**FINAL REPORT – GROUP 425**

MTE 100 / GENE 121

William Kwong, Aniket Phanse, Bharath Srinivas, Ryan Zhou

Table of Contents

[List of Tables iv](#_Toc500156755)

[List of Figures v](#_Toc500156756)

[Acknowledgements vi](#_Toc500156757)

[1 Summary 1](#_Toc500156758)

[2 Design Problem Definition 2](#_Toc500156759)

[3 Constraints and Criteria 3](#_Toc500156760)

[3.1 Constraints 3](#_Toc500156761)

[3.2 Criteria 3](#_Toc500156762)

[4 Mechanical Redesign and Implementation 4](#_Toc500156763)

[4.1 Mechanical Design Outline 4](#_Toc500156764)

[4.2 Components 5](#_Toc500156765)

[4.2.1 Chassis Base 5](#_Toc500156766)

[4.2.2 Intake and gear assembly 6](#_Toc500156767)

[4.2.3 Chain and Printhead Assembly 7](#_Toc500156768)

[4.2.4 Assembly Integration 8](#_Toc500156769)

[5 Software Design and Implementation 9](#_Toc500156770)

[5.1 Design Description 9](#_Toc500156771)

[5.2 Testing the program 11](#_Toc500156772)

[5.3 Trade-offs and redesigns to the software 11](#_Toc500156773)

[6 Project Management 13](#_Toc500156774)

[6.1 Roles 13](#_Toc500156775)

[6.2 Timeline 13](#_Toc500156776)

[6.2.1 Initial project plan 13](#_Toc500156777)

[6.2.2 Actual Project Plan 14](#_Toc500156778)

[7 Conclusions and Recommendations 15](#_Toc500156779)

[7.1 Robot Overall Evaluation 15](#_Toc500156780)

[7.2 Robot Design Reflection 15](#_Toc500156781)

[7.3 Software Design Changes 15](#_Toc500156782)

[8 References 16](#_Toc500156783)

[9 Appendices 17](#_Toc500156784)

[9.1 Source Code 17](#_Toc500156785)

# List of Tables

[Table 6.1 Initial Project Plan Gantt chart 13](#_Toc500156753)

[Table 6.2 Actual Project Plan Gantt chart 14](#_Toc500156754)

# List of Figures

[Figure 4.1 Side View of Final Design 4](#_Toc500156747)

[Figure 4.2 Final Design Front View (top) and Top View (bottom) 5](#_Toc500156748)

[Figure 4.3 View of gear arrangment 7](#_Toc500156749)

[Figure 4.4 View of final prototype 8](#_Toc500156750)

[Figure 5.1 Main Program Flowchart 9](#_Toc500156751)

[Figure 5.2 drawLine function flowchart 10](#_Toc500156752)

# Acknowledgements

Various parts of this project were inspired by different people, some on the internet while others include upper year students and TAs.

The various mechanical designs and working of few components were inspired by open source materials on the Internet. The gear ratio was calculated from the online LEGO Gear Ratio Calculator [1]. The belt component was inspired by the Plott3r robot designed by Jerry Nicholls [2].

We would like to acknowledge Christophe Avenel for the information on how to parse .svg files using the open source libraries on his GitHub account [3].

We would like to especially acknowledge our WEEF TA Amanda for her assistance with developing ideas for the sketching input and functionality and sharing her personal experience on working on a similar project.

We would also like to acknowledge our C++ textbook for reference to good coding practices and effective methods to define particular methods [4].

Finally we would like to express our gratitude to the University of Waterloo’s MME 100 faculty for providing us with the necessary resources such as the EV3 education set and for keeping our project on track.

# Summary

This report examines the build processes with respect to the mechanical design, software design, and project management of Group 425's Mechatronics 1A final project. The problem this robot attempts to solve is assisting MTE100 students with 2D and 3D shape visualization for their EGAD assignments. This examines the problems the robot attempts to solve and the criteria and constraints the robot must meet. The mechanical redesign of the robot is detailed in this report with its subcomponents broken down into design considerations, design choice, trade-offs, and final design functionality. The software design of the robot is split into its functions and main program, all written in RobotC, detailing the various methods the robot will be able to draw various shapes on a sheet of paper. The report explains how the software design was made and its various iterations before the final prototype. With the help of flowcharts and diagrams, the fundamental functions are explained. Finally, the project management is demonstrated through the use of GANTT charts alongside the first and final iterations of the robot building timeline. Afterwards, it was determined that certain mechanical systems such as replacing the x axis movement assembly with a rack and pinion as opposed to the belt design that was used to improve overall drawing accuracy.

