Department of Mechanical and Mechatronics Engineering

SketchBot Report

A Report Prepared For:
MTE 100 & MTE 121

Prepared By:
Nour Abdelrahman (21122136)
Daniel Mehany (21123991)
Noah Harman (21148698)
Youssef Hamadache (21146832)

November 28, 2024

Summary

This report details the development of an automated device designed to facilitate the creation of engineering sketches, particularly for users with physical limitations. It highlights the robot's primary functionality of drawing proportional squares or rectangles with precise dimensions and spacing, achieved through a combination of mechanical components and user-friendly software inputs. The design process involved addressing various constraints, including programming limitations and mechanical challenges, which led to iterative improvements in both the robot's structure and functionality. Ultimately, SketchBot demonstrates a successful integration of safety features, such as an emergency stop button, while ensuring reliability and accuracy in sketching tasks, making it a valuable tool for engineering applications.

Acknowledgements

N/A

Table of Contents

Table of Figures

Table of Tables

Section 1: Introduction

1.1

While engineering sketches remain a crucial part of the design process, they can be tedious and time-consuming. Within the group's studies, the group learned that every sketch begins with a proportional square or cube, from which the rest of the drawing is constructed. The entire sketch will be off if the initial square or cube is disproportionate. The idea of creating a robot to start sketches began when one of the group members dislocated his shoulder and could not complete his drawing. As engineers, the group thought to themselves, "there must be a way to automate the process for him" and further automate the beginning of all engineering sketches by having a robot draw the initial square or cube, ensuring the sketch would be proportional.

Section 2: Scope

2.1 Main Functionality

SketchBot is an automated device designed to simplify the creation of engineering sketches, particularly for individuals unable to draw manually due to physical limitations or personal preferences. Its primary functionality is to generate up to four equally spaced squares or rectangles with uniform dimensions. The robot achieves this precision by integrating mechanical and coding elements to replicate smooth, consistent human-like movements.

SketchBot processes user inputs such as the number of shapes to draw, their dimensions, and spacing. These parameters are fed into the system via buttons and file inputs. The device is equipped with motorized components that control movement along the X and Y axes, and a pencil holster for drawing. Additionally, it incorporates features like an emergency stop button to enhance safety and reliability. By automating the sketching process, SketchBot not only ensures consistency and proportionality but also eliminates errors that might arise from manual drawing, making it a valuable tool for various engineering applications.

2.2 Inputs

SketchBot employs four distinct inputs to execute its task effectively. Each input serves a specific purpose, contributing to the robot's precision, user control, and safety. Together, these inputs ensure that SketchBot remains both functional and reliable for users in various engineering scenarios.

I. **File IO**

SketchBot incorporates file input as a critical method for obtaining dimensions and spacing information. Through this feature, the user can upload a file containing the necessary parameters: the length and width of the rectangles and the (x, y) separation distances between them. These values are stored as variables within the robot’s system, providing the foundation for its operations. This approach not only automates the setup process but also minimizes potential

errors in manual data entry. By integrating this method, SketchBot ensures precision and flexibility, making it well-suited for a variety of engineering tasks.

II. **Touch Sensor**

Safety and user control are paramount in SketchBot's design, and the touch sensor plays a key role in achieving this. This sensor functions as an emergency stop mechanism, allowing the user to halt the robot's operations immediately if an issue arises. For instance, if the dimensions stored in the input file are incorrect or inconsistent, the touch sensor can be used to safely interrupt the process. Additionally, it serves as a simple way to power down the device when not in use. This feature ensures that SketchBot operates within a controlled environment, prioritizing both the user's safety and the system's integrity.

III. **Buttons**

The SketchBot system integrates the four directional buttons of the EV3 console to streamline user interaction. These buttons allow the user to determine how many rectangles the robot will draw during its operation. Each button corresponds to a specific quantity, ranging from one to four rectangles. This intuitive input method simplifies the user experience, ensuring that even those unfamiliar with coding or robotics can easily operate the device. The button-based control system adds a layer of accessibility, making SketchBot a practical tool for a wide audience.

IV. **Motor Encoder**

To maintain precision and consistency in its movements, SketchBot relies on motor encoders. These components track the rotational movement of the robot's motors, ensuring accurate translation of distance and positioning. By continuously monitoring how far the motors or wheels have travelled, the encoders play a crucial role in executing the correct dimensions and separation distances for the rectangles. This feature enables SketchBot to produce high-quality sketches that remain consistent across multiple iterations, emphasizing the robot's reliability and accuracy.

By integrating these four inputs—file I/O, a touch sensor, buttons, and motor encoders—SketchBot achieves a seamless balance of user control, safety, and precision, solidifying its role as an innovative tool for automated engineering sketches.

2.3 Motors

SketchBot uses four motors to execute its tasks with precision and stability. Two motors control movement along the Y-axis, positioned on either side of the robot's base to ensure balanced and smooth vertical motion. The remaining two motors manage operations on the X-axis: one drives horizontal movement to define the rectangle's length, while the other raises and lowers the drawing arm for accurate line placement. This motor configuration enables SketchBot to produce clean, consistent engineering sketches with efficient and coordinated motion.

2.4 Task Completion and Shutdown

SketchBot provides a user-friendly interface through its EV3 screen, displaying essential information throughout its operation. At the start, it shows the dimensions of each rectangle, including the length, width, and spacing between them, offering users a clear understanding of the parameters before the sketching begins. During the drawing process, the screen keeps users informed about its progress, indicating how many rectangles it is set to draw, how many have been completed, and how many remain. Once the task is finished, SketchBot notifies the user that the drawing is complete, ensuring transparency and ease of use.

Safety and convenience are prioritised in SketchBot’s functionality. It features an emergency stop button that halts the operation immediately, allowing users to intervene in case of errors or unexpected issues. When the emergency stop is pressed or after completing its task, SketchBot automatically returns to its original starting position, ready for subsequent use. To conserve energy and streamline its operation, the robot shuts off automatically 10 seconds after reaching its starting position, whether the emergency stop was engaged or the drawing task concluded successfully.

2.5 Changes

To enhance the design and functionality of the SketchBot, the group initially remodelled its original structure, as the initial design failed to meet the project requirements. Instead of simply adding parts to the existing EV3 robot, the group opted for a complete redesign to ensure a more effective and adaptable solution. Another key improvement was transforming the dimensions of the rectangles from fixed constants to variables, enabling users to create a wider range of rectangle sizes. This flexibility enhanced the robot's versatility for various applications.

Additionally, the group iteratively redesigned the pencil holster multiple times to address issues with precision. The previous designs didn't allow the robot to consistently draw accurate rectangles or produce straight, parallel, and perpendicular lines. The group refined the holster to ensure the pencil was securely held, contributing to better drawing accuracy.

To reduce weight and improve simplicity, the group detached the EV3 console from the robot's main body. This modification minimized the number of components, streamlining the design and reducing structural complexity. Lastly, the group incorporated a Tetrix Kit to create a sturdy border around the robot and a custom mount for the EV3 console. This provided additional stability and ensured that the console was securely attached while maintaining the overall integrity of the design.

Section 3: Constraints and Criteria

3.1 General Requirements

Our robot required the following general requirements:

- A substantial mechanical re-design of the EV3 robot
- The use of at least 3 motors one of which must have controlled movement
- The use of at least 4 distinct types of inputs from the following list:
- The use of timers (e.g. time1[T1], time1[T2])
- Repetition (loops)
- Decisions (if statements)

- At least six non-trivial functions, not including main, that use appropriate parameter passing.
    - At least one function must return something.
    - At least one function must have parameters (passed by value and/or by reference)

3.2 SketchBot Specific Criteria

After working on the robot and discovering flaws and issues throughout the process, the group came to a finalized list of criteria that the robot had to satisfy for it to complete its task:

- SketchBot draws lines of correct lengths based on the dimensions given.
- SketchBot creates proportional rectangles (perpendicular/parallel lines).
- SketchBot can create consistent and accurate rectangles.
- SketchBot must create straight and continuous lines.
- SketchBot provides correct spacing between each drawing (if specified).
- SketchBot immediately stops if the emergency stop button is pushed.

