THE LETTERING ROBOT

FINAL REPORT

December 4, 2017

Group Number: 854

Group Members: Ayesha Ebrahim, Laila Hashi, Sumaiya Islam, Jessenia Becerra

Student Numbers: 20724712, 20729987, 20732102, 20681720

Ayesha Ebrahim

[illegible]

Laila Hashi

[illegible]

Sumaiya Islam

[illegible]

Jessenia Becerra

[illegible]

Table of Contents

List of Figures

List of Tables

1 SUMMARY

In order to provide a tool that writes simple lettering, our plotter robot was created. The plotter was built through a process that incorporated both mechanical and software design. The mechanical design was a result of an analysis of three design alternatives. One of these designs were implemented. Dev C++ was used to produce a word file which contained the vector coordinate systems for each letter in the word. This word file was read and interpreted by the RobotC program so that the word could be plotted by the robot. The testing and troubleshooting procedure consisted of isolating functions and testing them separately to identify and fix bugs. Moreover, Gantt charts were created to keep track of deadlines and milestones. Overall, there were many setbacks along the way. Nonetheless, we still managed to plot most of the letters in the alphabet.

2 DESIGN PROBLEM AND DEFINITION

In life, many individuals come across various instances in which they are required to write aesthetically pleasing lettering but do not have the skills or time to do it themselves. This can be applied in many different scenarios such as organizing a wedding which requires beautiful writing on invitations and name place cards. For this reason, we attempted to build a robot that has the ability to write with a brush pen in simple calligraphy.

3 CONSTRAINTS AND CRITERIA

3.1 Constraints

In order for the robot to function efficiently and effectively many constraints were put into place. These constraints include: the ability to hold the brush pen, write all the letters in the alphabet legibly, apply two different pressures onto the brush pen, move in three different directions and be able to write at least one word. The constraint in which the robot must move in three different directions was the main concern that helped guide the project. The reason for this is because the robot would be unable to write a letter without the movement of the brush pen in the x-, y- and z-directions. On the other hand, a constraint that should have been added is that the movement of the robot in all three directions must be stable. This is mainly because the end result of the lettering is based on how well the robot is able to move in these three directions. The robot ended up plotting poorly because the conveyor belt that was used to move the robot in the x-direction was unstable and jerked while moving. A constraint that was not valuable to our final project is that of applying two different pressures onto the brush pen. During testing it was discovered that the bush pen already creates two different thicknesses by default, thus proving to be unnecessary.

3.2 Criteria

The main goal of our robot was to create aesthetically pleasing lettering. In order to achieve this goal, we had established criteria to ensure that the output of our robot's lettering is of high quality. These criteria included: programming the line thicknesses to gradually go from thick to thin, being able to write both upper and lower case letters, and lastly the ability to write more than one word. Since calligraphy is mainly dependent on the various line thicknesses and how they blend together, this was the criteria that we mainly focused on achieving. A big part of this was how we chose to position our brush pen in order to maximize the result of our outputted line thicknesses. A criterion that was not valuable throughout the process of creating our robot was that of writing both upper and lower-case letters. Although it would have increased the versatility of our robot, it would have required creating additional point patterns for the lower-case letters. Seeing that we were having trouble with plotting only the upper-case letters, it was deemed to be unimportant. Also, the criterion of writing more than one word was not taken into consideration because the x- and y- direction on the robot did not have enough range of motion to write words greater than five letters, let alone more than one line. Therefore, a criterion that should have been taken into consideration was to increase the number of characters that could be written on the page.

4 MECHANICAL DESIGN

4.1 Design Process

Initially, three different designs were considered for our final project. These designs were created based on how we would move the brush pen in the x-, y- and z-directions. The methods that were considered were a conveyer belt, a rack and pinion gear system and wheels. In order to choose between the three designs, the following factors were taken into account: stability, ease of implementation, availability of parts and appearance. Stability was rated the highest because the stability of the robot directly affected the quality of the outputted lettering. Whereas appearance was rated the lowest because it does not affect the overall functionality of the robot. After weighing each design against these factors, we were able to make an educated decision on which design we should implement. The design that we chose incorporated a conveyor belt for the x-direction and a rack and pinion gear system for both the y- and z-directions as shown in Figure 4.1 below. The design process was a critical step in the planning of our robot because the main function of the robot heavily relies on how well it is able to move in the three directions.

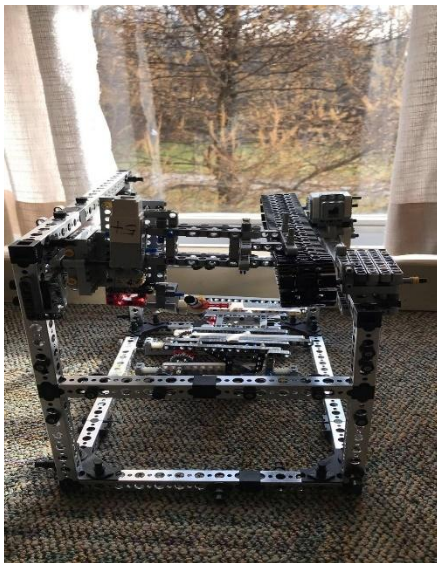


Figure 4.1 Final Design

4.2 Overall Assembly

To assemble the robot, we used Tetrix bars to build the skeleton. In the rectangular base, we placed a gear rack system for the y-direction. On the top, a metal rod, taken from the scrap resources in the student machine shop in E5, was held between two columns constructed from Tetrix pieces. A structure holding the brush pen and the z-motor slid across this rod. This structure contained a rack and pinion system to control the movement of the brush pen in the z-direction. Parallel and opposite to the same rod, a conveyor belt was attached along the top of the frame. On this frame the motor controlling the x-direction was mounted and connected to the conveyor belt. The structure on the rod had an arm that extended out and was attached to the conveyor belt so that when the conveyor belt moved, the structure glided across the rod. On the base, we placed two gear racks parallel to each other to create the path for the page holder. These gear racks were 3D printed from WatIMake (Figure 4.2) [1] [2]. The page holder was built upon a motor that was connected to gears such that the page holder and gear racks created a rack and pinion gear system in the y-direction.