# Design Problem Definition

The design problem our group attempted to solve was the issue with MTE100 students and EGAD. Students struggling in EGAD were found to have difficulty visualizing 2D and 3D objects in space. Thus, the problem of assisting MTE100 students with EGAD shape visualization was chosen.

Engineering Graphics and Design is a difficult subject especially since there is a lot of visualization skills required (particularly in the missing lines assignment) and hence many students struggle in their first few weeks of the class. Additionally, almost every assignment requires work from a computer software such as AutoCAD to be printed out and not every student has a printer to conveniently print their designs. Our project is a small step to try and help these students. Our goal is to build a robot that will be able to input a type of drawing file and convert those into instructions to use a drawing tool such as a pen, pencil or a marker to physically draw it out on a given paper. This way students can simply use this robot to print out their given work using tools they have at home, without having to pay for a printer or ink.

# Constraints and Criteria

## Constraints

The constraints for our design that our team decided to agree upon are:

* Must be able to draw with a minimum thickness of 0.05mm thickness when provided with a pencil of 0.5mm thickness.
* Must be able to draw on a standard A4 size plain paper.
* Drawing must reasonably resemble the original drawing in the file.
* Drawing tool must be able to reach every x and y coordinate mapped on the paper.
* Drawing tool must be restricted to the paper and not draw anywhere outside the paper.

## Criteria

Based on the above constraints and project requirements our designs will be judged in the following criteria:

* Complexity of code: The simpler the code, and the easier it is for the robot to understand where on the paper it is drawing, the higher the design scores in the category.
* Accuracy of drawing: Here, we talk about the quality of the drawing, including how straight the lines are, how accurate the angles are, and if the thickness of the lines is consistent or not.
* Time: The amount of time the robot takes to complete a drawing.

The robot must be able to draw on an A4 size sheet of paper since it is the most commonly used size sheet of paper for regular printers. The least valuable constraint would be the thickness of the drawing since it is unnecessary. As long as the user is able to understand what is being printed, the output is a success.

If we were given the opportunity to design the robot again, the criteria we might have added are:

* Speed: Able to control the speed of the printing process
* Portability: Able to transport the printer conveniently from one location to another

# Mechanical Redesign and Implementation

## Mechanical Design Outline

The team's idea for the printing robot was to move the pencil in the x axis, which is done so with a help of a belt which is attaches the pencil to a mount, while the paper moves along the y axis with the help of four wheels attached to one motor connected by two shafts parallel to each other [5].

This design that we decided to go for includes a moving belt on which the drawing tool is mounted. As this belt moves, the drawing tool moves horizontally across the paper. The paper is on a base on which few rubber wheels (controlled by a motor) rotate which either pushes the paper forward or pulls it back depending on direction of rotation.