Through iterative improvements, the group refined SketchBot to draw precise and proportional rectangles with accurate dimensions and spacing. Its ability to consistently produce straight and continuous lines, alongside a reliable emergency stop feature, underscores its functionality and safety. These efforts culminated in a robot capable of addressing the challenges of automated engineering sketches while meeting the specific requirements set for the project.

3.3 Constraints

When examining the available resources for the project, the group identified several constraints that shaped the design and implementation decisions. The Lego EV3 robot the group was required to use was limited to running the Robot C programming language, which constrained us to simpler code than the group might have preferred, thereby narrowing the scope of the project. Recognizing this, the group focused on prioritizing essential software functions to align with the vision for the robot's capabilities.

Mechanically, the design faced similar limitations. Initially the group considered implementing a two-axis system on a base plate, akin to the movement seen in 3D printers. However, this idea proved impractical due to the restricted Lego pieces available. The design would have required a compact motor capable of traversing along a double-sided gear rack to control x- and y-axis movements. Unfortunately, the size and power of the Lego motors were insufficient to meet these requirements. Moreover, additional constraints included working only within the whiteboard area, restricting dimensions to specified parameters, and limiting the design to no more than four squares or rectangles.

The speed limitations of the robot added another layer of complexity. The system had to operate within these performance constraints, influencing decisions involving movement mechanics and functionality.

Perhaps the most significant constraint was time, manifesting in two ways. The project deadline restricted how much could be iterated and refined, requiring a balance between innovation and practicality. Additionally, the group's busy schedules made it challenging to dedicate extensive time to the project. These time constraints left us with little room to develop a fully optimized solution. Nonetheless, the group adapted to these limitations by focusing on achievable goals and efficient decision-making throughout the project.

3.4 Changes to Criteria Based on Constraints

One change in the group's constraints came in the form of a mechanical development. The group lacked a gear rack, so the robot was initially designed without the use of one. Later, the group requested a spare 3-D printed gear rack from a previous group's project.

Criteria	**Constraints**
Can draw anywhere	Only within the whiteboard area
Can draw any number of squares the user inputs	Can't draw more than 4 squares
Draws the dimensions the user needs	Dimensions must be within the parameters
Can draw shapes	Must be squares or rectangles
Consistency and accuracy	Limited by its speed

Table 3.4 - Comparison of Criteria and Constraints

The comparison between the criteria and constraints highlights the necessary adjustments to the plan. While the group initially aimed for flexibility in drawing anywhere and any number of shapes, the group was limited to working within a whiteboard area and drawing no more than four squares. Similarly, the desire for customizable dimensions and diverse shapes was narrowed to parameters allowing only squares or rectangles. Furthermore, while the group prioritized consistency and accuracy, the robot's speed imposed practical limitations on precision. These constraints required us to change design.

Section 4: Mechanical Design and Implementation

4.1 The Completed Design

The SketchBot is a simple yet effective drawing device. It uses two axes, X and Y, to move left, right, forward, and backwards. A pencil mechanism raises and lowers for drawing, supported by a sturdy frame with an EV3 brick holder. As well as a whiteboard below acts as reusable paper with clear markers for the middle and start of the page.

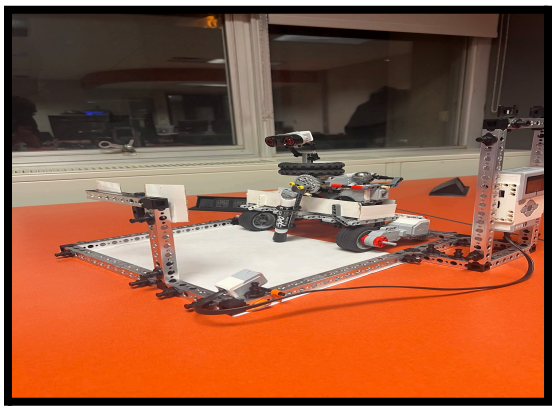


Figure 4.1.1 - Isometric view of robot assembly

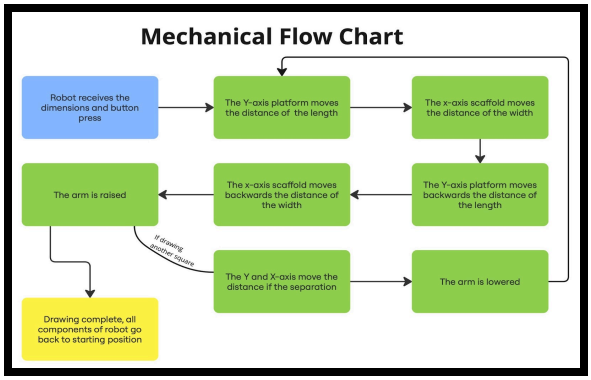


Figure 4.1.2 - Flowchart of mechanical design

4.2 The Y-axis Platform

The Y-axis of the structure is constructed from a large platform that serves as the foundation for the X-axis positioned above it. This platform is powered by two large motors, one on each side, a stable and controlled movement. To ensure smooth movements, each motor is supported by a caster ball positioned behind it, which helps in evenly distributing the weight and preventing any tilting. The platform is securely guided by a metal frame or railing, which ensures that it remains aligned and does not drift laterally during the performance. At the centre, between the two motors, the platform is built from multiple long pieces of Lego bricks, assembled to create a robust and sturdy surface. This sturdy design is essential, as the platform is engineered to hold significant loads, as it's essential for the rest of the robot

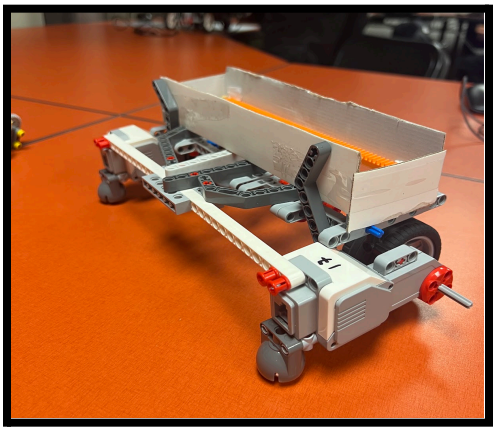


Figure 4.2.1 - Rear view of the Y-axis platform

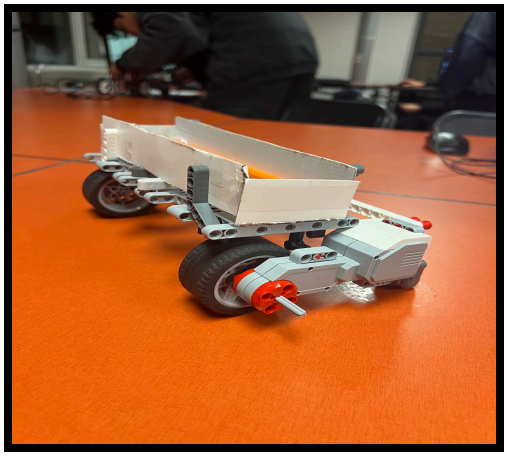


Figure 4.2.2 - Front isometric view of the Y-axis platform

4.3 The X-axis Scaffold

The x-axis scaffold is a smaller bot designed to sit atop the y-axis platform, enabling horizontal movement along the left and right directions. This scaffold is powered by a single motor that operates four gears, which interact seamlessly with two long racks to ensure smooth and precise motion along the x-axis. To prevent the bot from falling off or becoming dislodged during operation, it is enclosed by a lightweight cardboard border, which provides both containment and structural support without adding significant weight. This design ensures stability and efficiency in movement, making it a crucial component of the system.