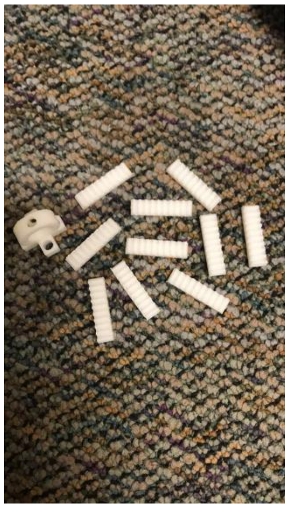


Figure 4.2 3D printed gear racks and pen holder

Throughout the assembling process we were focused in making each part durable and stable using the necessary number of pieces.

4.3 Base Frame Design

The overall frame of our robot was designed to make the base as structurally sound as possible. The initial idea was to use wheels for the y-direction; however, it proved to be inaccurate due to the amount of time it took to stop. Moreover, the size of this base was chosen such that it provided the movement in the x-direction enough leeway. The side that had the conveyor belt attached was set higher and the other side was staggered a little bit lower so that the pen holder was still aligned with the conveyor belt. In order to make sure that everything was attached securely, many triangular braces were used at the base of our robot to connect each piece. Additionally, screws were used to firmly lock pieces together.

4.4 Motor Driven Design

Our design used a total of three motors to power the movement of the brush pen in the x-, y- and z-directions.

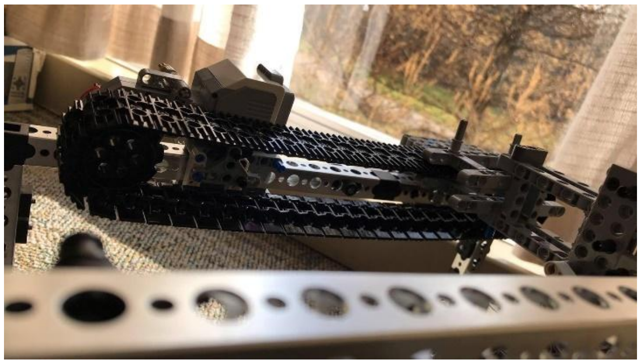


Figure 4.3 Conveyor belt system for x-direction

To start off, the motor used to power the x-direction was mounted to the top of our robot's main structure, shown in Figure 4.3 above. As stated previously, a conveyor belt was attached onto this motor and ran across the top of our robot in the x-direction. Two gears were attached on either side of the conveyor belt in order to guide it more smoothly across; however, due to the weight of the pen holder attached to the conveyor belt it continued to travel in rather jerky movements. This resulted in our robot's lettering in the x-direction to appear sloppy and in zigzags. In addition, we analyzed that the pen holder's motion to the right was steadier in comparison to when it was propelling to the left. One of the reasons that could have affected the x-direction movement is the fact that the motor was placed on the left side of the conveyor belt. We found that the conveyor belt given in the EV3 Kit could only travel smoothly in one direction because it was situated on the right side. We decided to keep the motor on left side because it increased the legibility, even though it was not completely accurate.

Furthermore, the second motor was used to power the y-direction of the robot, as seen in Figure 4.4 below. A gear rack was mounted at the bottom of our robot. The motor was then attached to the rack with lego rods and four gears that rolled on the top. A page holder was then created on top of the motor in order to move the page effectively across the y-direction of our robot. Due to the limited supply of gear racks provided within the lego kits we printed 12 of these using the 3D printing center in WATiMake. During the

testing stages of our robot we observed that the gear rack and pinion method for the y-direction was an excellent choice and drew each stroke with straight lines.

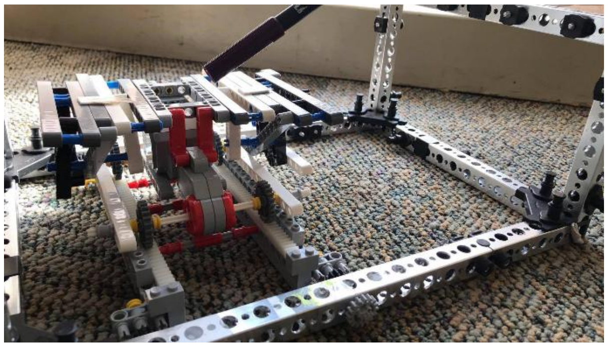


Figure 4.4 Y-direction system

Lastly, the final motor that was used was attached to the brush pen holder in order to move the pen on and off the page. A gear rack and pinion was also used to direct this motion of the robot. The overall pen holder structure was placed on a metal rod parallel to the conveyor belt. The pen holder structure was linked to the conveyor belt because the motor placed on the conveyor belt would be able to move it along the metal rod in the x-direction as seen in Figure 4.5. However, due to the pen holder weighing down the conveyor belt significantly and restricting its movements a considerable number of pieces were removed from the structure. Fortunately, the overall function of the z-direction was effective in moving the brush pen on and off the page due to use of a gear rack and pinion method.

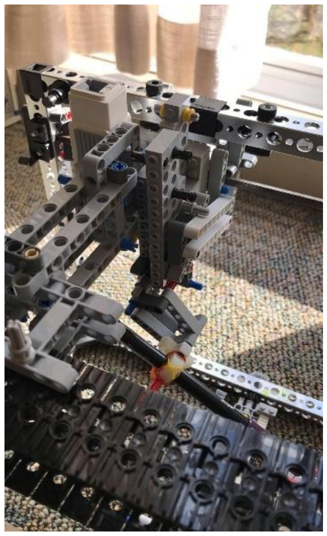


Figure 4.5 Pen holder structure

4.5 Sensor Attachment

The initial goal of the ultrasonic sensor was to detect the distance of the page relative to the brush pen. This would play a role in lifting the pen on and off the page as well as applying two different pressures. Unfortunately, the sensor was unable to function as we had hoped. Due to the close proximity to the page the ultrasonic sensor was unable to detect changes in the distance and always read in a value of 255. The sensor also added to the overall weight of the pen holder structure, thus causing additional problems for motion in the x-direction. Furthermore, after testing we observed that when positioned correctly the pen was still able to apply two different thicknesses onto the page without the help of the ultrasonic sensor calculating the distances. As a result of this, we decided to not include the ultrasonic sensor in our overall design.

5 SOFTWARE IMPLEMENTATION AND DESIGN

The overall function of the Dev-C++ program was to input a word from the user and then output to a word file the series of points corresponding to the vector path of each letter that the RobotC program could interpret. The RobotC program reads the file and plots each letter according to the points in the word file. The end result should be a completely plotted word.

5.1 Program Break-Down

Our programs were divided into smaller tasks in order to simplify the main function and make the programs easier to follow and understand. The design flowcharts can be found in Appendix 1 from pages 20-22.