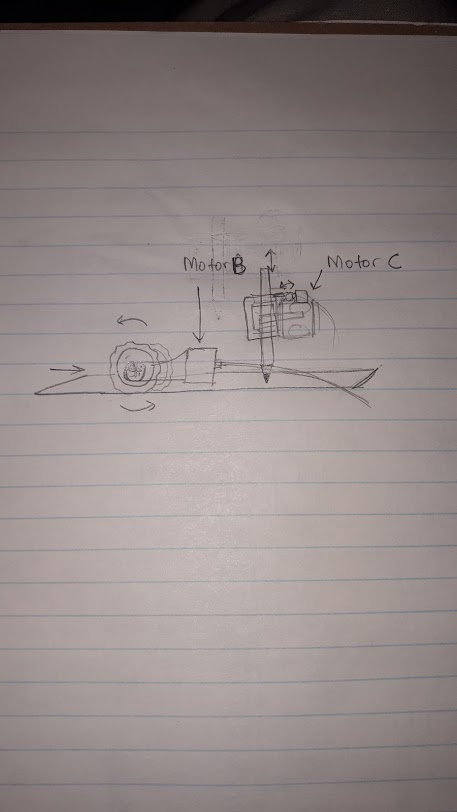


Figure . Side View of Final Design

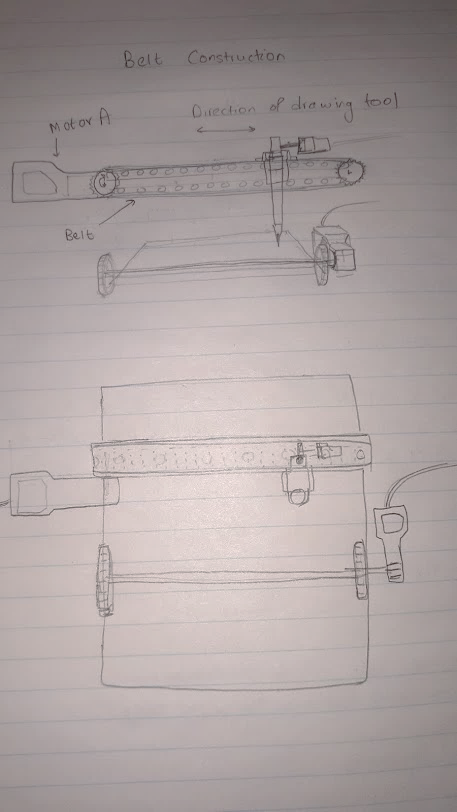


Figure . Final Design Front View (top) and Top View (bottom)

## Components

The mechanical design of our robot went through multiple iterations and prototypes. In order to coordinate and be able to have the mechanical design built my multiple team members in parallel, modular components had to be designed and built. The various components were designed independently of other components, however keeping to a specific set of requirements that each component had to be met.

### Chassis Base

The base design had the following constraints: a flat and sturdy platform to mount the printhead and a mounting mechanism for two towers to hold the gear assembly. Before building the chassis, we debated on varying factors such as if the base should be elevated to minimize vibration or if the base should be constructed using wood or Lego. Building our base with Lego would allow for an easier transition from the base to the intake assembly and would make connecting the various components together a much easier task. However, using Lego instead of wood would cause our base to be less study and rigid than if we were to use wood, as Lego pieces are susceptible to bending and flexing. We decided on building the base using Lego parts due to their ability to be taken apart and rebuilt easily compared to a wood base, but losing the stability and sturdiness of a wood base. Our final design consisted of flat Lego pieces locked together side by side in parallel running across the length of the robot. At one end, the Mindstorms EV3 brick was mounted to the chassis. Two towers were them mounted onto the chassis to hold the gear assembly and intake mechanism for the paper.

### Intake and gear assembly

In order to ensure that the paper would be able to move back and forth along the y axis consistently a design was required to be able to grip onto an A4 sized sheet of paper firmly. There were two designs we debated on using. The first design involved two sets of wheels run in parallel to sandwich the paper and roll the paper back and forth. The second design involved using a moving platform where the paper would be secured onto the platform. Using the moving platform design would allow for a much more accurate drawing as the canvas is effectively translated as the platform itself, allowing for consistent calibration and consistent paper positioning. Using the roller design would reduce the size of our robot and would be a much less complex design, allowing for more time to prototype and test each iteration of the design. We ended up choosing the roller design for its less complex design and its ability to move the sheet of paper much quicker than using a large platform, decreasing overall print time, however it would be less accurate than the platform design leading to decreased print quality. This resulted in utilizing four small diameter wheels in sets of two to sandwich the paper and rotate to move the paper back and forth. Due to the limited amount of motors available to us we decided on using one motor to power the intake assembly. This presented the challenge of gearing the entire assembly to run off of one motor and space the intake in such a way as to leave the shortest amount of distance between each set of wheels. Using a website called LEGO Gear Ratio Calculator [1], we determined the gearing possibilities to retain the gearing to a 1:1 ratio. The two sets of wheels were then mounted onto the two towers to determine the shortest distance between the two to ensure that a sheet of paper could pass through between the two. The final design involved two large gears offset from each other in a 1:1 ratio, followed by three small gears connected to one of the large gears, also in a 1:1 ratio, which then connected to the shaft driving the wheels. The bottom shaft was driven by the lowest large gear and the upper shaft was driven by the last small gear along the drivetrain. The figure below details the gear and intake assembly.