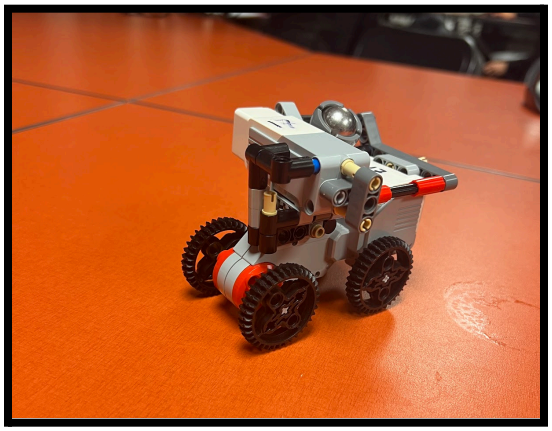


Figure 4.3.1 - Front Isometric view of the X-axis scaffold

Figure 4.3.2 - Rear view of the X-axis scaffold

4.4 The Arm

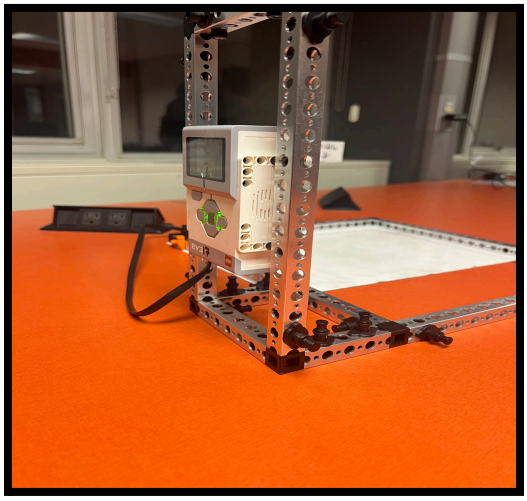
The arm is an essential component of the system, specifically designed to control the vertical movement of the pencil, allowing it to raise and lower depending on whether it is actively drawing. This feature ensures precise and efficient operation, as the pencil is only in contact with the paper when necessary, preventing unwanted marks and maintaining clean transitions. It is attached to a small x-axis robot, which provides the horizontal movement needed for creating the shapes and drawings. The arm's movement is powered by a small motor, which is engineered to provide reliable and smooth control. To further enhance the system’s performance, an additional weight is strategically placed on top of the arm. This added weight ensures that the pencil applies consistent pressure to the paper, resulting in steady, even strokes that maintain a uniform appearance throughout the drawing process. This combination of thoughtful design and

mechanical efficiency allows the arm to function as a key component of a highly effective drawing system.

Figure 4.4 - Isometric view of the arm

4.5 The Outer Structure

The outer structure of the robot is constructed using a metal frame from the Tetrix kit, designed to provide durability and support for the overall mechanism. This frame functions as a railing for the large robot, specifically along the y-axis, ensuring stability and controlled movement. Incorporated into the structure is a large holder for the LEGO EV3 brick, which is positioned to manage the weight distribution and keep significant weight off the main body of the robot. Additionally, a touch sensor is installed near the end of the structure, serving as an emergency stop button to halt operations if the robot veers off course. To further enhance functionality, the frame includes an L-shaped beam that indicates key positions, such as the starting point and the middle of the page, addressing alignment challenges caused by the complexities of the robot's dimensions.



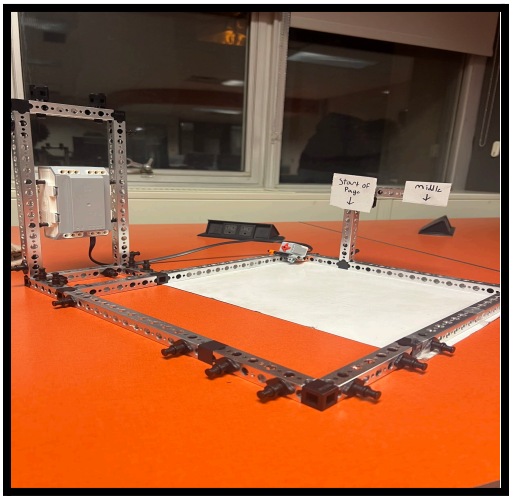


Figure 4.5.1 and Figure 4.5.2 - Views of the outer structure

Section 5: Software Design and Implementation

5.1 Software Breakdown

The software design for SketchBot is a structured and modular system that efficiently integrates hardware components with logical operations to execute its drawing tasks. The code is written in ROBOTC, leveraging the `PC_FileIO.c` library for file handling, and employs a combination of motor control, sensor inputs, and user interaction to achieve its objectives.

The program begins with defining key constants and variables for motor encoders, dimensional conversions, and user-defined parameters such as rectangle dimensions and spacing. These initializations allow the robot to maintain precision and adapt to varying user inputs. The code follows a modular approach, with dedicated functions for each key task, including file handling, motor movement, and user interactions. Finally, the system ensures neat operation by retracing its movements to return to the starting position and shutting down after completing its task.

