Graphing Robot

Group 814

MTE 100/GENE 121

December 3rd, 2019

Avery Chiu 20820648

Chuting Wang 20833432

Kerui Liu 20827426

Table of Contents

Table of Figures

Table of Tables

Summary

This introduction describes the purpose of creating the graphing robot and the definition of a function in mathematics. Functions are an important concept taught in high school and university. The graphing robot can help with education and it is a good resource for students.

The scope describes how the robot operates at a high level. The robot's main functionality is to receive a function entered by the user, to process the function, draw a Cartesian plane, then graph the function on a piece of paper. Some examples of inputs are inputs from a file for coordinates and input from a colour sensor to ensure that there is paper to draw on.

The constraints and the criteria outline the importance of the robot being lightweight, efficient, and accurate in terms of its graphing. There are some important constraints that constrict the size of the page or the weight of the robot for efficiency. There are also criteria such as being able to graph the function accurately within a tolerance of $\pm 1\ cm$.

The mechanical design outlines the design of the pen, arm, roller, and frame of the graphing robot with photos. The pen is controlled by a motor to move it off the page. The arm is controlled by a motor with wheels on a track that allows the pen to move horizontally. The roller is controlled by a motor that slides the paper into the frame horizontally.

The software design describes how both the C++ program and RobotC program operate together. In the C++ program, the program is broken into functions to process each type of graphical function. In the RobotC program, the program is broken into small tasks, such as moving to the origin or moving to the next coordinate. The software design also describes how the tests are performed, such as testing whether the C++ program will output the correct coordinates and how the RobotC program will move to each coordinate.

The verification describes how most constraints were met. In particular however, the robot itself did not have good control of the paper itself. This was due to the fact that the roller would tilt the paper sideways rather than vertically. This created a distorted image of the graph, which could be fixed if there was a roller on each side of the paper.

The project plan compares how the original plan deviates from what actually happened. In particular, the testing was not performed on time due to minor issues with the RobotC program. There were many small issues that destroyed the program, which delayed the team. These bugs needed to be fixed before testing could be performed.

The conclusion explains that the task was to create a robot that can accurately graph functions. It outlines how the pen, motor, arm, and frame are designed. It also describes how the software accepts functions and allows for the robot to graph different points.

Acknowledgements

We would like to acknowledge the teaching assistants and professors from the department of Mechanical and Mechatronics Engineering from the University of Waterloo for providing us guidance with the project. We are grateful for the resources and equipment they have provided us, specifically the Lego EV3 parts and the Tetrix Kit.

Introduction

Functions are an integral part of mathematics are taught in high school and university. An example of a function y=x+2. A value of x is plugged into the equation which results in a y value. Since there are an infinite amount of x values that can exist, when all of them are plugged in and plotted on a cartesian (x,y) plane, it results in a graph of a function.

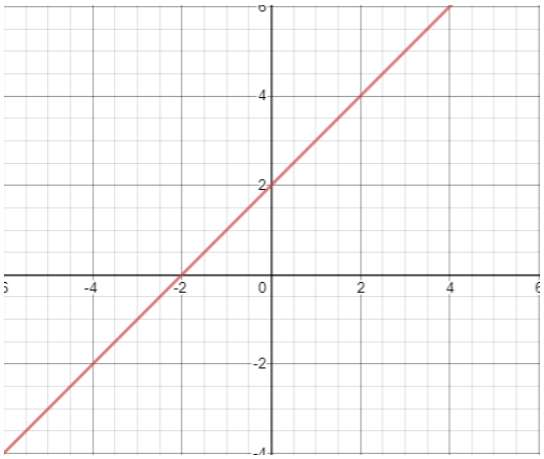

Figure 1:An example of a graph of the function y=x+2 [1]

The Graphing Robot was created to help graph functions on a physical page and for educational purposes. Many students in high school and university struggle with concepts in calculus. A robot that creates visual copies of graphs helps them to visualize the functions they are seeing in the classroom. This is important for students since it can help them solve their calculus problems (e.g. curve sketching) in a fascinating way. In turn, this also helps attract more students into the STEM field as the robot will help strengthen their calculus skills while also showing them a great example of a mechatronic system [2].

Scope

The main functionality of the robot is that it graphs polynomial, rational, trigonometric, exponential, and logarithmic functions. It receives a function from the user when they enter it into the computer. The computer creates a file of coordinates for the robot which receives each coordinate and moves its arm and the pen to that coordinate. At the end, the robot should successfully graph the function that the user entered.

Table 1: Interactions with environment and Motor Inputs

Interaction	**Description**
Receive user input on computer	The computer prompts the user and guides the user to enter the function they want.
File Input	The robot receives a file that is created by the C++ program.
Colour Sensor	The robot will not start unless it can detect white, meaning there is a page to draw on.

Button Press	The robot does not start automatically, instead it waits for the user to press the button to start the robot.
Motor for roller	The roller drags the paper under the robot to move it in the vertical direction, which helps graph the y-values of the coordinates. The motor spins the roller which drags the paper.
Motor for arm	The arm moves horizontally which helps to graph the x-values of the coordinates. The motor is used to move the arm around on the track.
Motor for pen	The pen spins sideways. This is used so the pen will not draw lines that are not part of the graph. The motor helps rotate the pen up and back down onto the page.

The robot will recognize that the task has been completed when all the valid coordinates have been processed and have been drawn onto the graph. In the overall shutdown procedure, the robot should output a message onto the screen of the robot to inform the user that it is finished graphing. Then the robot should output the total time it takes to graph the function in seconds as well as prompt the user to press a button. After the button press, the arm will be rotated to a down position and it will stop.

Since the design was changed, there were minor changes to the scope. Instead of simply starting at the bottom left side of the page and connecting each coordinate, the robot draws the cartesian plane first and starts at the origin instead. This change was made so it would be more clear for the user to see the graph. Otherwise, for example the two functions y=x and y=x-1 would look the same without a Cartesian plane.

Instead of sensing if the robot hits an object, the robot was changed so it would detect when there is no paper under it instead. The original plan was to have the robot drive and graph the page. However, due to changes made to the design, the robot now uses a colour sensor to detect when there is paper under it. This change was made since it would not make sense for the robot to detect when it hits something, it is more important for the robot to simply make sure it is drawing on paper and not on a table or the floor.

Criteria

Requirements:

The following requirements for the design of the robot and the entire system accentuates on the ability to draw a graph and calibrate the arm to move to the origin within the specified tolerance after each task is performed. The table includes a brief summary of each of the requirements that were mentioned in the preliminary report and the changes that were made to the previous requirements with sufficient explanations. It will also explain how each of the requirements support the development of the project (or did not support them).

Table 2: Requirements for the project and any changes made to it

Requirements			
Original (From preliminary report)	**Changes to the requirements**	**Explanation of changes to requirements**	**Did it help in the development of the project?**
The robot should draw a clear graph. This requires the robot to draw a smooth connection between two points.	None (Kept the same)	N/A	**Yes.** To construct a clear representation of the graph by drawing smooth lines requires very concise coordination between the motor. We have derived formulas representing the relationships between the two motor speed to meet the requirements.
The robot arm should keep the pen up when the robot is not drawing points for the graph.	None (Kept the same)	N/A	**Yes.** With this requirement, we designed and created a function in RobotC that controls the rotation of the pen. The function is being called when the pen is not drawing on the page (ie. calibrating to the origin)
All the points plotted by the robot come from the function entered by the user. This requires the computer to check that the robot receives the correct output file with the right coordinates from the latest function user inputs.	None (Kept the same)	N/A	**Yes.** It is important to ensure that all the points are coming from the right function. This can be checked by plugging in the x-values from -10 to +10, and compare the y-values outputted by the computer to the file.
N/A	The robot should be able to calibrate itself and move to the origin after performing certain	This requirement is added to increase the accuracy of the graph since the robot is	**Yes.** Being able to move to the origin within a tolerance of $\pm$ 1 $cm$is a

	tasks such as drawing the Cartesian plane or plotting graphs within a tolerance of maximum $\pm 1\ cm$.	drawing the Cartesian plane and graph based off of the origin. It is important that the pen can return to the origin consistently.	very crucial feature that increase the preciseness of the function that is graphed.
N/A	The robot should draw the Cartesian plane which starts from the origin and draws the positive x-axis and negative x-axis. The robot will then return to the origin and start to draw the positive and negative y-axis.	This allows the person to interpret the function more accurately. .	**Yes.** Graphing the cartesian plane shows the user how each section of the Cartesian plane appears and how the positive and negative part of the function is plotted.

Constraints:

The following constraints for the design of the robot and the entire system shows the significance of the limitations on the size of the paper, input of functions and analysis of coordinates. The table includes a brief summary of each constraint that was mentioned in the preliminary report and any changes that were made to the previous constraints with sufficient explanations. It will also explain how whether the constraint was helpful in the development of the project or not.

Table 3: Constraints for the project and any changes made to it

Constraints			
Original (From preliminary report)	**Changes to the requirements**	**Explanation of changes to constraints**	**Guided through project?**
The movement of the robot should be restricted, within the page to prevent the robot from drawing off the platform. The size of the page must be taken into account and at no point should the arm draw off the page.	The size of the paper must be kept consistent otherwise the robot will be unable to determine where to draw.	In our previous design, the EV3 robot was on top of a platform. It was important for the robot to not fall off the platform. With changes to the design, we made the robot stay on the same level as the paper. The robot no longer has to worry about falling off a platform, but rather simply not drawing off of the paper.	**Yes.** We have decided to limit the size of the paper to $12cm \times 25cm$, and keep it consistent for each graph. This will allow for consistent functions.
	The paper must not slide out of the robot's frame and must slide vertically along the side of the robot.	The old design did not move the page.	**Yes.** We have added two vertical bars to aid the paper to slide vertically along the page.
The pen must not draw off the paper, and must not fall off the holder.	None (Kept the same)	N/A	**Yes.** Following the constraint, we secured the pen in place with two lego blocks on the side. Tape has wrapped around the pen to secure the pen to prevent it from falling

			off the holder.
The input of a function is being restricted to a single function and the robot should not create combined functions.	None (Kept the same)	N/A	**Yes.** We did not put any composite functions in the choice of functions when user starts the program. If user wants to draw f(x) = sin(x)*x, and input sin(x) as a coefficient for polynomial function, the computer will not process the input, and will ask for a new input. Thus no graph will be drawn onto the paper.
	The coordinates calculated by the computer should not be greater than the domain and range allowed by the Cartesian (x,y) plane created on the paper.	With the new design, we have restricted the scope that the robot is allowed to graph in. So it is important for the computer to check whether the coordinate is within the range.	**Yes.** We have created functions in RobotC that checks whether the point is going out of the range (going into infinity). If the system detects that the point is not in the range, it will not plot it.

Mechanical Design

Overall Mechanical Design

The overall mechanical system consists of three motors: two large motors control the arm and the paper; a medium motor controls the rotation of the pen. There are two bars that act as a platform for the arm to move vertically, these bars act as a track. A colour sensor was placed to the side of the robot to make sure paper can be detected before drawing.

A large motor is connected to the two gears turning the wheel that helps the arm move along the platform. With the limited number of wheels, two sets of small Lego EV3 wheels were used to prevent the arm from tilting. Two hooks are connected to the arm that prevent it from falling off the platform. Another large motor is attached to the other wheel that rolls the paper along the side of the robot. Similar to the design of the arm, there are two sets of small Lego EV3 wheels which were used to help guide the paper along the side. The pen is attached to the medium motor that is connected to the arm. The motor controls the pen's rotation and it adjusts the pen into up whenever it is not graphing.

