People’s Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M’Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering Department of Electronics

Braille Printing Machine

License Degree Project

Date Of Submission:

- August 2020

Authors:

- Benbouali Mohamed Amine
- Berrichi Riad
- Belhadj Aymen

Supervisor:

- Ms. Derragui Nabila

ABSTRACT:

Braille is the language impaired individuals use to read books, letters, research papers and any kind of written data, printed braille paper is uncommon and very rare to come across and requires visually able individuals to print these kinds of books and papers on, making a visually impaired individual dependent on others. In this project, we introduce a CNC printing machine that prints regular text from a computer into braille text on a real life paper using a syringe, this project aims to simplify an impaired individual's reading process and is designed for ease of use and accessibility on the financial aspect. Allowing visually impaired individuals to join the research sector would change their effect on society by an order of magnitude.

CONTENTS LIST:

GENERAL INTRODUCTION:

Braille is the language visually impaired people use to read books, news papers and any type of literature. Braille was invented by Louis Braille at the age of 15 and published 5 years later, he was one of the few visually impaired teachers in the world at that time. His invention, a language that consists of raised dots on a piece of paper allows reading information without looking at it by passing a finger over them and feeling distinct patterns according to pre-set rules. [1]

, In braille, both letters and numbers have the same braille characters. So A has the same braille character as number 1, and letter B has the same braille character as number 2. The distinguishing factor is the letter coming before which is called an indicator, it indicates if the following characters are numbers or letters.

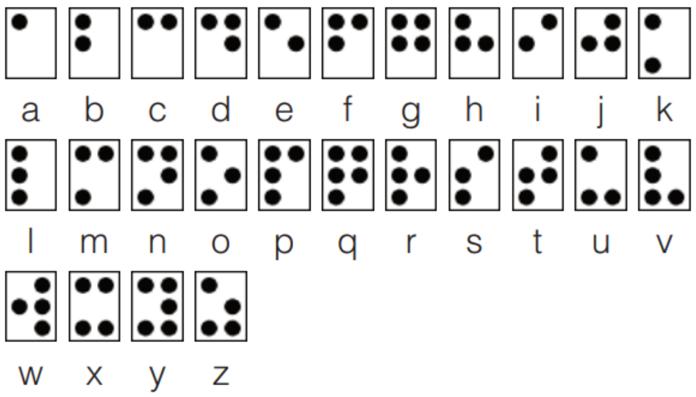


Figure 1: Braille alphabet [2]

This tactile language requires special methods for producing its books and papers, braille language is printed by distinctly raising dots on a paper with either a strong or a sharp object requiring high precision and just the right amount of pressure for raised dots that don't go away easily, machinery companies and book publishers own these types of machines (figure 2).

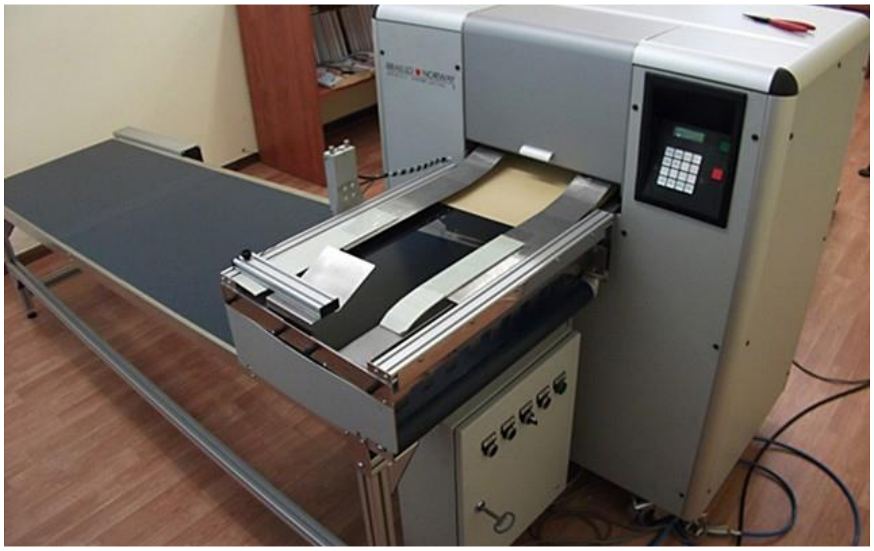


Figure 2: Industrial braille printing machine [3]

A visually impaired person can not easily have pdf files, research papers nor any online content easily printed without the need of a machinery company or a braille book publisher, it either requires a visually capable individual and requires a lot of time, or requires expensive machines which could not be easily altered by regular everyday people. There exist braille slates (figure 3), cheap plastic frames that allow raising dots according to the braille language, but that still requires reading the pdf file and manually writing the contents using it, that requires a visually able person.

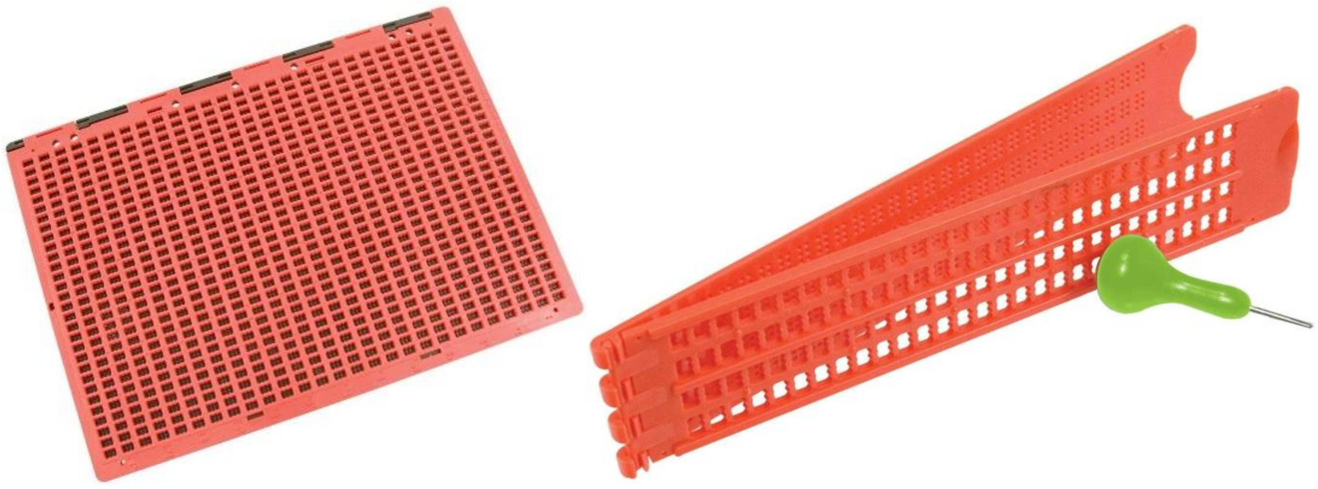


Figure 3: Braille slates [4]

A braille typewriter exists (figure 4) and allows the operation of a machine to raise dots on paper using special material, this machine however still requires a visually capable individual to both type on, and operate.

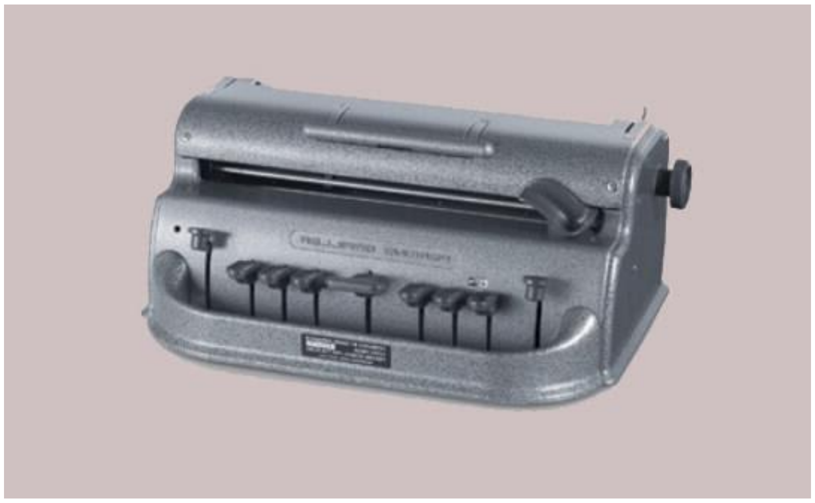


Figure 4: Braille typewriter [5]

Our purpose is to allow visually impaired people to join the race of research, teaching and improving human life, this can allow a previously disconnected portion of the population to contibute into open source content and to learn about it in much easier and more attainable ways.

A small, cheap and easy to operate machine would allow the printing of braille language with little money, little to no maintenance and simple operation, producing braille printed paper without the supervision of a visually capable individual, a machine that can print pdf files on the go into regular A4 paper.

1 CHAPTER 1: HARDWARE

1.1 INTRODUCTION

The solution is a printing machine that could thread a paper of size A4 to print braille language on it, it requires a very strong and consistent structure along with motors, timing belts, linear shafts, and specific structural parts to ensure precise movement.

This machine has to be significantly smaller than industrial printers and is easy to use.

Our purpose is to allow visually impaired people to contribute into research and to ease the consuming of information from any online source such as research papers, pdf files and news.

1.2 MATERIALS

1.2.a) Nema 17 Stepper motors: DC motors that contain two coils inside which allows stepping the motors by precise steps each time, they are also used in 3D printers (figure 1.1)

Figure 1.1: Nema 16 stepper motor [6]

1.2.b) Servo motors: precise motors just like stepper motors but their interior functionality contains a potentiometer which allows moving the motors to specifically given angles, these motors usually only spin a maximum range of 180 degrees. (figure 1.2)

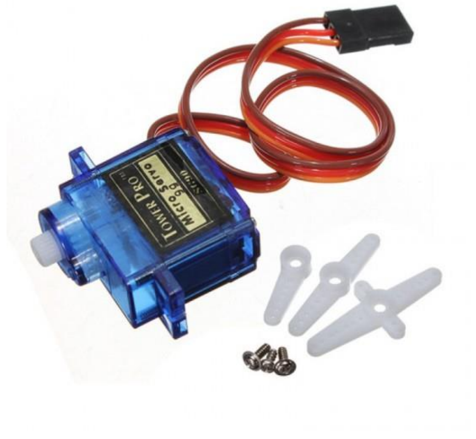


Figure 1.2: SG90 micro-small 9g Servo Motor 1.5KG [7]

1.2.c) Timing Belts: Timing belts are long stripes of plastic or iron containing ridges on one surface, allowing motors to move structures on linear axies. (figure 1.3)

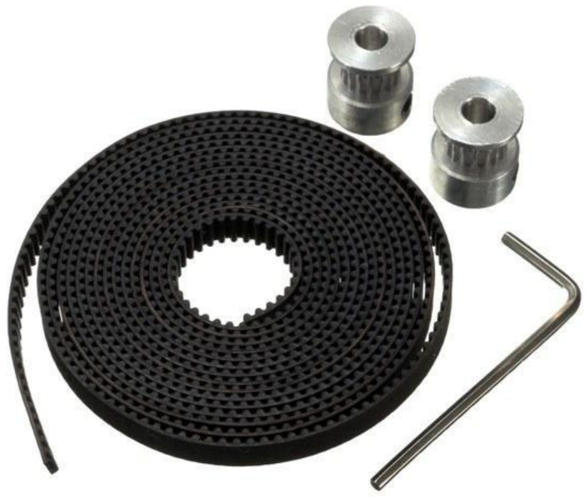


Figure 1.3: 2-meter timing belt 16 teeth & heads [8]

1.2.d) Linear Shafts: long smooth surfaced iron bars that act like an axis for a structure to move on a specific axis,also mainly used in 3D printers and any type of machinery Figure 1.4

Figure 1.4: Linear shafts 400mm 8mm[9]

1.2.e) Skeleton:

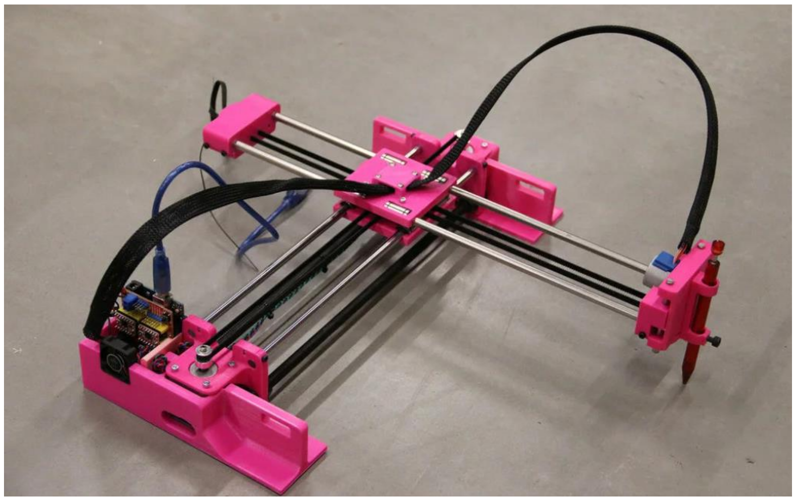


Figure 1.5: Example of a CNC machine [10]

We want a structure similar to figure 1.5 but on a larger scale, the benefits of this structure are that both motors can be situated on one axis only but still be able to steer the head anywhere on the x and y-axis (more in-depth later on).