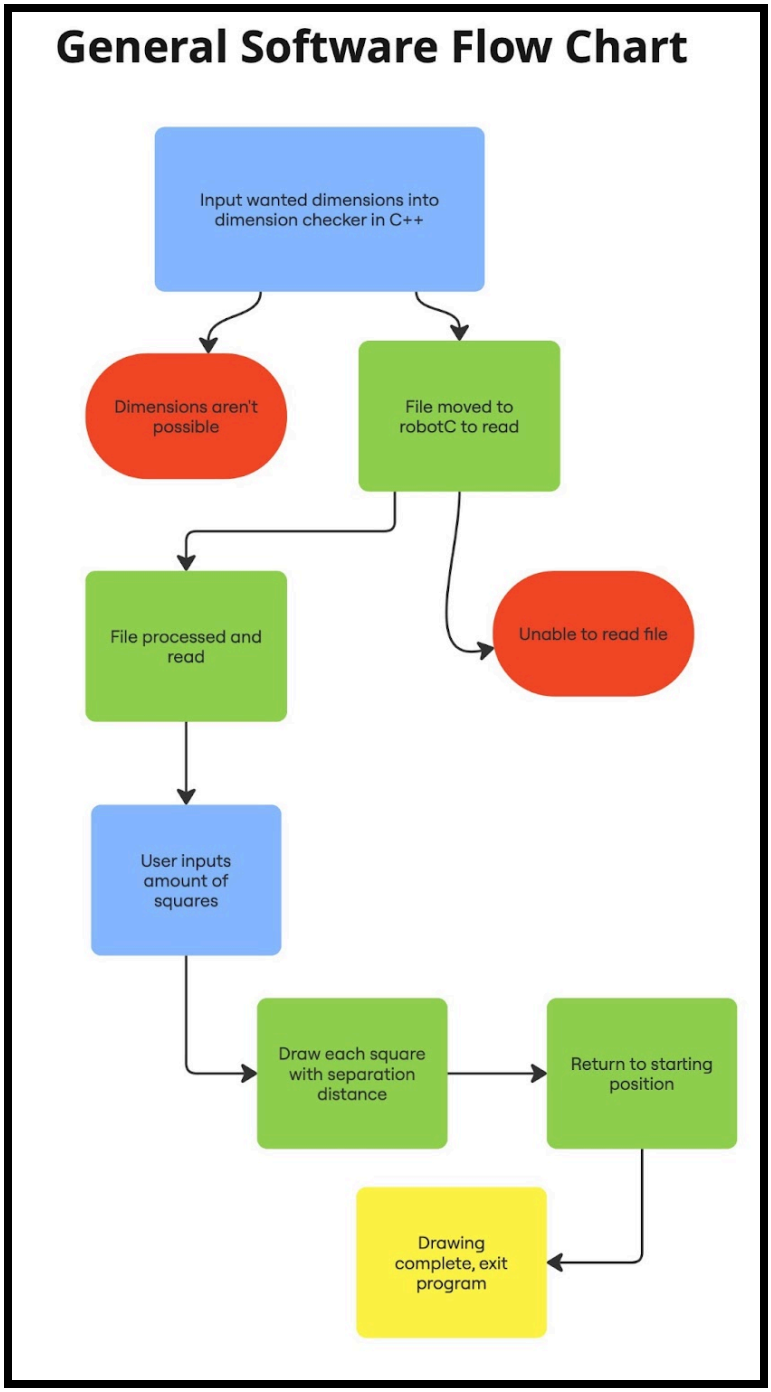


Figure 5.1 - Flowchart breakdown of RobotC program

5.2 File Creation

This program on C++ runs before SketchBot and it calculates the dimensions of a drawing based on user inputs, ensuring that the dimensions fit within a predefined area suitable for the robot's workspace. The user is prompted to input the length and width of the squares to be drawn, the number of squares, and the separation distances from the starting position of the robot. Using these inputs, the program computes the adjusted separations and the total dimensions of the drawing area. It then checks if the total length is less than or equal to 15 units and the total width is less than or equal to 19 units. If these constraints are met, a file containing the dimensions is created and sent to the robot for execution. If the dimensions exceed the limits, the program provides specific feedback on whether the length, width, or both are too large, ensuring clarity for the user.

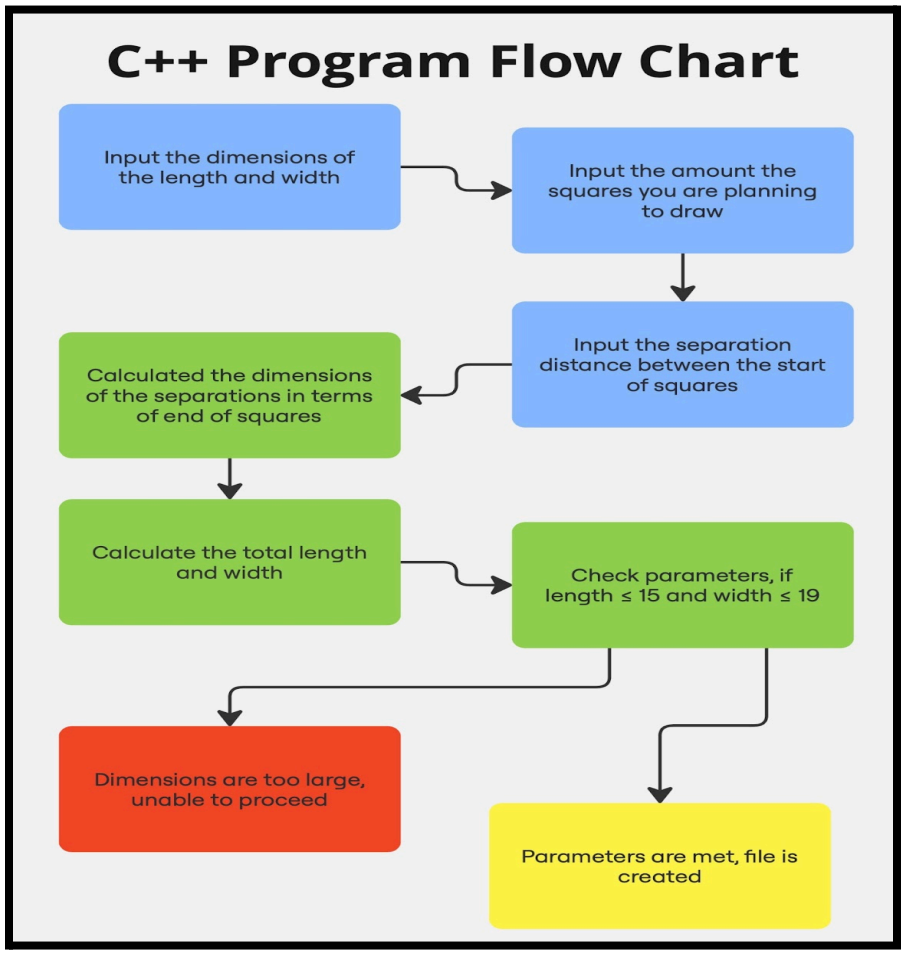


Figure 5.2 - Flowchart breakdown of C++ program

5.3 Data Management and Storage

The program efficiently manages data by storing critical information in variables, ensuring seamless communication between its components. Most of the data is initially read from an external file containing the dimensions necessary for the drawing process. These dimensions—such as the length, width, and spacing of the rectangles—are extracted and stored in corresponding variables in the main program. These variables serve as the foundation for the robot's operations, allowing the functions to access and utilize the information required to execute precise movements and sketches.

In addition to these file-based variables, SketchBot relies on two other crucial variables that play key roles in its functionality:

- **int numOfSq**:

This variable is initialized based on user input from the EV3 console. The user selects the number of rectangles to draw by pressing one of four buttons, each associated with an integer value (1-4). The buttonPress() function detects the user’s selection, returns the corresponding integer, and stores it in **numOfSq**. This value determines the number of iterations the robot will perform during the drawing process, making it essential for tailoring SketchBot's operation to the user's needs.

- **float distanceTraveledX** and **distanceTraveledY**:

These variables are dynamic trackers that accumulate the total distances travelled along the X and Y axes, respectively. Both variables are initialized at 0 and are updated after each movement the robot makes. By summing these distances, the program enables SketchBot to retrace its path accurately and return to its original position once its tasks are completed. These variables are critical for ensuring precision and reliability, especially in scenarios where multiple rectangles are drawn, and the robot must navigate a complex sequence of movements.

By combining data read from files with user inputs and dynamically updated variables, SketchBot's program ensures a robust and adaptable system. This approach to data storage and management is key to the robot's ability to execute its tasks with accuracy and consistency.

5.4 Function Overview

Function	Return Type	Parameters	Method of Passage	Author
file	void	&length, &width, &Xsep, &Ysep	Reference	Nour/Daniel
Xaxis	void	&distanceTraveledX, width	Reference/Value	Daniel
XRaxis	void	&distanceTraveledX, width	Reference/Value	Daniel
Yaxis	void	&distanceTraveledY, length	Reference/Value	Nour
YRaxis	void	&distanceTraveledY, length	Reference/Value	Nour
returnToStart	void	distanceTraveledY, distanceTraveledX, numOfSq, width, length, Xsep, Ysep	Value	Nour
draw	void	&distanceTraveledX, &distanceTraveledY, width, length	Reference/Value	Daniel
separation	void	Xsep, Ysep	Value	Nour
raise	void	N/A	N/A	Noah
lower	void	N/A	N/A	Noah
buttonPress	integer	N/A	N/A	Youssef

Table 5.1 - List of functions

5.4.1 File I/O

The file() function reads data from a file named "Dimensions.txt" and stores the values in the provided reference variables: length, width, Xsep, and Ysep. It first attempts to open the file using openReadPC(), and if the file cannot be opened, it displays an error message and exits. Once the file is successfully opened, the function sequentially reads four integer values from the file—representing the length, width, Y-axis separation, and X-axis separation—and stores them in the corresponding reference variables. If any value fails to be read, the function displays an error message for the specific variable and stops further execution. The use of reference variables allows the function to modify the values directly, making them accessible in the calling code.