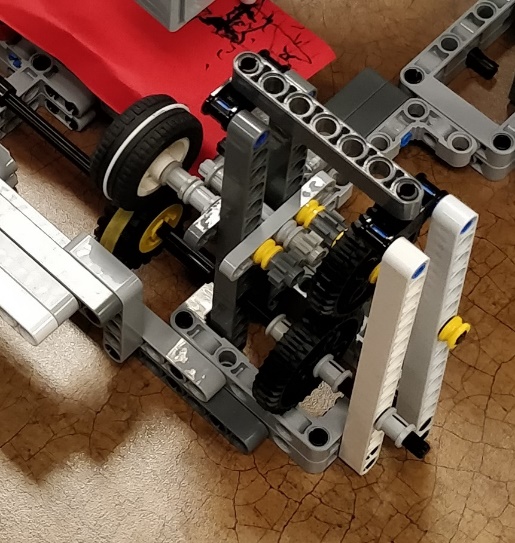


Figure . View of gear arrangment

### Chain and Printhead Assembly

In order for our robot to be able to print along the x axis the need for an assembly to mount and move the writing implement was needed. We debated between using either a bicycle chain type mechanism or utilizing a rack and pinion to move the printhead. Using the bicycle chain design would allow for extremely quick movement along the x axis and would allow for a reduced overall size of the printhead assembly. It would also make mounting the pen holding assembly much easier due to the bicycle chain having mounting holes. Utilizing the rack and pinion design would allow for a greater degree of accuracy with x axis movements due to the use of gears and motor encoders. We settled on the bicycle chain design as it offered more benefits than the rack and pinion design. It was faster and easier to build than the rack and pinion. Using the bicycle chain design would have to sacrifice accuracy of the sketching, but its benefits outweighed its cons. The printhead was mounted directly onto the bicycle chain. In order to prevent the printhead from moving past the specified print area touch sensors were mounted as limit switches. The printhead was powered by a small motor to allow the pen to move up and down for further print control.[3]

### Assembly Integration

Once the various components were completed, the entire assembly had to be constructed by connecting each component to one another. In order to achieve this a layout for the overall design had to be planned.

A few issues arose with spacing issues. Some components would be off by a few holes and connecting pieces had to be creates to solve this issue. Due to the need for the accurate placement of the various motors and sensors the spacing of certain components was heavily criticized and redone over multiple build cycles. Additional mounting points were required for drive motors and sensors which led to an increased build time than planned. The finished mechanical design is demonstrated in the figure below.