To move the machine in X and Y axies we use the stepper motors mentioned

To allow the poking head to move up and down we use the servo motors mentioned, this motor will be directly above the paper operating our threader operation by commands from the computer in the form of two angles, an up angle and a down angle that can be varied until we achieve the amount of threading we need for a usable braille paper.

To hold all of these elements in place we need specifically cut pieces of structure, In this case, we will be 3D printing these pieces, figures 1.6, 1.7, 1.8, 1.9 are all of the pieces we designed in the 3D modeling program called Blender.

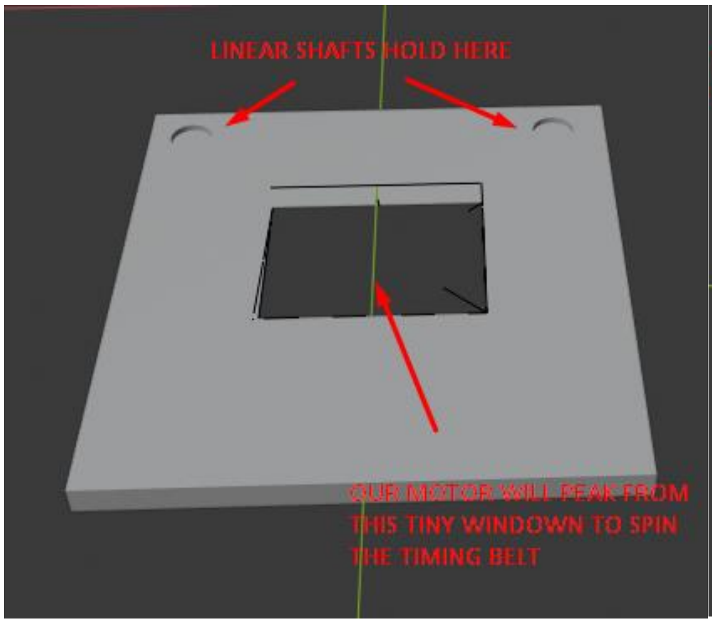


Figure 1.6: Side Panel

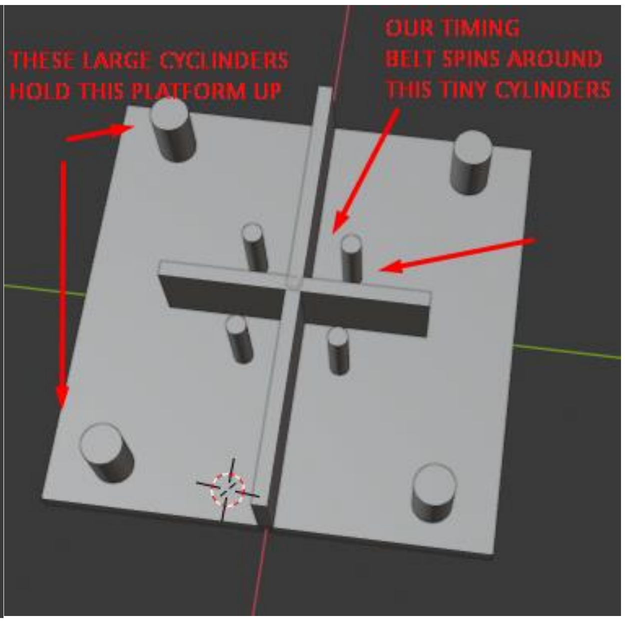


Figure 1.7: Center Panel

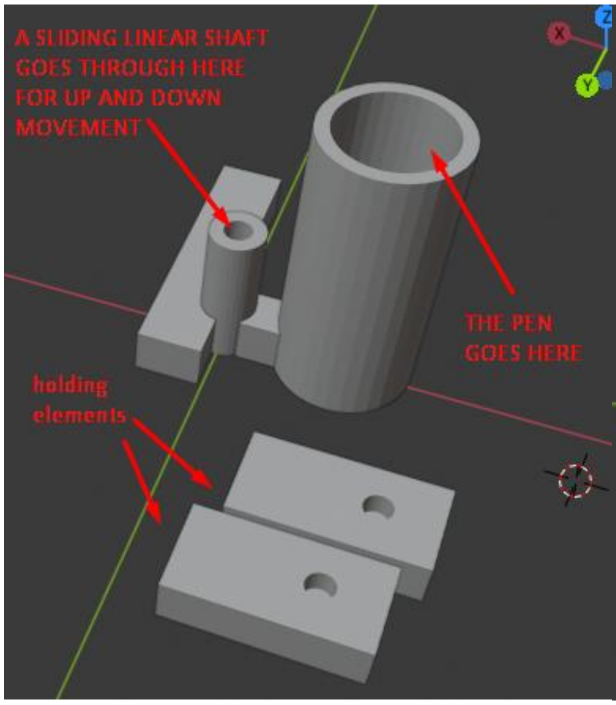


Figure 1.8: Drawing mechanism

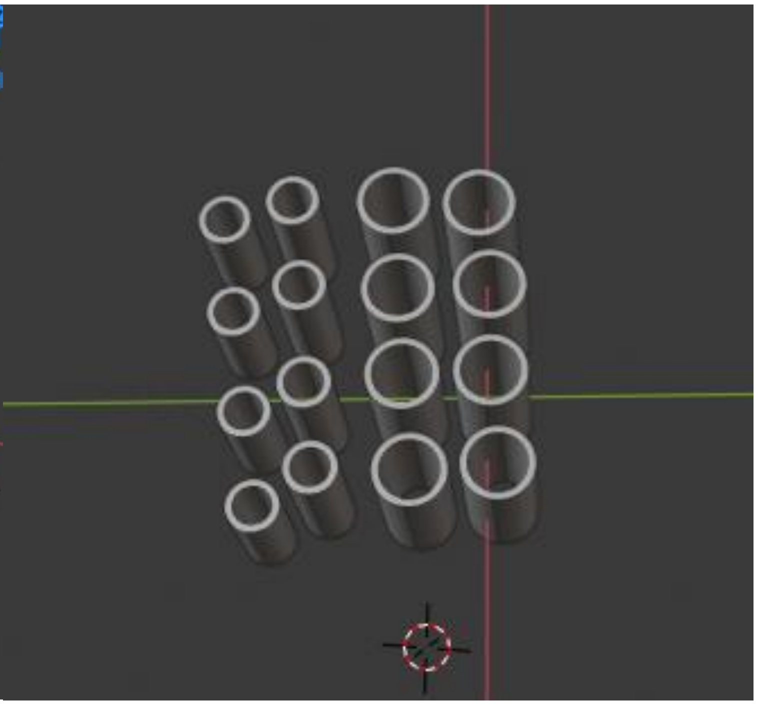


Figure 1.9: Spinning Bolts

1.3 <u>Electrical Circuit</u>

1.3.1 Full circuit and wiring:

The circuit that brings all of this together is very simple and logically ties all of the hardware together. (figure 1.10).

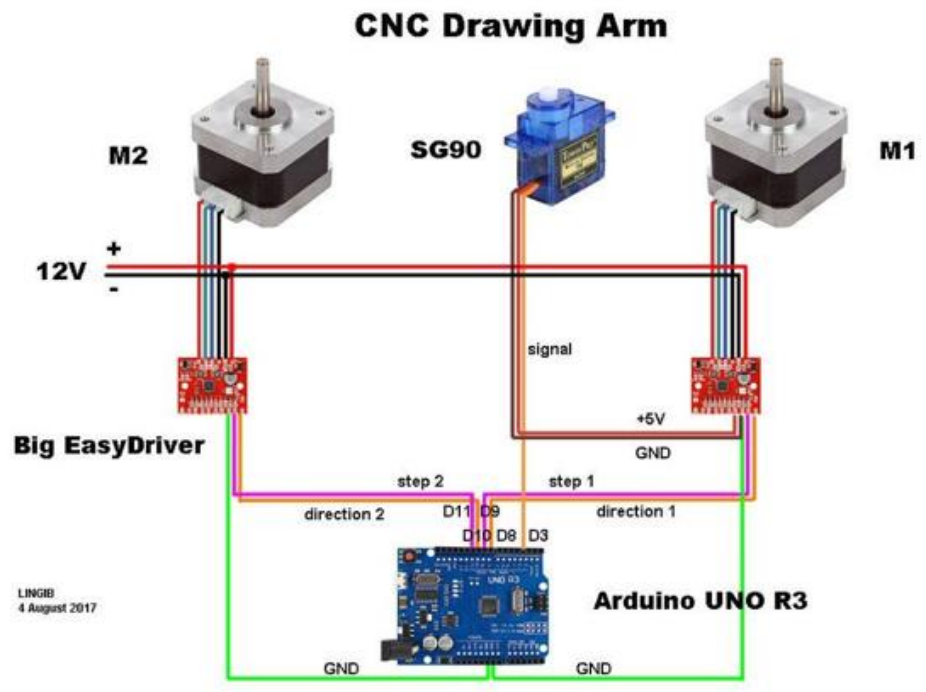


Figure 1.10: electrical circuit [11]

After acquiring all of the materials including an Arduino board we were able to wire all of the materials together to have the exact circuit, but without the 3D models shown previously. (figure 1.11)

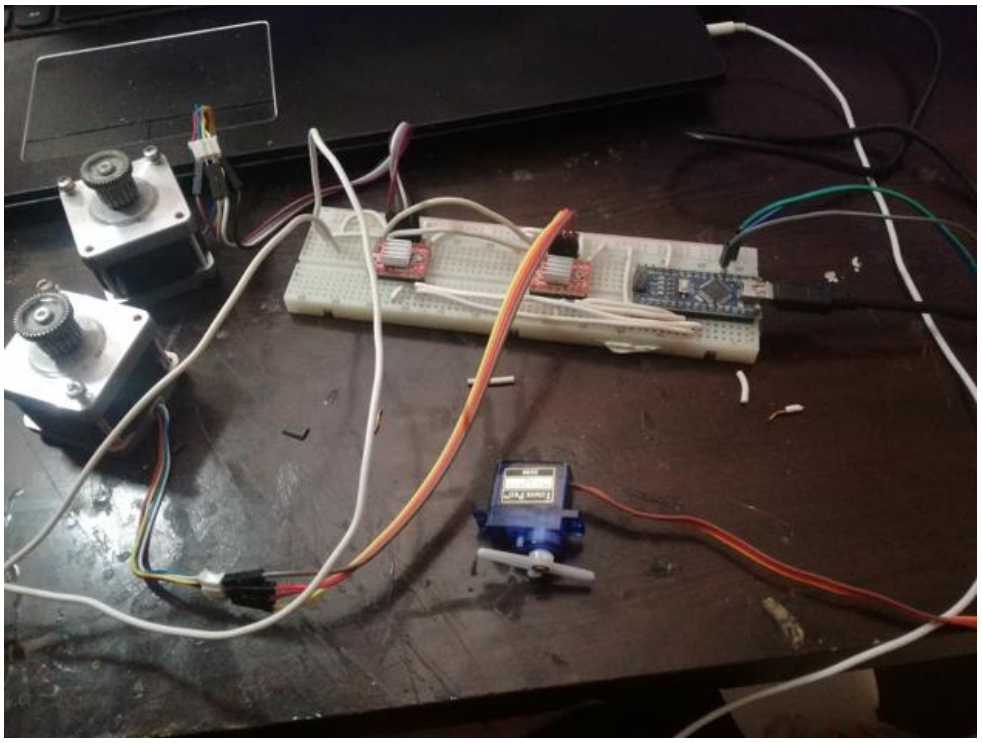


Figure 1.11: Picture of circuit built

After a few tests run we decided the 9g SG servo motor is too tiny and weak for poking paper so we had to replace it with another nema16 stepper motor just like the x-y axis motors along with its driver, the circuit came out to the following: (figure 1.12)

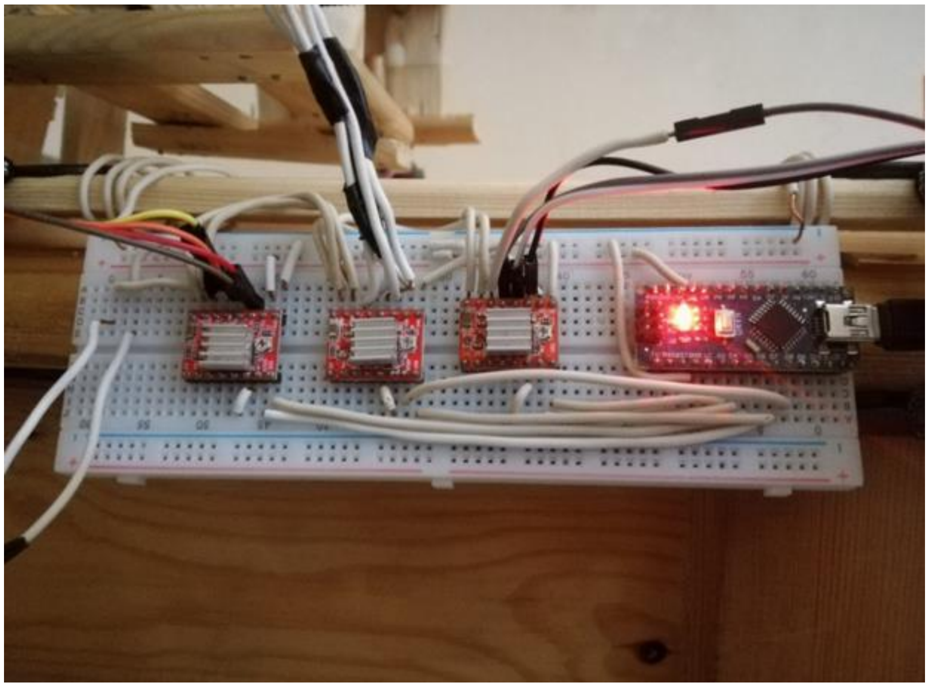


Figure 1.12: Driver Wiring

After that we used the timing belt along with the stepper motor to make the z-axis mechanism, after a few poking tests with all types of paper we concluded the best poking element was a syringe, it was possible to attach to our machine and was thin enough to poke paper consistently and every single time.

