# DotMatic: The Braille Printer


## Department of Mechanical and Mechatronics Engineering

**A Report Prepared For:**

The University of Waterloo

**Prepared By:**

Neha Kasoju (21118778), Hazel Bains (21137376), Khajani Sivapala (21133827), Ali Khoja (21149346)

150 University Ave W.

Waterloo, Ontario, N2N 2N2

December 3, 2024

# Table of Contents

# List of Figures

# List of Tables

# Summary

DotMatic is a braille-printing robot capable of reading in a text file with English words and printing the translation in braille. It correctly reads and interprets words and performs the consequent movement to punch dots. Motors help it move in 3 different directions to access all parts of the paper, with a crank moving up and down, a cart moving the stylus across the paper right and left, and a paper-roller moving the paper down after each line.

# 1.0 Introduction

The printing of the braille language often requires access to a high-end, expensive braille printer. Braille is printed for numerous reasons, from translating information in public spaces for users of the language to teaching a curious individual or new learner how to read it. The limitations of printing it, however, can present a problem for those who may want to produce braille. DotMatic serves as a cheaper alternative and possible solution to this problem. It is a portable and affordable braille printer capable of printing words and small sentences in braille when given text in the English language. One example of a use case of this project is in classrooms where students might be taught how to read braille. Rather than investing in an expensive, professional braille printer as that level of precision likely would not be required, DotMatic can easily be acquired and used for its efficiency and investment value.

## 2.0 Scope

The robot carries out one main task of printing braille on a piece of paper using a stylus. Sub-tasks that it carries out to achieve the overall goal include: reading a text file and converting to braille, moving in 3 different directions to reach different locations and waiting for input to start, shut down and perform an emergency stop. These tasks did not change from earlier stages of the project. When DotMatic is set up and the code is run, it downloads a text file which the user can update beforehand to write in the words that they want printed. It then waits for the enter button to be pressed by the user to commence printing. The entire text file is read for the robot to print. As the cart moves across the rack using a motor, the motor counts the distance it moves across using motor encoder, and moves back by that distance once the printing of that row is complete to return to its original spot. A second motor on the cart turns a crank to move the stylus vertically and press it into the paper when necessary. A motor controlling wheel rollers on the paper then moves to roll the paper down to allow the cart to start printing on the next line. The E-stop uses a touch sensor that waits for input and signals when it is pressed. The robot then stops all motors and follows a set procedure. Otherwise, it carries out the printing process until the end of everything read is reached and is printed. Once all movement is complete, the robot will display a message and wait for the down arrow to be pressed to exit the program. The user can then remove the paper.

## 3.0 Constraints and Criteria

- Correctly Interpret Words from Text File
    - The first constraint of DotMatic is to correctly read and interpret words written in the text file.
    - Originally, the plan was to read words separated by space characters. However, the plan was changed to read words separated by diagonal (“/”) character and an exclamation mark (“!”) at the end of the statement. This change was made because RobotC does not seem to detect whitespace accurately.
- Punch Letters (Vertical Movement)
    - The second constraint for DotMatic is to punch letters into the paper. This is vertical movement done by the crank.
- Left-Right Movement
    - Another key constraint for this project is moving the cart horizontally. This is done so that the cart can move to the next space to print the next dot.
- Able to Read Braille
    - Finally, the last constraint for DotMatic is to accurately read the braille lettering printed on paper. This means that the dots are punched enough to read but not enough to rip the paper.

# 4.0 Mechanical Design and Implementation

## 4.1 Paper Roller

A large motor was attached to the frame as a paper roller. Two large wheels were connected to the motor. The wheels rotated clockwise or counterclockwise to feed paper forward or backwards. This was the y-axis component. This considerably reduced the range of motion the stylus and cart had to achieve to print braille row by row. It also distributed tension effectively with the frame and mat for uniform punching.

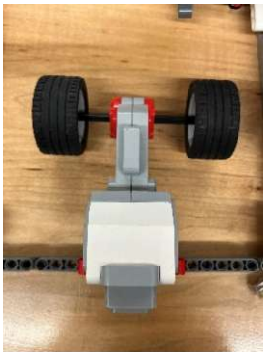

Fig 1: Paper Roller

Originally, the paper was to be suspended in between pairs of wheels as shown in Fig 1. This idea aimed to maintain tension on the paper using the grip with the wheels holding it. The team realized that this would not create enough tension to hold the paper taut while the stylus applies vertical pressure, and thus would make punching braille challenging.