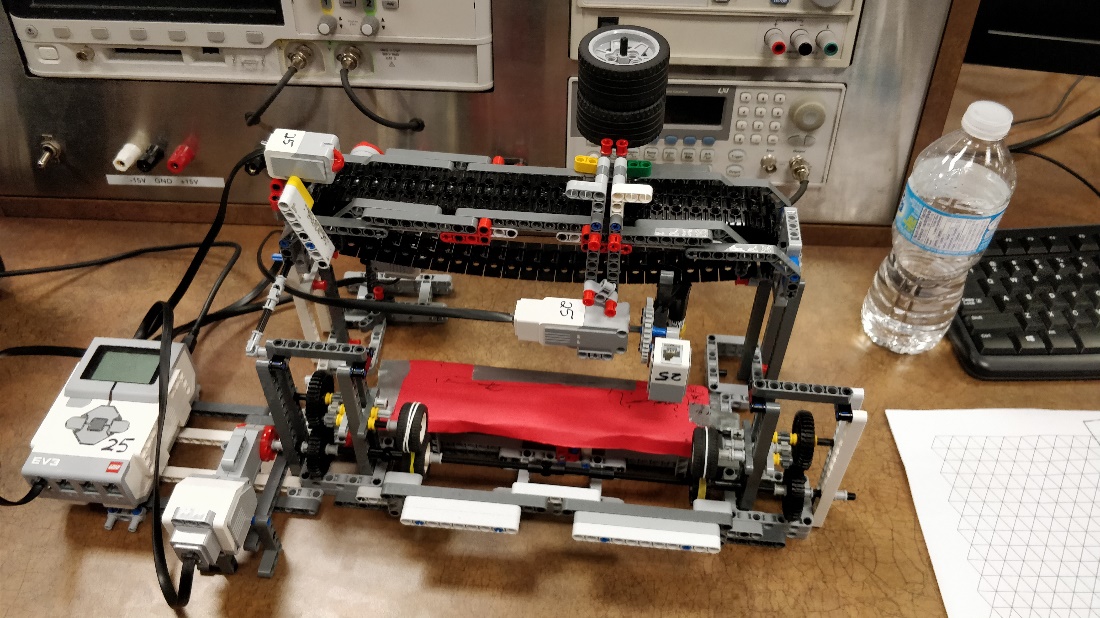


Figure . View of final prototype

# Software Design and Implementation

## Design Description

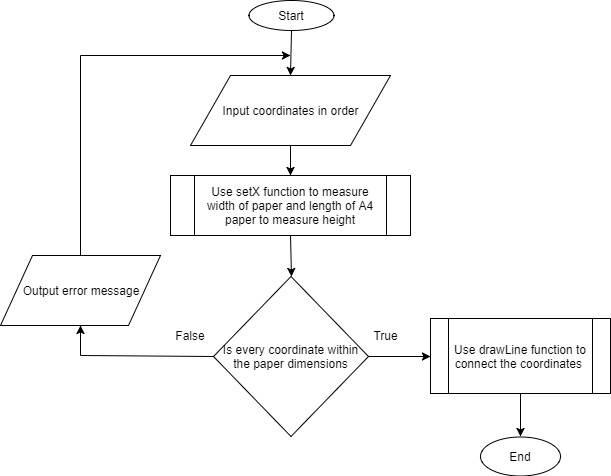


Figure . Main Program Flowchart

The main program of the robot was designed to handle data related to the drawing that was to be drawn. We decided to convert this information to a format that a computer can easily understand. We broke the drawing into individual lines and inputted this information to the program in the form of pairs of coordinates. These coordinates represented the distance (in cm) from the top right corner of the page since that would make programming the motors easier. Due to the limit of knowledge and time we decided to omit complex curves and instead break them into small lines at small angles with each other. Thus, the program is designed to input coordinate points and draw straight lines between them. This also required for us to define a function that would draw a line between to points.

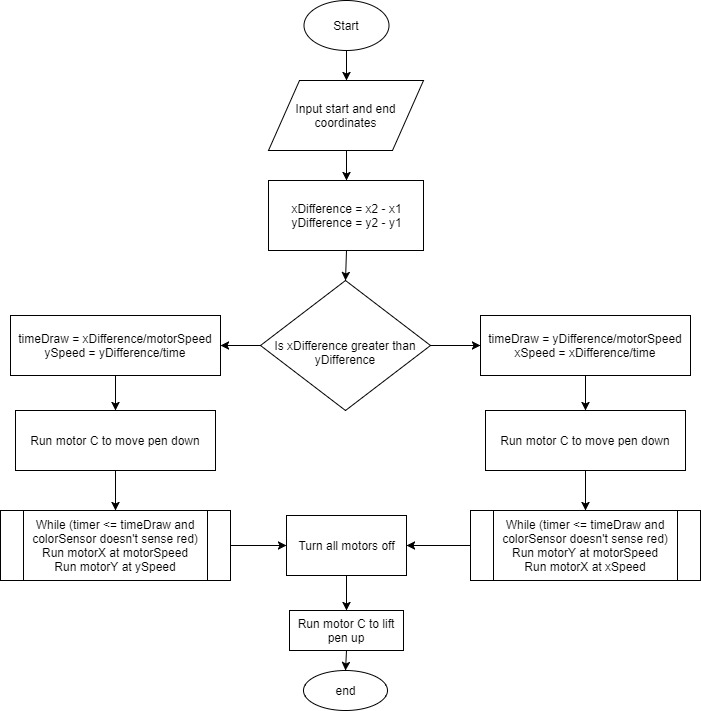


Figure . drawLine function flowchart

The void function called drawLine takes in the start coordinates and end coordinates of the line. Using basics of linear algebra, we run the two motors (X and Y axis) at appropriate speeds to result in a straight line from the start point to the end point. Another function called moveTo was defined so that the pointer (in this case the pen) can move to any desired coordinate on the paper. It used the same logic as the above function with the difference of the penUp and penDown functions. In this case the pen is constantly in the “up” position so that the pointer can move without drawing anything.

Another function that is very important for the operation of the robot is the setX function which runs the x-axis motor from the origin (top-right corner) until the touch sensor is pressed. The touch sensor indicates the left end of the x-axis. The encoder value for the motor at this point gives us the width of the paper. This width of the paper is used in all the functions to check whether the coordinates inputted to us are within this paper dimension. As for the length of the paper, we pre-programmed it to measure 29.7 cm (length of an A4 paper) minus the distance from the rubber wheels and the pen.