1.3.2 Structure and building

Due to the inability to obtain 3D printed pieces due to the school club not being able to provide them in time we resorted to building it fully in wood by cutting it and nailing it together, this is still a remarkable prototype with what we can obtain, so after a long time of sawing and nailing, we got the build! We designed the threading head with zip ties and nails with wood, aswell as the X and Y axies, we reinforced it with as much wood as possible for most accuracy. (figures 1.13, 1.14 and 1.15)

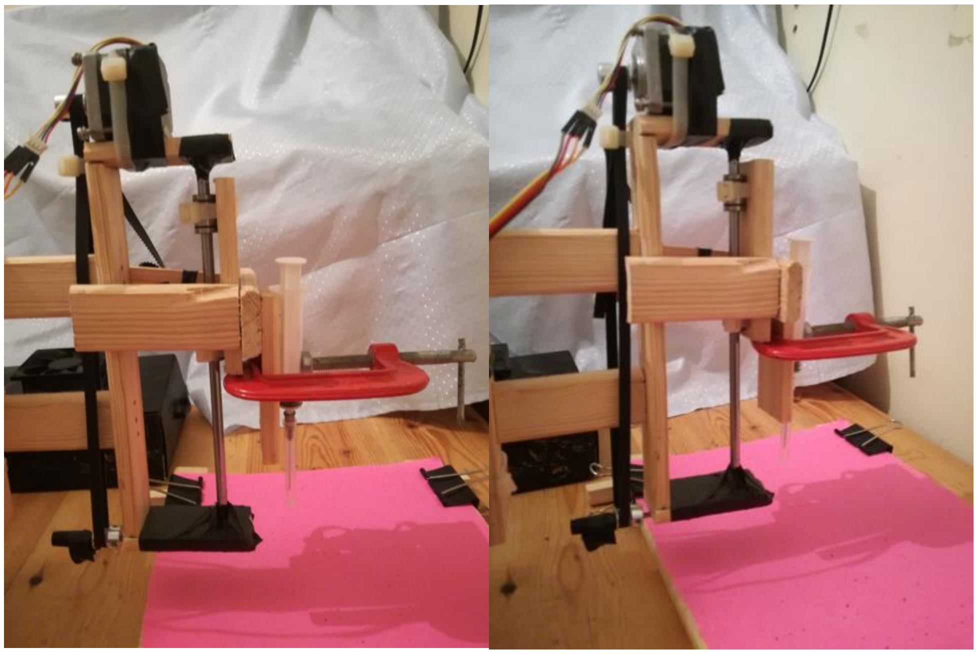


Figure 1.13: Wooden threading head mechanism

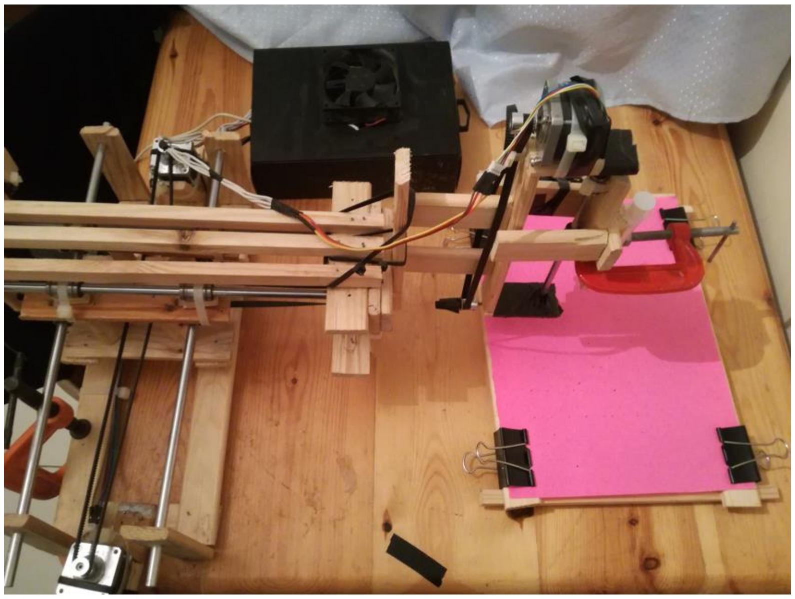


Figure 1.14: Top View of machine

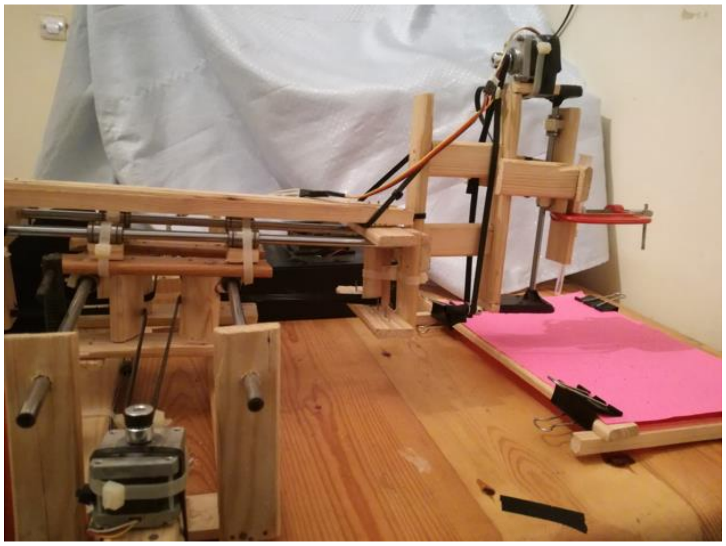


Figure 1.15: Side view of machine

1.3.3 The machine's motion: The way our machine moves left, right, back and forth is a result of the spinning combinations of the motors:

> Motors rotate in opposite directions: the machine goes back/forth depending on direction. (Figures 1.16 and 1.17)

> Both rotate clockwise: the machine goes left. (figure 1.18)

> Both rotate anti-clockwise: the machine goes right. (figure 1.19)

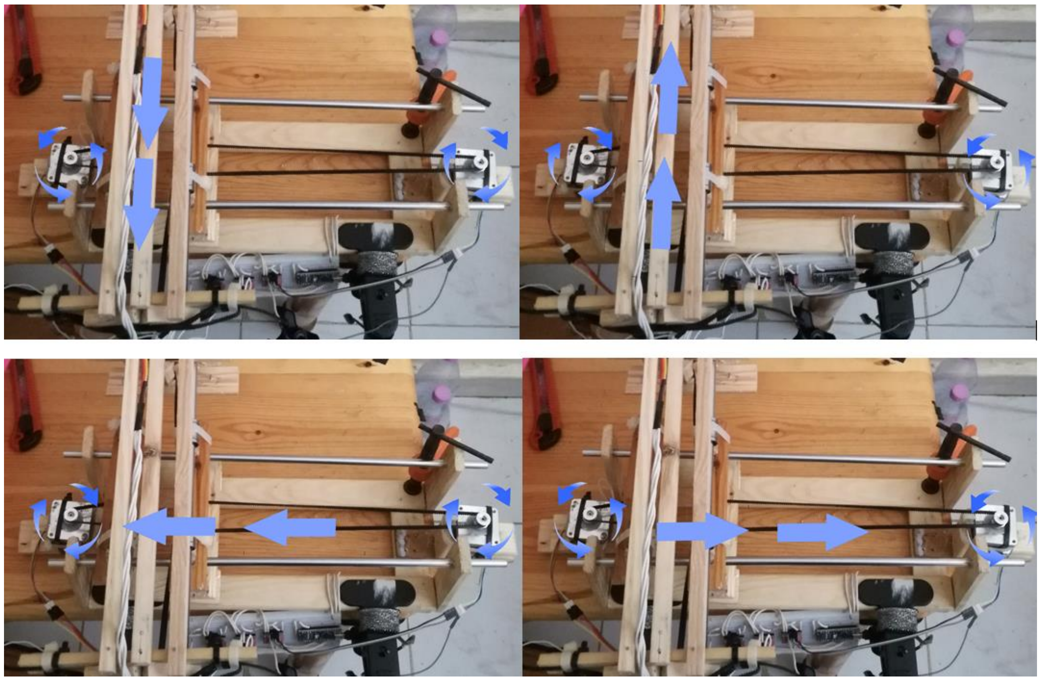


Figure 1.16: Going backwards(top left) **Figure 1.17: Going forward (top right)**

Figure 1.18: Going left (bottom left) **Figure 1.19: Going right (bottom right)**

We also have made a wooden frame that can hold a paper in an elevated manner which allows the syringe we attached to go through and not break on the table

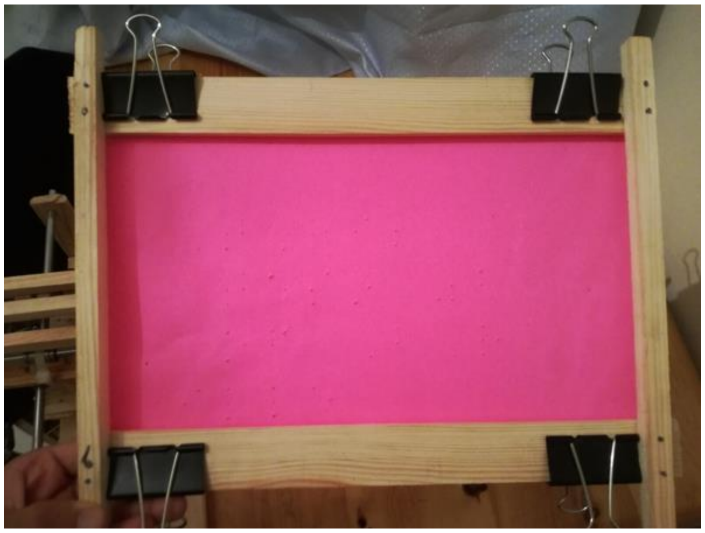


Figure 1.20: Paper holding frame

2 CHAPTER 2: SOFTWARE

No publicly available communication protocol converts regular text into machine coordinates for Braille printing so we have to write our own, our software follows the flowchart in figure 2.1 to control the machine

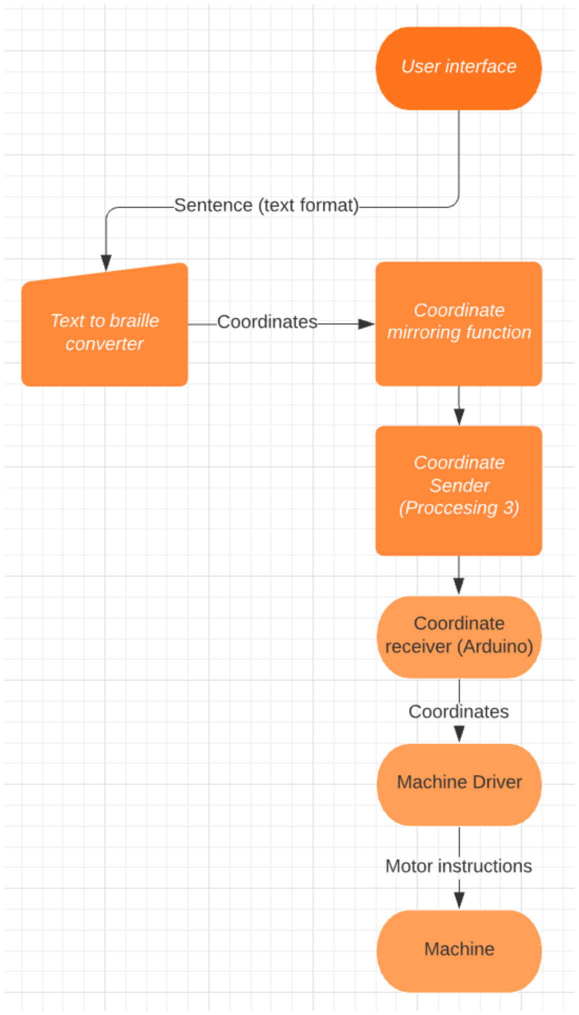

User interface
Sentence (text format)
Text to braille converter
Coordinates
Coordinate mirroring function
Coordinate Sender (Proccesing 3)
Coordinate receiver (Arduino)
Coordinates
Machine Driver
Motor instructions
Machine