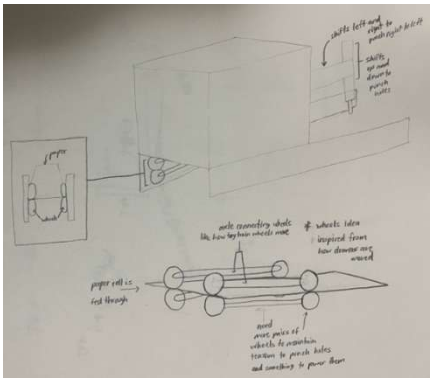

Fig 2: Original Paper Rolling Design

## 4.2 Frame

Paper is fed through a Lego frame (Fig 2). The frame is approximately 27 centimeters by 21 centimeters, while the paper was 4-8 millimeters smaller than the frame. This created some tension on the paper and secured it for accurate printing.

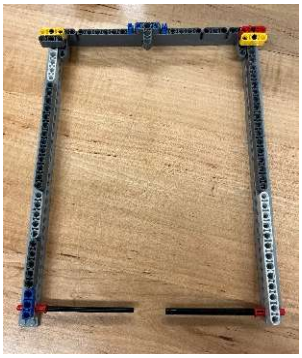

Fig 3: Frame

Originally, there was supposed to be a taller frame because of the original design for the paper rolling feature, shown in Fig 2.

## 4.3 Cart

The cart design was the horizontal (z-axis) movement component. The design with the stylus was modified numerous times with trial and error. The main challenge was moving it forward incrementally with the use of a single motor while minimizing space. Initially, a motor with two wheels was placed beside the cart and connected to one of its four wheels. As the motor would run, it would move forward and simultaneously pull the cart with it (Fig 4). After some testing, it was seen that this design took unnecessary space and materials to pursue, without the precision that was desired.

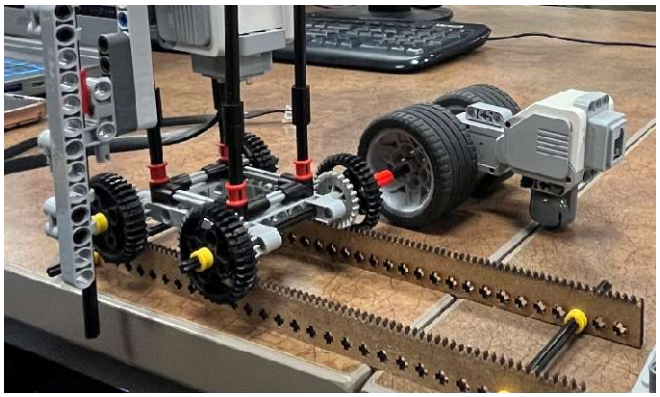

Fig 4: Old Cart Assembly

Thus, the motor placement was changed to be placed inside the cart, which resulted in the construction of a supportive frame to hold it in place. Sufficient gears were added to link the motor and wheels, resulting in a gear assembly powering two wheels and moving the other two as a result (Fig 5). This design was more compact and allowed for more control over the cart via the motor.

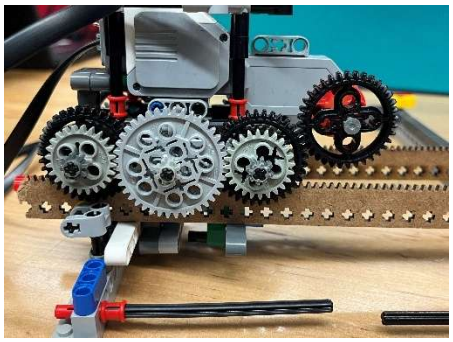

Fig 5: Gear Assembly

## 4.4 Wooden Rack

Originally, the team planned to use a threaded axle instead of a wooden rack. The stylus was going to be mounted to an assembly that moved across the threaded axle (Fig 6). The team realized this design was inefficient and had many sources of error.

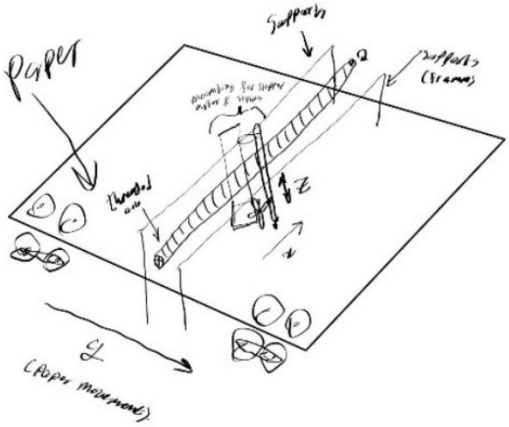


Fig 6: Old Axle Design

Two pre-laser cut wooden racks were used to facilitate horizontal (x-axis) movement. The gears on the cart aligned with the ridges on the rack, which allowed it to move left or right depending on the motor speed (Fig 7). The wooden rack was secured to the frame and used two yellow stoppers to reduce vibration on either side.