5.1.1 Dev-C++ Portion

To program our Dev-C++ code, we chose to divide the program into three parts: letter_value function, make_word_file function and the main function. We divided up the code into these three parts because the code consists of many loops that could be hard to understand and distinguish if they had all been coded in main. Furthermore, by dividing up the code in this way, it was easier to identify bugs in our program as it allowed us to isolate the problem to a certain block of code. The purpose of the letter_value function is to find the numerical value that corresponds to the value we have assigned each letter, of an inputted character. This value is then used in the make_word_file function in order to be able to make the word file, containing all of the vector coordinates for that word, and then read by the RobotC program. In main, we decided to only initialize the three arrays used in the program and then call the make_word_file function. By doing so, we reduced the complexity of main and thus, increased its readability.

5.1.2 RobotC Portion

Similar to our Dev-C++ code, in RobotC we used functions to break up the blocks of tasks we needed to perform in task main. By doing so, we are able to clearly organise the different tasks we are performing and it is easy to take a look at the task main and understand how all the subtasks are performed together. The RobotC program has four functions, excluding main. The first function, move, is used to move the brush pen from a certain starting point to an end point. Secondly, the plot_letter function repeatedly calls the move function to plot a single letter. Next, there is the space function that creates white space after each letter has been completed. All of these functions are called in main to plot the entire word, along with the efficiency function that calculates and displays the number of letters the robot has plotted per minute.

5.2 Major Algorithm Design Decisions

5.2.1 Dev-C++ Portion

Within our Dev-C++ program, three arrays were created in order to make the word file. Firstly, the alphabet array is a one dimensional, character array that contains all the capital letters in alphabetical order. Secondly, the font array is a two-dimensional, double array that becomes initialized with the values from the font file. The font file contains all the points needed to create each letter. We chose to make it into an array rather than simply reading the file so that we are able to loop through the values in the file multiple times. Lastly, the third array is a one dimensional, character array called "word" that is initialized with the letters inputted by the user.

As aforementioned, we decided to use the alphabet array to find the numerical values of each letter the user inputs. This is done by looping through the alphabet array comparing the word array and the alphabet array values until the character in each is equal. The letter value is then the corresponding index. This letter value is used to look up the row index of the font file and the entire row in the font file is outputted to the word file.

5.2.2 RobotC Portion

Initially, in RobotC we planned to write 26 functions in order to be able to plot each letter of the alphabet. These functions would then be called within main to plot the letters inputted by the user. However, with the guidance of our tutor, Michael Stachowsky, we realized that this was a relatively rudimentary method to accomplish the task at hand. Therefore, we created a file that contains a set of coordinates that can be read and interpreted within the program in order for the robot to move to each point. Theoretically, this method would allow us to plot anything with a given set of coordinates.

Furthermore, while writing the space function we ran into many issues with our code. Initially, we had used motor encoders to calculate the distance needed to create the space between each letter, but after each letter was plotted, a space was not created and the pen would drag across the page in the opposite direction. Due to time constraints, we made the decision to use the "wait1Msec" function to perform this task. Despite this method being rather inefficient, we were able to successfully create the space that was needed.

5.3 Tradeoffs

Our software was created for the sake of readability and understandability; however, as a result of this our code became more inefficient. This is apparent when one focuses on the font file and the font array. Instead of simply using the array index as the letter value, we incorporated the letter value into the font file as the first number in each row so that it was easier for us to locate different letters in the font files and make changes to them when needed as each row was numbered.

5.4 Program Testing

5.4.1 Dev-C++ Portion

We coded and tested our C++ program simultaneously. For example, after programming each array we would output it to the screen to make sure that it was being initialized correctly. Furthermore, we tested each letter to make sure that the correct point pattern was being outputted to the font file. When there were errors, we observed the points being outputted to the word file for several different letters in order to outline the error pattern. This way we were able to pinpoint the specific problem.

5.4.2 RobotC Portion

Similarly, while coding the RobotC program we tested our code countless times to make sure that it worked. Firstly, to ensure that the robot was reading in the file correctly, we displayed the values being read into the program on the brick so that we could identify their correctness. After that, we coded the entire program and then started to debug the issues with our program that were causing the robot to plot incorrectly. To isolate the problems, each function was tested separately by commenting out the other functions. Through this process we discovered that our problems were contained in the space and move functions. Also, to test within the move function, we would display the start and endpoint being used in the move function to the brick. Additionally, benchmarks in the code were displayed to the brick so that we were able to map out where the problems were occurring in the code. This method helped us discover many issues, some of which we were able to solve and others we worked around instead of solving because of the time constraint.

Furthermore, for many aspects in the program, we found the values that worked best through trial and error. For instance, this was used to find the most suitable wait times to be used in the space function. Moreover, within the move function, we tested different motor powers and font scales so that the letters would be plotted clearly.

5.5 Significant Problems

5.5.1 Dev-C++ Portion

While testing the program, we encountered a few obstacles with the values from the font array being incorrectly outputted to the word file. We were certain that the issues were related to the actual outputting of the values into the word file because we checked all our arrays and they were correctly initialized. Knowing this, we were able to pinpoint the function that was programmed incorrectly: the make_word_file function. After debugging, we realized that one of the issues was that we were not taking into account the fact that the number of x and y values that needed to be outputted were equal to twice the number of coordinates in the row because in the font file, we treated each x and y pair as one coordinate. Similarly, the second issue was that for the loop to output values to the font file we made the condition so that it looped around for the number of coordinates multiplied by two. The issue was that we forgot to take into account the first two values in the row, the letter value and the number of coordinates, and therefore we were always missing one coordinate pair. We realized our mistake after trying many test cases and seeing the pattern of the lack of the final two coordinates.

5.5.2 RobotC Portion

The plotter had trouble plotting diagonal lines. Due to the time constraint, we were unable to solve this issue and thus, resorted to rewriting our font file so that all the letters consisted of horizontal or vertical lines. Furthermore, despite making sure that we reset our motor encoders in the space function, the motors performed seemingly unpredictable activities. For this reason, we decided to simply use the wait1Msec function to program the space between the letters which we found to be more reliable. The DevC++, RobotC and the font file can be found in the Appendix.

6 PROJECT MANAGEMENT

The tasks were initially split according to skills and preference of each person; however, we also made sure that each team member had experience with working on all components of the project, such as mechanical design, construction, coding and testing. This can be seen in the tables, Table 6.1 and Table 6.2, below.