Figure 2.1: Software flowchart

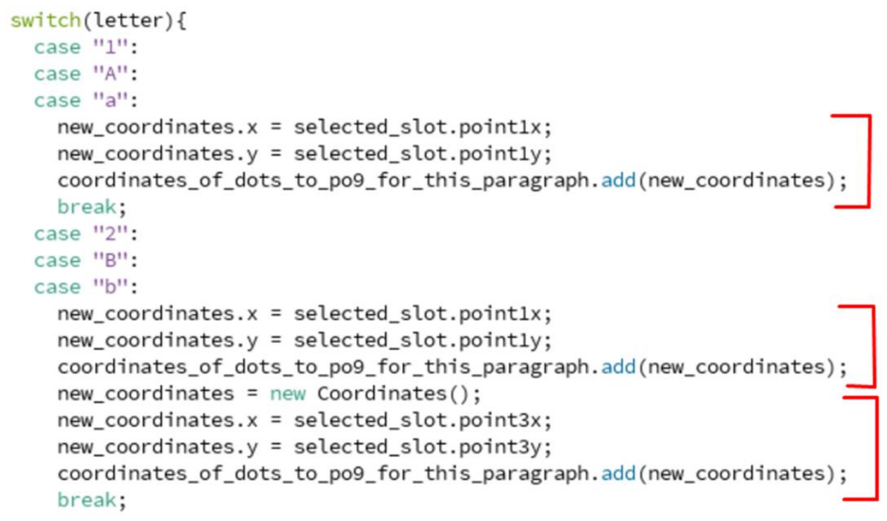
2.1 TEXT TO BRAILLE CONVERSION

We'll be writing this code on a program called Processing 3, it was originally made to code games but it's also easy in dealing with machines as it allows us to treat real-life machines as games that allow for many industrial innovations.

2.1.1 Text to braille dots switch case function:

We coded a basic converter, for starters we can use a simple switch case statement to decide which letter we have, for example, the letter A and the number 1 is designated by one dot on the top left, the letter B and number 2 are designated by two dots.

(in the code in figure 2. 2, we are taking the X and Y coordinates of each dot and adding them to our printing machine's list of dots that it needs to poke)



```
switch(letter){
  case "1":
  case "A":
  case "a":
    new_coordinates.x = selected_slot.point1x;
    new_coordinates.y = selected_slot.point1y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    break;
  case "2":
  case "B":
  case "b":
    new_coordinates.x = selected_slot.point1x;
    new_coordinates.y = selected_slot.point1y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    new_coordinates = new Coordinates();
    new_coordinates.x = selected_slot.point3x;
    new_coordinates.y = selected_slot.point3y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    break;
```

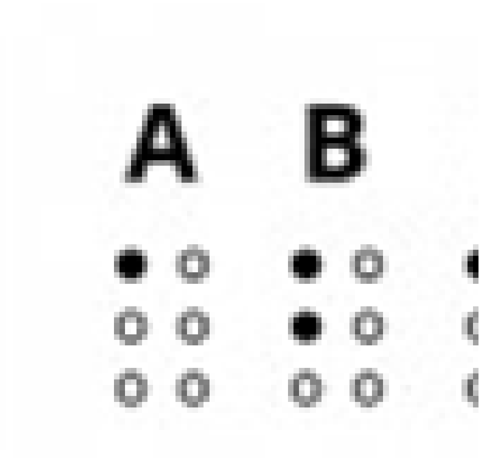
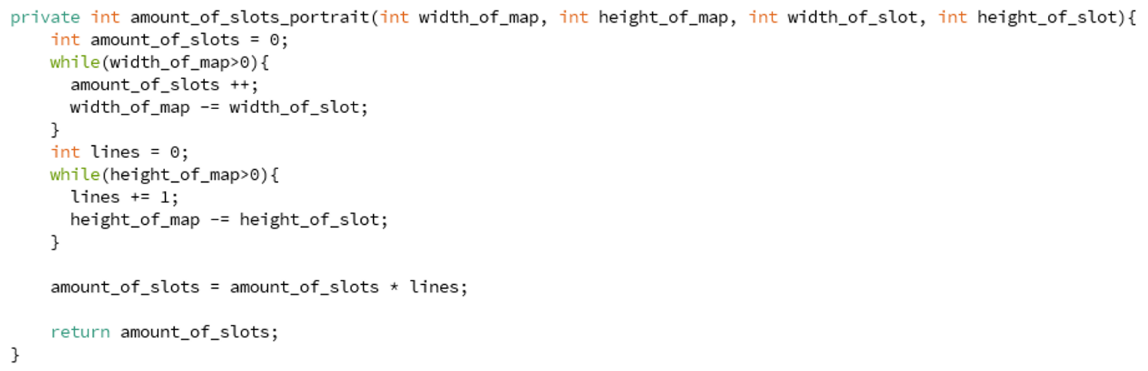


Figure 2.2: Text to braille conversion

2.1.2 Surface mapping function: One of the most important features we added was the size of the braille language on the real paper and it can be done with a simple function (figure 2.3)



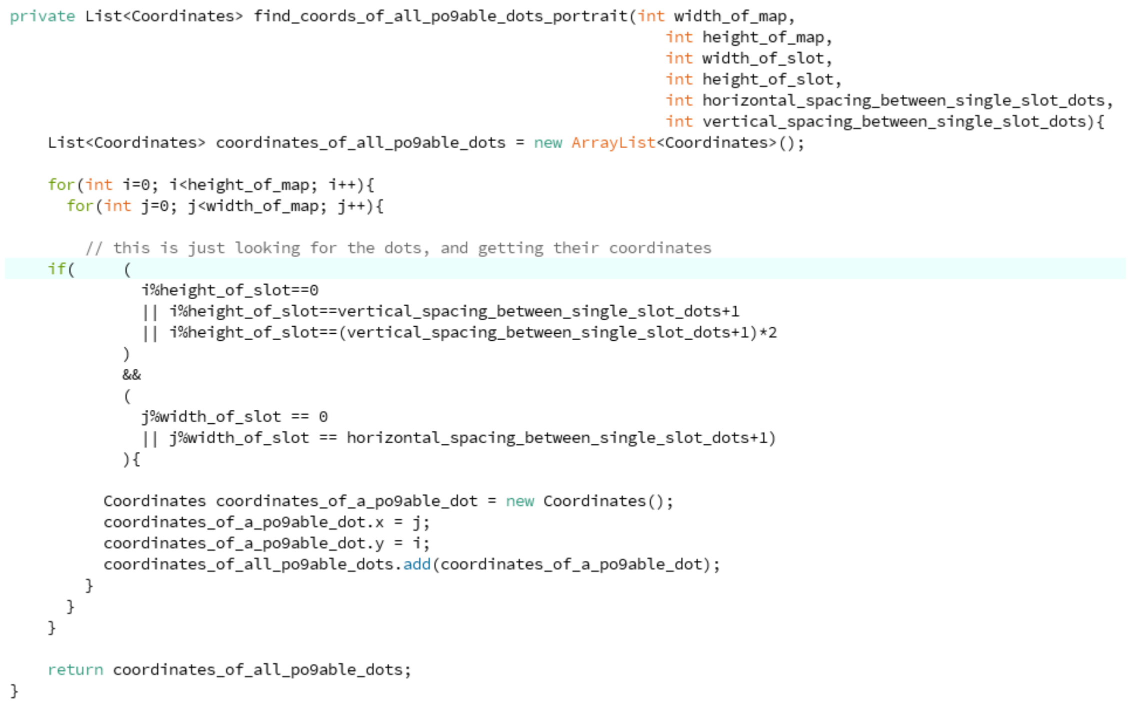
```
private int amount_of_slots_portrait(int width_of_map, int height_of_map, int width_of_slot, int height_of_slot){
    int amount_of_slots = 0;
    while(width_of_map>0){
      amount_of_slots ++;
      width_of_map -= width_of_slot;
    }
    int lines = 0;
    while(height_of_map>0){
      lines += 1;
      height_of_map -= height_of_slot;
    }

    amount_of_slots = amount_of_slots * lines;

    return amount_of_slots;
}
```

Figure 2.3: Mapping of the given space

This function can be given the total width and height of the entire paper, and the width and height of each letter and it would count how many letters it could fit in it and returns amount_of_slots, this variable is essential in the printing process as it allows the code to quickly place any sentence into the required slots and instantly send it to the machine.

How this function works is looping over the available resolution of the map and count how many times we can fit our designated width_of_slot and height_of_slot.

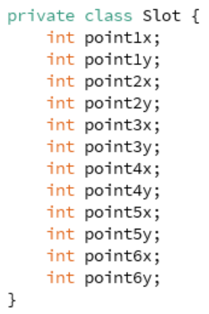
2.1.3 Slot coordinate finding function: Now that we got the amounts of slots available, we need to know the exact coordinates of all of those slots and this next function does just that, it takes in the spacing between each slot that is given by the user and the given height & width to calculate the exact coordinates of each dot it needs to poke, this function is what ensures each dot is properly placed relative to the others and letters are consistently spaced away from each other. (figure 2.4)



```
private List<Coordinates> find_coords_of_all_po9able_dots_portrait(int width_of_map,
                                                                      int height_of_map,
                                                                      int width_of_slot,
                                                                      int height_of_slot,
                                                                      int horizontal_spacing_between_single_slot_dots,
                                                                      int vertical_spacing_between_single_slot_dots){
    List<Coordinates> coordinates_of_all_po9able_dots = new ArrayList<Coordinates>();

    for(int i=0; i<height_of_map; i++){
      for(int j=0; j<width_of_map; j++){

        // this is just looking for the dots, and getting their coordinates
    if(     (
              i%height_of_slot==0
              || i%height_of_slot==vertical_spacing_between_single_slot_dots+1
              || i%height_of_slot==(vertical_spacing_between_single_slot_dots+1)*2
            )
            &&
            (
              j%width_of_slot == 0
              || j%width_of_slot == horizontal_spacing_between_single_slot_dots+1)
            ){

          Coordinates coordinates_of_a_po9able_dot = new Coordinates();
          coordinates_of_a_po9able_dot.x = j;
          coordinates_of_a_po9able_dot.y = i;
          coordinates_of_all_po9able_dots.add(coordinates_of_a_po9able_dot);
        }
      }
    }

    return coordinates_of_all_po9able_dots;
}
```

Figure 2.4: Slot coordinate finding function

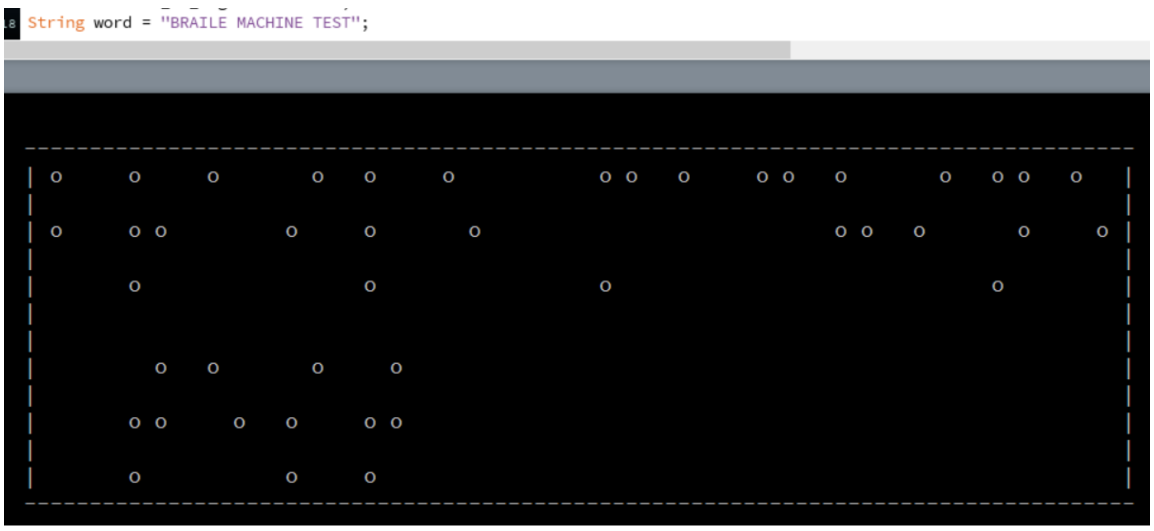
2.1.4 Slot class: We are storing all of these coordinates and so we need a Slot architecture and it contains the x and y for each of its 6 slots. (figure 2.5)



```
private class Slot {
    int point1x;
    int point1y;
    int point2x;
    int point2y;
    int point3x;
    int point3y;
    int point4x;
    int point4y;
    int point5x;
    int point5y;
    int point6x;
    int point6y;
}
```