In sum, the fundamental function was the drawLine function which called the moveTo function and the penUp and penDown function. Other functions that made calculations easier and were not the most important functions included the toEncoder functions for the X and Y axis, which converted the coordinate distance to encoder values.

## Testing the program

Testing process included testing each individual function and then testing combinations of functions.

Void functions such as the moveTo function was used to test movements in X axis and Y axis separately by inputting simple values that would result in horizontal or vertical lines. After the robot passed this test, we would give it some more difficult values which would result in it drawing diagonal lines. Through these tests we had to constantly update values such as the radius of the motors used in the conversion from encoders to distance, so that the distance the motors moved were more accurate. We also had to take into account few time delays such as the time for the motor to start, time delay caused by many gear transfers and the time delay caused due to the friction between the pen.

After the robot passed these tests, we ran the robot through the main program giving various values for coordinates. Since the robot passed through the tests involving each individual function it did not take much effort to pass the main program.

## Trade-offs and redesigns to the software

Our initial software design was to make inputting the drawing much easier and accurate. The original plan included inputting an SVG file as it is a text-based image which can be parsed using a scripting language such as Python [3]. As we researched about this idea we realized that most of our code would include the python code and libraries leaving very little to code in C. Giving importance to time and originality we decided to completely rethink the way we will be coding this robot.

Another design we came up with is to input a file full of coordinates (similar to the GENE 121 assignments) which would act as a pseudo-programming language, since we would use letters such as ‘U’ or ‘D’ to indicate penUp or penDown respectively. This idea was later simplified to simply changing our coordinates in the main part of the code, because of the limitations of using EV3 and ROBOTC to input and read files. The most effective way to get an accurate drawing while also keeping the level of complexity of the code low was to break the drawings into straight lines and coordinates which we would input ourselves manually in the ‘main’ part of the code.

# Project Management

## Roles

The roles were split equally among all team members since the project had a lot of different components.

It was split up accordingly:

* Aniket Phanse: Responsible for the software (specifically coding functions) and building pen mounting mechanism.
* Bharath Srinivas: Responsible for mechanical design (specifically assembly and one function of code).
* William Kwong: Responsible for software, more specifically test cases and building base of robot.
* Ryan Zhou: Responsible for mechanical design, specifically belt system and overall idea and one function of code.

## Timeline

### Initial project plan

Table . Initial Project Plan Gantt chart [6]

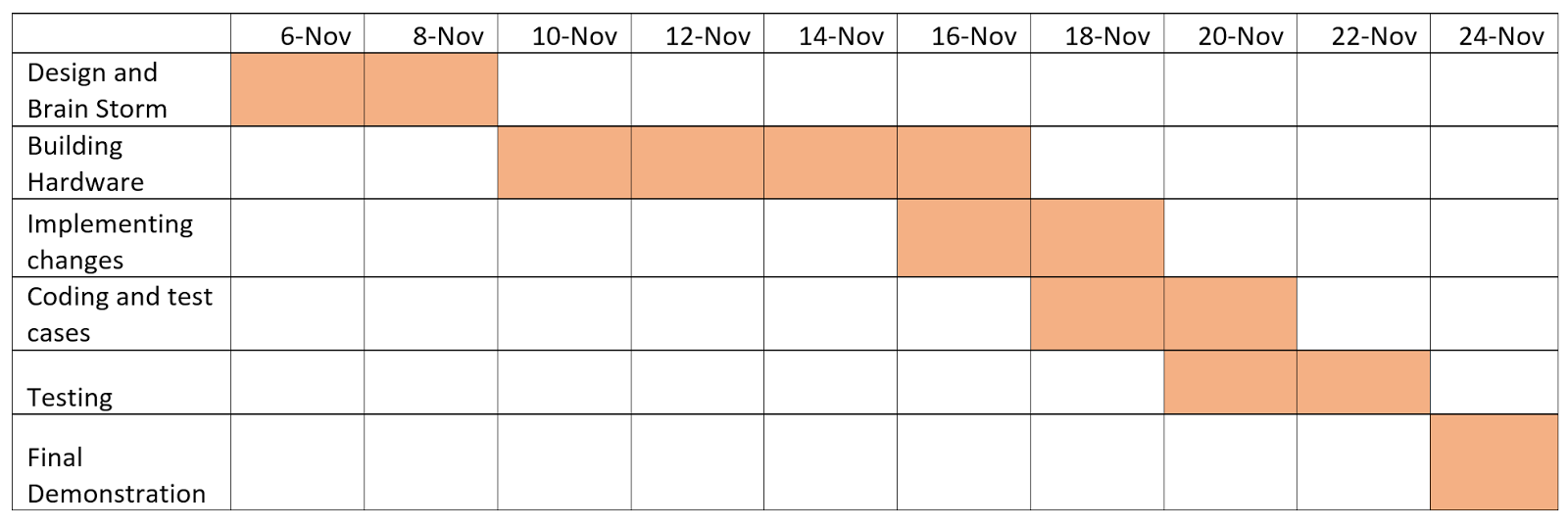
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | This week | Week 1 (12 - 18) | | Week 2 (19 - 25) | |
| Design and Brain Storm |  |  |  |  |  |
| Building |  |  |  |  |  |
| Coding and test cases |  |  |  |  |  |
| Testing |  |  |  |  |  |
| Final Demonstration |  |  |  |  |  |