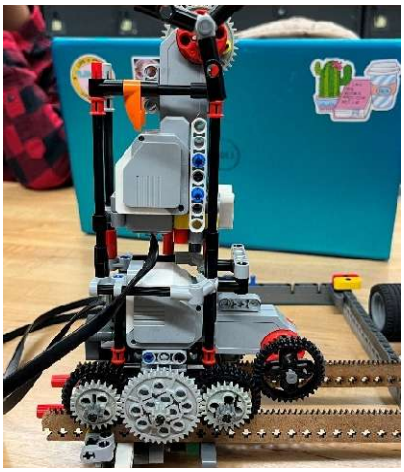

Fig 7: Cart Assembly

Before, the cart used to rise and go off the wooden rack with every punch. This made punching less accurate. Then the cart was secured onto the rack from the bottom using Lego pieces to prevent it from going off (Fig 8). This also created a margin for printing braille since the bottom connectors hitting the frame prevented it from going all the way right (Fig 8).

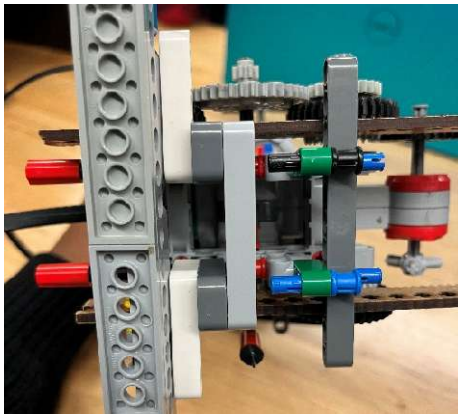

Fig 8: Bottom Connectors

## 4.5 Crank

The crank was powered by a large motor. It converted rotational movement into linear movement, inspired by the piston movement in an engine. This was the vertical, y-axis, component. The stylus was fed through a black connector, shown in (Fig 8). This stylus was not tightly secured by the connector, which allowed it to move vertically. Keeping the stylus at the perfect height within the crank impacted punching. Two grey gears secured the stylus to the connector at the correct height for punching.

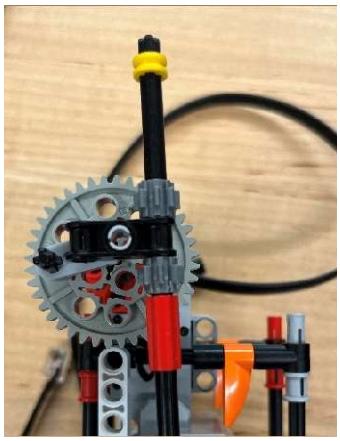

Fig 9: Upper Crank Assembly

The crank was secured at the stylus's midpoint with a U-shaped connector (Fig 10). The connector was moving with the crank rotations, so the team secured it down to the assembly with a hair band (Fig). The stylus was moving freely through the connector to perform linear movement.

Fig 10: Middle Crank Assembly

Originally, the crank was not punching perpendicular to the mat (Fig 11), which made the braille less accurate. The stylus height also kept adjusting with every rotation. The stylus was vibrating after every punch because it dragged onto the paper when released. The team realized the stylus needed to be secured inside a U-shaped connector instead. The crank was modified, focusing on keeping the stylus as perpendicular to the mat as possible at the position where it punches the paper. The stylus height was adjusted, through trial and error, at that specific position.

Fig 11: Original Crank Setup

## 4.6 Stylus & Mat

The stylus was made of a broken push pin inserted into a connector and attached to a Lego axle rod (Fig 12). The push pin was sharp and fine enough to pierce the paper uniformly.

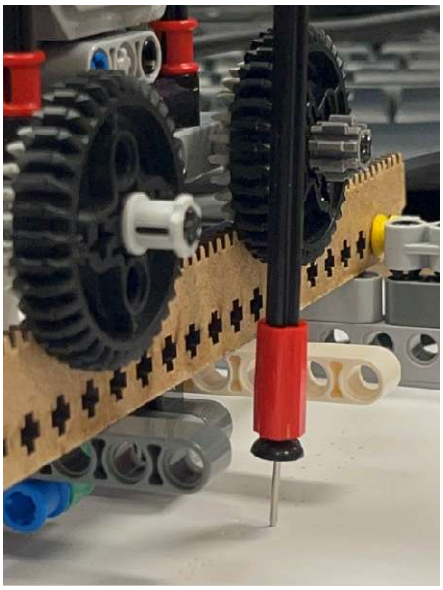

Fig 12: Stylus

Originally, the team used cardboard for the mat. Cardboard was too dense and prevented the pin from punching, especially considering the motor power. The team pivoted to using a foam mat, which was less dense. It allowed the push pin to pierce through it with minimal resistance. The foam mat worked well with the sketchbook paper, which was more flexible than regular paper.