Figure 2.5: Slot class

2.1.5 Console test printing: Now everything is organized and we can pass the sentences to our code and it would return a bunch of coordinates that our machine could use, we can plot that on the computer as an example temporarely: (figure 2.6)



```
String word = "BRAILE MACHINE TEST";
```

```
-------------------------------------------
|o  o  o   o o  o     oo o  oo o   o oo o |
|                                         |
|o  oo    o  o   o             oo o   o  o|
|                                         |
|   o        o        o              o     |
|                                         |
|                                         |
|    o o   o  o                           |
|                                         |
|   oo  o o  oo                           |
|                                         |
|   o     o  o                            |
-------------------------------------------
```

Figure 2.6: Console test of sentence "BRAILLE MACHINE TEST"

We've made it super easy to use the function, the user inputs a sentence and it would return a set of coordinates. (figure 2.7)

```
Pair<String, List<Coordinates>> p = converter(word);
```

Figure 2.7: converter function use

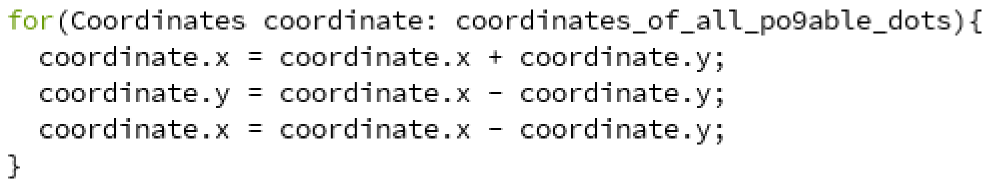
2.1.6 Main function: In this converter function, we use all of the functions 2.1.1 to 2.1.5 to determine the necessary information to be able to convert words (figure 2.8)

```
// Preparation 1: find the resolution of each slot
int height_of_slot = vertical_spacing_between_single_slot_dots*2 + vertical_spacing_between_full_slots + 3;
int width_of_slot = horizontal_spacing_between_single_slot_dots + horizontal_spacing_between_full_slots + 2;

if(rotate_90_degrees){
  height_of_slot++;
}
// Preparation 2: find the resolution of our print
int width_of_map = slots_per_line*width_of_slot;
int height_of_map = lines*height_of_slot;



// Preparation 3: find how many slots there can be
int amount_of_slots = amount_of_slots_portrait(width_of_map, height_of_map, width_of_slot, height_of_slot);
```

Figure 2.8: Main function layout

The user can also choose the rotation of the text allowing for horizontal and vertical braille prints on paper.

If the user had chosen vertical prints all we need to do is to switch between our x and y coordinates for each dot and that's it!

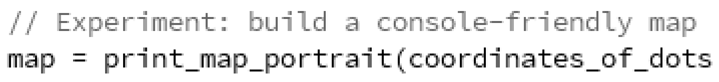


```
for(Coordinates coordinate: coordinates_of_all_po9able_dots){
  coordinate.x = coordinate.x + coordinate.y;
  coordinate.y = coordinate.x - coordinate.y;
  coordinate.x = coordinate.x - coordinate.y;
}
```

Figure 2.9: Reordering coordinates

Figure 2.9 seems unnecessary but it is a CPU friendly way of exchanging values between two variables, we first add the two values and store them in the first variable and then we subtract the results and store them in the second variable which eliminates one of the values and then we just subtract that value from the total and get the second value, essentially this for loop in figure 2.7 is just exchanging values between both variables x and y.

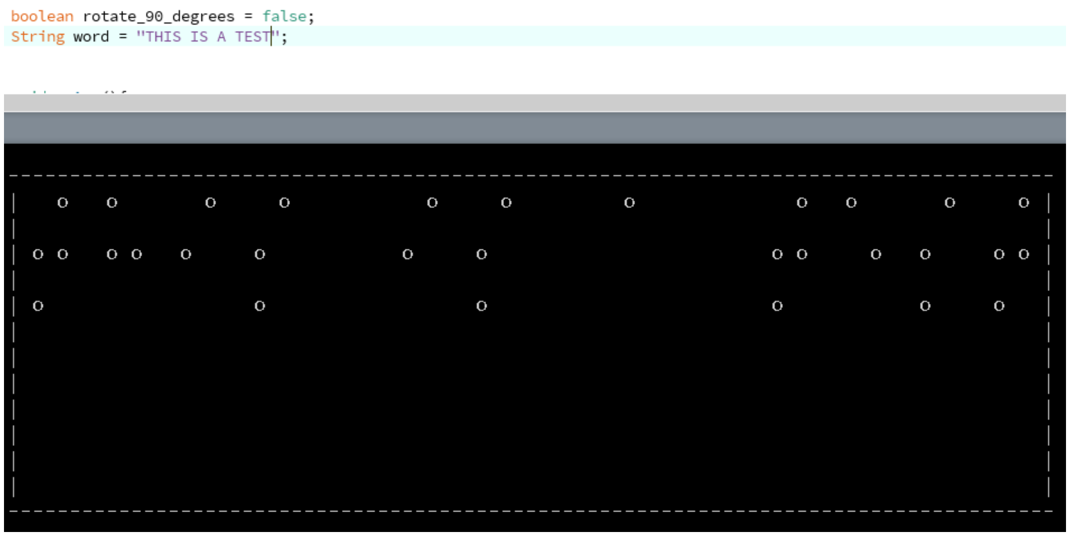
Using a simple print function to debug the example on the console and we named that function print_map_portait. (figure 2.10)



```
// Experiment: build a console-friendly map
map = print_map_portrait(coordinates_of_dots
```

Figure 2.10: Usage of console test function

This gets us this test print in figure 2.11!



```
boolean rotate_90_degrees = false;
String word = "THIS IS A TEST";
```

```
-------------------------------------------------------------------------------------
|   o   o       o     o           o     o         o             o   o       o     o |
|                                                                                   |
| o o   o o   o     o           o     o                       o o     o   o     o o |
|                                                                                   |
| o                 o                 o                       o           o     o   |
|                                                                                   |
|                                                                                   |
|                                                                                   |
|                                                                                   |
|                                                                                   |
|                                                                                   |
|                                                                                   |
-------------------------------------------------------------------------------------
```

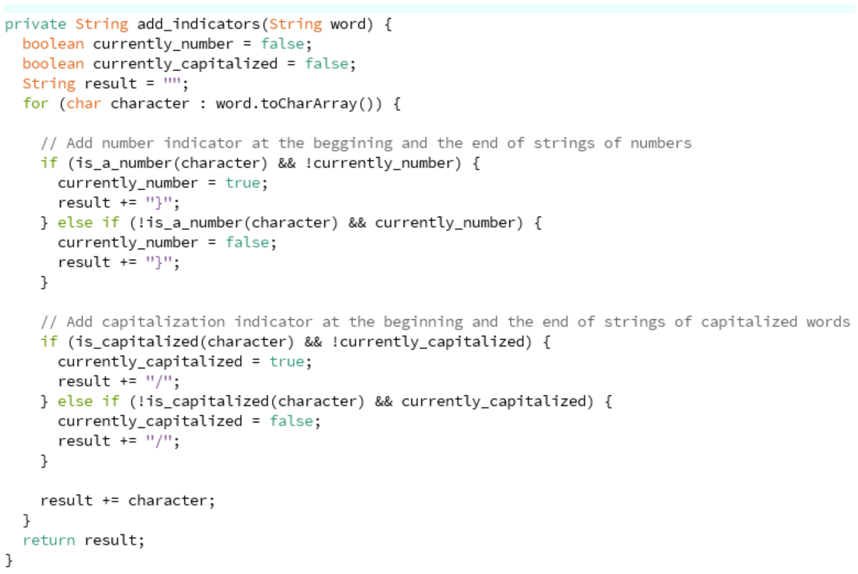
Figure 2.11: Console test of sentence "THIS IS A TEST"

As apparent in the console rotate_90_degrees Is set to false indicating this is a landscape regular print, and the sentence printed is THIS IS A TEST, comparing this to the string we notice it is missing the letter indicating symbols that are put between letters and numbers to distinguish between them so let's add them!

2.1.7 Adding number & capital letter indicators: We can easily name them, for example, the symbol } is for the Numeric Indicator, and / for the Capitalized Letter Indicator and have them be added to this example "{/This is test number }1" of course the braille reader won't be reading these brackets in his paper, our code will convert these brackets into the desired indicators, now all we need to do is write a function that adds these indicators into our sentence.

We will create the add_indicators() function will add these indicators according to our sentence's capitalization and numbers.

And it simply works by looping over the entire sentence and checking if we have switched from letters to numbers and vice versa, it also checks for capitalization, we repeat, these brackets are not going to be read by anyone but our machine to convert them into the appropriate indicators in braille language to be read by visually impaired people. (figure 2.12 and figure 2.14)



```java
private String add_indicators(String word) {
  boolean currently_number = false;
  boolean currently_capitalized = false;
  String result = "";
  for (char character : word.toCharArray()) {

    // Add number indicator at the beggining and the end of strings of numbers
    if (is_a_number(character) && !currently_number) {
      currently_number = true;
      result += "}";
    } else if (!is_a_number(character) && currently_number) {
      currently_number = false;
      result += "}";
    }

    // Add capitalization indicator at the beginning and the end of strings of capitalized words
    if (is_capitalized(character) && !currently_capitalized) {
      currently_capitalized = true;
      result += "/";
    } else if (!is_capitalized(character) && currently_capitalized) {
      currently_capitalized = false;
      result += "/";
    }

    result += character;
  }
  return result;
}
```

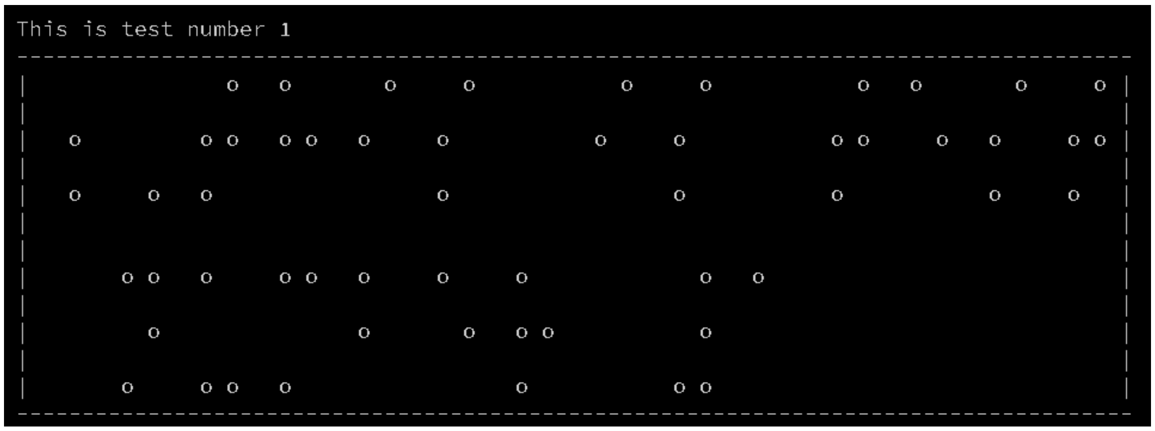
Figure 2.12: Adding indicators

Last but not least we add the symbols into our switch case to have them converted into their designated symbols:

This is essentially just adding the x and y-axis of the dots

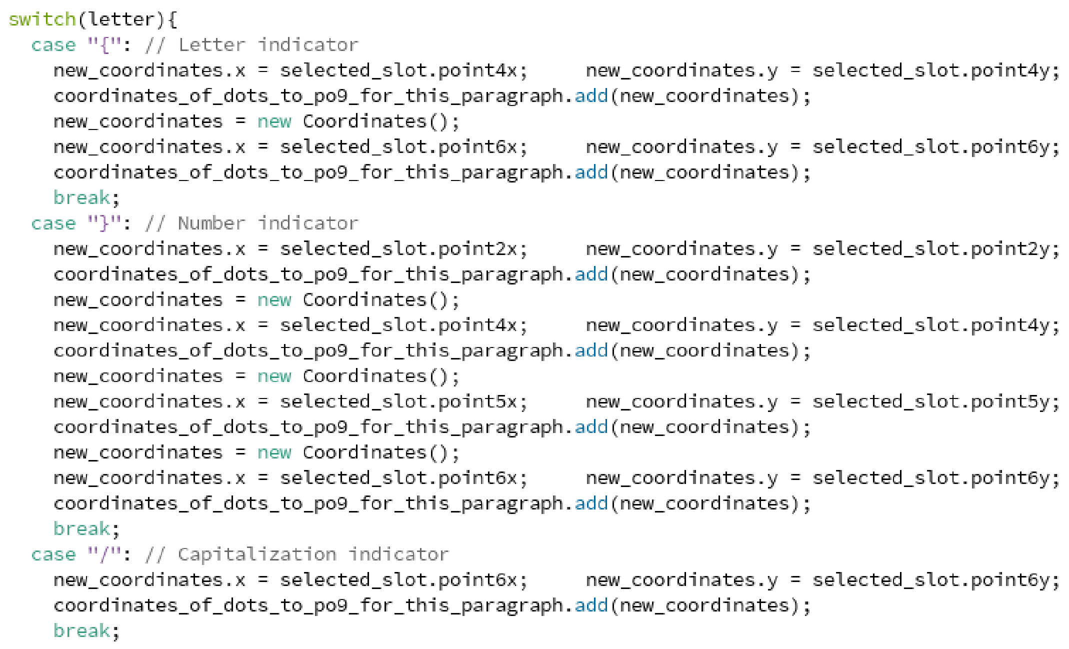
- ➢ Number Indicator: checks the dots number 2, 4, 5, and 6.
- ➢ Capitalization Indicator: only checks the dot number 6.