5.4.2 returnToStart

The returnToStart() function is responsible for returning the robot to its starting position after it has completed drawing the specified number of rectangles. It takes into account the total distance travelled along the X and Y axes, as well as the number of rectangles drawn and the spacing between them.

First, the function checks if more than one rectangle has been drawn (numOfSq > 1). If so, it initiates the return sequence by resetting the motor encoders and then moving the robot backwards along the Y-axis, retracing the distance it travelled in the vertical direction. The robot moves a distance equal to the total distance travelled in the Y direction, divided by the number of rectangles drawn, adjusted for the Y-axis separation (Ysep).

Next, the function moves the robot along the X-axis, again retracing the total distance travelled horizontally, divided by the number of rectangles and adjusted for the X-axis separation (Xsep). Once the robot has returned to the starting point, the function displays a completion message on the screen, indicating that the return to the start position is complete.

This function ensures that the robot finishes its drawing task in a controlled manner, retracing its steps to the original starting location.

5.4.3 buttonPress

The buttonPress() function allows the user to select how many rectangles SketchBot will draw by pressing one of the four directional buttons on the EV3 console. The function starts by displaying a prompt on the screen, instructing the user to press any button to make a selection. Once a button is pressed, the function checks which button was pressed using the getButtonPress() function.

- If the **Up** button is pressed, the function returns 1, indicating the selection of 1 rectangle.
- If the **Right** button is pressed, the function returns 2, selecting 2 rectangles.
- If the **Down** button is pressed, the function returns 3, selecting 3 rectangles.
- If the **Left** button is pressed, the function returns 4, selecting 4 rectangles.

After detecting the button press, the function displays a confirmation message on the screen showing the number of rectangles selected. If no button is pressed (which is unlikely with the buttonAny condition), it returns 0, but this case is mainly a safeguard. The value returned by this function (1, 2, 3, or 4) determines how many times the robot will repeat its drawing task in the main program.

5.4.4 Xaxis, XRaxis, Yaxis, YRaxis

The Xaxis(), XRaxis(), Yaxis(), and YRaxis() functions are responsible for moving SketchBot along the X and Y axes, both forwards and backwards. These functions utilize motor encoders to track the robot's movement and ensure precise distance coverage.

- Xaxis()

This function moves the robot forward along the X-axis. It starts by resetting the motor encoder of motor C (responsible for horizontal movement) to ensure accurate tracking. The motor is set to move in the positive direction (right), and the robot continues to move until the encoder reads the distance equivalent to the specified width. Once the target distance is reached, the motor stops and the total distance travelled along the X-axis is updated.

- XRaxis()

This function moves the robot backwards along the X-axis. Similar to Xaxis(), the motor encoder for motor C is reset, and the motor is set to move in the reverse direction (left). The robot moves until it retraces the exact distance specified by the width. After the movement, the motor stops, and the distance travelled is updated.

- Yaxis()

This function moves the robot forward along the Y-axis. It resets the encoders for motors A and B (which control vertical movement) and sets both motors to move in the positive direction (forward). The robot continues moving until the encoder reaches the distance equivalent to the length. Once the target distance is covered, the motors stop, and the distance travelled along the Y-axis is updated.

- YRaxis()

This function moves the robot backwards along the Y-axis. It operates similarly to Yaxis(), but the motors are set to move in the negative direction (backward). The robot moves until it retraces the distance specified by the length, after which the motors stop, and the distance travelled is updated.

Together, these functions allow SketchBot to precisely navigate both axes, enabling it to draw the required shapes and move between them with consistent accuracy. They are critical for the robot's ability to draw rectangles and retrace its movements when necessary.

5.4.5 Separation

The separation () function moves the robot a specified distance along both the X and Y axes to create the necessary space between drawn shapes. This function is called between each rectangle to ensure proper spacing as defined by the user. The function first checks the touch sensor value == 0 to ensure the emergency stop operation isn't active. Once confirmed, it begins the horizontal movement.

- X-axis Movement

The function resets the motor encoder for motor C and sets the motor to move in the positive direction (right). It continues moving until the distance travelled equals the specified Xsep (X-axis separation distance). Once the target distance is reached, the motor stops.

- Y-axis Movement:

Next, the function resets the motor encoders for motors A and B and sets both motors to move forward (upward along the Y-axis). It continues to move until the distance travelled equals the specified Ysep (Y-axis separation distance). Once this movement is completed, the motors are stopped.

This sequential movement ensures that the robot correctly spaces the rectangles as it moves between them, maintaining consistency and accuracy in the overall sketching process. The separation() function is crucial for positioning multiple rectangles and for providing adequate spacing as required by the design.

5.4.6 raise, lower

The raise() and lower() functions control the movement of SketchBot's drawing arm, allowing it to be lifted and lowered as needed during the drawing process.

- raise()

This function is responsible for lifting the drawing arm. It starts by checking the touch sensor to ensure the robot is ready to act. The motor encoder for motor D (which controls the drawing arm) is reset, and the motor is set to move in the positive direction (upward). The robot continues to move until the motor encoder reaches the set distance, which is typically the amount needed to lift the arm off the surface of the paper. Once the target position is reached, the motor is stopped, leaving the drawing arm raised and ready for the next movement.

- lower()

This function lowers the drawing arm back down to the surface to begin drawing. It follows a similar process: the touch sensor is checked to ensure the robot is ready, the motor encoder for motor D is reset, and the motor is set to move in the negative direction (downward). The robot continues to move until the encoder reads the set distance, indicating that the arm has been lowered sufficiently. Once the arm reaches the desired position, the motor is stopped, allowing the drawing process to resume.

Together, the raise() and lower() functions ensure that the robot can control the positioning of its drawing arm, allowing for precise lifting and placement during the drawing and spacing between shapes. These actions are essential for maintaining accurate drawings and preventing smudging or overlapping while the robot moves between tasks.

5.4.7 draw

The draw() function is responsible for executing the primary task of SketchBot: drawing a rectangle. It controls the movement of the robot along both the X and Y axes, ensuring the proper execution of the drawing process.

The function begins by checking the touch sensor to ensure the robot is ready to start the drawing operation. If the sensor indicates the robot is in a ready state, the function proceeds to execute the drawing actions.

- X-axis Movement

The function first calls the Xaxis() function to move the robot forward along the X-axis, drawing one side of the rectangle. It moves the robot at a distance equal to the specified width.

- Y-axis Movement

Once the robot has completed the horizontal movement, the Yaxis() function is called to move the robot forward along the Y-axis, drawing the first vertical side of the rectangle. This movement covers the distance specified by the length.

- Reverse X-axis Movement

After completing the first vertical side, the function calls the XRaxis() function, which moves the robot backwards along the X-axis to retrace the previously drawn horizontal line in the opposite direction.

- Reverse Y-axis Movement

Finally, the YRaxis() function is called to move the robot backwards along the Y-axis, retracing the last vertical side of the rectangle and completing the full outline.

The draw() function ensures that the robot precisely follows the dimensions specified for the rectangle, completing each side in sequence and maintaining consistency and accuracy throughout the process.

5.5 Testing Procedures

During the process of building and programming SketchBot, the group prioritized frequent testing to ensure steady progress and verify that each step of the development process was functioning as intended. The primary focus of the project was to achieve consistent and accurate drawings. To test this, the group adopted a simple yet effective method: measuring the dimensions of each rectangle produced by the robot after every adjustment to its mechanical or coding features. This approach allowed us to quickly identify and correct inaccuracies, ensuring that the robot consistently met the desired precision.