Table 6.1 Division of labor-Mechanical Design

Task	Team member responsible
Base structure	All the members
Z-direction system (pen holder structure, rack and pinion, motor mounting)	Ayesha and Jessenia
X-direction system (conveyor belt, attachments on the left and right side)	Laila and Jessenia
Y-direction system (hexagonal structure, rank and pinion path)	Sumaiya

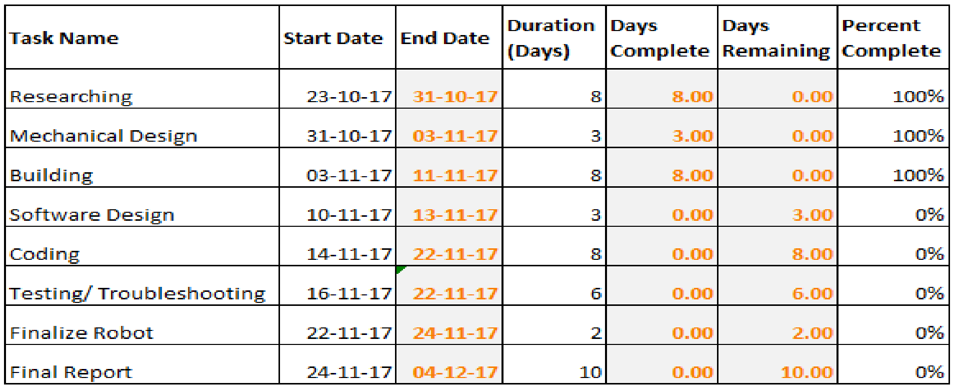
Table 6.2 Division of labor - Software

Task	Team member responsible
C++ and RobotC functions	All team members were equally assigned functions. The name of the person responsible is included as comments above each function.
Testing/troubleshooting in C++	Ayesha, Sumaiya and Laila
Testing/troubleshooting in RobotC	All team members.

For the duration of our project up until software design, we mostly followed the schedule presented in the interim report. We started to deviate from the time periods allocated in the schedule while working on writing the code. Here we realized it would take much less than eight days to write the code and more time to test and troubleshoot.

Since testing, troubleshooting and finalizing the robot were parallel tasks towards the end of the project, they lasted up until the demo day. Besides the changes in durations of certain modules, we were successfully able to meet all deadlines. Additionally, the final report will be completed by Sunday afternoon, resulting in a total of eight days to complete this task instead of an expected ten. The discrepancies in expected and actual progress can be seen in tables Table 6.3 [3[and Table 6.4 below. [4]

Table 6.3 Expected project timeline



Task Name	Start Date	End Date	Duration (Days)	Days Complete	Days Remaining	Percent Complete
Researching	23-10-17	31-10-17	8	8.00	0.00	100%
Mechanical Design	31-10-17	03-11-17	3	3.00	0.00	100%
Building	03-11-17	11-11-17	8	8.00	0.00	100%
Software Design	10-11-17	13-11-17	3	0.00	3.00	0%
Coding	14-11-17	22-11-17	8	0.00	8.00	0%
Testing/ Troubleshooting	16-11-17	22-11-17	6	0.00	6.00	0%
Finalize Robot	22-11-17	24-11-17	2	0.00	2.00	0%
Final Report	24-11-17	04-12-17	10	0.00	10.00	0%

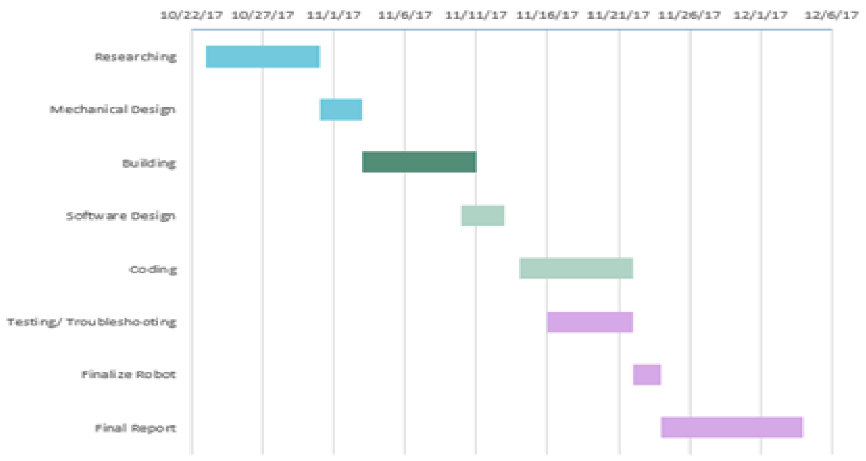
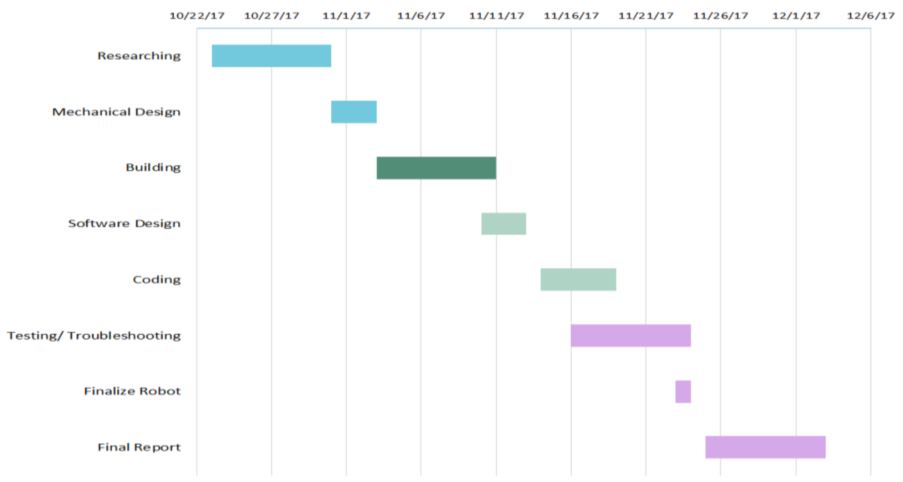


Table 6.4 Actual project timeline

Task Name	Start Date	End Date	Duration (Days)	Days Complete	Days Remaining	Percent Complete
Researching	10/23/2017	10/31/2017	8	8.00	0.00	100%
Mechanical Design	10/31/2017	11/3/2017	3	3.00	0.00	100%
Building	11/3/2017	11/11/2017	8	8.00	0.00	100%
Software Design	11/10/2017	11/13/2017	3	3.00	0.00	100%
Coding	11/14/2017	11/19/2017	5	5.00	0.00	100%
Testing/ Troubleshooting	11/16/2017	11/24/2017	8	8.00	0.00	100%
Finalize Robot	11/23/2017	11/24/2017	1	1.00	0.00	100%
Final Report	11/25/2017	12/3/2017	8	8.00	0.00	100%



7 CONCLUSIONS AND RECOMMENDATIONS

Overall, although our design did not completely solve the problem we aimed for, our robot was still able to write most of the letters in the alphabet legibly. We conducted a short survey during the demo day where spectators and our assigned tutor and TA all agreed on the readability and distinguishable thicknesses of the plotted lines as shown in Figure 7.1and Figure 7.2 below. During the testing stages we realized that it was a difficult task to get accurate results without the use of controllers, such as a proportional–integral–derivative (PID), which is taught in upper years. Thus, we were satisfied with the end result of our plotter robot.