And now every sentence includes all the required indicators as appears in the figure 2.13



```
This is test number 1
-------------------------------------------------------------------------------------
|               o   o       o     o           o     o           o   o       o     o |
|                                                                                   |
|   o         o o   o o   o     o           o     o           o o     o   o     o o |
|                                                                                   |
|   o     o   o                 o                 o           o           o     o   |
|                                                                                   |
|                                                                                   |
|       o o   o     o o   o     o     o             o   o                           |
|                                                                                   |
|         o               o       o   o o           o                               |
|                                                                                   |
|       o     o o   o                 o           o o                               |
-------------------------------------------------------------------------------------
```

Figure 2.13: Console test including indicators of sentence "This is test number 1"

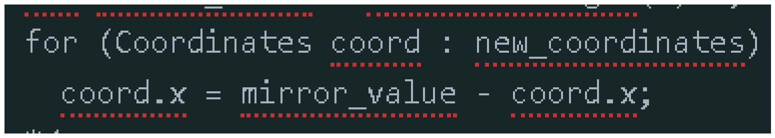


```
switch(letter){
  case "{": // Letter indicator
    new_coordinates.x = selected_slot.point4x;     new_coordinates.y = selected_slot.point4y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    new_coordinates = new Coordinates();
    new_coordinates.x = selected_slot.point6x;     new_coordinates.y = selected_slot.point6y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    break;
  case "}": // Number indicator
    new_coordinates.x = selected_slot.point2x;     new_coordinates.y = selected_slot.point2y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    new_coordinates = new Coordinates();
    new_coordinates.x = selected_slot.point4x;     new_coordinates.y = selected_slot.point4y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    new_coordinates = new Coordinates();
    new_coordinates.x = selected_slot.point5x;     new_coordinates.y = selected_slot.point5y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    new_coordinates = new Coordinates();
    new_coordinates.x = selected_slot.point6x;     new_coordinates.y = selected_slot.point6y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    break;
  case "/": // Capitalization indicator
    new_coordinates.x = selected_slot.point6x;     new_coordinates.y = selected_slot.point6y;
    coordinates_of_dots_to_po9_for_this_paragraph.add(new_coordinates);
    break;
```

Figure 2.14: Function which detects indicators

2.1.8 Mirror effect problem:

Since our machine is threading the paper up-to-down this means the raised dots are at the bottom, which means our braille paper needs to be read from the oppsite side, how we've achieved this is by inverting the coordinates of the dots, this is done by substracting every coordinate from the maximum size of the map, for only X axis like in figure 2.15



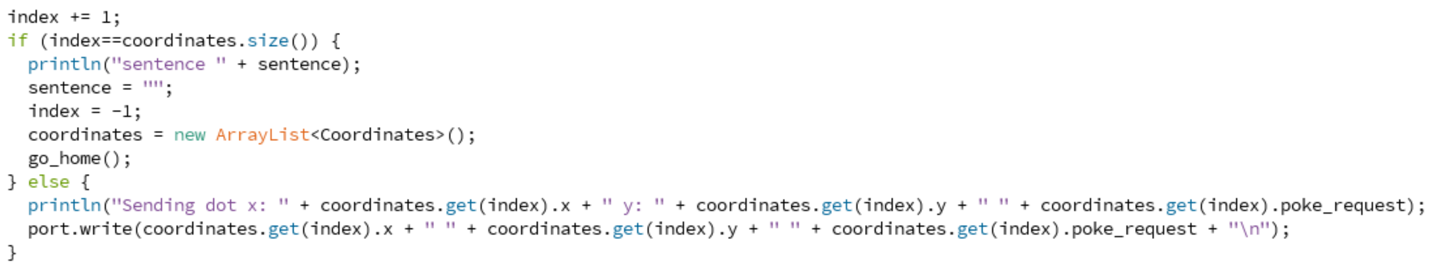
```
for (Coordinates coord : new_coordinates)
  coord.x = mirror_value - coord.x;
```

Figure 2.15: Coordinate mirror effect

2.2 COMMUNICATION BETWEEN SOFTWARE AND HARDWARE

2.2.1 SENDING SIDE: PROCESSING 3

2.2.1.a) coordinate looping function: All we need to do is to loop over our coordinates received from the converter and send them through Serial to our Arduino (figure 2.16)

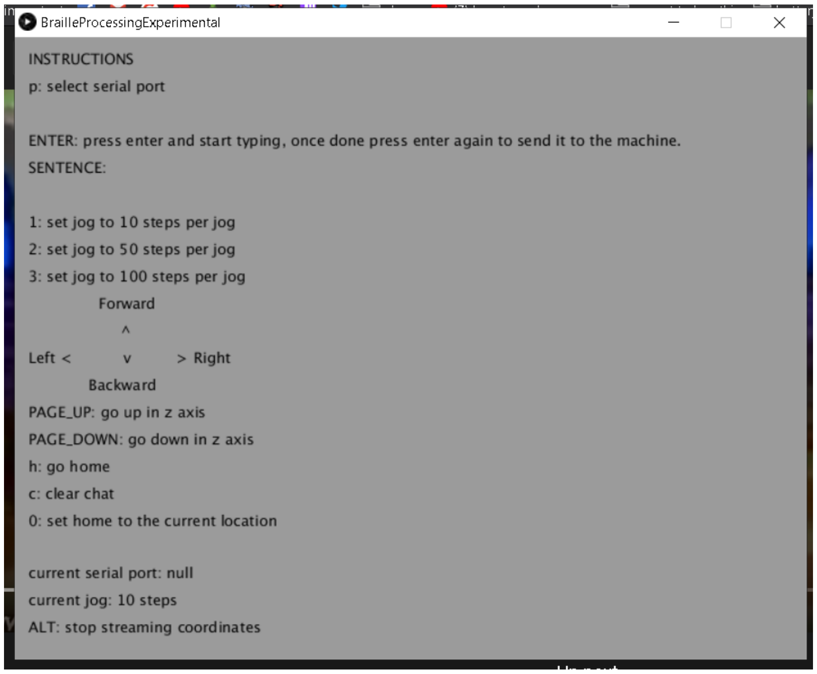


```
index += 1;
if (index==coordinates.size()) {
  println("sentence " + sentence);
  sentence = "";
  index = -1;
  coordinates = new ArrayList<Coordinates>();
  go_home();
} else {
  println("Sending dot x: " + coordinates.get(index).x + " y: " + coordinates.get(index).y + " " + coordinates.get(index).poke_request);
  port.write(coordinates.get(index).x + " " + coordinates.get(index).y + " " + coordinates.get(index).poke_request + "\n");
}
```

Figure 2.16: Function which loops over coordinates to transmit to the machine

We also expect a response with the text “Done” after the machine has poked the previous dot so we don’t overlap requests.

2.2.1.b) Building a GUI (Graphical user interface): We built a simple GUI (figure 2.17) that lets the user type a sentence and send it, this GUI is thankfully very easy to make as processing 3 allows us to add lines with 1 line of code.



```
BrailleProcessingExperimental

INSTRUCTIONS
p: select serial port

ENTER: press enter and start typing, once done press enter again to send it to the machine.
SENTENCE:

1: set jog to 10 steps per jog
2: set jog to 50 steps per jog
3: set jog to 100 steps per jog
            Forward
               ^
Left <         v          > Right
          Backward
PAGE_UP: go up in z axis
PAGE_DOWN: go down in z axis
h: go home
c: clear chat
0: set home to the current location

current serial port: null
current jog: 10 steps
ALT: stop streaming coordinates
```

```
void draw() {

  background(155);
  fill(0);
  int y = 24, dy = 24;
  text("INSTRUCTIONS", 12, y);
  y += dy;
  text("p: select serial port", 12, y);

  textSize(13);
  y += 2*dy;
  text("ENTER: press enter and start typing, once done
  y += dy;
  text("SENTENCE: " + sentence, 12, y);
  y += 2*dy;

  text("1: set jog to 10 steps per jog", 12, y);
  y += dy;
  text("2: set jog to 50 steps per jog", 12, y);
  y += dy;
  text("3: set jog to 100 steps per jog", 12, y);
  y += dy;
  text("               Forward", 12, y);
  y += dy;
  text("                    ^", 12, y);
  y += dy;
  text("Left <           v          > Right", 12, y);
  y += dy;
  text("             Backward", 12, y);
  y += dy;
  text("PAGE_UP: go up in z axis", 12, y);
  y += dy;
  text("PAGE_DOWN: go down in z axis", 12, y);
  y += dy;
  text("h: go home", 12, y);
  y += dy;
  text("c: clear chat", 12, y);
```

Figure 2.17: GUI code and visualization

2.2.2 RECEIVING SIDE: ARDUINO IDE

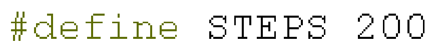
2.2.2.a) Stepper motor control:

We are using Arduino IDE to program our microcontroller to receive a single type of instruction which consists of a string of three numbers spaced between:

$$X-\text{coordinate}\quad Y-\text{coordinate}\quad \text{Poke}-\text{request}$$

The coordinates our Text To Braille Converter are going to be sent in a simple string one by one to our machine, the string will contain x and y coordinates as well as a poke-request this poke request will either be a 0 or a 1, it is a 1 if we want our machine to poke the paper at the coordinates provided and 0 means it will not poke the paper and will just go to the x and y coordinates, this is needed to send the machine back to 0 0 after finishing the printing so it is ready for future prints.

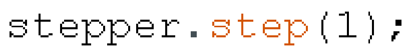
Now that we have built the circuit, we are going to control it using Arduino IDE, this code is written in a modified version of the C language made for Arduino and it will help us control our motors to steer a proper braille machine.



```
#define STEPS 200
```

Figure 2.18: Defining the motor's steps

If we determine the stepper motor's STEPS variable to be 180, it will step 1 degree each time, but a universally used value is 200 for more accuracy yet less jittering for NEMA 16/17 stepper motors. (figure 2.18)

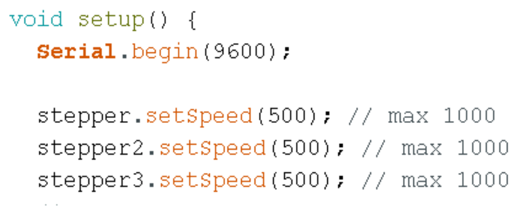


```
stepper.step(1);
```

Figure 2.19: Stepping the motor clockwise

Now it is as easy as one line of code to step each of our stepper motors in any direction by simply entering positive and negative values into the step() function for both stepper and stepper2.