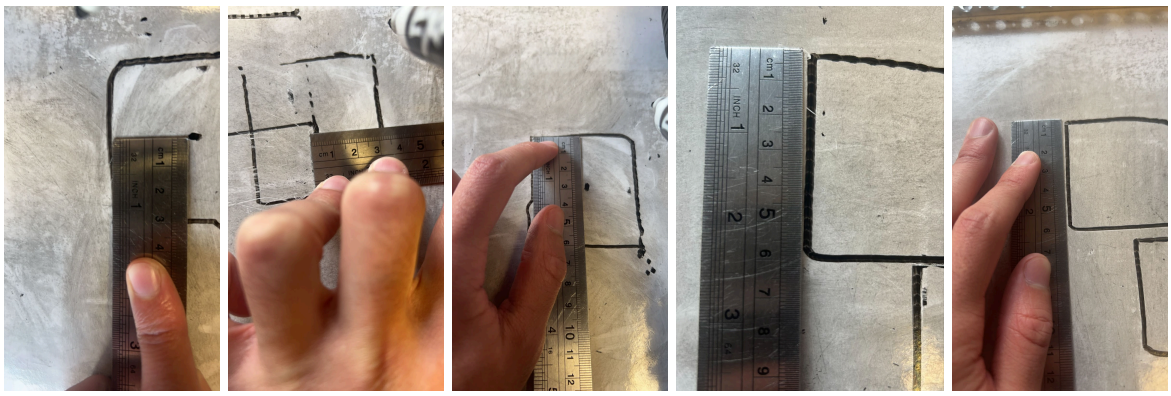


Figure 5.5 - Images of dimensioning for testing

Another key testing strategy involved isolating each function and verifying that it worked independently before integrating it into the main program. For example, functions responsible for drawing, movement, spacing, and returning to the start were tested individually to confirm their functionality. By breaking down the project into manageable components, the group was able to pinpoint errors efficiently and backtrack to identify the source of any issues. This methodical approach to testing not only minimized the risk of compounding errors but also provided a clear pathway for debugging and refining both the code and the mechanical design.

5.6 Software Design Decisions

The design of SketchBot's software was guided by the need to balance simplicity, functionality, and reliability while meeting the project's requirements. Several decisions were made to ensure that the system could achieve its objectives efficiently, but these decisions also involved trade-offs that influenced the overall design.

- Modular Programming

The software employs a modular structure, dividing functionality into discrete functions such as draw(), raise(), lower(), and separation (). This approach enhances readability and simplifies debugging, allowing each function to handle a specific aspect of the robot's operation. However, this modularity comes with a trade-off: increased reliance on parameter passing. Ensuring proper data flow between functions adds complexity to the code, requiring careful management of variables to avoid errors.

- Use of File Input

The decision to use file input for dimension data provides flexibility and user convenience, allowing adjustments without modifying the code. This approach simplifies the process for users who need to specify different dimensions frequently. However, it introduces a dependency on external files and error handling. If the file is missing or improperly formatted, the program cannot proceed, and robust error-checking mechanisms must be in place to handle such scenarios gracefully.

- Safety Mechanisms

Incorporating the touch sensor as an emergency stop adds an essential layer of safety, ensuring that the robot can halt operations immediately in case of errors or unexpected behaviour. While this improves reliability and user control, it also requires additional checks in almost every function, slightly increasing the complexity of the code and the time taken to process movements.

- Motor Control and Precision

Using motor encoders to control movement ensures precise and consistent execution of tasks. This decision was crucial for achieving the accuracy required to draw rectangles with correct dimensions. The trade-off here lies in performance: encoder-based movements are slower compared to open-loop control, as the robot constantly monitors and adjusts its position to stay within the desired parameters.

- Trade-offs in Error Handling

The program prioritizes user feedback and error handling by displaying detailed messages on the EV3 screen for issues such as failed file reads or improper inputs. While this improves user experience and reduces troubleshooting time, it adds to the program's execution time and increases the lines of code, making the software slightly less efficient.

- The decision to Return to Start

The inclusion of the returnToStart() function ensures that the robot returns to its starting position after completing its task, maintaining a clean and predictable workflow. However, this decision introduces additional computations and movements, slightly increasing the overall time to complete the operation.

By carefully weighing these trade-offs, the software design for SketchBot achieves a balance between user experience, precision, safety, and functionality. The decisions made reflect a commitment to building a reliable and versatile system while accepting reasonable compromises to meet the project’s objectives.

5.7 Problems Encountered

1. Inaccurate Movement Along Axes

During early testing, the group noticed that the robot's movements along the X and Y axes were inconsistent. For instance, when attempting to draw a rectangle, the horizontal and vertical sides would not meet perfectly at the corners due to encoder miscalculations or drift in motor movements. In the final code, the group utilized motor encoders with conversion factors (conversionX and conversionY) tailored to the specific radius of the wheels. This ensured accurate distance tracking based on the number of motor rotations. Additionally, the group implemented checks in the Xaxis() and Yaxis() functions to stop the motors precisely when the required distance was reached. This eliminated drift and ensured the robot's movements matched the dimensions specified in the input file.

2. File Input Failures

During development, the robot would fail to start due to an improperly formatted "Dimensions.txt" file, which caused the program to never run or read any of the variables properly. To handle this, the group implemented an error-checking statement in the file() function that would display if an error occurred after the code ran through each step. The program now verifies whether the file is successfully opened and checks each parameter individually as it is read. This ensures the user is aware of the issue and can correct the file.

3. Failure to Return to the Start Position

Early versions of the returnToStart() function did not accurately retrace SketchBot's movements, causing it to stop short or overshoot the starting position. This was due to errors in the calculations of the cumulative distances. The group refined the returnToStart() function by directly referencing the distanceTraveledX and distanceTraveledY variables, which continuously racked the robot's movements. Additionally, the group incorporated slight adjustment factors to the calculations to avoid overshooting or falling short of the starting position.

5.8 Task List for Demo

- Draws a square of proper dimensioning
- Consistently draws the same squares
- Displays timer
- Displays progress
- Draws the correct amount of squares
- Proper separation distances
- Returns to the original position
- The robot shuts down by itself
- C++ file indicates if dimensions are possible or not

****Note**** No changes were made to the task list from the initial conception of the robot.

Section 6: Verification

6.1 Meeting Constraints

The updated constraints were met by careful testing of the robot before the demo. Before the demo, group members came together to test the robot's ability to complete the task given a wide range of scenarios. These included but were not limited to, a varying number of squares, different placement positions, and varying dimensioning parameters. Different scenarios were tested to be ready for a wide variety of possible tests the TAs could ask of us during the demo. However, should no specific parameters be drawn upon by the TAs, the group created an ideal scenario for the robot to draw on. This allowed the robot to showcase its ability to meet constraints in a near-ideal setting, providing a view of what the robot is truly capable of.

6.2 Failing to Meet Constraints

Failure to meet constraints was present in some of the tests the group ran. The accuracy of straight lines, which is the “straightness” of lines, was often inconsistent. The group blames this on the imprecise nature of the Lego EV3 robot. Arm positioning was often difficult as the gears within the motor were sensitive and prone to inaccurate rotation. Furthermore, the robot battery began to fail closer to the project's demo, meaning the motors' power output often fluctuated during the sketching process. Thus, the criteria of accuracy are not fully met. A possible design update which would fix this issue is a new battery component.

Section 7: Project Plan

7.1 Timeline

The project timeline began with group formation on October 6, followed by the project proposal submission on October 11. After the proposal, a major setback occurred as the robot design had to be redefined, delaying progress by one to two days. Despite this, the informal presentation and robot distribution were completed ahead of schedule on October 31. Furthermore, the physical system was completed early on November 6. The final report was started immediately after the formal presentation and was completed well before the deadline, allowing ample time for fine-tuning. This proactive approach ensured that the team stayed on track, with key milestones completed efficiently.