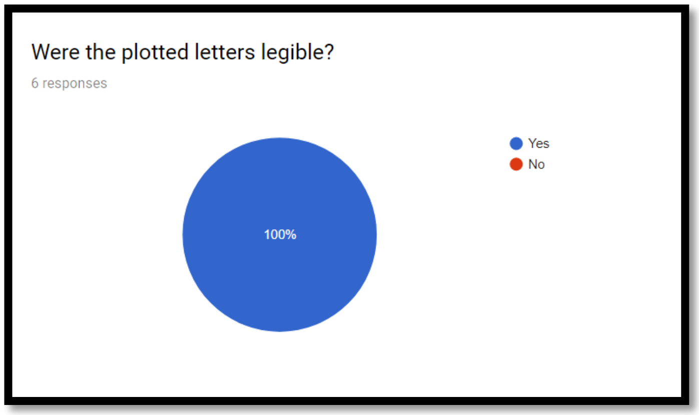


Figure 7.1 Survey results regarding legibility

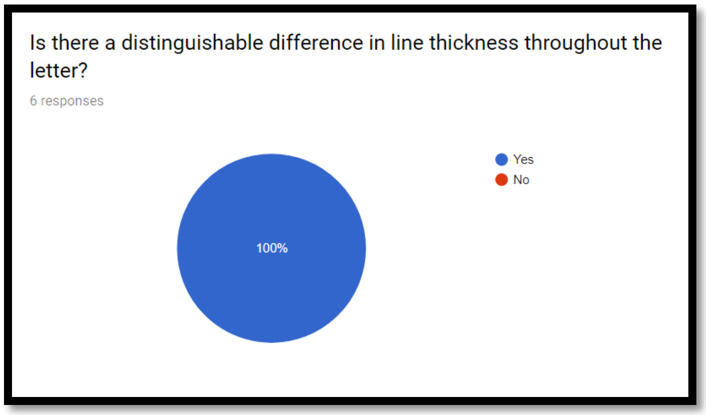


Figure 7.2 Survey results regarding thicknesses

During the testing process, we recognized that we should have chosen a gear rack and pinion system for the x-direction similar to the ones chosen for both the y- and z-directions. This would provide more stability as well as increase the range of motion that the brush pen is able to move across the page. With our existing combination of pen holder structure and x-direction system we were only able to plot a maximum of five letters before we had to manually reset the motors to the starting positions.

Due to time constraints, many tradeoffs were made during the testing and troubleshooting stages. When the robot was unable to plot diagonal lines, we decided to change the font file to exclude diagonals and only plot in horizontal and vertical lines; however, finding a way to plot at angles would improve the aesthetic appeal of the robot's lettering. Additionally, the space function created within RobotC could be further improved to incorporate motor encoders. This way we would calculate the distance between each letter accurately without employing the trial and error method for different power and wait combinations. Lastly, as further improvement to our robot we could make changes to the code to successfully string the plot letter and space functions together to enable the plotting of an entire word.

8 REFERENCES

[1] Thingiverse.com. “Customizable Lego Rack Gear by Buffington.” Thingiverse, Buffington, www.thingiverse.com/thing:58194.

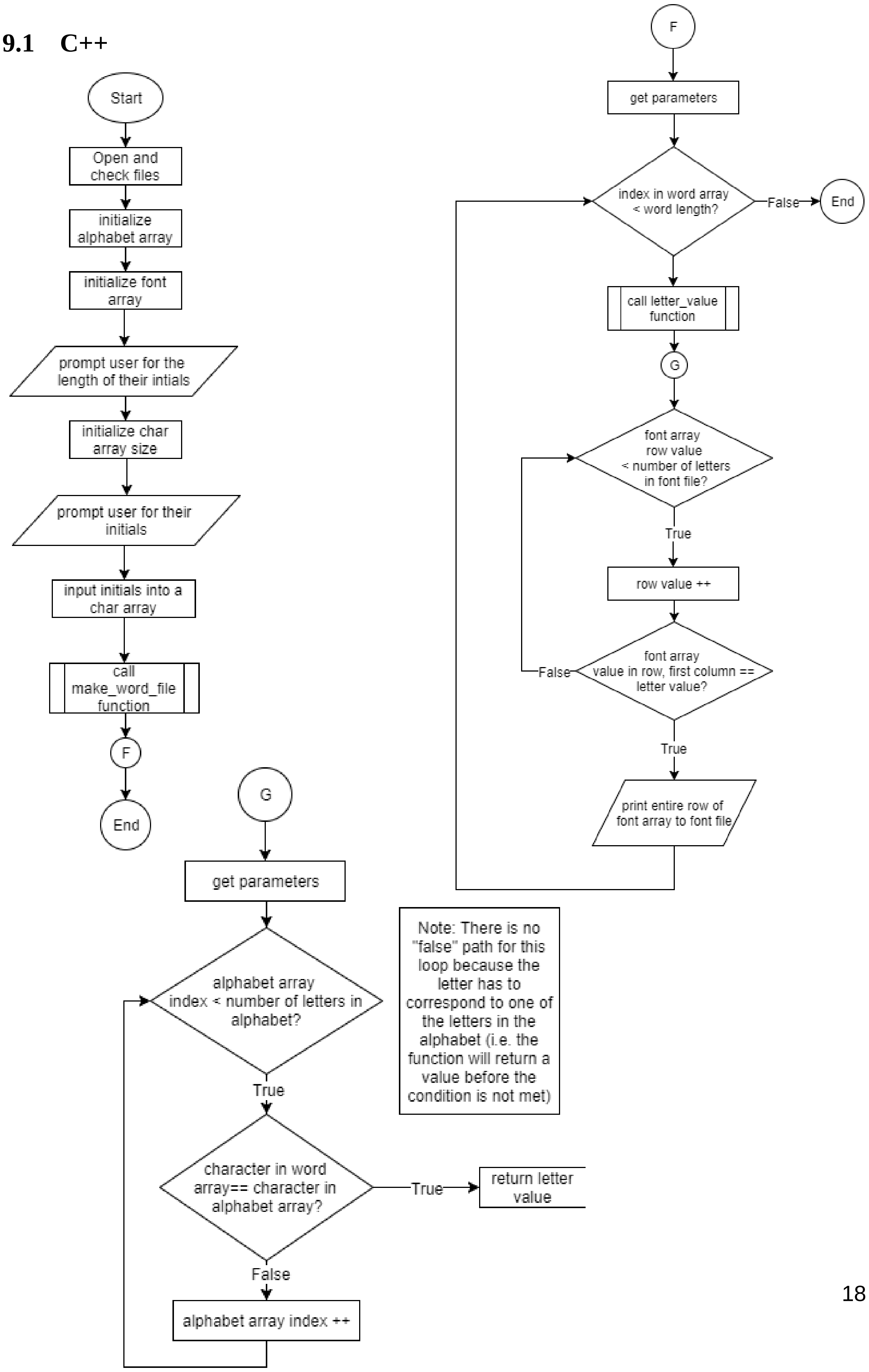
[2] Thingiverse.com. “LEGO Set Screw Pen Holder by Thetinkeringstudio.” Thingiverse, Thetinkeringstudio, www.thingiverse.com/thing:1486961.