## 4.7 E-Stop

The E-stop was constructed using the touch sensor and serves as a fast and effective method of stopping the braille printer in the case of any emergencies. Once pressed, all motors are instantly shut off and a certain amount of time is given before paper is rolled out to help the user ensure their safety. The overall design concept remained simple and similar throughout the design and construction process, however the design of the "button" to be pressed evolved. Initially, a more complex lever was designed to push the touch sensor. Later on however, this was changed to a large gear instead of placed on top of the touch sensor (Fig 13). This change helped maximize the surface area of the "button" to make it easier to press, as well as have a faster reaction rate from the user's touching of the button to the printer stopping.

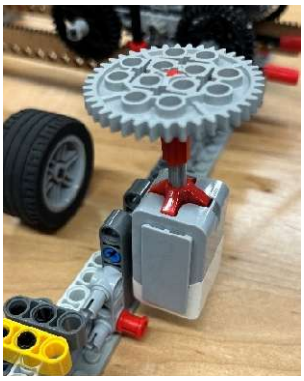

Fig 13: E-Stop

## 4.8 Overall Assembly

DotMatic (Fig 14) operates on 3 axes of movement, similar to a CNC machine, utilizing a roller to move paper up and down, a cart to move the stylus across the page and a crank to move the stylus up and down. The cart sits on wooden racks and starts at the far right, punching braille from right to left in a line. The cart maintains a margin away from the inner edge of the frame, due to connectors under the cart that secure it to the wooden rack. The cart moves right or left depending on the motor speed, moving back to the start after punching each row. A structure built on the cart surrounds a large motor that powers the crank. The crank converts rotational movement into vertical movement. The stylus is placed inside the crank structure which moves up and down perpendicular to the paper to punch braille into it. Sketchbook paper, roughly under 27 centimeters by 21 centimeters, is placed inside a frame at the bottom and is secured under the paper roller's wheels.  A simple thumbtack is used to press into the paper and create indents that show as raised dots once the paper is flipped.

The "enter" button is pressed to start the printing after the paper is secured and the text file specifying the words to be printed is updated as desired. The printer prints out everything specified and

signals to the user when the braille is complete and the paper is ready to be removed. An emergency stop button is also attached to the side of the frame, available for use at any given time if needed. The user can press it to end the printing operation and stop the machine. The printer will display a certain time before it rolls out the paper for removal.

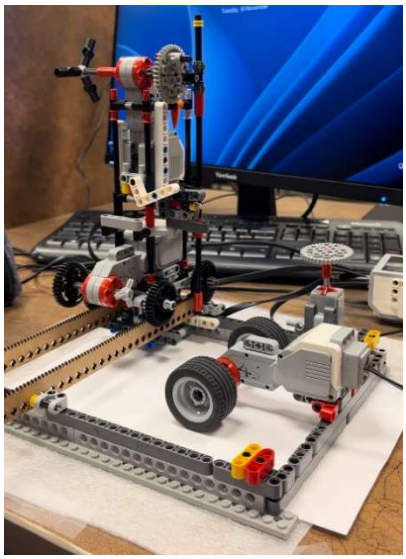

Fig 14: Braille Printer

# 5.0 Software Design and Implementation

The software program for DotMatic is split up into 3 components: start-up, standard operation, and shut down.

## 5.1 Task List

Startup:

- Configure touch sensor to be used as emergency stop
- Open text file and check for errors
- Prompt the user to press the enter button to start the printing process

Standard Operation:

- Read the contents of the text file and store it in a character array
- Determine the length of the array
- Determine the index of the letter of the alphabet corresponding to each character in the characters array
- The first row of each of the letters in the characters array, repeat this for all three rows.

Shut down:

- Display that the printing was successful and prompt the user to press the down button to exit the program

Emergency stop procedure: [1]

- Warn the user that the emergency stop is activated and that the paper will be rolled out automatically after a given period of time.
- Once the paper is rolled out, prompt the user to exit the program.

1

Note: the emergency stop function is called before, during and after each of the above mentioned tasks. If the condition is satisfied (i.e. the estop button is pressed) the program will exit according to the emergency stop procedure rather than completing the print job and executing the regular shut-down procedure.

## 5.2 Functions

Table 1: Software Functions

Data showing each function and their parameters.

| Function Name | Return Type | Parameters | Written By |
|---|---|---|---|
| *readWordFromFile* | int | TFileHandle &fin | Neha Kasoju |
| *systemStart* | void | N/A | Khajani Sivapala |
| *eStop* | void | N/A | Neha Kasoju |
| *systemStop* | void | N/A | Neha Kasoju |
| *printRow* | void | int *ptr, int len | Hazel Bains |
| *printWrd* | void | int wrdLen | Ali Khoja |
| *movePaper* | void | float deg | Hazel Bains |
| *moveCart* | void | float deg, int pwr | Ali Khoja |
| *moveCrank* | void | float deg | Khajani Sivapala |

### 5.2.1 readWordFromFile

This function reads characters from a file into the global array wrd until it reaches a maximum of 16 characters, encounters an end condition, or cannot read more characters. The function null-terminates the string and returns the number of characters read, excluding the null terminator.