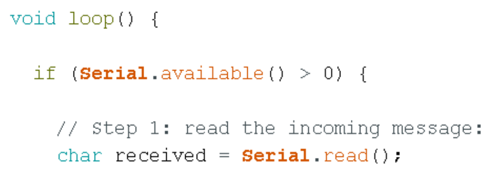
Now, we receive the string and order the motors to move in the following way in figure 2.19



```
void setup() {
  Serial.begin(9600);

  stepper.setSpeed(500); // max 1000
  stepper2.setSpeed(500); // max 1000
  stepper3.setSpeed(500); // max 1000
```

Figure 2.20: Setting speeds of motors at half speed

We set the speeds at 50% as they were enough to steer while not pulling as much current, we also initialized the Serial to start listening for coordinates! (figure 2.20)



```
void loop() {

  if (Serial.available() > 0) {

    // Step 1: read the incoming message:
    char received = Serial.read();
```

Figure 2.21: Serial function to receive instructions

A void loop is an internal function from Arduino that keeps running again and again as long as the Arduino is on, we keep checking for messages and if they arrive we read them to add them to our full string. (figure 2.21)

2.2.2.b) Serial coordinate decoding and processing:

```
// Step 3.4: if both coordinates were inputted then go to them
if(coords_are_queued){

  // Step 3.4.1: go to coordinates
  // Step 3.4.1.1: x axis
  int x_difference = coordinates_x.toInt()-x;
  step_motors_this_much(x_difference, "x");

  delay(630);
  // Step 3.4.1.2: y axis
  int y_difference = coordinates_y.toInt()-y;
  step_motors_this_much(y_difference, "y");

  // Step 3.4.1.3: z axis
  run_poke_request();

  // Step 3.4.3: aftermath cleanup & preparation to continue
  x = coordinates_x.toInt();
  y = coordinates_y.toInt();
```

Figure 2.22: Extracting coodinates coordinates

We calculate the difference between the current position of the machine and the received coordinates and we start stepping it towards it, we also run poke_request which uses step() if poke_request = 1. (figure 2.22)

After that, we send a message “Done” to confirm the machine has indeed poked the paper. (figure 2.23)

```
Serial.println("Done");
```

Figure 2.23: Microcontroller response

And that is it, we press upload and our code gets uploaded on the Arduino board!

3 CHAPTER 3: EXPERIMENTATION & RESULT

In this section, we will be experimenting with the machine and testing its running conditions.

3.1 Power consumption: After switching the power supply on, we can remark that the three motors are pulling 1.21A from our DC power supply that we set at 11.9V. (figure 3.1)

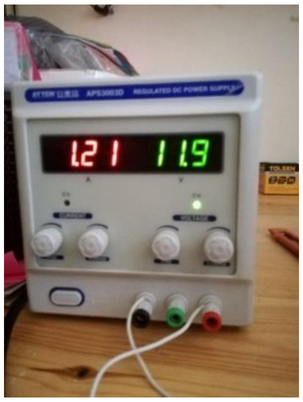
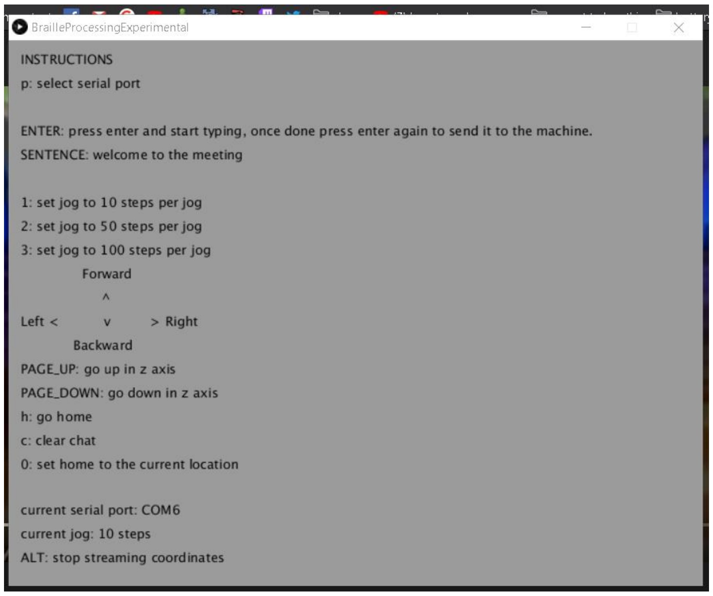


Figure 3.1: Voltage and current

We run our Processing 3 code and the GUI shows up with the Arduino on COM6 (a USB port) (figure 3.2)



```
INSTRUCTIONS
p: select serial port

ENTER: press enter and start typing, once done press enter again to send it to the machine.
SENTENCE: welcome to the meeting

1: set jog to 10 steps per jog
2: set jog to 50 steps per jog
3: set jog to 100 steps per jog
           Forward
               ^
Left <         v        > Right
          Backward
PAGE_UP: go up in z axis
PAGE_DOWN: go down in z axis
h: go home
c: clear chat
0: set home to the current location

current serial port: COM6
current jog: 10 steps
ALT: stop streaming coordinates
```

Figure 3.2: State of GUI when connected

3.2 Practical Tests & Inspection: We wrote the sentence “Welcome to the meeting” as a test runs for our machine, this sentence was suggested by our promoting teacher during a zoom conference call to test the machine

We press enter and the machine starts moving after having converted the sentence into coordinates and started the organized stream of coordinates.

We noticed the stepper motors being loud due to vibrations, that thankfully can be fixed by adding capacitors in parallel with the terminals of each coil of our stepper motors which lessens jittering and vibrations.

Our machine was traveling towards the coordinates and pulling down the syringe towards the paper and effectively poking through it leaving consistent and strong dots that can be felt with the human hand very comfortably, thankfully due to our mirrored map in the converter code, our machine was printing in a mirrored way which allows the reader to read from the other side of the paper which has holes poking out of the paper instead of into the paper which made it way easier to read.

After our machine has finished printing and went back to 0 0 we unmounted the paper from the simple wooden holder we made for it, the first thing we see is a very clearly aligned set of dots for our sentence.

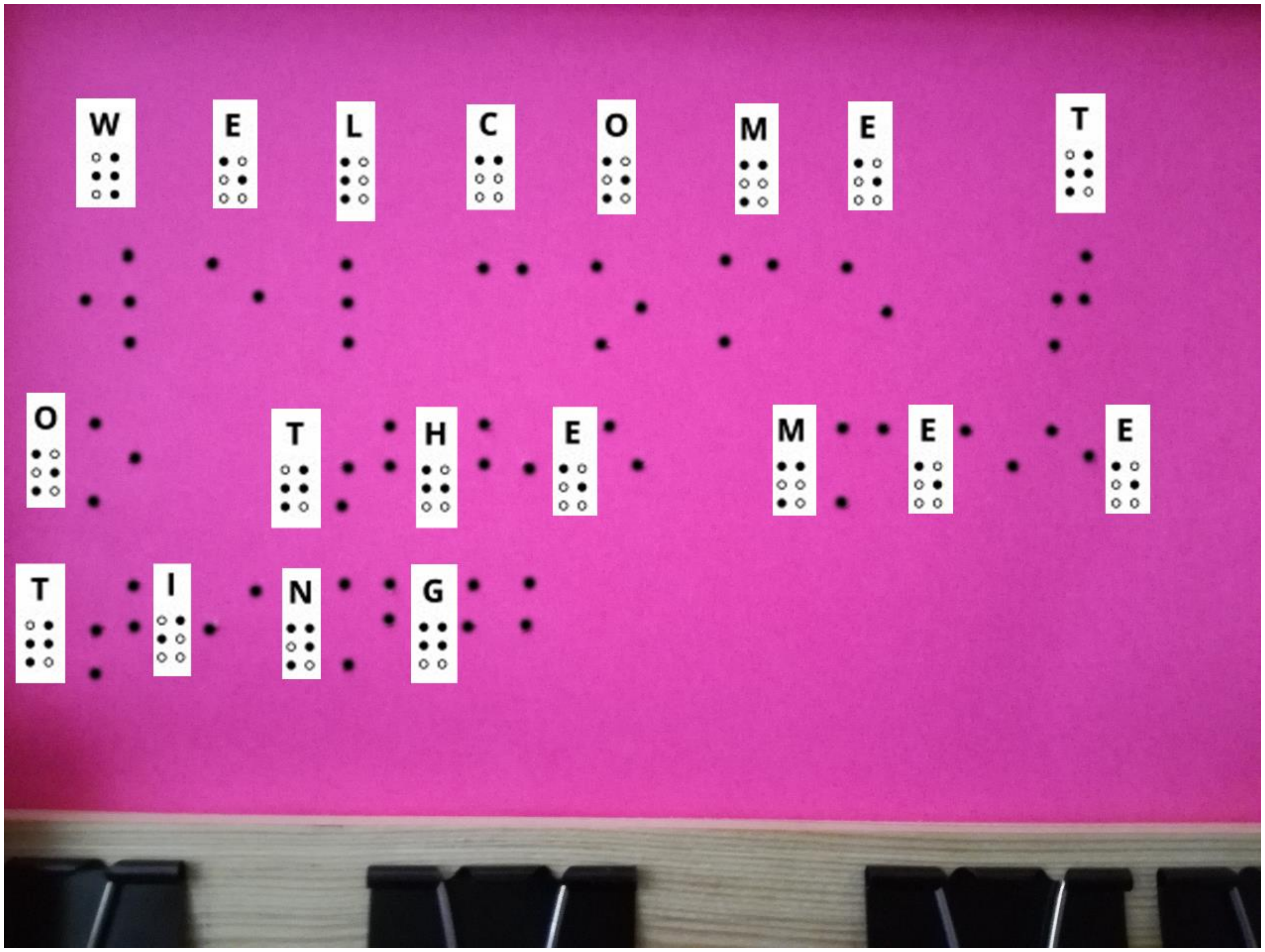


Figure 3.3: Inspected print

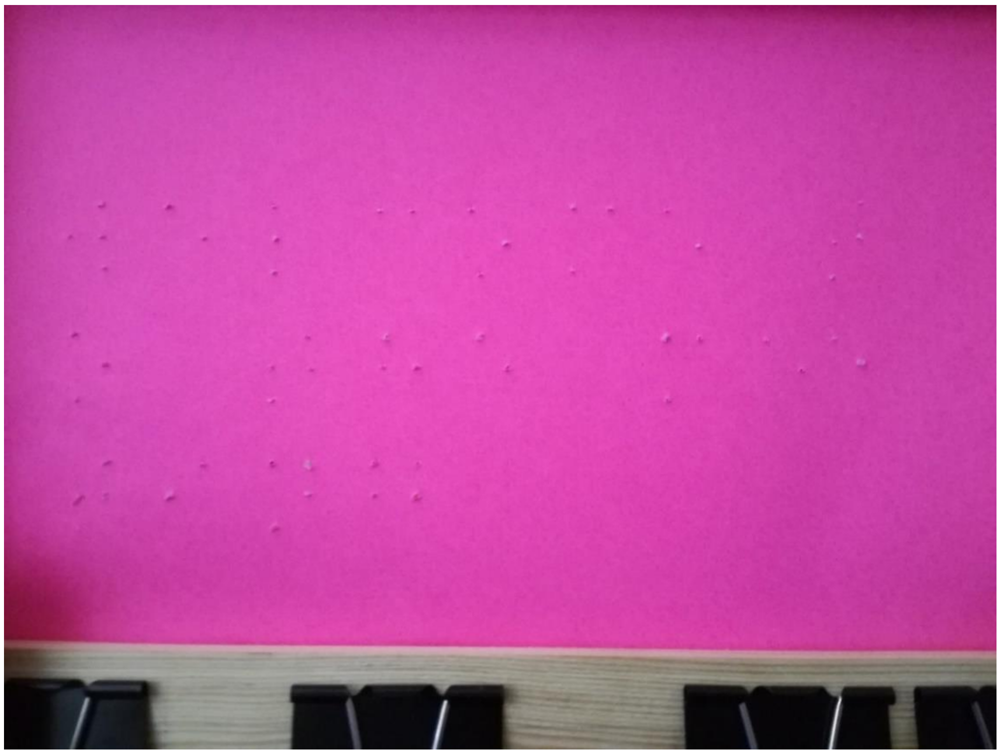


Figure 3.4: Uninspected print

The sentence tested in figures 3.3 and 3.4 was “welcome to the meeting”. The next test is to cover the treatment of numbers and capitals by our machine (number indicator and the capitalization indicator).