Important Milestones:

* Finish building final robot – November 15, 2017.
* Finish coding program and start testing robot – November 18, 2017.
* Final demonstration – November 24, 2017.

### Actual Project Plan

Table . Actual Project Plan Gantt chart



Design and brainstorm: The team discussed various designs that the robot should look like and try to find the design which is easiest to execute.

Building hardware: The hardware took approximately 4 days to be completely built. Many changes have been implemented while building the robot (Shown in Gantt chart) for example, changing how the pencil should be mounted and how it should be brought down to the paper without bending the paper.

Coding and test cases: As changes were being implemented to the hardware, the software had to be changed as well and this process took around 6 days in total for completion.

Testing: Final testing and changes took places for two days before the final demonstration which was on 24th of November.

The schedule that was presented in the interim report was just a basic idea. The team followed most of the chart except for a few things that were unexpectedly difficult like assembling the pencil mounting mechanism.

# Conclusions and Recommendations

## Robot Overall Evaluation

In terms of the robot's result, it had done most of what we anticipated but not to the extent that we had hoped for since we had dropped the .svg file idea due to extreme complexities. The main problem that we had faced during the entire project assignment was trying to convert the .svg file into RobotC commands. This would have achieved our end goal in a more effective way but as we had to redesign the software to input coordinates manually, the student would have to turn his/her drawing into a set of coordinates and edit the code to get the robot to draw the desired drawing.

## Robot Design Reflection

As far as what things went well, our team had a solid idea of the overall design of the hardware as well as the coding process, which made the project run smoothly. Any difficulties that came up were resolved easily by the team's unanimous judgement.

Instead of the belt, we would change it to a rack and pinion system since it is more accurate. We would also improve the pen mounting system by somehow restricting the pen to move only in one vertical axis which would decrease the error as well as time taken for the robot to be ready for the drawing procedure.

Extra sensors would be beneficial for restricting movement of the belt and keeping the marker controlled. We would also have worked on the .svg file to improve the drawing quality of the robot.

## Software Design Changes

Even though all the functions were well written, and the robot could accurately draw lines between two different points, the way we inputted these points was not very useful when considering the end goal of our project. If we could somehow get our laptop to run a program that would accept coordinate values from the user and then send the instructions to the EV3, it would have achieved our goal in a better way. The most effective way would have been to use Python to script .svg file but since very few of us had any knowledge in Python, we would have end up depending solely on the resources off the internet with very little of original thinking.

# References

|  |  |
| --- | --- |
| [1] | "LEGO Gear Ratio Calculator," Gears.sariel.pl, 2017. [Online]. Available: http://gears.sariel.pl. [Accessed 16 November 2017]. |
| [2] | J. Nicholls, "LEGO Plott3r," Jander, 9 June 2016. [Online]. Available: http://www.jander.me.uk/LEGO/plott3r.html. [Accessed 24 November 2017]. |
| [3] | C. Avenel, "EV3 Pix3l Plott3r," Independent, 2016. [Online]. Available: https://github.com/cavenel/ev3-print3rbot/. [Accessed 20 November 2017]. |
| [4] | T. Gaddis, Starting Out with C++ from Control Structures to Objects, 8th Edition, Pearson, 2015. |
| [5] | "Telegraph Machine and Printer," JK Brickworks, 2017. [Online]. Available: http://jkbrickworks.com/telegraph-machine-and-printer/. [Accessed 16 November 2017]. |
| [6] | F. W. Trabold, W. N. Polakov and C. Wallace, The Gantt chart, a working tool of management, New York: The Ronald press company, 1922, p. 184. |