Subsections for the various components (pictures of following parts):

Frame

The top view of base structure of the system is shown in the figure below. It contains a rectangle base that was constructed using Tetrix Kit and two LEGO bars that act as the platform.

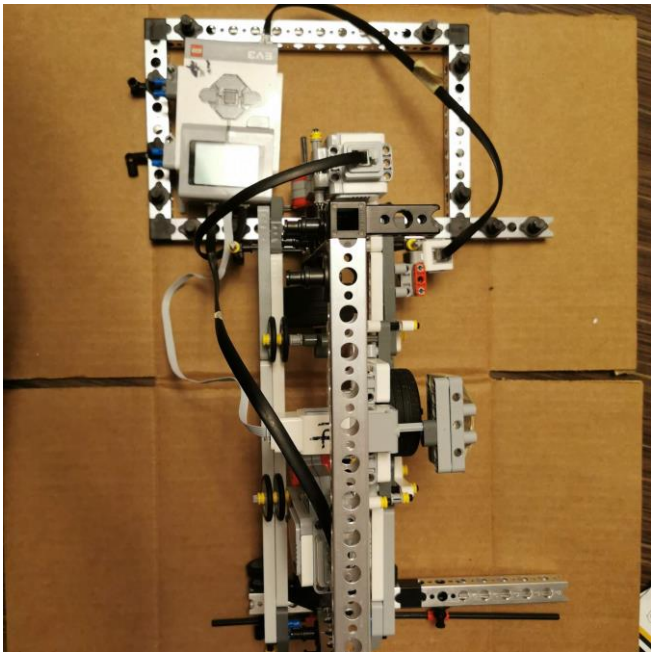

Figure 2: Top view of the system

The following view shows the left side of the robot which there is a large motor connected to the side. There is also a colour sensor attached to the platform.

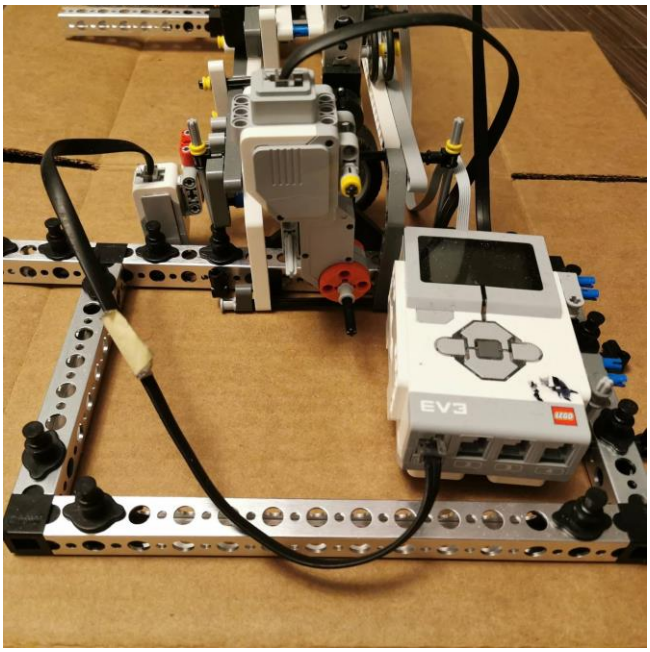

Figure 3: Left side view of the system

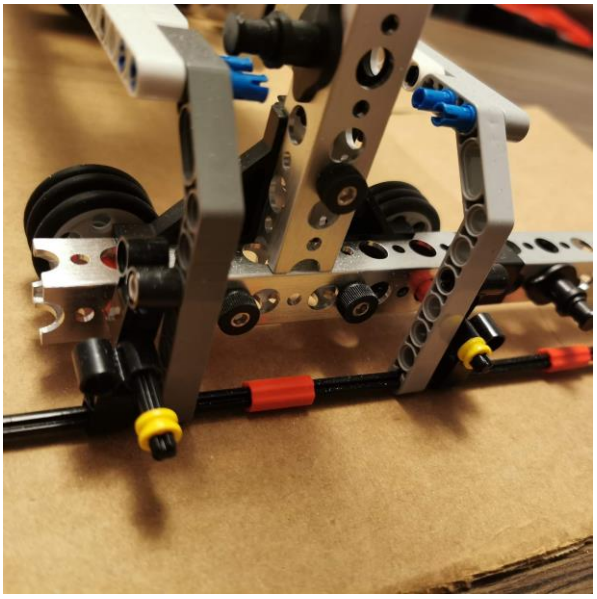

Figure 4: Right side view of the system

Motors

- Large Motor that control the paper

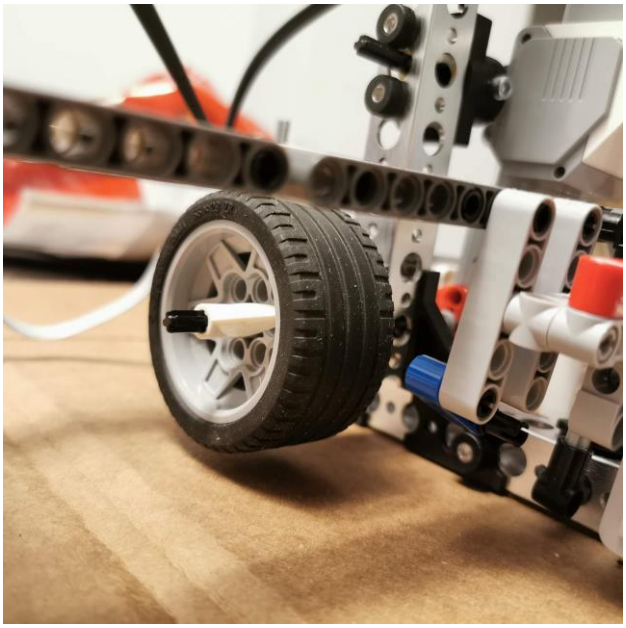

Figure 5: Large motor that connects the wheel controlling the paper

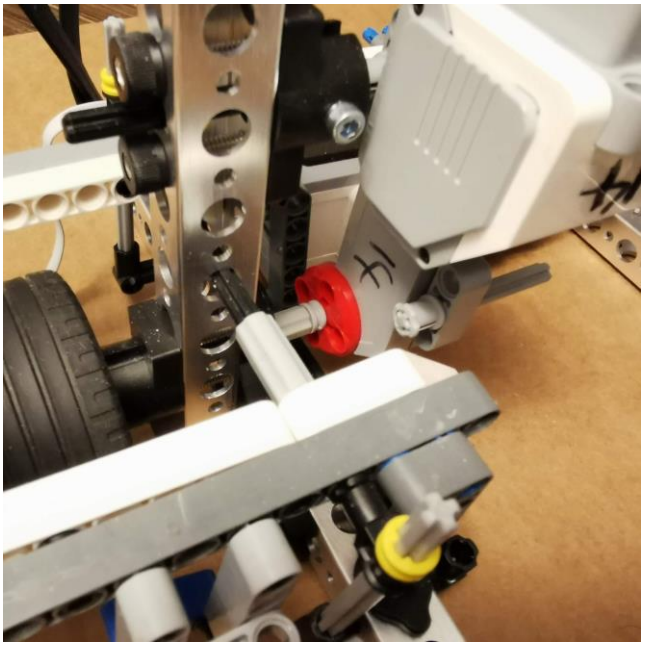

Figure 6: Connections between wheel and motor

- Small wheels that aid the motor to move the paper vertically

Figure 7: Two sets of small wheels that aid the paper to move vertically

- Large Motor that control the arm

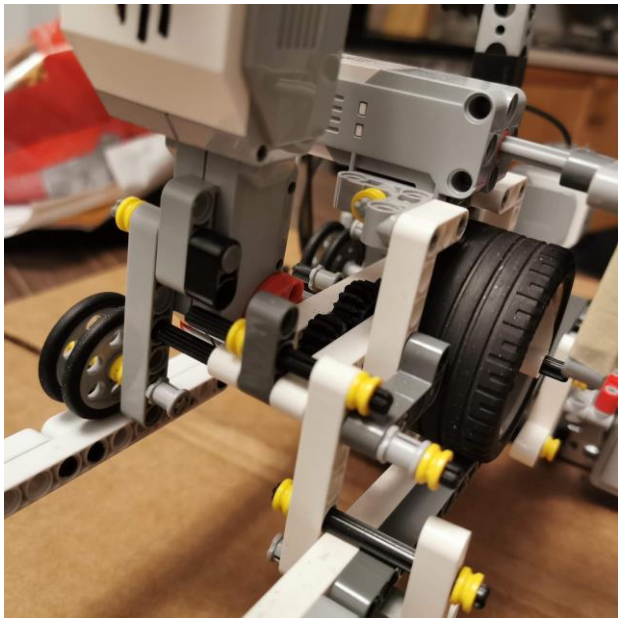

Figure 8: Large motor that is connected to the arm

- Small wheels that help the motor move along the platform

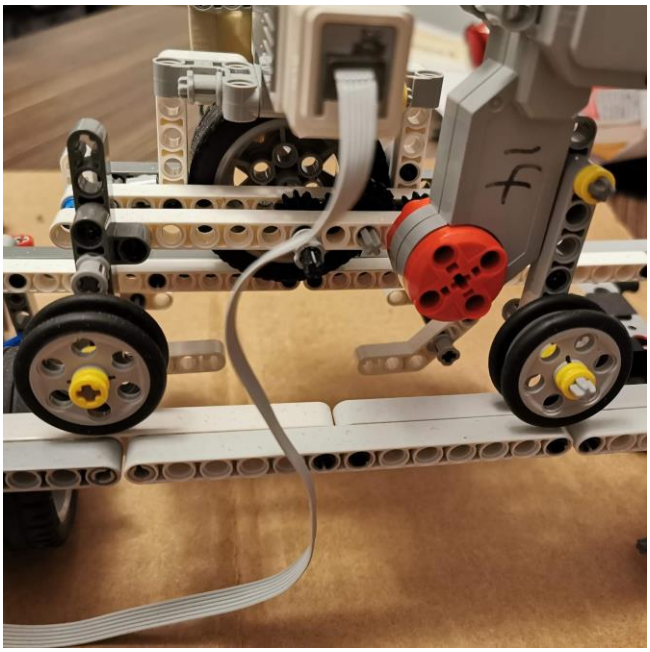

Figure 9: Two sets of small wheels that guide the arm to move along the platform

- Medium Motor that control the pen

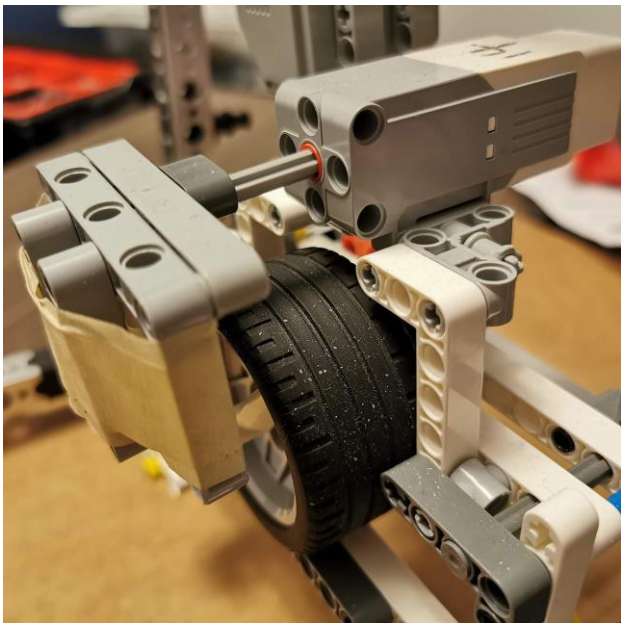

Figure 10: Medium motor that control the rotation of the pen

Sensor attachment design

- Colour sensor

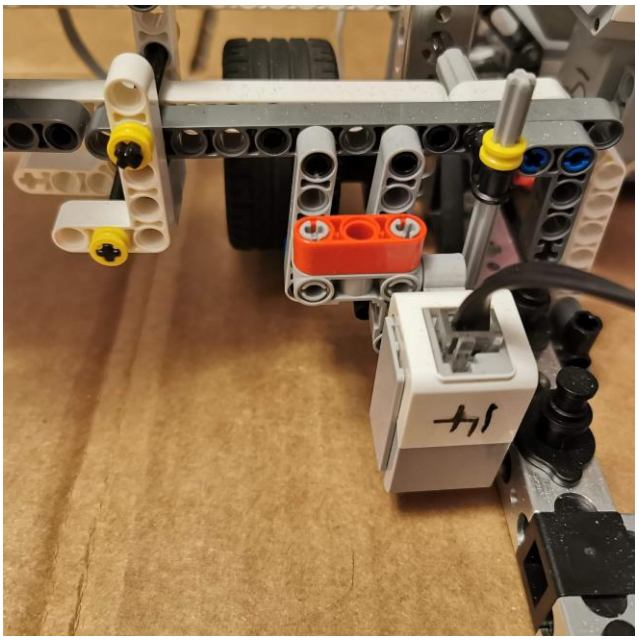

Figure 11: Colour Sensor that attach to the side of the robot

- Hooks that holds the arm in proper position (preventing the arm fall off the platform)