### 5.2.2 systemStart

The systemStart function displays a message prompting the user to press the Enter button to start. It continuously checks for the Enter button press while calling the eStop function for safety. Once Enter is pressed, it waits for all buttons to be released and then clears the display.

### 5.2.3 eStop

The eStop function is an emergency stop mechanism triggered by a sensor (connected to S1). It halts motors C and D, displays an emergency stop message, and initiates a 5-second delay for a paper roll-out process, during which motor A rotates slightly (180 encoder units). The function then waits for the user to press and release the Down button to confirm exiting the system, displaying corresponding messages throughout.

### 5.2.4 systemStop

This function prompts the user to press the Down button to exit and continuously checks for the button press while calling the eStop function for safety. Once the Down button is pressed and other buttons are released, it stops all motors (C, D, and A) and displays a "System Stopping" message. It then waits for 15 seconds, repeatedly checking the eStop function throughout to maintain emergency handling.

### 5.2.5 printRow

The printRow function processes a row of data represented by an integer array pointer (ptr) and its length (len). It iterates through the array, moving a cart (motorD) to specified positions and optionally rotating a crank (motorA) if the current data value is nonzero. It tracks the total distance moved and resets the cart to its starting position after processing the row. Finally, it advances the paper (motorC) by a specified distance to prepare for the next row. The eStop function is called throughout to ensure safety.

### 5.2.6 printWrd

The printWrd function processes a word (stored in wrd) and prints it by converting each character to its corresponding index based on ASCII values. It then breaks the word into rows, where each character is represented by two values from the alpha array. The function prints each row using the printRow function and moves the paper after printing the entire word. The eStop function is used throughout to ensure safety.

### 5.2.7 movePaper

The movePaper function advances the paper by a specified degree (deg) using motorA. It resets the motor encoder, activates the motor to move at a speed of -10, and continuously checks the encoder until the desired degree of rotation is reached. Once completed, it stops the motor and ensures safety by repeatedly calling eStop.

### 5.2.8 moveCart

The moveCart function moves the cart by a specified degree (deg) and power (pwr) using motorD. It resets the motor encoder, sets the motor speed to the specified power, and continuously checks the encoder until the desired degree of movement is reached. Once the movement is complete, the motor stops, and eStop is called throughout for safety.

### 5.2.9 moveCrank

The moveCrank function rotates the crank by a specified degree (deg) using motorC. It resets the motor encoder, sets the motor speed to -10, and continuously checks the encoder until the desired degree of rotation is reached. Once the rotation is complete, the motor stops, and eStop is called throughout for safety.

## 5.3 Data Stored

- The braille alphabet is stored in the program as a [26][6] 2-dimensional integer array.
- The contents of the text file are read and stored into a character array
- The length of the array is stored in a variable
- The distance moved by the cart in printing each row is stored and then used to move the cart back to its initial position
- The number of times the cart is moved is stored as an integer and is used to leave a larger gap between each letter
- The indices of each character in the character array are stored in an integer array and are used to determine which letters to print from the alphabet array.
- The corresponding row of letter in the characters array is stored in a rows array to enable printing by rows

## 5.4 Software Decisions

### 5.4.1 Predetermining encoder counts for movement:

The decision to calculate the encoder counts for moving the cart and the paper was made in order to provide consistency and accuracy in the placement of each braille dot. This also simplified the project by eliminating the need to implement a system of limit switches or the like. This enabled the team to focus on other aspects of the work that were more relevant to the constraints and criteria outlined in a previous section.

### 5.4.2 Printing row by row vs letter by letter

As mentioned before, each letter in the braille alphabet is composed of a 3x2 matrix of convex dots. There were two proposed methods of printing words made of such letters. The first was to print the first row of each letter in one line, then moving the paper and repeating the process for the remaining two rows. The second was to print each letter in its entirety (all 3 rows and 2 columns) before moving on to the next letter in the word.

The decision was made to follow the first method and print row by row to avoid the back-and-forth movement of the paper and the cart as this would likely increase inaccuracy in printing due to the unreliability of the EV3 motors. Printing each character in its entirety would require the paper to be moved multiple times for each letter and then reinitialized to the starting position for the next letter, however, by going row by row the reinitialization is simplified as it only involves repositioning the cart rather than both cart and paper. Finally, this decision also simplified the code as everything the printing occurs in a logical sequence.

