Producing a Drawing From the Equations

There would be many ways to implement this into the code but the simplest is to have a function that generates many points along the generated line and makes many small operations to keep accuracy of the motor equation from dissociating too far. The implementation would use the slope to determine the ratio of speeds between the arm and conveyor so the motors would move at the exact right rate to create a non-radial straight line. Other equations can also be derived for other shapes if desired using a similar method.
The second method would be to use the current file IO coordinate system and move angularly then radially between the many points generated by the graphing program giving the illusion of a straight line in the sand due to the ball bearing’s natural shape.

References

[1] S. Industries, "Sisyphus Industries," 2022. [Online]. Available: https://sisyphus-industries.com/product/wood-table/} . [Accessed 06 12 2022].

Appendix A - Main Robotc Code

```
#include "PC_FileIO.c"

void MoveBelt(float Position,float &CurrentBeltPosition);
void MovePivot(float Position,float &CurrentPivotPosition);
void Clear();
void SelectPattern(bool &IsSomethingWrong);
bool OpenPatternFileOne();
bool OpenPatternFileTwo();
bool OpenPatternFileThree();
bool OpenPatternFileFour();
bool OpenPatternFileFive();
Void          ReadInstructions(float          &CurrentBeltPosition,float
&CurrentPivotPosition,bool &IsSomethingWrong);
void rave(bool &isItLit, int flashes);

TFileHandle FILE_IN;
//global variable used as
//TFileHandle cant be initialised with a value
const float DEGREES_TO_ENCODER = 7.5;
const float ENCODER_TO_DEGREES = (1/7.5);
const float CM_TO_ENCODER = 30;
const float ENCODER_TO_CM = (1.0/30.0);

task main()
{
      bool isItLit = true;

      for(;;)
      {
      float CurrentBeltPosition = 0;
      float CurrentPivotPosition = 0;
      displayString(3,"Select Pattern");
      bool IsSomethingWrong = false;

      SelectPattern(IsSomethingWrong);
      if(IsSomethingWrong == true)
```

```
			break;

		displayString(5,"FileOpened");

		while(SensorValue[S1]==0)
			{}
		displayString(6,"Holup 2 seconds homie");
		wait1Msec(2000);

		ReadInstructions(CurrentBeltPosition,CurrentPivotPosition,IsSomethin
gWrong);
		if(IsSomethingWrong == true)
			break;

		rave(isItLit,3);

		eraseDisplay();
		}
}

void SelectPattern(bool &IsSomethingWrong)
{

while  (!getButtonPress(buttonEnter)  &&  !getButtonPress(buttonLeft)  &&
!getButtonPress(buttonRight)        &&!getButtonPress(buttonUp)        &&
!getButtonPress(buttonDown) && SensorValue[S1]!=1)
		{}

		if (getButtonPress(buttonEnter))
			IsSomethingWrong=OpenPatternFileOne();

		else if (getButtonPress(buttonLeft))
			IsSomethingWrong=OpenPatternFileTwo();


		else if (getButtonPress(buttonRight))
			IsSomethingWrong=OpenPatternFileThree();

		else if (getButtonPress(buttonUp))
			IsSomethingWrong=OpenPatternFileFour();

		else if (getButtonPress(buttonDown))
			IsSomethingWrong=OpenPatternFileFive();

		else
			IsSomethingWrong=true;
}
```

```
bool OpenPatternFileOne()
{
      bool fileOkay = openReadPC(FILE_IN,"1.txt");

      if (!fileOkay)
      {
            return true;
      }
      return false;
}

bool OpenPatternFileTwo()
{
      bool fileOkay = openReadPC(FILE_IN,"2.txt");

      if (!fileOkay)
      {
            return true;
      }
      return false;
}

bool OpenPatternFileThree()
{
      bool fileOkay = openReadPC(FILE_IN,"3.txt");

      if (!fileOkay)
      {
            return true;
      }
      return false;
}

bool OpenPatternFileFour()
{
      bool fileOkay = openReadPC(FILE_IN,"4.txt");

      if (!fileOkay)
      {
            return true;
      }
      return false;
}

bool OpenPatternFileFive()
{
	bool fileOkay = openReadPC(FILE_IN,"5.txt");

	if (!fileOkay)
	{
		return true;
	}
	return false;
}

void		ReadInstructions(float		&CurrentBeltPosition,float
&CurrentPivotPosition,bool &IsSomethingWrong)
{
	int LinesInFile = 0;


	nMotorEncoder[motorA]=nMotorEncoder[motorD]=0;

	readIntPC(FILE_IN,LinesInFile);
	displayString(7,"DrawingStarted");

	clearTimer(T1);
	int TimeElapsed = 0;

	for(int Lines=0;Lines<LinesInFile;Lines++)
	{
		int Pivot = 0;
		float Position = 0;


		displayString(9,"Number of Instructions: %d",LinesInFile);
		displayString(10,"Reading Instuction %d",(Lines+1));
		wait1Msec(500);
		readIntPC(FILE_IN,Pivot);
		readFloatPC(FILE_IN,Position);
		displayString(11,"%d %f",Pivot,Position);
		wait1Msec(250);
		if(Pivot==1)
			MovePivot(Position,CurrentPivotPosition);

		else if(Pivot==0)
			MoveBelt(Position,CurrentBeltPosition);

		TimeElapsed = time100[T1];
		displayString(13,"Time Elapsed: %f",(TimeElapsed)/10.0);

		if(SensorValue[S1]==1)
		{
```

```
            IsSomethingWrong = true;
            break;
            }
      }
}

void MovePivot(float Position,float &CurrentPivotPosition)
{
      displayString(12,"Now Moving Pivot");
      wait1Msec(1000);
      if(Position > CurrentPivotPosition)
      {
            motor[motorA]=-20;
            while(nMotorEncoder[motorA]*-
1<Position*DEGREES_TO_ENCODER&&SensorValue[S1]==0)
            {}
            motor[motorA]=0;
      }
      else
      {
            motor[motorA]=20;
            while(nMotorEncoder[motorA]*-
1>Position*DEGREES_TO_ENCODER&&SensorValue[S1]==0)
            {}
            motor[motorA]=0;
      }
      CurrentPivotPosition = nMotorEncoder[motorA]*(-1*ENCODER_TO_DEGREES);
}

void MoveBelt(float Position,float &CurrentBeltPosition)
{
      displayString(12,"Now Moving Belt ");
      wait1Msec(1000);
      if(Position > CurrentBeltPosition)
      {
            motor[motorD]=20;

      while(nMotorEncoder[motorD]<Position*CM_TO_ENCODER&&SensorValue[S1]=
=0)
            {}
            motor[motorD]=0;
      }
      else
      {
            motor[motorD]=-20;
```

```
	while(nMotorEncoder[motorD]>Position*CM_TO_ENCODER&&SensorValue[S1]=
=0)
		{}
		motor[motorD]=0;
	}
	CurrentBeltPosition = nMotorEncoder[motorD]*(ENCODER_TO_CM);
}

void Clear (float Position, float & CurrentBeltPosition)
{
//kabir wrote a c++ program to write a pattern file for this
}

void rave(bool &isItLit, int flashes)
{
	int flashNum = 0;

	if(isItLit == true)
	{
		flashNum = 2*flashes;
	}
	else
	{
		flashNum = 2*flashes+1;
	}

	for(int count=0; count<flashNum; count++)
	{
		motor[motorC] = -30;
		wait1Msec(500);
		motor[motorC] = 0;
		wait1Msec(50);
		motor[motorC]=30;
		wait1Msec(100);
		motor[motorC]=0;
		wait1Msec(100);
	}

	isItLit = !isItLit;
}
```