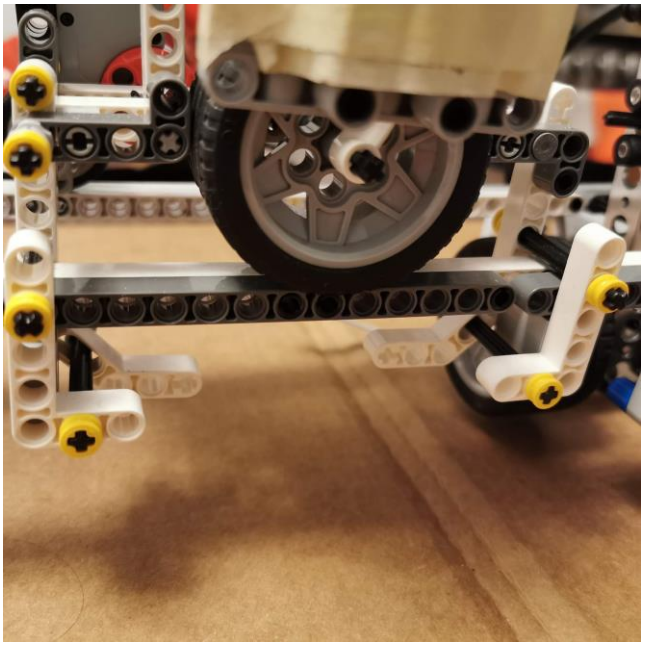

Figure 12: Hooks that holds the arm in proper position

- Connections between Tetrix and Lego EV3.

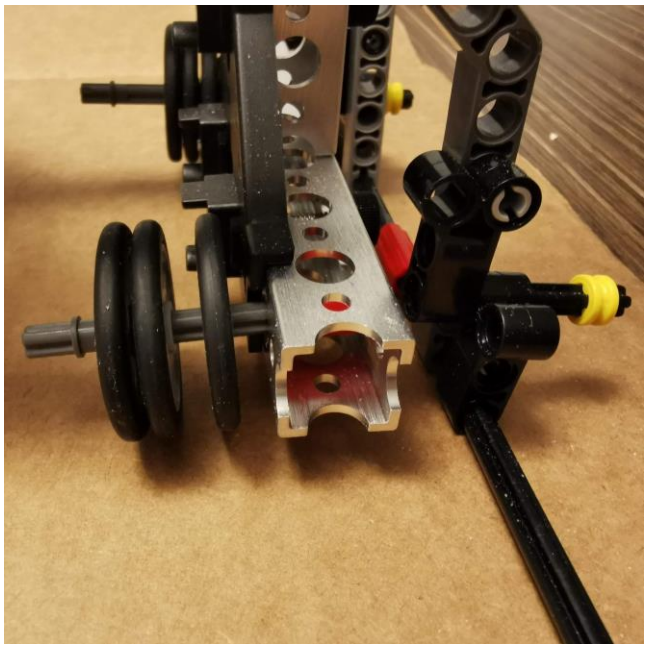

Figure 13: Connections between LEGO EV3 and Tetrix Kit

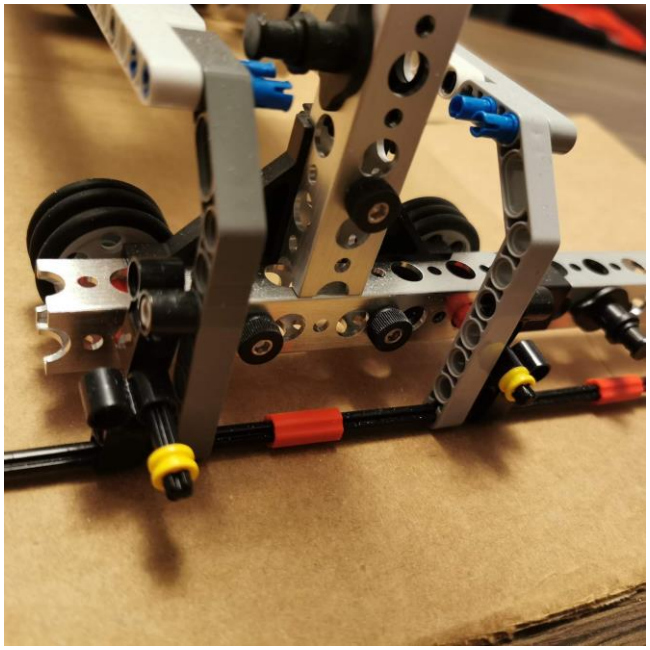

Figure 14: Assembly between parts

Table 4: Mechanical Design Decisions and any Trade Offs

Design Decision	**Trade Offs**
Using medium motor to control the movement of the pen	The connection between the motor and the pen is only connected with a single piece of Lego which cause the pen

	to tilt or bend while drawing. This lead to unclear graphs.
Using two sets of small Lego EV3 wheels to help guide the paper to slide under the platform	The larger wheels are not at the same level as the smaller wheels which caused some unbalance in the system. This also created some difference in friction that could cause the paper to slide sideways which would affect the graphing of the function.
Two parallel bars were constructed to act as a track for the robot	The length of the platform restricts the size of the graph which can only be $8\ cm$. A smaller graph makes it harder to draw precise graphs.

Software Design

How the program was broken into smaller tasks:

For the C++ program, the design was broken into getting the user's function of their choice, allowing them to scale it, then writing coordinates of this function to a file. This was chosen since it is a robust way to get user input. Letting the user type in any function they want can be hard to control. In this design, the user can only choose one type of function, which makes dividing up the blocks easier. Each block will compute a specific function (e.g. a trigonometric function), allow the user to scale the function (such as stretching it), then output the coordinates of that function to a file.

For the RobotC part, the design was broken into multiple steps which involved reading input from the file, drawing the cartesian plane, then driving to each coordinate. These steps were chosen since they are all distinct tasks that need to be accomplished for a user to be able to see their function. It seemed to be a logical way of graphing a function since it is similar to how it is done by a human. A human would think of the function they are drawing, draw the Cartesian plane, then graph out the function keeping in mind the key points of the graph.

Flow chart for the program:

C++ Flow Chart:

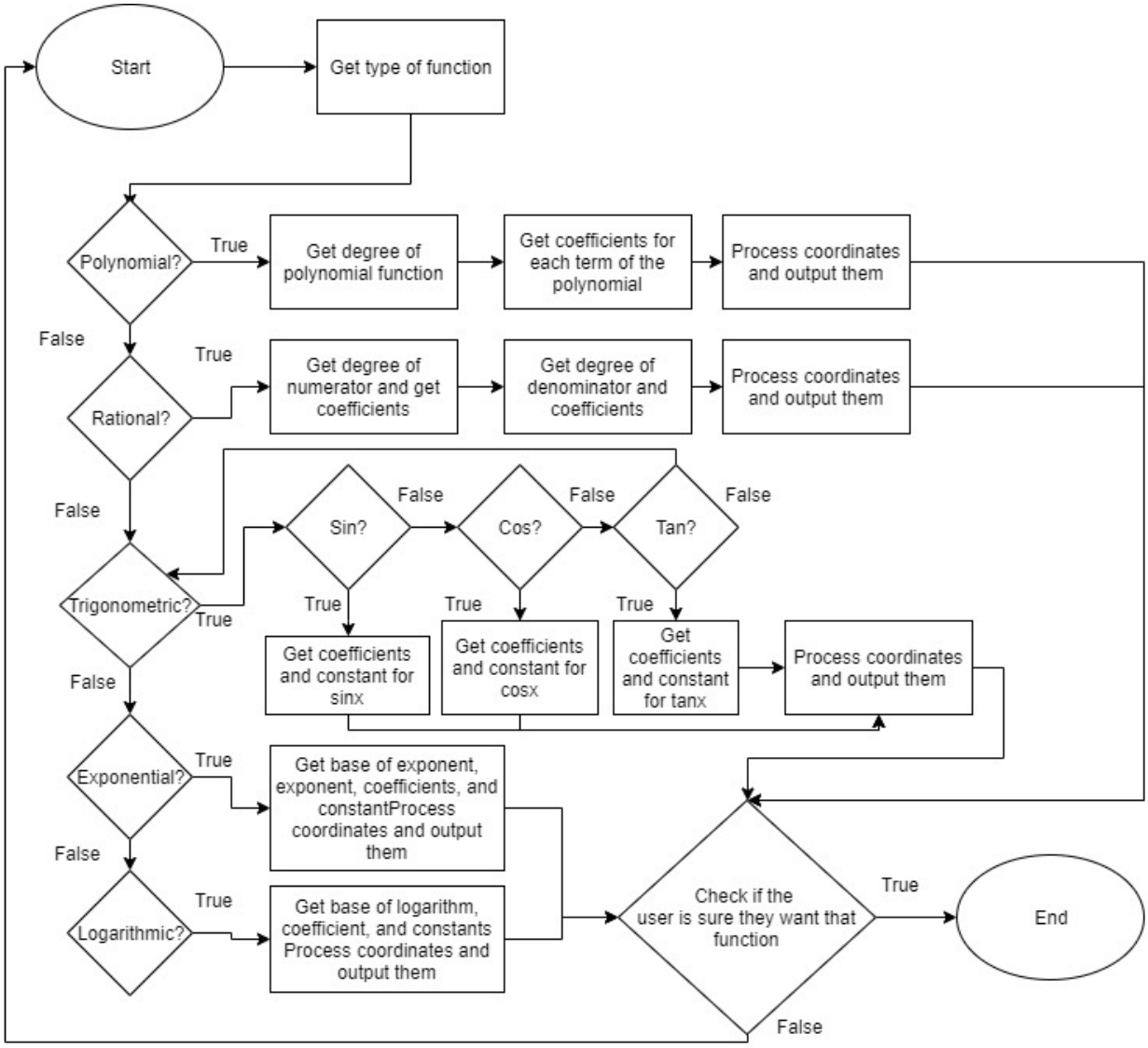

Figure 15: Flow chart for C++ Program

RobotC Flow Chart:

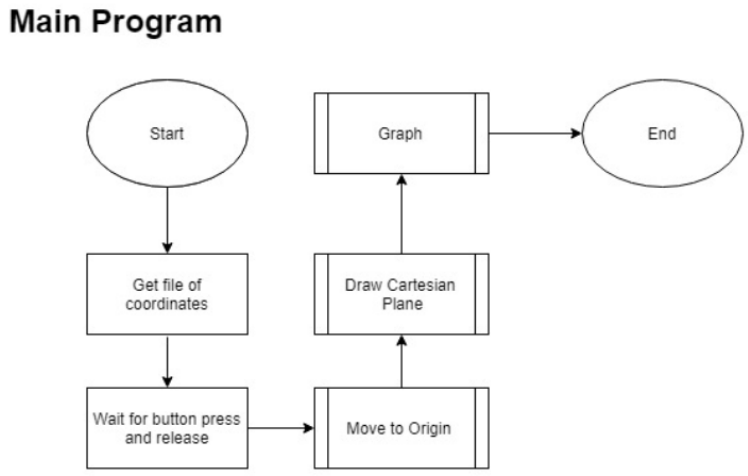

Figure 16: Flow chart for RobotC - Main Program

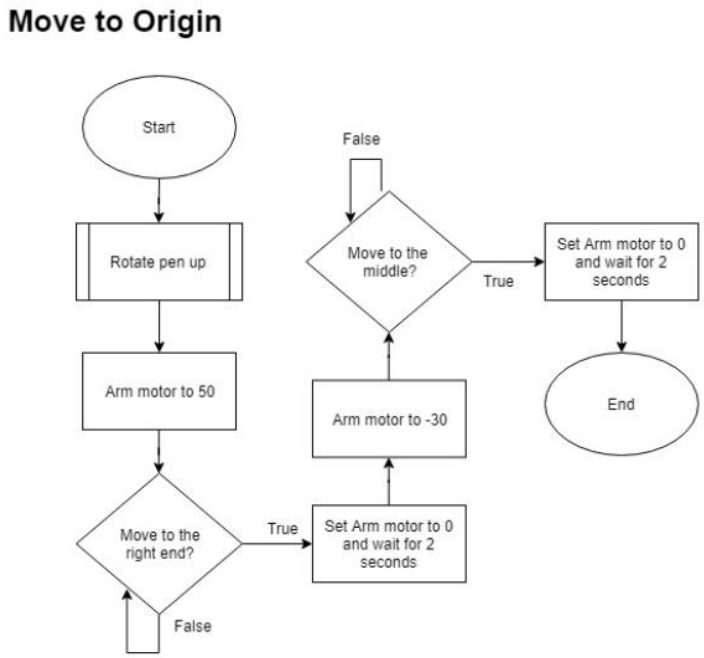

Figure 17: Flow chart for RobotC - Move to Origin

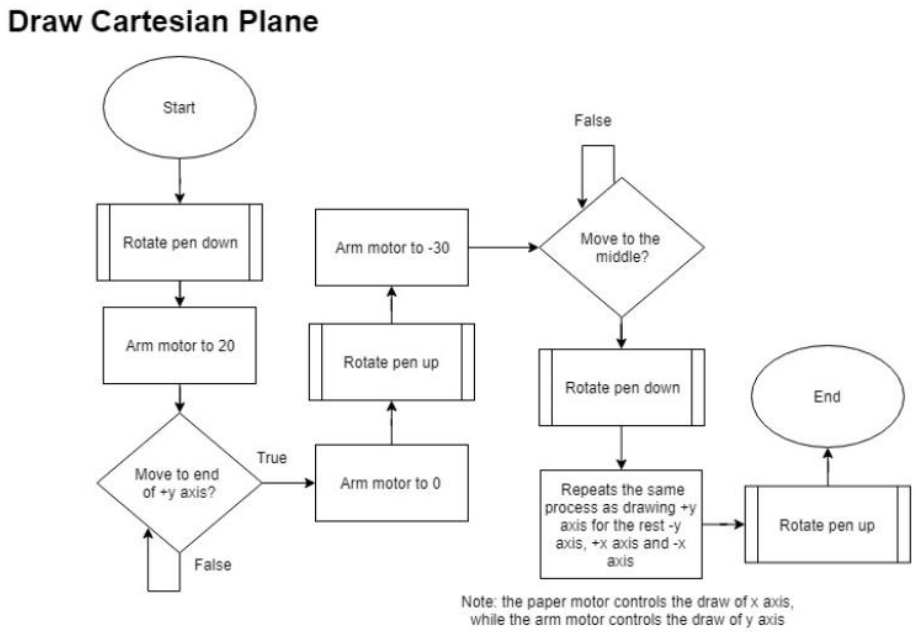

Figure 18: Flow chart for RobotC - Draw Cartesian

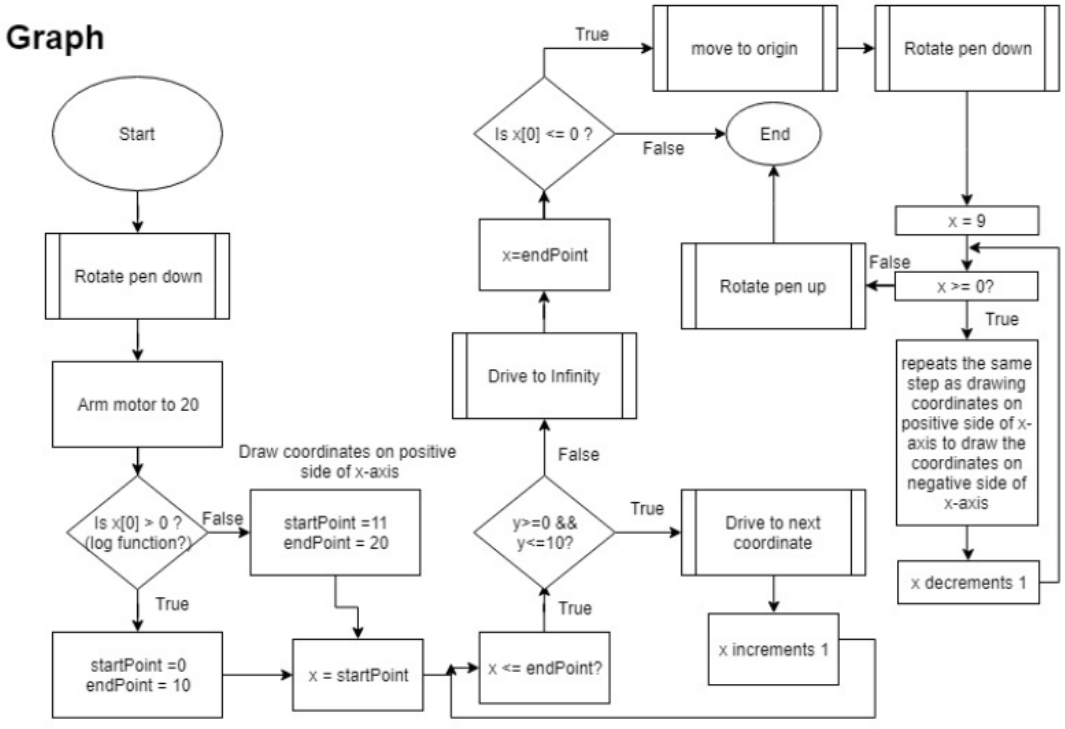

Figure 19: Flow chart for RobotC - Graph

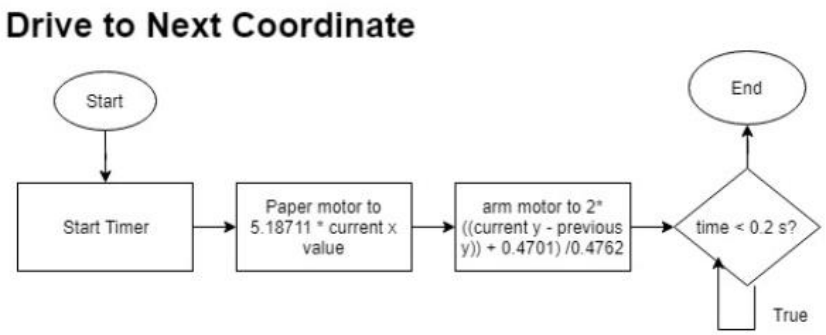

Figure 20: Flow chart for RobotC - Drive to Next Coordinate

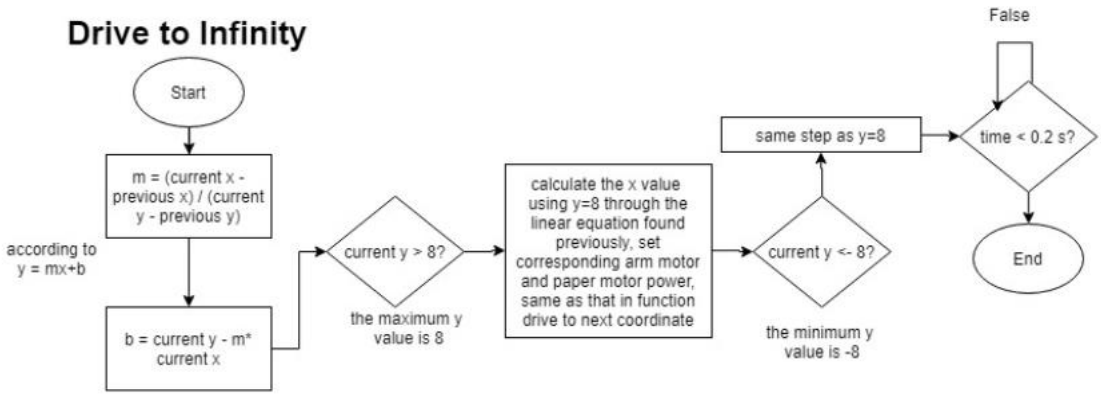

Figure 21: Flow chart for RobotC - Drive to Infinity

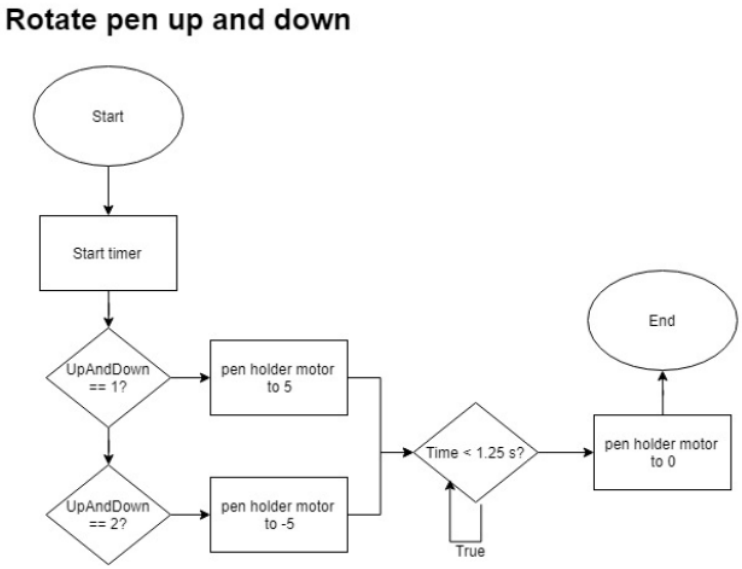

Figure 22: Flow chart for RobotC - Rotate pen up and down

Updated Task List:

- Receive function from user
- Write file of coordinates to robot
- Receive file of coordinates from computer
- Wait for a button press before starting and prompt the user to press the button
- Draw Cartesian plane
- Plot the graph on the right side of x axis
- Plot the graph on the left side of the x axis
- Wait for button press to end the program

One major change to the tasks list is drawing the cartesian plane, which was added for clarity so the user could interpret the function more clearly.