[3] J. Becerra, A. Ebrahim, L. Hashi, S. Islam, "Final Project Interim Report", 2017.

[4] “The Best Free Gantt Chart Excel Template.” Free Gantt Chart Excel Template: Download Now | TeamGantt, www.teamgantt.com/free-gantt-chart-excel-template.

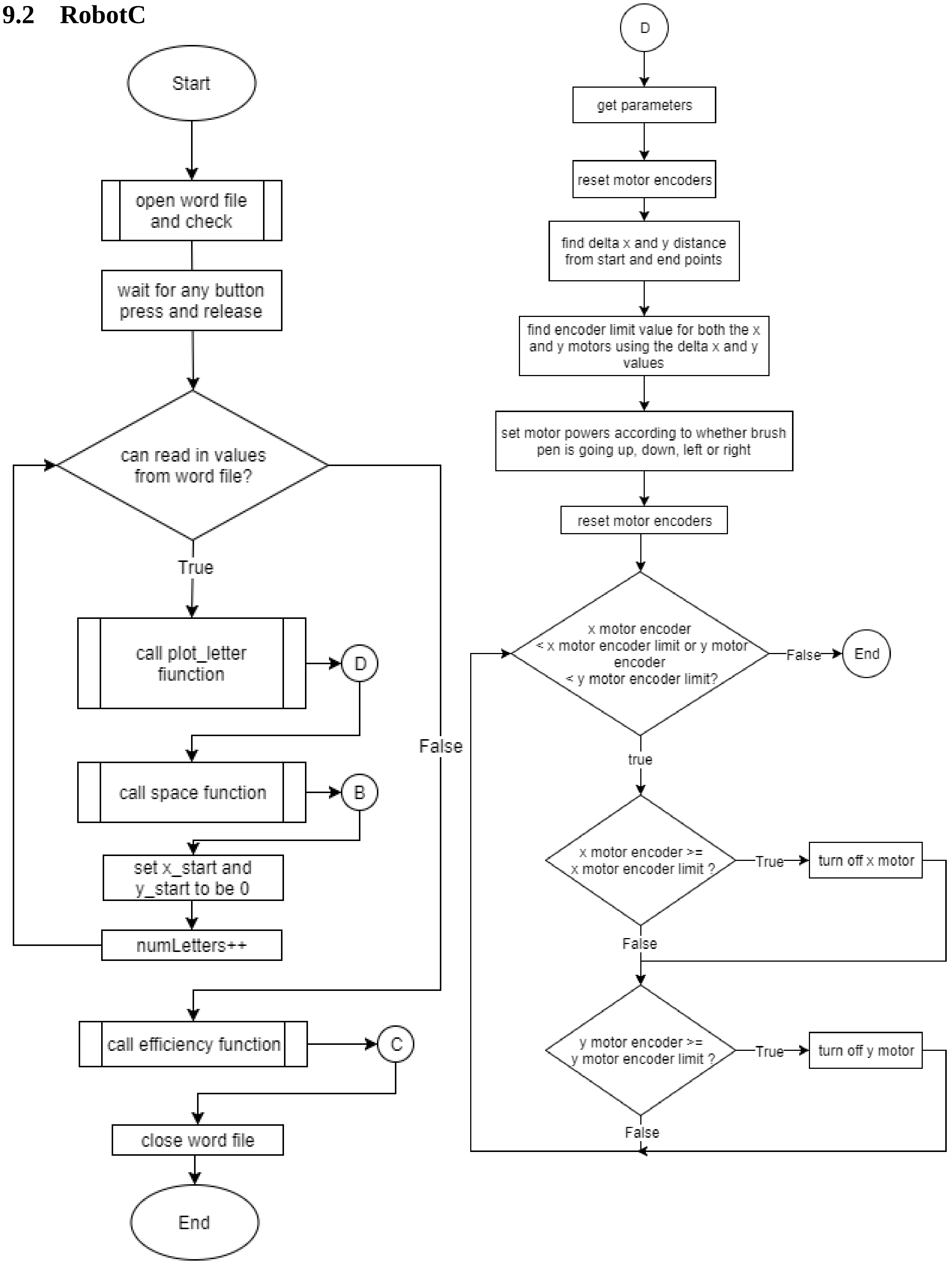
9 APPENDIX 1

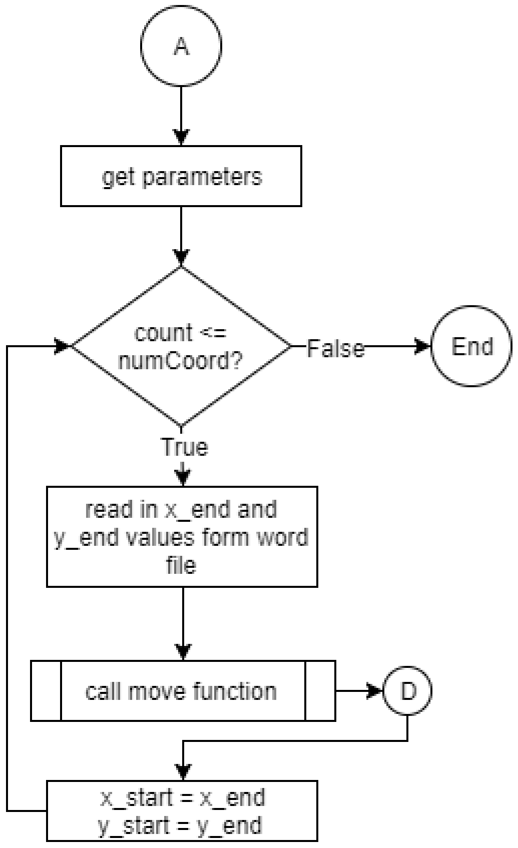
9.1 C++

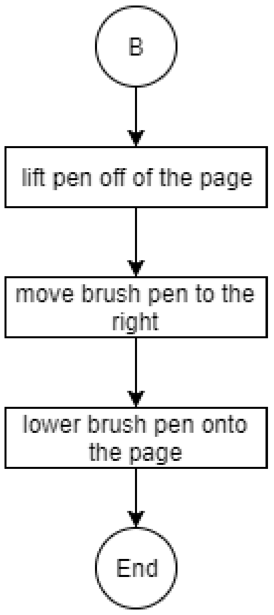


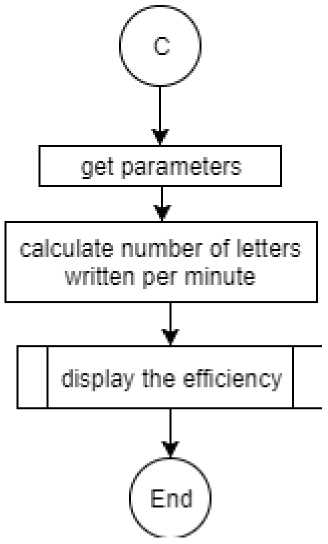
Start
Open and check files
initialize alphabet array
initialize font array
prompt user for the length of their intials
initialize char array size
prompt user for their initials
input initials into a char array
call make_word_file function
F
End
G
get parameters
alphabet array index < number of letters in alphabet?
True
character in word array== character in alphabet array?
True
return letter value
False
alphabet array index ++
Note: There is no "false" path for this loop because the letter has to correspond to one of the letters in the alphabet (i.e. the function will return a value before the condition is not met)
F
get parameters
index in word array < word length?
False
End
call letter_value function
G
font array row value < number of letters in font file?
True
row value ++
font array value in row, first column == letter value?
False
True
print entire row of font array to font file

9.2 RobotC


Start
open word file and check
wait for any button press and release
can read in values from word file?
True
call plot_letter fiunction
D
call space function
B
set x_start and y_start to be 0
numLetters++
False
call efficiency function
C
close word file
End
D
get parameters
reset motor encoders
find delta x and y distance from start and end points
find encoder limit value for both the x and y motors using the delta x and y values
set motor powers according to whether brush pen is going up, down, left or right
reset motor encoders
x motor encoder < x motor encoder limit or y motor encoder < y motor encoder limit?
False
End
true
x motor encoder >= x motor encoder limit ?
True
turn off x motor
False
y motor encoder >= y motor encoder limit ?
True
turn off y motor
False


A
get parameters
count <= numCoord?
False
End
True
read in x_end and y_end values form word file
call move function
D
x_start = x_end
y_start = y_end


B
lift pen off of the page
move brush pen to the right
lower brush pen onto the page
End


C
get parameters
calculate number of letters written per minute
display the efficiency
End

10 APPENDIX 2

10.1 C++

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

using namespace std;

const int NUM_LETTERS = 26;
const int MAX_COORD = 26;

//letter_value function - All team members
//find numerical value corresponding to letter
int letter_value (char alphabet[NUM_LETTERS], char word[], int word_length,
                           int index_word)
{
       char letter;