### 5.4.3 Printing from right to left

Once again it must be emphasized that the braille alphabet is composed of **convex** dots that are read using one's fingers. The decision was made that it would be best to punch the dots into the paper from above (creating concave dots) and then flipping the paper in order to read the convex impressions. The team determined that the simplest way to ensure that the convex dots were in the right order once the page was flipped would be to have the robot start printing on the right side of the page and move towards to left.

### 5.4.4 Storing the braille alphabet

Originally the decision was made to store the braille alphabet in a [26][3][2] 3-dimensional array, which can be thought of as a collection of 26 [3][2] 2-dimensional arrays. This seemed to be a clean way to store the alphabet in the code. However, the group came to the realization that RobotC does not support the use of 3-dimensional arrays. This was discovered at a later point in the design process and with limited time remaining the group decided to pivot to a [26][6] 2-dimensional array which removed the explicit distinction between the rows and columns of the letters. Implicitly, the first two indices would correspond to the first row, the second two to the second row, with the remaining two indices corresponding to the third row. At this point in the process the use of structs was considered, however it was deemed impractical and unnecessary, especially within the remaining time.

## 5.5 Testing

### 5.5.1 Printing Correctly

To ensure the braille printer was printing the correct words and statements, the program included tests such as output (cout) statements to see that the printer was about to punch the correct letters.

### 5.5.2 Unit Testing

Before combining all of the functions and testing it all together in main, every function was tested individually. This was done by sending in arbitrary parameters to each function and seeing if it outputs the correct value. Furthermore, for more complex functions, different sections within each function were

tested in detail. For example, the *readWordFromFile* function was tested by inputting a test file with one word and seeing if the program would display the word on the LEGO EV3 Robot screen. This shows that the word from the file is being read correctly.

### 5.5.3 Testing Together

Finally, all the functions were tested together in main. Some functions call on previous functions, thus, the program was run to ensure that this process worked seamlessly.

### 5.5.4 Debugging

Using the debug feature in Visual Studio Code, the program was run and tested to fix errors.

## 6.0 Verification

Every constraint was met.

- Correctly Interpret Words from Text File
    - This constraint was met as the program correctly read and interpreted the words from the text file.
- Punch Letters (Vertical Movement)
    - The second constraint for DotMatic is to punch letters into the paper. This constraint was met as there is accurate vertical movement done by the crank to punch into the paper.
- Left-Right Movement
    - The third key constraint for this project is moving the cart horizontally. This constraint was met as the cart accurately shifted left and right when needed.
- Able to Read Braille
    - Finally, the last constraint for DotMatic is to accurately read the braille lettering printed on paper. This constraint was met as we were able to accurately read the braille and understand it.

# 7.0 Project Plan

This section discusses how the workload was divided among group members and the timeframe allocated for completing each aspect of the project.

## 7.1 Debugging

Table 2: Task Allocation

| Member Name | Mechanical Tasks | Software Tasks |
|---|---|---|
| Hazel Bains | • Build cart<br>• Connect all mechanical pieces together | • Write and test *printRow*<br>• Write and test *movePaper* |
| Neha Kasoju | • Build and test e-stop<br>• Attach racks together | • Write and test *readWordFromFile*<br>• Write and test *eStop*<br>• Write and test *systemStop* |
| Ali Khoja | • Build bottom frame<br>• Manually test paper rolling | • Write and test *printWrd*<br>• Write and test *moveCart* |
| Khajani Sivapala | • Build and manually test crank | • Write and test *systemStart*<br>• Write and test *moveCrank* |

## 7.2 Schedule

- Finalize design (Nov 1)
- Finish physical system (Nov 5)
    - Test stylus printing
    - Testing motor movement
    - Creating smooth rolling mechanism for paper
- Finish software (Nov 8)
    - 2D arrays for each letter
    - Read from text file
    - Finish testing
- Project Demo (Nov 19)

## 7.3 Revisions to Plan

The original plan was to finish all of the software program then test it with the mechanical system. However, the plan was changed to test each function of the program with the mechanical system. Once one function is functioning properly and completes the mechanical tasks, then the next function is edited and tested accordingly. This new plan extended the timeline as the testing was completed November 17, instead of the original plan of finishing testing by November 8. Although the timeline was extended, detailed unit testing ensured the program ran accurately.

## 8.0 Conclusions

In conclusion, DotMatic solves the problem of lack of access to braille printers as they are expensive. The four constraints outlined were correctly interpreting the words from the text file, punching letters, moving the left and right cart, and able to read the braille printed on the paper. DotMatic meets all the constraints. The key features of the mechanical design includes the crank, the cart, and the paper rolling wheels. The key features of the software design include reading from the file and reading the array.