The plotting was also changed, the robot now graphs the x axis on the right side before going to the left, rather than start from the far left and going to the right. This change was made so the robot could be calibrated more accurately.

Functions for C++ Program:

Table 5: Functions for C++ Program

Function Name	Return Type	Parameters	Description	Programmer
getFunction	void	none	Lets the user choose the function they want	Avery Chiu

degreeInput	int	none	Allows the user to choose the degree of their function	Avery Chiu
polynomialDegree	int	int *coefficients int degree	Processes the polynomial and gets the coefficients/constants	Avery Chiu
processPolynomial	void	int *coefficients int constant int degree	Gets coordinates for the polynomial and writes them to a file	Avery Chiu
processRational	void	int *coefficientsNumerator int degreeNumerator int constantNumerator int *coefficientsDenominator int degreeDenominator int constantDenominator	Gets coordinates for the rational function and writes them to a file	Chuting Wang
trigonometric	void	none	Gets the type of trigonometric the user wants	Avery Chiu
wantFunctionOrNah	bool	none	Makes sure the user entered the right function and confirms if they want it	Avery Chiu
trigFunctionOutput	void	int choice int coefficient int innerCoefficient int constant	Scales the trigonometric function and outputs it to a file	Avery Chiu
processLogarithmic	void	none	Gets base, constants, and coefficients and writes the coordinates of a logarithmic function to a file	Chuting Wang
processExponential	void	none	Gets base, constants, and coefficients and writes the coordinates of an exponential function to a file	Kerry Liu
endOfProgram	bool	none	Confirms if the user wants the their function and ends the program	Avery Chiu

Functions for RobotC Program:

Table 6: Functions for RobotC Program

Function Name	Return Type	Parameters	Description	Programmer
motorSpeed	void	motorInfo & m int newSpeed	Set motor speed for specific motor	Chuting Wang
readFile	void	xyCoord* coords	Read coordinates from file that C++ has outputted	Avery Chiu
pressAndReleaseButton	void	None	Wait for user to press and release any button on the robot	Avery Chiu
moveToOrigin	void	None	Rotate the pen up, and calibrate the arm to the origin (middle)	Chuting Wang
rotatePenUpAndDown	void	int upOrDown	The function controls the up and down position of the pen. upOrDown = 1 (up) upOrDown = 2 (down)	Chuting Wang
drawCartesianPlane	void	None	Draw the cartesian plane starting from the origin, and then draw the positive x-axis and the negative x-axis. Then calibrating to the origin to draw the positive and negative y-axis.	Chuting Wang
graph	void	xyCoord* coords	Graph the function starting from origin and move to the next coordinates while testing if the value is going to infinity.	Chuting Wang, Avery Chiu
undefined	bool	xyCoord coords	Check if the x,y coordinates is out of range	Avery Chiu
toInfinity	bool	xyCoord coords int motorEncoderValue	Check if the y-value is out of range (going to draw off the page)	Avery Chiu
moveToNextCoo	void	xyCoord coords	Using data from the motor speed and the	Kerry Liu

rd		int yOld	range available for the robot to draw in to calculate appropriate motor speed for both the arm and paper	
driveToInfinity	void	xyCoord coords int yOld int xOld	Set appropriate motor speed for arm and paper by deriving an equation representing the motor speed	Kerry Liu
colorTest	bool	None	Check if the color sensor detects white to start running the robot	Chuting Wang
end	void	int time	End the program by setting all motor speed to 0, and output the time that the robot takes to graph the function	Avery Chiu

How Data is Stored in the Program

For the C++ program, the coefficients of the polynomial and rational functions are stored in an array. Each aspect of the other functions, such as the coefficient and constant, are stored in variables. For example y=5sin2x + 3, does not require an array to store its coefficients. In fact, the majority of the program simply uses variables to store data. When writing coordinates to a file, the program simply calls upon the coefficient and constant variable to scale coordinates accordingly to solve for their respective y-values.

Explanation of software design and any trade offs

Table 7: Explanation of the C++ program design and Trade Offs

Design	Explanation	Trade Offs
Users select the type of functions	Users are prompted to enter number corresponding to the type of the functions they want to enter.	Combined functions cannot be entered in this program and also the types of functions are limited to polynomial, rational, primary trigonometric, logarithmic, and exponential functions only.
Users enter degree of the functions	For polynomial and rational, the degrees of the functions are limited to at least 1 and at most 5.	Users cannot enter functions higher than degree 5.
Polynomial functions	Users select degree of functions, enter coefficients for each degree and finally the constant	Requires computer to prompt the user for each aspect, rather the user simply typing in the whole function in one go.
Rational functions	Users select degree for numerator, enter coefficients for each degree and finally the constant. Repeat the same step for the	The form of the rational functions entered is limited to the form of y = f(x)/g(x). (i.e.

	denominator.	if users want to enter y = 1/x + 1, they have to convert it to y = x+1/x.)
Primary trigonometric functions	Users select the type of trigonometric functions, which are sine, cosine, and tangent. The functions entered is in the form of y = a f(kx) + c.	User cannot choose inverse trigonometric functions.
Logarithmic functions	Users enter the function in a form of y = a log(kx)/log b +c, by receiving the outer and inner coefficients, the base, and the constant of the functions.	Requires computer to prompt the user for each aspect, rather the user simply typing in the whole function in one go.
Exponential functions	Users enter the function in a form of y = a (b)^x + c, by receiving the inner coefficients, the base, and the constant of the functions.	Requires computer to prompt the user for each aspect, rather the user simply typing in the whole function in one go.
The maximum x value that can be calculated is +10, while the minimum is -10.	For each function entered, not all coordinates can be calculated.	The domain is restricted from [-10,10].

Table 8: Explanation of the RobotC program design and Trade Offs

Design	Explanation	Trade Offs
Read coordinates from the file generated by the C++ program	Users have to download the file into rc-data on RobotC each time a new function is entered and a new file of coordinates is generated.	Only one function can be graphed when the program runs once. The program cannot prompts the users to enter a new function. This is also less user friendly.
The robot starting position is at the origin	The robot arm (moving horizontally) moves to the right end of the platform, and then moves back to the middle of the platform, which is the origin of the graph.	The robot cannot draw the graph from one end to the other end, instead, the robot has to start drawing one side of the Cartesian plane starting from the origin, then come back to the origin to draw the other side.
The robot rotates pen up and down using timer	Through testing, when the medium motor is set to power 5, the motor turns 90 degrees clockwise in 1.25 s. As a result, if the pen's starting position is straight down, the pen rotates up with motor power of 5 in 1.25 s and then rotates down back to	Cumulative errors happen since each time the pen is rotated, the timer is reset. As a result, the rotation of the pen has to be used as least as possible, otherwise when the pen is rotated down to the drawing position, it might not be accurate anymore.

	drawing/starting position when motor power is set to -5.	
The robot draws the Cartesian Plane when it is adjusted itself properly to the origin position.	The robot draws +y, -y, +x, and -x respectively, which means each time the robot finishes one end of the axis, it has to rotate the pen up and comes back to the origin before drawing the next end of the axis.	N/A
The physical distance between each two x values is 0.4cm	As the maximum horizontal movement is 8 cm, the distance between each two x values is calculated to be 0.4 cm on paper. As such, the y value calculated is converted based on this scale.	In order to limit the y value generated from x values from -10 to 10, the coefficients entered for each degree is limited to -5 to 5 in the C++ program.
The robot draws on each side of the x-axis by drawing lines from one coordinate to the next till the one that has y value out of the range or the robot reaches x=10 or -10	Starting from the origin, the robot graphs the coordinates on the positive side of the x-axis first. The x value is incremented by 1 each time the robot is moving to the right. Once the corresponding y value converted is greater than 8 cm, the robot will take the 8 cm as the y value and calculate the corresponding x value, forming a new coordinate which the robot drives towards before moving back to the origin position and start drawing the other side of the axis.	N/A

How is the program tested throughout the project

In the C++ program, as each function was created it was tested to see whether it would provide correct output. This means that each y value generated should correlate to its x value. For example, if the user entered the function y=2x, an x value of 5 should correlate to a y value of 10. This data is seen in the file that is created by the C++ program.

In the RobotC program, each individual tasks was tested. This meant isolating all the other functions and only calling a specific one. This was to ensure that the function would operate properly. A good example is the moveToOrigin function. This was tested to make sure that the function would move the arm back to the origin regardless of where it is located.

Testing for C++ Program

Table 9: Testing procedure for C++ Program

Test	Expected Output	Actual Output	How to ensure the program ran correctly?	Why this test was performed?
y=2x^2+2 When x = 5	52	52	The expected and actual output should be the same	The test makes sure the coordinates for the polynomials are correct
y = (2x+1)/x^2 When x=2	1.25	1.25	The expected and actual output should be the same	The test makes sure the coordinates for the rational functions are correct and that fractions are handled well
y=sinx+5 When x=8 (in radians)	5.989358247	5.98936	The expected and actual output should be similar (to 5 decimal places)	The test makes sure the coordinates for the trigonometric functions are correct
y=2log10(x)+3 When x=-5	None	None	The expected and actual output should be the same, there should be no values since the function is noted for negative x-values	The test makes sure the coordinates for the logarithmic functions are correct and that there are no coordinates for negative x-values
y=2(3)^x + 1 When x-019	1.00003387	1.00003	The expected and actual output should be similar (to 5 decimal places)	The test makes sure the coordinates for the exponential functions are correct and that there are proper coordinates for negative x-values

Testing for RobotC Program

Table 10: Testing procedure for RobotC Program

Test	Expected Event	Actual Event	How to ensure the program ran correctly?	Why this test was performed?
moveToOrigin	The graph should move to the middle	The graph moves to the middle	The arm should be equidistant and located in the midpoint between the two sides (with a tolerance of + or - 1cm)	This test was performed to ensure that the origin is consistent for the graph and so other values drawn by the graph will not be skewed
moveToNextCo	The robot	The robot	The next coordinate	This test was performed

ord For y=x^2 from (1,1) to (2,4)	should draw a curve from the previous coordinate to the next	draws the curve as specified but is skewed slightly due to the rotation of the page.	should be 1 cm away horizontally and 3 cm away vertically.	to ensure that the robot can graph functions properly
drawCartesianPlane	The robot should draw a Cartesian plane of size 8cm x 8cm	The robot draws the plane of size 6cm by 7cm	The robot should draw the cartesian plane (with a tolerance of + or - 2 cm)	This test was performed to make sure that the robot can properly plot a graph

Significant problems encountered and how they were solved

Table 11: Program encountered with software design and solutions

Problem	Solution
The files being written to the robot were empty and did not contain any coordinates. The robot did not receive the file input properly.	The file was downloaded into the robot (rc_data) so that it could be read.
The pen would continuously spin and would not stop until someone forcefully shut down the robot	Two of the arguments were swapped when attempting to call a function. This led to the pen to continuously spin rather than the roller. Initially the roller would spin until a touch sensor was pressed. However, instead the pen spun continuously instead.
The robot needs to coordinate the motor speeds so it can draw the graph accurately. Both motors must move at the same time to draw diagonal lines or curves.	A program is written to test the distance that the wheel connected to the large motor can move with its corresponding motor power. The power is incremented by 5 each time. A table is generated based on this, seen in table 1 of Appendix A.
The pen holder should turn 90 degrees each time it rotates up or down, but the speed of rotation of the pen holder turns needs to be converted to corresponding medium motor power.	Similar to the large motor, the turning speed of the pen holder with corresponding medium motor power is tested out through a written program. A table is generated as the reference for calculations from the motor power to the pen holder turning speed, seen in Table 2 of Appendix B.