7.2 Mechanical Redesign

After the initial project proposal, the group's original idea did not meet the specified requirements. The initial design was a small modification of the standard EV3 robot, featuring an additional motor to control the mechanical components. However, it lacked significant changes in appearance and did not include enough inputs, prompting a redesign. This redesign, while more complex to implement, ultimately proved beneficial as it resulted in a robot that was more efficient and accurate than initially anticipated.

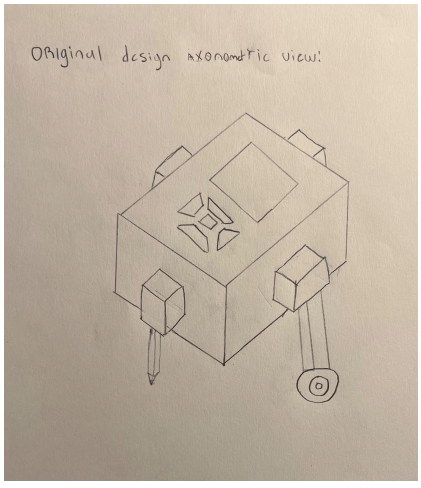


Figure 7.2.1 - Isometric view of original design

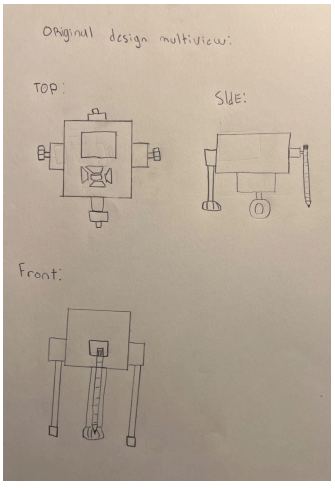


Figure 7.2.2 - Orthographic views of original design

7.3 Splitting of Tasks

The early phase of the project began without any splitting of tasks. Initially, all group members simply brainstormed ideas for the robot. After an idea of the robot was formed, the group decided the mechanical aspect of the design had to be worked on first before the software could be tackled. The design of the robot was split into four parts, being, the frame, the "up/down" axis, the "left/right" axis, and the mechanism which would raise and lower the drawing device. Group members worked on their assigned portions together, bouncing ideas off one another while working on their respective parts. Once the mechanical aspect was dealt with, the splitting of tasks was done categorically by assigning group members to a task which played to their strengths or weaknesses. This allowed for optimal time management, ensuring group members didn't waste time performing tasks which were better suited to a different group member. It was redundant to have all four group members working on the same part of the software, so some group members were allocated to different tasks. Two were assigned to work on the software, one to work on the sketching or PowerPoint, for the formal and informal presentations, and one was assigned to work on the report. This allowed the group to prepare for the future by getting ahead on some tasks which may conventionally be completed later in the project process, lessening the stress on the group by lowering the net workload during a rather busy time.

7.4 Revisions and Changes

Instead of splitting the members into three groups after the mechanical design phase, the group initially intended for all four group members to work on the software. Almost immediately, the group realised this plan was not realistic. The functionality of each independent function was heavily dependent on how other functions would be written. This means that for functions to operate in tangent with each other, a certain vision would have to be laid out and executed from beginning to end. This was especially difficult in the early parts of the project, where a concrete direction of the software was foggy, as specific issues regarding the methods were bound to appear later on in the software design process. Thus, the group decided one person working on the code for the robot was a better way to begin the software writing process, with another member joining when how functions would operate with one another was determined.

Meanwhile, the remaining group members either worked on preparing for the formal presentation, or demo, or writing very raw pseudocode.

The way group members worked together was also revised. The initial idea was to coordinate specific time slots for all four group members to work on the project together, but this plan was found to be problematic. The day-to-day lives of group members varied greatly, with all group members having varying commitments such as religious, athletic, design-based, and social commitments to name a few. This meant that finding time where all group members could meet together was challenging. To avoid this headache while also ensuring the team could coordinate the project vision in a social environment, group members made attempts to meet amongst themselves regardless if the presence of all four members was guaranteed.

Section 8: Conclusions

Sketch Bot embodies the spirit of engineering design that solves world problems. It solved a mechanical issue while aiding people with disabilities. Through mechanical design, software engineering, and intentional demonstration, Sketch Bot was capable of creating proportionate and accurate replicative sketches, aiding the physically impaired while also relieving stresses associated with frustration from drawing.

The design was capable of meeting constraints and criteria through proper engineering practices and detailed planning aiming for the specific requirements of the robot. The limits set by specific constraints were very carefully considered.

Finally, the project fulfilled its purpose of demonstrating learned techniques regarding engineering design. It worked properly and has great potential for future development.

Ultimately, Sketch Bot fulfills the requirement of a user's device while functioning appropriately and safely. This was a successful project made out of collaborative efforts that champion moving forward in real-world applications when limited resources are at one's disposal.

Section 9: Recommendations

9.1 Mechanical Design Enhancements

To improve the mechanical design, several modifications are recommended based on the testing and observations. First, replacing the y-axis wheels with a gear system would provide greater precision in movement, ensuring a more controlled path and reducing the overall footprint of the robot. This adjustment would create additional workspace on the board, allowing for more extensive drawing operations. Second, using a larger motor to move the arm and incorporating additional support would significantly enhance the stability of the marker. This change would reduce smudging and ensure consistent contact with the drawing surface when raising or lowering the arm. Lastly, a smaller yet steadier frame constructed from long LEGO pieces rather than numerous smaller components would improve structural integrity. This approach would simplify maintenance and repair, as fewer connectors would reduce points of potential failure.

9.2 Software Recommendations

xOn the software side, adding the emergency stop mechanism through a while loop instead of having to check if the touch sensor has been activated every step of the way. This approach would allow the stop condition to be continuously monitored, enhancing both the efficiency and safety of the system, especially in industrial applications. Additionally, integrating a feature that checks the compatibility of dimensions directly on the robot would be much faster as it would eliminate the need for a separate verification program. This change would improve workflow and reduce user errors. Finally, implementing functionality to draw various sizes of squares would increase the device's versatility and usefulness in industrial settings, accommodating a broader range of tasks and user requirements.

Section 10: Back Matter

10.1 Appendices

10.1.1 Appendix A: RobotC Code

File: C:\Users\n3abdelr\Desktop\SketchBot.c

```
// File input/output library
#include "PC_FileIO.c"
// Conversion factors based on radius of wheels
const float conversionX = (2 * PI * 1.75) / 360;
const float conversionY = (2 * PI * 2.75) / 360;

// File input
void file(int &length, int &width, int &Xsep, int &Ysep) {
  TFileHandle fin;
  bool fileOkay = openReadPC(fin, "Dimensions.txt");

  if (!fileOkay) {
    displayTextLine(1, "File open failed");
    wait1Msec(10000);
    return;
  }

  if (readIntPC(fin, length)) {
    displayTextLine(1, "Length: %d", length);
    } else {
    displayTextLine(1, "Error reading length");
    wait1Msec(1000);
    return;
  }

  if (readIntPC(fin, width)) {
    displayTextLine(2, "Width: %d", width);
    } else {
    displayTextLine(2, "Error reading width");
    wait1Msec(1000);
    return;
  }

  if (readIntPC(fin, Ysep)) {
    displayTextLine(3, "Ysep: %d", Ysep);
    } else {
    displayTextLine(3, "Error reading Ysep");
    wait1Msec(1000);
    return;
  }

  if (readIntPC(fin, Xsep)) {
    displayTextLine(4, "Xsep: %d", Xsep);
    } else {
    displayTextLine(4, "Error reading Xsep");
    wait1Msec(1000);
    return;
  }
}
```