# 9.0 Recommendations

## 9.1 Mechanical

### 9.1.1 Crank improvements

Although the crank can punch braille accurately, it could perform linear movement better to punch with minimal vibration. To fix this problem, the U-shaped connector should be swapped with a small frame that would surround the stylus. The U-shaped connector does not enclose the stylus with enough height but gives it enough slip to move vertically. A new frame in place of the connector would do a more efficient job.

### 9.1.2 Wooden Rack

The wooden rack was roughly 5 millimeters thick, while the gears were three times thicker. If the wooden rack was thicker, it would accommodate the wheel size more. It would have made horizontal movement which reduces alignment issues since there is more surface area for the wheels to align to.

The wooden rack vibrates slightly when the printer is operating, and the stoppers used were moving out of place. This can create errors by reducing accuracy. To prevent this, the wooden rack could have used better stoppers that secured tighter.

## 9.2 Software

### 9.2.1 Increased Printing Functionality

Currently, the program only works for one, single line, text file and a given maximum length. Provided more time, the project would be made more practical for a broader range of use cases with less arbitrary limitations. Consider, as an example of this, the way in which the robot only prints one string of characters in one line before exiting the program. It was designed in this way as proof of concept and to be feasibly completed within the time limit. However, in the future, the software would be adapted to allow the user to enter multiline text files to broaden the practical applications of this machine. Additionally, this could be taken one step further by programming the robot to enable the execution of multiple print tasks, which would be implemented similarly to the way in which commercial printers have a print queue.

### 9.2.2 Input Validation

Something else that would greatly improve the useability of this program would be the implementation of input validation functions in the code which would parse through the text file and ensure that the text file conforms to the input parameters so as not to crash the program. For example, in the current case, if the text file contained an invalid character, there would be no corresponding index of

the alphabet array with which to match said character causing the corresponding element of the indices array to be undefined.  This machine is specifically intended for use in elementary schools. To that end, input validation could also be used to ensure that printed messages are always appropriate. This could be done by comparing each word in the input file to words in a text file containing a bank of all the words that are not allowed to be printed. If there is a match, the word would be skipped.

### 9.2.3 Allowing for More Characters and Words

Finally, one of the simplest yet most effective ways to expand the practical use of this project would be to add more characters other than just the letters of the alphabet. In the future, this would likely be done in a file rather than being stored entirely in the program through an array. This could be taken one step further by storing entire words, as in some cases the braille for a word can be different than that of letters that make up the word. The program would then have to not only match characters to indices but also check if the indices array contained a certain sequence of indices, and if so then it would use the braille of a preset word rather than printing out each individual letter contained in the word.