Verification

Constraints used in demo and how are they met

The table discussed the constraints that were used in the demo and how each of them were met. For those constraints that did not met were bolded and will be explained at the bottom.

Table 12: Updated constraints and how are they met

Constraints	**How is it met**
The size of the paper must be kept consistent. Otherwise the robot will be unable to determine the area to draw in.	The size of the paper is set uniformly to $12cm \times 25cm$, so we can set a boundary for the robot with in the code so that it will not go off the range.
The pen must not draw off the paper, and must not fall off the holder.	The pen has been set in place with two lego blocks on the side. Tape has been used for securing the pen to prevent falling off the holder.
The paper must not slide off the page and must slide vertically along the side of the robot.	There are two tetrix "bars" that are connected to the base of the robot so the paper could slide vertically along the side.
The input of a function is being restricted to a single function and the robot should not create combined functions.	A message displayed on the screen will notify the user that a combined function is not a proper option. In case the user wants to draw f(x) = sin(x)*x, and input sin(x) as a coefficient for polynomial function, the computer will not process the input, and will ask for a new input. Thus no graph will be drawn onto the paper.
The coordinates calculated by the computer should not be greater than the domain and range allowed by the cartesian (x,y) plane created on the paper.	Within the RobotC program, we construct a function that is able to check whether a coordinate is out of the allowed range. If the y coordinate for any x value is outside the range, the robot will automatically drive to infinity and sketch a line that reaches the edge of the domain or range.
The arm of the robot should not fall off the platform.	There are two hooks that are attached to the bottom of the arm so it can not move when it reaches the edge.

Table 13: Constraints that were not met and Explanation

Constraints that were not met	**Explanation**
The pen must not draw off the paper	This constraint was not met because the roller that controlled the paper was sliding the paper away from the robot sometimes. Thus the pen could not draw onto the paper.
The paper must not slide out of the robot frame.	This constraint was not met since there was no signal that was sent to the robot informing it when the paper is sliding off.

Project Plan and Revisions

Table 14: Project Plans and how work is been allocated

Member	**Role**
Avery Chiu	Develop user interface for function including the polynomial function and trigonometric function. Create functions in RobotC to drive the robot towards each point smoothly.
Kerui Liu	Develop user interface for function including the Exponential function. Create functions in RobotC that allows robot to draw (moveToNextCoord, driveToInfinity) Build the platform/base for the robot and part of the robot arm.
Chuting Wang	Develop user interface for function including the rational function and logarithmic function. Create functions in RobotC, including functions that rotate the pen up and down, graph, adjust the robot position (moveToOrigin), draw Cartesian Plane on the paper. Build part of the platform for the robot and the robot arm and add it to the robot.

The follow figure show is the Gantt chart for the final project that shows the distribution and dependency of task.

Gantt Chart For Final Project

Duration

The dates are not to scale

Sub Projects	Date Start	Date End	PERCENT COMPLETE	PERIODS					
				October 11, 2019 to October 25, 2019	October 26, 2019 to October 27, 2019	October 28, 2019 to October 31, 2019	November 1, 2019 to November 4, 2019	November 4, 2019 to November 7, 2019	November 8, 2019 to November 20, 2019
Develop user interface to receive functions and interpret the function they want	October 11, 2019	October 27, 2019	**100%**						
Testing code and file output	October 28, 2019	October 31, 2019	**100%**						
Write RobotC code to interpret file and output date	November 1, 2019	November 7,2019	**100%**						
Create robot arm to draw	November 4, 2019	November 7, 2019	**100%**						
Create robot platform to drive on	November 4, 2019	November 7, 2019	**100%**						
Test graphing functions	November 8, 2019	November 20, 2019	**100%**						

Figure 23: Gantt Chart

The mechanical system was created before the RobotC code was developed. This was done so the RobotC program could be written and calibrated perfectly for graphing. Tests were performed to figure out what motor power would deliver a certain speed (in cm/s). These tests could not have been performed without first constructing the mechanical system.

Testing was not performed on time. There was minimal testing on November 5th due to delays in writing the RobotC program. This was due to many minor logical errors within the code that had to be fixed before testing could be performed.

Conclusion

In conclusion, the report is about the methods used to attempt to create a graphing robot that can receive a function on the computer and draw the function out physically on a piece of paper. However, these methods were incompetent and did not fully meet all the requirements. Even though the computer

properly interpreted the functions, the graphs that were drawn did not always meeting the expectations. There were many times where the graph was skewed and where the graph obviously did not represent what the user entered.

In the mechanical design, the arm, roller, and pen worked well together The pen would spin clockwise and counterclockwise to align the pen onto the page. The arm and roller would coordinate with each other to draw a diagonal line or curve from one point to the next. However, the main problem with this design was that the roller was only on one side of the page causing the page to become tilted whenever the roller tried to move sometimes. This created many inconsistencies.

In the software design, each block of code was broken down to process each graphical function. Each one of these would process their respective function and output coordinates to a file. In the RobotC part, there were several functions for each tasks that would allow the robot to move to each coordinate, including moveToOrigin, and drawNextCoord. These tasks were split up in chronological order so the robot could successful

Overall, the design behind the robot was good but with a few minor errors that could have been corrected in both the mechanical and software design.

Recommendation for Mechanical Design

The range of the platform can be enlarged both horizontally and vertically, so that when connecting the coordinates of the graph, the cumulative errors due to resetting timers and encoders would appear less obvious compared to the current 8 cm x 8 cm range which requires a lot of precision.

The roller that controls the paper should have been the same on both sides in order to maintain the orientation of the page and so it would not tilt sideways.

Recommendation for Software Design

The software design could have been more organized. If there is more time, the program could have been split up into multiple files rather than being in one block. In the C++ part, there were many tasks that were repeated constantly that could have been placed into a function. For example, one output function can be used to process the output for each type of function rather than processing the output separately under each type of function.

Another small issue was the long time it took for the robot to receive a file. Due to the actual software for the LegoEV3 robot, the user must manually download the file onto the robot before it can read the coordinates and draw them. It would be better if a script could automatically do this for the user.

For the RobotC program, the coordinates read in can be processed first in order to calculate the end points, the points that are out of the range, before drawing the graph. In this case, the robot will be able to draw the graph from one end to the other rather than drawing two sides of the coordinates separately and adjusting itself back to the origin, which can create errors due to all the different adjustments the robot needs to make and rotations of the pen holder.

References

[1] Desmos, "Desmos," Desmos, [Online]. Available: https://www.desmos.com/calculator. [Accessed 25 October 2019].

[2] C. Wang, A. Chiu and K. LIu, "Preliminary Report," University of Waterloo , Waterloo , 2019.

Appendix A

large motor			
power	time (s)	distance (cm)	average speed (cm/s)
0	5	0	0
5	5	8.63938	1.727876
10	5	17.27876	3.455752
15	5	34.55752	6.911504
20	5	43.1969	8.63938
25	5	60.47566	12.095132
30	5	69.11504	13.823008
35	5	77.75442	15.550884
40	5	95.03319	19.006638
45	5	103.67256	20.734512
50	5	120.95132	24.190264
55	5	129.5907	25.91814
60	5	138.23009	27.646018
65	5	155.50883	31.101766
70	5	164.14822	32.829644
75	5	172.7876	34.55752
radius (cm)		2.75	

Figure 24: The speed in cm/s that the wheel moves with the corresponding large motor power

medium motor			
power	time (s)	cycle number	turning speed (radian/s)
0	5	0	0.000000
5	5	1	1.256637
10	5	2	2.513274
15	5	3	3.769911
20	5	4	5.026548
25	5	5	6.283185
30	5	6	7.539822
35	5	7	8.796459
40	5	8	10.053096
45	5	9	11.309734
50	5	11	13.823008
55	5	12	15.079645
60	5	13	16.336282
65	5	14	17.592919
70	5	15	18.849556
75	5	16	20.106193

Figure 25:The turning speed in radian/s that the pen holder turns with the corresponding median motor power