       letter = word[index_word];

       for (int index_alphabet = 0; index_alphabet < NUM_LETTERS;
index_alphabet++)
       {
              if(alphabet[index_alphabet] == letter)
              {
                     int letter_number = 0;
                     letter_number = index_alphabet;
                     return letter_number;
              }
       }
}

//make_word_file function - Ayesha Ebrahim
void make_word_file(ofstream & fout, double font[][MAX_COORD], char word[],
                                int word_length, char alphabet[])
{
       //loop through the the entire word
       for (int index_word = 0; index_word < word_length; index_word++)
       {
              //find the value assigned to letter
              int letter_number = letter_value (alphabet, word, word_length,
 index_word);
              int row = 0;
              row = letter_number;

              int col = 0;
              //number of coordinates is always in second column
              col = 1;

              int num_coord = 0;
              num_coord = font[row][col];
```

```
            //print line to word file
            for(int col = 0; col <= (num_coord*2 + 2); col++)
            //num_coord*2 because coord in pairs
            //+2 to account for letter and num_coord values
            {
                  fout << font[row][col] << " ";
            }

            fout << endl;
      }
}

//main - All team members
int main()
{
      ifstream fin_font("font_file.txt");
      ofstream fout("word_file.txt");
      if(!fin_font)
      {
            cout << "Error! Could not find file! " << endl;

            return EXIT_FAILURE;
      }

      char alphabet[NUM_LETTERS] =
{'A','B','C','D','E','F','G','H','I','J','K',

      'L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

      double font[NUM_LETTERS][MAX_COORD] = {0};

      //initialize the font array
      for(int row=0; row < NUM_LETTERS; row++)
      {
            int col = 0;
            int letter = 0;
            int num_coord = 0;

            fin_font >> letter >> num_coord;

            font[row][col] = letter;
            col++;
            font[row][col] = num_coord;

            //num_coord *2 because it is the number of pairs and add 2
because
            //we start at column 2 because coordinates start in third column
            for(col = 2; col < (num_coord*2 + 2); col++)
            {
                  double coord = 0;
                  fin_font >> coord;
                  font[row][col] = coord;
            }
      }
```

```
	int word_length=0;

	//verification loop for word length
	do
	{
		//get word
		cout << "Please enter the number of letters in your initials (max
2)."
			<< endl;
		//converting word into characters and putting into an array
		cin >> word_length;
	}
	while(!(word_length <= 2 && word_length >= 1 ));