Page 1 of 5

File: C:\Users\n3abdelr\Desktop\SketchBot.c

```
// Moves robot forward along the X axis
void Xaxis(float &distanceTraveledX, int width) {
  nMotorEncoder[motorC] = 0;
  motor[motorC] = 10;
  while (abs(nMotorEncoder[motorC] * conversionX) < width) {}
  distanceTraveledX += width;
  motor[motorC] = 0;
}
// Moves robot backwards along the X axis
void XRaxis(float &distanceTraveledX, int width) {
  nMotorEncoder[motorC] = 0;
  motor[motorC] = -10;
  while (abs(nMotorEncoder[motorC] * conversionX) < width) {}
  distanceTraveledX += width;
  motor[motorC] = 0;
}
// Moves robot forward along the Y axis
void Yaxis(float &distanceTraveledY, int length) {
  nMotorEncoder[motorA] = 0;
  motor[motorA] = motor[motorB] = 10;
  while (abs(nMotorEncoder[motorA] * conversionY) < length) {}
  distanceTraveledY += length;
  motor[motorA] = motor[motorB] = 0;
}
// Moves robot backwards along the Y axis
void YRaxis(float &distanceTraveledY, int length) {
  nMotorEncoder[motorA] = 0;
  motor[motorA] = motor[motorB] = -10;
  while (abs(nMotorEncoder[motorA] * conversionY) < length) {}
  distanceTraveledY += length;
  motor[motorA] = motor[motorB] = 0;
}
// Returns robot to start
void returnToStart(float distanceTraveledX, float distanceTraveledY, i
  if (numOfSq > 1) {
    nMotorEncoder[motorB] = nMotorEncoder[motorC] = 0;
    displayTextLine(5, "Returning to start...");
    motor[motorA] = motor[motorB] = -10;
    while (abs(nMotorEncoder[motorB] * conversionY) < (distanceTravele
    motor[motorC] = -10;
    while (abs(nMotorEncoder[motorC] * conversionX) < (distanceTravele
    displayTextLine(6, "Return complete!");
  }
}
// Draws a singular square/rectangle
void draw(float &distanceTraveledX, float &distanceTraveledY, int widt


  SensorType[S1] = sensorEV3_Touch;
```

File: C:\Users\n3abdelr\Desktop\SketchBot.c

```
  if(SensorValue[S1] == 0){
    displayTextLine(5, "Starting draw()");
    Xaxis(distanceTraveledX, width);
    Yaxis(distanceTraveledY, length);
  }

  SensorType[S1] = sensorEV3_Touch;
  if(SensorValue[S1] == 0){
    XRaxis(distanceTraveledX, width);
    YRaxis(distanceTraveledY, length);
  }
}
// Moves robot along the X and Y axis by indicated seperation distance
void seperation(int Xsep, int Ysep) {

  SensorType[S1] = sensorEV3_Touch;
  if(SensorValue[S1] == 0){
    nMotorEncoder[motorC] = 0;
    motor[motorC] = 10;
    while (abs(nMotorEncoder[motorC] * conversionX) < Xsep) {}
    motor[motorC] = 0;

    nMotorEncoder[motorA] = 0;
    motor[motorA] = motor[motorB] = 10;
    while (abs(nMotorEncoder[motorA] * conversionY) < Ysep) {}
    motor[motorA] = motor[motorB] = 0;
  }
}
// Raises drawing arm
void raise() {

  SensorType[S1] = sensorEV3_Touch;
  if(SensorValue[S1] == 0){
    nMotorEncoder[motorD] = 0;
    motor[motorD] = 25;
    while (abs(nMotorEncoder[motorD]) < 25) {}
    motor[motorD] = 0;
  }
}
// Lowers drawing arm
void lower() {

  SensorType[S1] = sensorEV3_Touch;
  if(SensorValue[S1] == 0){
    nMotorEncoder[motorD] = 0;
    motor[motorD] = -25;
    while (abs(nMotorEncoder[motorD]) < 30) {}
    motor[motorD] = 0;
  }
```

```
}
// Takes user input to return an integer that ditctates the number of
int buttonPress() {
  displayTextLine(5, "Press a button to select squares.");

  while (!getButtonPress(buttonAny)) {}

  if (getButtonPress(buttonUp)) {
    displayTextLine(6, "Selected: 1 square");
    return 1;
    } else if (getButtonPress(buttonRight)) {
    displayTextLine(6, "Selected: 2 squares");
    return 2;
    } else if (getButtonPress(buttonDown)) {
    displayTextLine(6, "Selected: 3 squares");
    return 3;
    } else if (getButtonPress(buttonLeft)) {
    displayTextLine(6, "Selected: 4 squares");
    return 4;
  }

  return 0;
}

task main() {
  displayTextLine(1, "Starting program...");
  wait1Msec(1000);

  int length, width, Xsep, Ysep;
  file(length, width, Xsep, Ysep);

  int numOfSq = buttonPress();
  displayTextLine(7, "Squares to draw: %d", numOfSq);
  wait1Msec(2000);

  float distanceTraveledX = 0.0;
  float distanceTraveledY = 0.0;
//Starts timer
  time1[1] = 0;

  for (int i = 0; i < numOfSq; i++) {


    SensorType[S1] = sensorEV3_Touch;
    //if(SensorValue[S1] == 0){
      displayTextLine(8, "Drawing square %d of %d", i + 1, numOfSq);
      draw(distanceTraveledX, distanceTraveledY, width, length);
      wait1Msec(1000);
      raise();
```

```
      if (i < (numOfSq - 1)) {
        seperation(Xsep, Ysep);
        lower();
        wait1Msec(1000);
    }
  }
  float end_time = time1[1]; // Ends timer

  displayTextLine(10, "Drawing complete!");
  displayTextLine(11, "Total Time: ");
  displayTextLine(12, "%0.2f", (end_time / 1000)); // Displays time fr
  wait1Msec(3000); // Waits 3 seconds before returning to start

  returnToStart(distanceTraveledX, distanceTraveledY, numOfSq, width,
}
```

10.1.2 Appendix B: C++ Code

```
#include <fstream>
#include <iostream>

using namespace std;

int main() {

    int length = 0;
    int width = 0;
    int numSq = 0;
    int Lsep1 = 0;
    int Wsep1 = 0;
    int Lsep2 = 0;
    int Wsep2 = 0;
    int Ltotal = 0;
    int Wtotal = 0;

    ofstream fileObject1;
    fileObject1.open("/Users/danielmehany/Desktop/Dimensions.txt");

    cout<<"welcome to SketchBot!"<<endl;

    cout<<"enter length and width of the square(s):"<<endl;
    cin>> length >> width;

    cout<<"enter how many squares you would want to be drawn:"<<endl;
    cin>> numSq;

    cout<<"enter the seperation length and width between the start of the
     squares:"<<endl;
    cin>> Lsep1 >> Wsep1;

    Lsep2 = length - Lsep1;
    Wsep2 = width - Wsep1;

    Ltotal = ((length * numSq) + (Lsep2 * (numSq - 1)));
    Wtotal = ((width * numSq) + (Wsep2 * (numSq - 1)));

    if (Ltotal <= 15 && Wtotal <= 19) {
        cout<<"drawing is within parameter, file created!"<<endl;
        cout<<"You're dimensions have been transferred, please choose the
         amount squares you want directly on the bot"<<endl;
        fileObject1 << length << " " << width << " " << Lsep1 << " " <<  Wsep1;
    }

    else if(Ltotal > 15 && Wtotal <= 19) {
        cout<<"length is too large, unable to process"<<endl;
    }

    else if(Ltotal <= 15 && Wtotal > 19) {
        cout<<"width is too large, unable to process"<<endl;
```

```
    }
    else
        cout<<"length and width are too large, unable to process"<<endl;
}
```

10.2 References

N/A