Appendix B

```
/*-----------------------------------C++ Program-------------------------*/
/*                                                                        */
/*    This program is created to accept basic functions and interpret     */
/*    them for a LEGO EV3 Robot to plot out on a graph. This robot        */
/*    will draw the curve on either a whiteboard or a piece of paper.    */
/*                                                                        */
/*    To start, users select the type of functions, which are             */
/*    polynomial, rational, trigonometric, exponential, and               */
/*    logarithmic. Then users would be prompted to determine the          */
/*    outer and inner coefficients, base, constant and degrees for       */
/*    functions. The output coordinates will be starting from x=-10 to  */
/*    x=10.                                                               */
/*                                                                       */
/*    NOTE:                                                              */
/*    There will be many times the word function and constant may be    */
/*    used.These terms do not refer to the programming side of things   */
/*    but rather the mathematical terms                                  */
/*                                                                       */
/*    PROGRAM ID: C++ PROGRAM                                            */
/*    PROGRAMMER: GROUP 8-1                                              */
/*    RUN DATE: NOVEMBER 22, 2019                                        */
/*                                                                       */
/*-----------------------------------------------------------------------*/
//Libraries
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <iomanip>
#include <fstream>

//Function prototypes
void getFunction();
int degreeInput();
int polynomialDegree(int *coefficients, int degree);
void processPolynomial(int *coefficients, int constant, int degree);
void processRational(int *coefficientsNumerator, int degreeNumerator,
                              int constantNumerator, int
*coefficientsDenominator,
                              int degreeDenominator, int
constantDenominator);
void trigonometric();
bool wantFunctionOrNah();
void trigFunctionOutput(int choice, int coefficient, int innerCoefficient,
int constant);
void processLogarithmic();
void processExponential();
bool endOfProgram();
```

```
using namespace std;

//main, calls the other functions
int main()
{
	//Main Menu
	cout << "Welcome to the robot graphing calculator" << endl;
	cout << "This program accepts basic functions and graphs the robot" <<
endl;
	cout << "These functions will generally be in the form of y=af(kx)+c"
<< endl;

	getFunction();
	system("PAUSE");
	return 0;
}//end main

//users select the functions then process the functions
void getFunction()
{
	int userInput = 0;
	do
	{
		//Continually checks for proper input
		do
		{
			cout << "Please pick an option" << endl;
			cout << "\t1. Polynomial function" << endl;
			cout << "\t2. Rational function" << endl;
			cout << "\t3. Trigonometric function" << endl;
			cout << "\t4. Logarithmic function" << endl;
			cout << "\t5. Exponential function" << endl;
			cin >> userInput;
		} while (userInput < 1 || userInput > 5);

		//checks the option the user entered
		if (userInput == 1)
		{
			//Find degree of the polynomial
			const int DEGREE = degreeInput();
			int coefficients[DEGREE];
			processPolynomial(coefficients, DEGREE,
polynomialDegree(coefficients, DEGREE));
		}

		else if (userInput == 2)
		{
			//rational function is in the form of f(x)/g(x)
			//Find degree of f(x)
```

```
                  cout << "Please enter the information for the numerator of
the rational function" << endl;
                  const int DEGREENUMERATOR = degreeInput();
                  int coefficientsNumerator[DEGREENUMERATOR];
                  int constantNumerator =
polynomialDegree(coefficientsNumerator, DEGREENUMERATOR);

                  //Find degree of g(x)
                  cout << "Please enter the information for the denominator
of the rational function" << endl;
                  const int DEGREEDENOMINATOR = degreeInput();
                  int coefficientsDenominator[DEGREEDENOMINATOR];
                  int constantDenominator =
polynomialDegree(coefficientsDenominator, DEGREEDENOMINATOR);

                  processRational(coefficientsNumerator, DEGREENUMERATOR,
constantNumerator,
                                                coefficientsDenominator,
DEGREEDENOMINATOR, constantDenominator);
            }
            else if (userInput == 3)
            {
                  trigonometric();
            }

            else if (userInput == 4)
            {
                  processLogarithmic();
            }
            else if (userInput == 5)
            {
                  processExponential();
            }
      } while (endOfProgram());
      cout << "Thank you for using the program" << endl;
      cout << "Remember to press any button on the robot for it to start
graphing" << endl;
}

//Allows the user to choose a degree of their function(s)
int degreeInput()
{
      int degree = 0;
      cout << " What degree would you like your function to be?" << endl;
      do
      {
            cout << " Please enter a number between 1 and 5 inclusive?" <<
endl;
            cin >> degree;
      } while (degree < 1 || degree > 6);
```

```
	cout << " You have chosen to make a " << degree << "th degree function"
<< endl;
	return degree;
}

//Procceses the degree of the polynomial and shows the user their polynomial
//Returns the constant value, c , of the polynomial
int polynomialDegree(int *coefficients, int degree)
{
	int constant = 0;
	bool want = 0;
	//Loop makes sure the user is satisfied with their polynomial
	do
	{
		//Accepts each coefficient of the function and then the constant
value c
		for (int countDegree = degree - 1; countDegree >= 0; countDegree-
-)
		{
			do {
			cout << "Please enter the coefficient for x^" <<
countDegree + 1 << endl;
			cout << "between -5 and 5 inclusive" << endl;
			cin >> coefficients[countDegree];
			}while(coefficients[countDegree] < -5 ||
coefficients[countDegree] > 5 );
		}
		cout << "Please enter the constant value c" << endl;
		cin >> constant;

		cout << "Your polynomial is ";

		//Prints out the function for the user to see
		for (int countDegree = degree - 1; countDegree >= 0; countDegree-
-)
		{
			if (countDegree != 0 && coefficients[countDegree] != 0)
				cout << coefficients[countDegree] << "x^" <<
countDegree + 1 << " + ";
			else if (countDegree == 0 && coefficients[countDegree] !=
0)
				cout << coefficients[countDegree] << "x"
					<< " + ";
		}
		cout << constant << endl;

		//Makes sure the user likes the polynomial
	} while (wantFunctionOrNah());
	return constant;
}
```

```
//Outputs points of the polynomial to the file
void processPolynomial(int *coefficients, int degree, int constant)
{
	ofstream fout("roboCoords.txt");
	double x = -10;
	double y = 0;
	//Goes through 20 x values and finds their respective y value
	for (x = -10; x <= 10; x++)
	{
		//Resets the y value
		y = 0;
		//Gets the y value
		for (int poly = 0; poly < degree; poly++)
		{
			y += (coefficients[poly] * pow(x, poly + 1)); //ax^n
		}
		y += constant;				//add the constant at the
end
		fout << x << " " << y << endl; // write the coordinate to the the
file for the robot
	}
	fout.close();
	cout << "The coordinates have been placed into the file
'roboCoords.txt' " << endl;
}

//Outputs points of the rational function to the file
void processRational(int *coefficientsNumerator, int degreeNumerator,
				int constantNumerator, int
*coefficientsDenominator,
				int degreeDenominator, int
constantDenominator)
{

	ofstream fout("roboCoords.txt");
	double x = -10;
	int yDenominator = 0;
	int yNumerator = 0;
	double y = 0;

	//Goes through 20 x values and finds their respective y value
	for (x = -10; x <= 10; x++)
	{
		//Resets the y value
		yDenominator = 0;
		yNumerator = 0;
		y = 0;

		//Gets the y value in the numerator of f(x)
```

```
            for (int poly = 0; poly < degreeNumerator; poly++)
            {
                  yNumerator += (coefficientsNumerator[poly] * pow(x, poly +
1));
            }
            yNumerator += constantNumerator;

            //Gets the y value in the denominator of g(x)
            for (int poly = 0; poly < degreeDenominator; poly++)
            {
                  yDenominator += (coefficientsDenominator[poly] * pow(x,
poly + 1));
            }
            yDenominator += constantDenominator;

            //Gets the y value using f(x)/g(x)
            y = (double)yNumerator / yDenominator;

            fout << fixed << setprecision(4) << x << " " << y << endl; //
write the coordinate to the the file for the robot
      }
      fout.close();
      cout << "The coordinates have been placed into the file
'roboCoords.txt' " << endl;
}

//process trigonometric function
void trigonometric()
{
      int choice = 0, coefficient = 0, innerCoefficient = 0, constant = 0;

      do
      {
            do
            { //continually checks to make sure the user enters proper input
                  cout << "Please choose a trigonometric function" << endl;
                  cout << "\t1. sin" << endl;
                  cout << "\t2. cos" << endl;
                  cout << "\t3. tan" << endl;
                  cin >> choice;
            } while (choice < 1 || choice > 3);

            do {
            cout << "please enter the coefficent a for y=af(x)" << endl;
            cout << "between -5 and 5 inclusive" << endl;
            cin >> coefficient;
            }while(coefficient > 5 || coefficient < -5);
```

```
            cout << "Please enter the inner coefficient k for y=f(kx)" <<
endl;
            cin >> innerCoefficient;
            cout << "Please enter the constant c for y=f(x) + c " << endl;
            cin >> constant;
            if (choice == 1)
            {
                  cout << "y=" << coefficient << "sin(" << innerCoefficient
<< "x) + " << constant << endl;
            }
            else if (choice == 2)
            {
                  cout << "y=" << coefficient << "cos(" << innerCoefficient
<< "x) + " << constant << endl;
            }
            else
            {
                  cout << "y=" << coefficient << "tan(" << innerCoefficient
<< "x) + " << constant << endl;
            }

      } while (wantFunctionOrNah());
      trigFunctionOutput(choice, coefficient, innerCoefficient, constant);
}

//Outputs points of trigonometric function a file
void trigFunctionOutput(int choice, int coefficient, int innerCoefficient,
int constant)
{
      ofstream fout("roboCoords.txt");
      for (double x = -10; x <= 10; x++)
      {
            fout << x << " ";
            if (choice == 1)
            {
                  fout << coefficient * sin(innerCoefficient * x) + constant;
            }
            else if (choice == 2)
            {
                  fout << coefficient * cos(innerCoefficient * x) + constant;
            }
            else
            {
                  fout << coefficient * tan(innerCoefficient * x) + constant;
            }
            fout << endl;
      }
      fout.close();
      cout << "The coordinates have been placed into the file
'roboCoords.txt' " << endl;
```

```
}

//Procceess the logarithmic function
void processLogarithmic()
{
	int base = 0, coefficient = 0, innerCoefficient = 0, constant = 0;
	do
	{
		cout << "please enter the base of logarithmic function: " <<
endl;
		cin >> base;
		do {
		cout << "please enter the coefficent a for y=af(x)" << endl;
		cout << "between -5 and 5 inclusive" << endl;
		cin >> coefficient;
		}while(coefficient > 5 || coefficient < -5);
		cout << "please enter the inner coefficent k for y=f(kx)" <<
endl;
		cin >> innerCoefficient;
		cout << "Please enter the constant c for y=f(x) + c " << endl;;
		cin >> constant;

		//print the function
		cout << "y = " << coefficient << "log" << base << "("
			<< innerCoefficient << "x) + " << constant << endl;
	} while (wantFunctionOrNah());

	ofstream fout("roboCoords.txt");
	for (double x = -10; x <= 10; x++)
	{
		fout << x << " " << coefficient * (log(innerCoefficient * x) /
log(base)) + constant << endl;
	}
	fout.close();
	cout << "The coordinates have been placed into the file
'roboCoords.txt' " << endl;
}

//Proccesses the exponential function and writes to the file
void processExponential()
{
	int base = 0, coefficient = 0, constant = 0;
	double x = -10;
	double y = 0;
	do
	{
		cout << "Please enter the base of the exponential function" <<
endl;
		cin >> base;
		do {
```

```
            cout << "Please enter the coefficient a for the fucntion
y=a(b)^x + c" << endl;
            cout << "between -5 and 5 inclusive" << endl;
            cin >> coefficient;
            }while(coefficient< -5 || coefficient> 5 );
            cout << "Please enter the constant c for y= (b)^ x + c" << endl;
            cin >> constant;

            cout << "The expoential function you have entered is " <<
coefficient << "(" << base << ") ^ ";
            cout << "x + " << constant << endl;
      } while (wantFunctionOrNah());

      ofstream fout("roboCoords.txt");
      for (x = -10; x <= 10; x++)
      {
            y = 0;
            fout << x << " " << coefficient * (pow(base, x)) + constant;
      }
      fout.close();
      cout << "The coordinates have been placed into the file
'roboCoords.txt' " << endl;
}

//Checks if the user wants the function or not
bool wantFunctionOrNah()
{
      int choice = 0;
      cout << "Are you sure you want this function?" << endl;
      cout << "Please pick an option" << endl;
      cout << "\t1. Yes" << endl;
      cout << "\t2. No" << endl;
      cin >> choice;
      return choice == 2;
}

//Checks if the user wants to end the program
bool endOfProgram()
{
      int choice = 0;
      cout << "Please upload the file to the robot" << endl;
      cout << "and press the button on the robot for it to start graphing" <<
endl;
      cout << "You may now either end the program and run the robot or" <<
endl;
      cout << "enter a new function for the robot which will overwrite the
old one" << endl;
      cout << "Please pick an option" << endl;
      cout << "\t1.End program" << endl;
      cout << "\t2.Re-enter the function" << endl;
```

```
	cin >> choice;
	return choice == 2;
}
```