# Appendix A: Final Code:

```
#include "PC_FileIO.c"


/*
    global array storing word (17 characters + null terminator), number of character is arbitrary, changes
    depending on what is being printed.
*/

char wrd[18] =
{NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,
  NULL};


//global array for the alphabet in braille

int alpha[27][6] =

{

   {1,0,0,0,0,0}, //A

   {1,0,1,0,0,0}, //B

   {1,1,0,0,0,0}, //C

   {1,1,0,1,0,0}, //D

   {1,0,0,1,0,0}, //E

   {1,1,1,0,0,0}, //F

   {1,1,1,1,0,0}, //G

   {1,0,1,1,0,0}, //H

   {0,1,1,0,0,0}, //I
```

```
    {0,1,1,1,0,0}, //J
    {1,0,0,0,1,0}, //K
    {1,0,1,0,1,0}, //L
    {1,1,0,0,1,0}, //M
    {1,1,0,1,1,0}, //N
    {1,0,0,1,1,0}, //O
    {1,1,1,0,1,0}, //P
    {1,1,1,1,1,0}, //Q
    {1,0,1,1,1,0}, //R
    {0,1,1,0,1,0}, //S
    {0,1,1,1,1,0}, //T
    {1,0,0,0,1,1}, //U
    {1,0,1,0,1,1}, //V
    {0,1,1,1,0,1}, //W
    {1,1,0,0,1,1}, //X
    {1,1,0,1,1,1}, //Y
    {1,0,0,1,1,1}, //Z
    {0,0,0,0,0,0}  //
};

//fileio function
int readWordFromFile(TFileHandle &fin);

//start stop functions (including emergency case)
```

```
void systemStart();

void systemStop ();

void eStop ();


//printing functions

void printRow(int *ptr, int len);

void printWrd(int wrdLen);


//movement functions

void movePaper(float deg); //y-axis

void moveCart(float deg, int pwr);  //x-axis

void moveCrank(float deg); //z-axis


task main() {

//configure touch sensor for eStop

SensorType [S1] = sensorEV3_Touch;

wait1Msec(50);


//reading from file

string fileName = "input.txt";

TFileHandle fin;

  bool fileHandle = openReadPC(fin, fileName);


  if (fileHandle) {
```

```
        systemStart();


eStop();
int wrdLen = readWordFromFile(fin);


eStop();
  printWrd(wrdLen);
  eStop();


  systemStop();
}
else {
displayTextLine(4, "Error opening file");
}
closeFilePC(fin);
}


int readWordFromFile(TFileHandle &fin) {
    int charCount = 0;
    char c = ' ';
    int wrdEnd = 0;


    while (readCharPC(fin, c) && charCount < 17 && wrdEnd == 0) {
        wrd[charCount] = c;
```

```
    charCount++;
  }

  wrd[charCount] = '\0';  // null-terminate the string

  return charCount;  // return the number of characters in word (excluding null terminator)
}

//---------------------------------
void systemStart(){
displayTextLine(3, "Press Enter to Start Printing");
eStop();
while(!getButtonPress(buttonEnter))
{eStop();}
 while(getButtonPress(buttonAny))
 {eStop();}
eraseDisplay();
}

//---------------------------------
void eStop (){
  if (SensorValue[S1]){
    displayTextLine(4, "Emergency Stop Activated!");
```

```
        motor[motorC] = 0;
        motor[motorD] = 0;

        const int TIME_UNTIL_ROLL = 5;

        displayTextLine(5, "%d secs until paper roll out.", TIME_UNTIL_ROLL);
        wait1Msec (5000) ;
eraseDisplay();

        motor[motorA] = 10;
        while (abs(nMotorEncoder[motorA]) <= 180){}
        motor[motorA] = 0;

displayTextLine(5, "Press down button to exit");

                    while (!getButtonPress(buttonAny)){eStop();}

                    while (getButtonPress(buttonDown)){eStop();}

                    displayTextLine(4, "System Stopping.");
    }
}

//-------------------------------
```

```
void systemStop (){

displayTextLine(4, "Press down button to exit");
eStop();
   while (!getButtonPress(buttonDown)){eStop();}

   while (getButtonPress(buttonAny)){eStop();}
eStop();
   displayTextLine(4, "System Stopping.");

   motor[motorC] = 0;
   motor[motorD] = 0;
   motor[motorA] = 0;

eStop();
   wait1Msec (15000) ;
   eStop();
}

void printRow(int *ptr, int len) {
int countCrank = 0;
float distMoved = 0;
nMotorEncoder[motorD] = 0;
```

```
for (int i = 0; i<len; i++) {

eStop();
moveCart(8.25, -10);
distMoved+=abs(nMotorEncoder[motorD]);
eStop();


if (countCrank % 2 == 0){
moveCart(8.25, -10);
distMoved+=abs(nMotorEncoder[motorD]);
}

countCrank++;

if (*ptr) {
eStop();
moveCrank(360);
wait1Msec(50);
}
ptr++;
}
eStop();
moveCart(distMoved, 10);
```

```
eStop();
movePaper(8.25);
}

void printWrd(int wrdLen) {
   int row[34];              //max of wrdLen*2
   int indices[17]; //max of wrdLen

   for (int i = 0; i < wrdLen; i++) {
                  eStop();
      //determining indices to be used for alpha (using aski codes)
      //i.e. what letter is in wrd[i]
                  if (wrd[i] == '/') {
                        indices [i] = 26;
                        eStop();
                  } else {
         indices[i] = wrd[i] - 'a';
         eStop();
      }
   }

   for(int i = 0; i <3; i++) {
                  eStop();
      int k = 0;
```

```
        for (int j = 0; j < wrdLen; j++) {
                        eStop();
          row[k]=alpha[indices[j]][i*2];
          row[k+1]=alpha[indices[j]][i*2 + 1];
          k+=2;
          eStop();
        }
        eStop();
        printRow(&row[0], wrdLen*2);
    }
    eStop();
movePaper(3*8.25);
}


void movePaper(float deg){
nMotorEncoder[motorA] = 0;
eStop();
motor[motorA] = -10;
while (abs(nMotorEncoder[motorA]) <= deg) {eStop();}
motor[motorA] = 0;
eStop();
}
```

```
void moveCart(float deg, int pwr){
eStop();
nMotorEncoder[motorD] = 0;
eStop();
motor[motorD] = pwr;
while (abs(nMotorEncoder[motorD]) <= deg) {eStop();}
motor[motorD] = 0;
eStop();
}

void moveCrank(float deg){
nMotorEncoder[motorC] = 0;
eStop();
motor[motorC] = -10;
while (abs(nMotorEncoder[motorC]) <= deg) {eStop();}
motor[motorC] = 0;
eStop();
}
```