Appendix B - Clear File Instruction Writer

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <iomanip>

using namespace std;

const int DIAMETER = 36;

int main()
{
	cout << "DIAMETER*2" << endl;
	for(int i=DIAMETER; i>0; i--)
	{
		if((i%2)==0) //if even
		{
			cout << "0 " << i << endl;
			cout << "1 360" << endl;
		}
		else if((i%2)==1) // if odd
		{
			cout << "0 " << i << endl;
			cout << "1 -360" << endl;
		}
	}
	cout << "0 0" << endl;
}
```

Appendix C - Random File Instruction Writer (Oliver Chamoun)

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main ()

{
	cout <<"101"<<endl;
	for(int i=0;i<100;i++)
	{
		if((i%2)==0)
		{
			cout <<"0 ";
			int maxBelt=36;
			int minBelt=0;
			int randBelt= rand()%(maxBelt-minBelt+1)+minBelt;
			cout<<randBelt<<endl;
		}
		else if ((i%2)==1)
		{
			cout <<"1 ";
			int maxPivot=360;
			int minPivot=0;
			int randPivot=rand()%(maxPivot-minPivot+1)+minPivot;
			cout<<randPivot<<endl;
		}

	}
	cout <<"0 0"<<endl;

	return 0;
}
```

Appendix D – Details, Calcuclations, and Simulations

Figure 47: https://www.geogebra.org/m/pe5vuabr
Figure 48: https://www.geogebra.org/m/uqpxcht5