Appendix C

```
/*------------------------------RobotC Program-------------------------*/
/*    This program use structs and functions. This program will read the */
/*    file generated from C++ program and draw the graph on the paper    */
/*                                                                       */
/*    To start, the robot will adjust itself to origin position, then    */
/*    draw the Cartesian Plane. The robot then starts the actual graphing */
/*    after it finishes the Cartesian Plane, starting from the origin.   */
/*    The robot will draw the graph on positive x-axis first and then come*/
/*    back to the origin to draw the graph on negative x-axis.           */
/*                                                                       */
/*    SETTING IN THIS PROGRAM:                                           */
/*    1.The side that has pen holder is the front side of the robot      */
/*    2.the starting position of the pen holder of the robot: pen down   */
/*    3.the direction paper moves is x-axis, the direction pen moves     */
/*    is y-axis.                                                         */
/*    PROGRAM ID: DrawFunction                                           */
/*    PROGRAMMER: GROUP 8-14                                             */
/*    RUN DATE: NOVEMBER 22, 2019                                        */
/*                                                                       */
/*-----------------------------------------------------------------------*/
#include "PC_FileIO.c"
//strucs
//xy coordinates
typedef struct{
	float x;
	float y;
}xyCoord;

//motor speed
typedef struct{
	tMotor port;
	int speed;
}motorInfo;

//declare global constants
const int NUM_OF_COORDS=20;
const int MAX_ARM_COUNTS=8*(180/(PI*2.75)); //prevents arm from going off
page
//the max_arm_counts is 9 since the maximimum horiozntal movement of the arm
wheel is 9cm
const int MIN_ARM_COUNTS=0;

//Function Prototypes
void motorSpeed(motorInfo & m , float newSpeed);
```

```
void readFile(xyCoord* coords);
void pressAndReleaseButton();
void moveToOrigin(motorInfo&arm,motorInfo&paper,motorInfo&pen);
void drawCartesianPlane (motorInfo&arm, motorInfo&paper, motorInfo&pen);
void rotatePenUpAndDown (int upOrDown,motorInfo&pen);
void graph(xyCoord* coords,motorInfo&arm,motorInfo&paper,motorInfo&pen);
void moveToNextCoord(float numX,float numY,int
yOld,motorInfo&paper,motorInfo&arm);
void driveToInfinity(float numX,float numY, int yOld, int
xOld,motorInfo&paper,motorInfo&arm);
bool undefined(float num);
bool toInfinity(float num, int motorEncoderValue);
bool colorTest();
void end(int time,motorInfo&arm,motorInfo&paper,motorInfo&pen);

//Controls calling of other functions
task main() {
      //naming motors
      motorInfo pen; //motor controls the movement of pen holder
      pen.port = motorC;
      pen.speed = 0;

      motorInfo paper; //motor controls the movement ot paper
      paper.port = motorB;
      paper.speed = 0;

      motorInfo arm; //motor controls the movement of arm
      arm.port = motorD;
      arm.speed = 0;

      //sensor type and sensor mode
      SensorType[S1] = sensorEV3_Color;
      wait1Msec(50);
      SensorMode[S1] = modeEV3Color_Color;
      wait1Msec(50);

      //Start a timer to see how long it takes to graph
      time1[T1]=0;

      //Variables
      xyCoord coords[NUM_OF_COORDS];

      //initialize variables
      for(int row=0; row<NUM_OF_COORDS;row++){
            coords[row].x=0;
            coords[row].y=0;
      }
      int count=0;
            while( colorTest() &&count < 1){
            //function call
```

```
            //starting robot
            readFile(coords); //read file and store data into arrays
            displayString(3,"Please press the down button to start the
robot");
            //press button to start moving robot
            pressAndReleaseButton();
            //robot move to origin
            moveToOrigin(arm,paper,pen);
            //robot draws Cartesian Plane
            drawCartesianPlane(arm,paper,pen);
            //actual graphing
            graph(coords,arm,paper,pen);
            count++;
      }

      //ending robot
      end(time1[T1],arm,paper,pen);
      rotatePenUpAndDown(2,pen); //rotate pen down as the ending position,
get ready for the next grpah
}//end of main

//set robot speed
void motorSpeed (motorInfo & m , float newSpeed){
      m.speed = newSpeed;
      motor[m.port] = m.speed;
}

//Reads the data from the file
void readFile(xyCoord* coords){
      TFileHandle fin;
      int f = 0;
      int f2 = 0;
      //Check if file can open
      if(!openReadPC(fin,"roboCoords.txt")) {
            displayString(3,"Error with the file");
      }
      else{
            //store all coordinates in an array
            for(int numCoords=0;numCoords<NUM_OF_COORDS;numCoords++){
            readIntPC(fin,f);
            readIntPC(fin,f2);
                  coords[numCoords].x=f;
                  coords[numCoords].y=f2;
            }
      }
      closeFilePC(fin);
}

//Waits for user to push button and release
void pressAndReleaseButton(){
```

```
	while(!getButtonPress(buttonAny)){}
	while(getButtonPress(buttonAny)){}
	displayString(3,"");
}

//move the robot to the origin of the cartesian plane
void moveToOrigin(motorInfo&arm, motorInfo&paper, motorInfo&pen){
	//rotate pen up before moving the robot to the starting position
	rotatePenUpAndDown(1,pen);
	//move the robot arm to the RIGHT end
	nMotorEncoder[arm.port] = 0;
	wait1Msec(500);
	motorSpeed(arm ,50);
	while(nMotorEncoder[arm.port] < MAX_ARM_COUNTS){}
	motorSpeed(arm,0);
	wait1Msec(2000);
	//move the robot arm to the CENTER of the horizontal graphing range
	motorSpeed(arm,-30);
	while(nMotorEncoder[arm.port] > MAX_ARM_COUNTS/2){}

	motorSpeed(arm ,0);
}

//rotate pen holder up and down
void rotatePenUpAndDown (int upOrDown, motorInfo&pen){
	//starts a timer to record the time taken to rotate the pen up and down
	time1[T2] = 0;
	//when the motor power is set to be 5, it takes 5 seconds to rotate a
full cycle
	//rotate pen up 90 degrees
	if (upOrDown == 1){
		motorSpeed(pen,5);
	}
	//rotate pen down
	else if (upOrDown == 2){
		motorSpeed(pen,-5);
	}
	while(time1[T2]<1250){}
	motorSpeed(pen,0);
	wait1Msec(1000);
}

//robot draws the cartesian plane
void drawCartesianPlane (motorInfo&arm, motorInfo&paper, motorInfo&pen){
	//rotate pen down when drawing cartesian plane
	rotatePenUpAndDown (2,pen);

	//variable
	const int ARM_COUNT = 4*(180/(PI*2.75)); //half of the horizontal
movement range
```

```
	nMotorEncoder[arm.port] = 0;
	nMotorEncoder[paper.port] = 0;

	for(int count=0; count<2; count++){
		if(count==0){     //draw +y axis
			motorSpeed(arm,20);
			while(nMotorEncoder[arm.port] < ARM_COUNT){}
		}

		else if(count==1){      //draw -y axis
			motorSpeed(arm,-30);
			while(nMotorEncoder[arm.port] > 0) {} //back to the origin
			motorSpeed(arm,0);
			rotatePenUpAndDown(2,pen); //pen down
			motorSpeed(arm,-20);
			while(nMotorEncoder[arm.port] > -ARM_COUNT){}
		}
		motorSpeed(arm,0);
		rotatePenUpAndDown(1,pen); //pen up
		wait1Msec(5);
	}

	for(int count=0; count<2; count++){
		if(count==0){//draw +x axis
			motorSpeed(arm,30);
			while(nMotorEncoder[arm.port] < 0){} //back to the origin
			motorSpeed(arm,0);
			rotatePenUpAndDown(2,pen); //pen down
			motorSpeed(paper,20);
			while(nMotorEncoder[paper.port] < ARM_COUNT){}
		}
		else if(count==1){//draw -x axis
			motorSpeed(paper,-30);
			while(nMotorEncoder[paper.port] > 0){} //back to the origin
			motorSpeed(paper,0);
			rotatePenUpAndDown(2,pen); //pen down
			motorSpeed(paper,-20);
			while(nMotorEncoder[paper.port] > -ARM_COUNT){}
		}
		motorSpeed(paper,0);
		rotatePenUpAndDown(1,pen); //pen up
		wait1Msec(5);
	}
	motorSpeed(paper,30);
	while(nMotorEncoder[paper.port] < 0){} //back to the origin
	motorSpeed(paper,0);
	wait1Msec(5);
}
```

```
//Function to graph everything
void graph(xyCoord* coords,motorInfo&arm,motorInfo&paper,motorInfo&pen) {
	nMotorEncoder[arm.port]=0;

	//variables
	int yOld = 0;
	int xOld = 0;
	int startPoint = 11;
	int endPoint = NUM_OF_COORDS;

	rotatePenUpAndDown(2,pen); //pen down
	if(coords[0].x > 0){ //if log function
		startPoint = 0;
		endPoint = 10;
	}

	//draw the coordinates in +x

	for(int numCoords=startPoint;numCoords<=endPoint;numCoords++){
		yOld = coords[numCoords-1].y;
		xOld = coords[numCoords-1].x;
		if(!undefined(coords[numCoords].y)){

	moveToNextCoord(coords[numCoords].x,coords[numCoords].y,yOld,paper,arm)
;
			//stops the robot briefly to reset
			motorSpeed(arm ,0);
			motorSpeed(pen ,0);
			motorSpeed(paper ,0);
			wait1Msec(1000);
		}
		else if(toInfinity(coords[numCoords].y,nMotorEncoder[arm.port])){

	driveToInfinity(coords[numCoords].x,coords[numCoords].y,yOld,xOld,paper
,arm);
			//exit the for loop when it goes to first coordinate that
has y value out of range
			numCoords = NUM_OF_COORDS+1;
		}
	}


	if(coords[0].x<=0){ //log function will not be included
		//return to origin before drawing the coordnates in -x
		moveToOrigin(arm,paper,pen);

		rotatePenUpAndDown(2,pen); //pen down
		//draw the coordinates in -x
		for(int numCoords = 9;numCoords>=0;numCoords--){
```

```
                  yOld = coords[numCoords+1].y;
                  xOld = coords[numCoords+1].x;
                  if(!undefined(coords[numCoords].y)){

      moveToNextCoord(coords[numCoords].x,coords[numCoords].y,yOld,paper,arm)
;
                  motorSpeed(arm ,0);
                  motorSpeed(pen ,0);
                  motorSpeed(paper ,0);
                  wait1Msec(1000);
                  }

                  else
if(toInfinity(coords[numCoords].y,nMotorEncoder[arm.port])){

      driveToInfinity(coords[numCoords].x,coords[numCoords].y,yOld,xOld,paper
,arm);
                        //exit the for loop when it goes to first coordinate
that has y value out of range
                        numCoords = -1;
                  }
            }
            rotatePenUpAndDown(1,pen); //pen up
      }
}

//Checks if a coordinate is viable in range
bool undefined(float num){
      return(num>10 || num < -10);
}

//Checks if the next coordinate goes to infinity
bool toInfinity(float num, int motorEncoderValue) {
      return(num>10 && motorEncoderValue < MAX_ARM_COUNTS ||
      num<0 && motorEncoderValue > MIN_ARM_COUNTS);
}

//drive to next coordinates
void moveToNextCoord(float numX, float numY,int
yOld,motorInfo&paper,motorInfo&arm){
      // check what is the y-value for when x=0, and move to the position
      // rotate down the pen to start
      time1[T2] = 0;
      // Draw connection between two points
      motorSpeed(paper, 5.18711*numX);
      motorSpeed(arm , ((2*((numY -yOld)) + 0.4701) /0.4762));
      while( time1[T2] < 200){}
      // The distance for every x-value is 0.4 cm
      // The distance foe every y-value is (y-value new - y-value old)*0.4 cm
      // Speed for the arm and the paper should be uniform
```

```
}

//drive to infinity
void driveToInfinity(float numX, float numY, int yOld, int
xOld,motorInfo&paper,motorInfo&arm){

      float m = 0, b = 0, xNew = 0, speedX = 0, speedY = 0;
      // Find a linear equation between two points
      // The out of range point and the point before it
      m = (numX - xOld)/(numY - yOld) ;
      b = numY - m*numX;
      // Find the corresponding x value when y = 8
      // Calculate the proper speed for the arm and the paper to move
      if (numY > 8){
            xNew = (8-b)/m;
            speedX = (xNew - xOld)*0.4/0.2;
            speedY = ((8-yOld) * speedX) /(xNew - xOld);
      }
      // Find the corresponding x value when y = -8
      // Calculate the proper speed for the arm and the paper to move
      if (numY < -8){
            xNew = (-8-b)/m;
            speedX = (xNew - xOld)*0.4/0.2;
            speedY = (fabs(-8-yOld) * speedX) /(xNew - xOld);
      }

      time1[T2] = 0;
      // Draw connection between two points
      motorSpeed(arm , (speedY + 0.4701)/0.4762);
      motorSpeed(paper , (speedX + 0.4701)/0.4762);
      while (time1[T2] < 200){}
}

//check for proper paper loaded
bool colorTest (){
      return SensorValue[S1] == (int)colorWhite;
}

//display message when the graph is finished
void end(int time,motorInfo&arm,motorInfo&paper,motorInfo&pen) {
      motorSpeed(arm ,0);
      motorSpeed(pen ,0);
      motorSpeed(paper ,0);
      displayString(3, "Finished graphing");
      displayString(5,"The program ran for %d seconds",time/1000);
      displayString(7,"Please press any button to end the program");
      pressAndReleaseButton();
}
```