	//MIKE SAYS THIS IS FINE
	char word[word_length]={};

	cout << "Please enter your initials in all capitals." << endl;

	for(int index = 0; index < word_length; index++)
	{
		cin >> word[index];
	}

	//outputting to word file
	make_word_file(fout, font, word, word_length, alphabet);

	fin_font.close();
	fout.close();

	return EXIT_SUCCESS;
}
```

10.2 RobotC

```
#include"EV3_FileIO.c"

const float RAD_X=2; //cm
const float RAD_Y=1.25; //cm

//x_s and y_s are the starting coordinates and x_e and y_e are the ending
coordinates

//Move Function - Laila Hashi
void move(float & x_s, float & y_s, float & x_e, float & y_e)
{

	float delta_x = fabs(x_e - x_s);
	float delta_y = fabs(y_e - y_s);


	const float FONT_SCALE_X = 1.5;
	const float FONT_SCALE_Y = 2;

	float enc_limit_x = FONT_SCALE_X*(delta_x*360)/(2*PI*RAD_X);
	float enc_limit_y = FONT_SCALE_Y*(delta_y*360)/(2*PI*RAD_Y);

	//these powers were decided by thorough testing
	float power_y = 5;
	float power_x = 4;
	float power_neg_x = 4;


	if (x_e < x_s) //right stroke
	{
		motor[motorA] = power_x;
	}
	else //left stroke
	{
		motor[motorA] = -power_neg_x;
	}
	if (y_e < y_s) //down stroke
	{
		motor[motorB] = -power_y;
	}
	else //up stroke
	{
		motor[motorB] = power_y;
	}

	nMotorEncoder[motorA] = nMotorEncoder[motorB] = 0;
	while(fabs(nMotorEncoder[motorA]) < enc_limit_x ||
fabs(nMotorEncoder[motorB]) < enc_limit_y)
	{
		if (fabs(nMotorEncoder[motorA]) >= enc_limit_x)
			motor[motorA] = 0;

		if (fabs(nMotorEncoder[motorB]) >= enc_limit_y)
			motor[motorB] = 0;
	}

}

//Plot letter function - Jessenia Becerra
void plot_letter(TFileHandle & fin, int numCoord, float & x_s, float & y_s)
{
	float x_e = 0;
	float y_e = 0;
	//numCoord - 1 because we already read in one coordinate in main
	for(int count=0; count < numCoord - 1; count++)
	{
		readFloatPC(fin, x_e);
		readFloatPC(fin, y_e);

		move(x_s, y_s,x_e,y_e);

		x_s = x_e;
```

```
            y_s = y_e;
        }

}

//Efficiency function - Jessenia Becerra
void efficiency(int time, int numLetters)
{
        //assume at least one letter was written so no division by 0
        float efficiency_time = (time/1000)/numLetters;
        displayString(0,"Letters per minutes %f" , efficiency_time);
        wait1Msec(2000);
}

//Space function - All team members
void space()
{
        motor[motorA]=motor[motorB]=0;
        wait1Msec(1000);

        //moving pen up
        motor[motorC] = -10;
        wait1Msec(1000);
        motor[motorC] =0;


        //moving pen to the right
        motor[motorA] = -5;
        wait1Msec(750);
        motor[motorA] = 0;

        wait1Msec(1000);

        //moving pen down
        motor[motorC] = 10;
        wait1Msec(1000);
        motor[motorC] =0;
}

//main - Sumaiya Islam
task main()
{
        TFileHandle fin;
        bool infileOkay = openReadPC(fin,"word_file.txt");

        float x_start=0, y_start=0, x_end=0, y_end=0;
        int numCoord=0, numLetters = 0, thickness = 0, letter = 0;

        //robot will start after any button is pressed
        while((!getButtonPress(buttonAny)))
        {}
        while(getButtonPress(buttonAny))
        {}

        ////figure out button stuff
```

```
	clearTimer(T1);

	while (readIntPC(fin, letter) && readIntPC(fin, numCoord) &&
readFloatPC(fin, x_end) && readFloatPC(fin, y_end))
	{
		plot_letter(fin, numCoord, x_start, y_start);

		space();
		x_start = 0;
		y_start = 0;

		numLetters++;
	}

	efficiency(time1[T1], numLetters);

	closeFilePC(fin);
}
```

11 APPENDIX 3

font_file.txt

```
0 6 0 0 0 1 1 1 1 0 1 0.5 0 0.5
1 8 0 0 0 1 0.75 1 0.75 0.5 0 0.5 1 0.5 1 0 0 0
2 9 0 0 0 1 1 1 1 0.75 1 1 0 1 0 0 1 0 1 0.25
3 5 0 0 0 1 1 1 1 0 0 0
4 9 0 0 0 1 1 1 0 1 0 0.5 0.5 0.5 0 0.5 0 0 1 0
5 8 0 0 0 1 1 1 0 1 0 0.5 0.5 0.5 0 0.5 0 0
6 10 0 0 0 1 1 1 1 0.75 1 1 0 1 0 0 1 0 1 0.25 0.75 0.25
7 6 0 0 0 1 0 0.5 1 0.5 1 1 1 0
8 3 0 0 0 1 0 0
9 5 0 0 0 0.5 0 0 1 0 1 1
10 8 0 0 0 1 0 0.5 0.75 0.5 0.75 1 0.75 0.5 1 0.5 1 0
11 4 0 0 0 1 0 0 1 0
12 7 0 0 0 1 0.5 1 0.5 0.5 0.5 1 1 1 1 0
13 4 0 0 0 1 1 0 1 1
14 5 0 0 0 1 1 1 1 0 0 0
15 5 0 0 0 1 1 1 1 0.5 0 0.5
16 8 0 0 0 1 1 1 1 0 0 0 0.75 0 0.75 0.25 0.75 -0.25
17 7 0 0 0 1 0.75 1 0.75 0.5 0 0.5 1 0.5 1 0
18 6 0 0 1 0 1 0.5 0 0.5 0 1 1 1
19 4 0 0 0 1 1 1 -0.5 1
20 5 0 0 0 1 0 0 1 0 1 1
21 4 0 0 -1 1 0 0 1 1
22 8 0 0 0 1 0 0 0.5 0 0.5 0.5 0.5 0 1 0 1 1
23 5 0 0 1 1 0.5 0.5 0 1 1 0
24 5 0 0 0 1 0 0.5 0.5 0.5 0.5 1
25 7 0 0 1 0 0 0 0 0.5 1 0.5 1 1 0 1
```