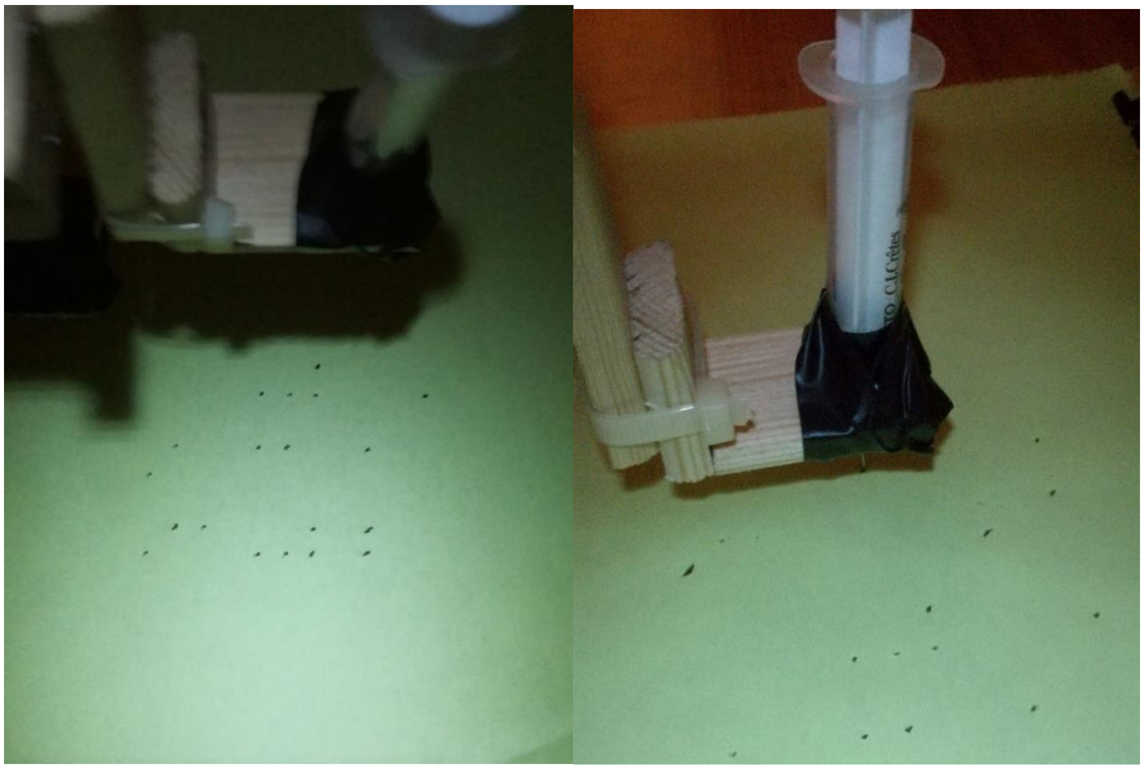


Figure 3.5: Threading mechanism in action

So after a few minutes of printing, we got the following result in figure 3.6 (the dots couldn't appear extremely well on camera but they're physically very apparent and easy to read, this is due to the machine being hand-made with wood and the inaccuracy that comes with it, nonetheless it's still a remarkable result for a fully handmade machine that's missing proper 3D printed pieces.

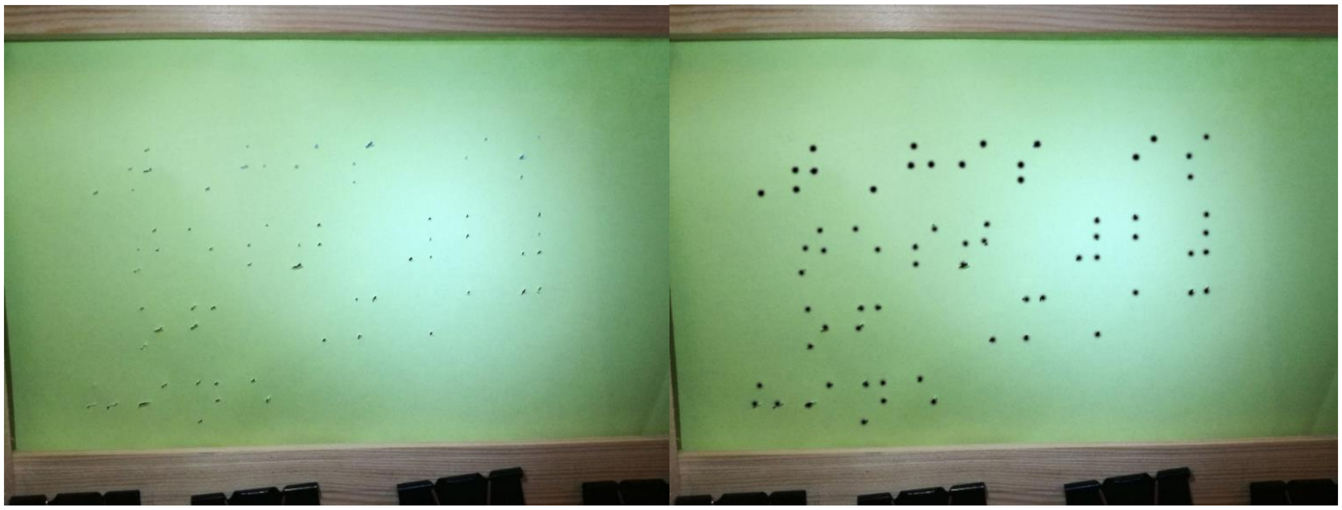


Figure 3.6: Second test, highlighted and dehighlighted

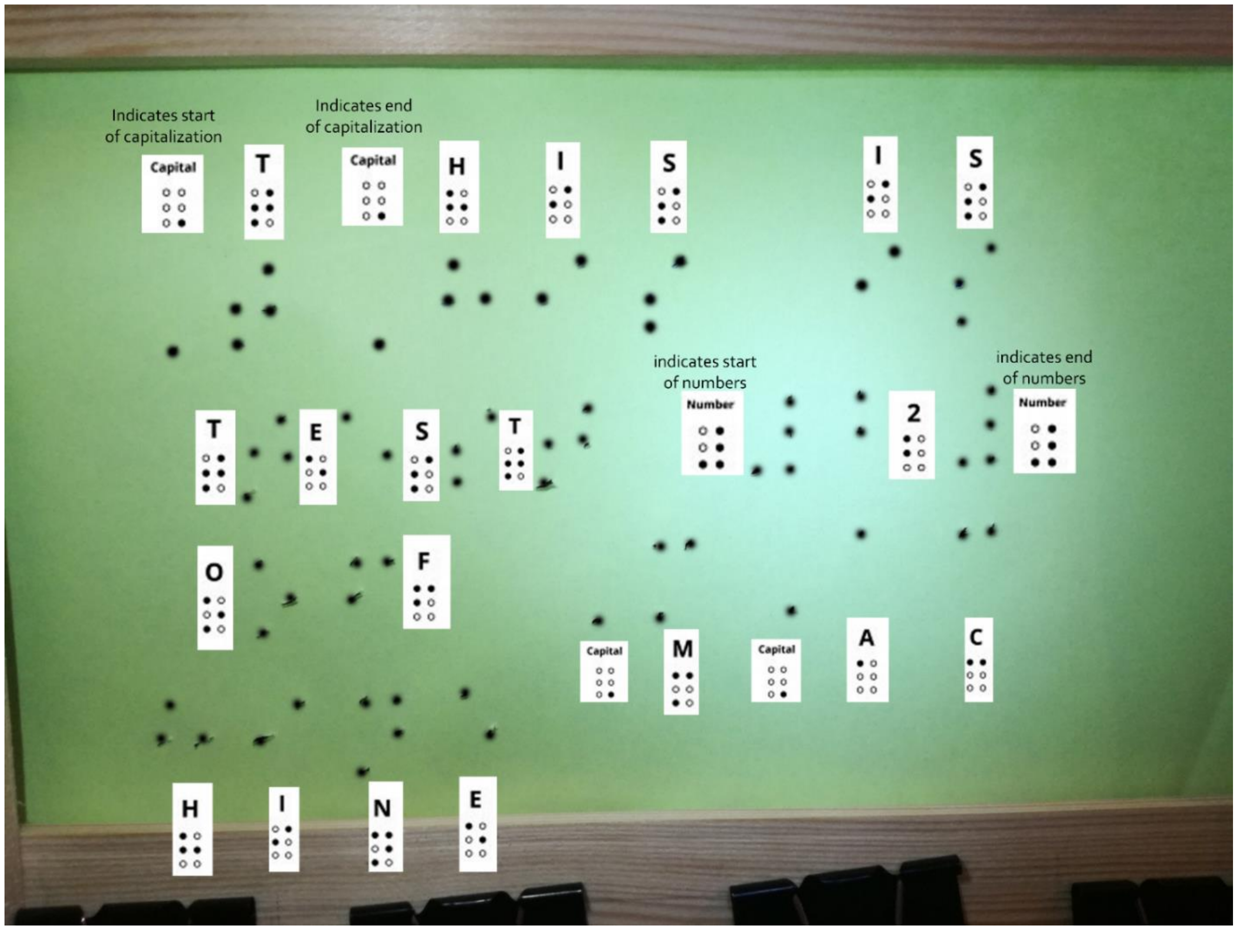

Indicates start of capitalization
Capital
T
Indicates end of capitalization
Capital
H
I
S
I
S
indicates start of numbers
Number
2
indicates end of numbers
Number
T
E
S
T
O
F
Capital
M
Capital
A
C
H
I
N
E

Figure 3.7: Inspected test print

GENERAL CONCLUSION

A visually impaired individual has always depended on visually capable individuals for sourcing their braille books that are vastly different from regular books, printing braille is hard and that makes these books rare and hard to come by.

The goal of this machine is was to allow visually impaired individuals to get ahold of research papers, pdf files and sheets from the internet in a real life printed paper using a syringe that threads the dots according to the letters without the supervision of visually capable individuals.

During the building of the hardware of this machine we encountered hardware limitations when using servo motors making us change the threading head mechanism by a large delta in size and so it became heavier and harder to move around the canvas, the single axis double stepper motor mechanism was able to effectively move it around by pulling more power from the power supply.

Writing the code for the machine was challenging as we had to convert regular text into machine coordinates taking into consideration all the mirroring of the writing and given size of the paper, the proper reading side is on the bottom of the paper where the threading is pointed outwards meaning all the coordinates had to be mirrored.

After building and programming this CNC braille printing machine and testing it with all use cases including the mirror effect, we have achieved our general goal of convenience and ease of use to up to 90%.

We conclude that a personal braille printing machine is very useful for a visually impaired individual as we were able to obtain readable sheets within 5 minutes, we also conclude that printing braille is vastly different from regular text printing, it isn't easy to build a mechanism that forces a syringe on a paper to thread it without tearing it apart while still having it be readable.

To improve this machine the wooden casing that holds the paper needs to be replaced with a machinery that automatically replaces paper for it to be used in printing full books completely automatically, the skeleton has to be replaced with precise 3D printed pieces instead of wood. The syringe can be replaced with proper dot raising mechanism made for braille printing as a syringe is just a workaround we found. The noise can be improved with better rated capacitors in parallel with the power supply and motors for less jittering during operation, better stepper motors and more costly drivers should be used for longer periods of constant printing, special oil for the linear shafts can be dipped on these shafts for smoother shifting of the axies and the threading head, aswell as more robust design modifications for a more stable structure can be put in place to prevent the machine's parts from shifting around due to the high power.

Finally the threading head of this machine is heavy enough for us to have had to add iron special pinning mechanims that hold the machine onto the table and keeping it up right, better implementations for this pinning such as adding an entire platform specially for the machine can be better as it will be pinned onto the platform and allows removing the huge iron pinners that are not robust enough.

REFERENCES:

[1] https://fr.wikipedia.org/wiki/Louis_Braille

[2] https://www.royalblind.org/national-braille-week/about-braille/braille-facts

[3] https://friendsoftheblind.org/programs-we-support/braille-printing-shop

[4] https://www.amazon.com/MaxiAids-Braille-Slate-Plastic/dp/B00I5PNG4Y

[5] https://journals.openedition.org/traduire/386

[6] https://french.alibaba.com/product-detail/nema-16-stepper-motor-size-39mm-stepper-motor-catalogue-60557322173.html

[7] https://www.amazon.fr/AZDelivery-Digitales-Micro-Servo-Parent/dp/B081Z5V3M2

[8] https://www.amazon.com/Aluminum-Pulley-2Meters-Timing-Belt/dp/B01AN3ZK9E

[9] https://fr.aliexpress.com/item/32332956978.html

[10] https://www.instructables.com/id/Build-Your-Own-Drawing-Machine/

[11] https://www.instructables.com/id/SIMPLEST-Arduino-Vertical-Plotter/

[12] https://www.thesun.co.uk/news/5541046/braille-alphabet-how-read-louis-braille-writing-system/

GENERALITIES & DEFINITION

G-code: a term well-known in the 3D printing industry which is used to refer to any method used to convert digital content into machine coordinates which are then used to print something in 2D or 3D, this method is machine-specific and must be tailored for each size and purpose of the machine, g-code is generally code programmed in a programming language chosen by the programmer and isn't language-specific.

CAD: a well-know term in 3D printing and is used to refer to any program that is used to design objects in 3D, CAD itself is a 3D design program too.

Servo Motor: it is similar to a Stepper Motor but this one uses a potentiometer to spins to exact angles specified by code, it isn't as accurate as a Stepper Motor but is cheaper.

Linear Shafts: an iron rod used as an axis for the motor to keep its motion accurate, it is vastly used in 3D printers.

Timing Belt: a belt with teeth on it, it helps a motor convert its spinning motion into an axis motion, this axis relies on Linear Shaft to say in its path.

IDE: a program that allows coding of any other program within, IDE stands for Integrated Development Environment (examples: Processing 3, Pycharm, Atom.io, CodeBlocks, Dev++).

Processing 3: a programming IDE used to make games and is used a lot in machinery.

Numeric Indicator: a symbol in the braille language that is added before every string of numbers to indicate that what's coming next are numbers, it is necessary as numbers and letters in braille have the same letters.