# Appendices

## Source Code

The following code was used to run the demonstration on November 24, 2017 in front of the WEEF TAs.

const float rX = 2.70;

const float rY = 1.25;

const int motorspeed = 30;

const float maxX = 0;

const float maxY = 0;

bool receive\_Size(string paper, int & x, int & y)

{

bool result = false;

if (paper=="A4")

{

x = 297;

y = 210;

result = true;

}

return result;

}

//coverts distance to encoder for the X axis

int toEncoderX(float distance)

{

return distance/(2\*PI\*rX) \* 360;

}

//converts distance to encoder for the y axis

int toEncoderY(float distance)

{

return distance/(2\*PI\*rY) \* 360;

}

//measures width of paper assuming we set it in the right position

int setxmotor(void)

{

while (SensorValue[S1] != 1)

{

motor[motorD] = 30;

}

motor[motorD] = 0;

int width = nMotorEncoder[motorD];

while (nMotorEncoder[motorD] > 0)

{

motor[motorD] = -30;

}

motor[motorD] = 0;

return width;

}

//all coordinates are relative to the top-right corner of page because of motor orientation

//Converts encoders to distance for Y axis

float toCoordinateY(int encoder)

{

return (encoder/360)\*2\*PI\*rY;

//Converts encoders to distance for X axis

float toCoordinateX(int encoder)

{

return (encoder/360)\*2\*PI\*rX;

}

//moves pointer to desired location assuming start location is current location

void moveTo(float startx, float starty, float finalx, float finaly, int width)

{

float xdiff = finalx - startx;

float ydiff = finaly - starty;

if (finalx <= width)

{

if (abs(xdiff) >= abs(ydiff))

{

float yTime = toEncoderX(xdiff)/motorspeed;

int ySpeed = toEncoderY(ydiff)/yTime;

time1[T1] = 0;

motor[motorA] = motorspeed;

motor[motorB] = ySpeed;

while (time1[T1] <= yTime && sensorValue[S3] != 5)

{}

motor[motorA] = motor[motorB] = 0;

}

else

{

float xTime = ydiff/motorspeed;

float xSpeed = xdiff/xTime;

time1[T1] = 0;

while (time1[T1] <= xTime && sensorValue[S3] != 5)

{

motor[motorA] = xSpeed;

motor[motorB] = motorspeed;

}

}

}

}

void penDown(void)

{

nmotorEncoder[motorC] = 0;

while (nMotorEncoder[motorC] < 50)

{

motor[motorC] = 10;

}

motor[motorC] = 0;

}

void penUp(void)

{

nMotorEncoder[motorC] = 0;

while (nMotorEncoder[motorC] > -50)

{

motor[motorC] = -10;

}

motor[motorC] = 0;

}

void drawline(int startX, int startY, int finalX, int finalY,int width)

{

int currentX = toEncoderX(nMotorEncoder[motorA]);

int currentY = toEncoderY(nMotorEncoder[motorB]);

moveTo(currentX, currentY, startX, startY, width);

wait1Msec(100);

penDown();

wait1Msec(100);

moveTo(startX, startY, finalX, finalY, width);

wait1Msec(100);

penUp();

}

void end\_program(int sensor)

{

if (canvas\_Present(sensor)==false)

{

motor[motorA] = 0;

motor[motorB] = 0;

motor[motorC] = 0;

motor[motorD] = 0;

}

}

//Sets pointer to origin, calls setxmotor

int originSet()

{

penUp();

wait1Msec(1000);

int width = setxmotor();

wait1Msec(1000);

nMotorEncoder[motorB] = 0;

while (nMotorEncoder[motorB] < 240)

{

motor[motorB] = 30;

}

motor[motorB] = 0;

nMotorEncoder[motorB] = 0;

wait1Msec(1000);

penDown();

return width;

}

int runtime()

{

return time1[T2]/10;

}

task main()

{

int width = originSet();

//coordinates below can be modified as per user's choice

drawLine(0, 0, 0, 10, width);

drawLine(0, 10, 10, 10, width);

drawLine(10, 10, 10, 0, width);

drawline(10, 0, 0, 0, width);

}