Brailliant

Yujun Lee, Samay Sahoo, Ziyu Li

Department of Electrical and Computer Engineering, Carnegie Mellon University

*Abstract*—Brailliant aims to provide a system capable of translating English text to braille and displaying braille on a pad of actuators controlled by an Arduino. It is integrated with a web-application to transfer text input to the pad with ease and a speaker to read out the word. It is around 4 times cheaper than commercial braille displays and designed to offer a widely accessible braille translation/education tool to the visually impaired.

*Index Terms*—Stepper motor, Braille, Database, Web Application, PCB Design, Arduino, Python,

# Introduction

I

n 2015, it was revealed that globally 36 million people suffer from blindness, not to mention the 217 million that are suffering with severe visual impairment. While the cases of visual impairment has been showing a decreasing trend over the last 30 years, the challenges are more prominent than ever due to the growing and aging population [1]. Despite the growing need of assistive technology, most of them today are accessible through auditory features, failing to provide sufficient help in loud public settings, as well as to provide a complete command over written language.

The current standard of reading for the blind is through Braille, a patterned cell of 6 protruded dots representing each alphabetic character. Yet, public schools including those for blind students have few teachers who are able to read braille. Such circumstances explain the low literacy rate among the visually impaired, merely capping at a low 10% [2].

With the continuous technological breakthroughs over the recent years, there has been numerous devices in the market designed to combat the literacy issue for the blind such as portable braille displays. However, these products continue to be inaccessible to the general blind population due to their high prices averaging around $5000. Aside from the pricing, there are also battery life and portability problems as the large number of intricate solenoids and actuators in them require a large amount of energy.

To address these short comings, Brailliant proposes an affordable, highly portable, and low energy consuming braille pad with a built in text to braille translator and an audio guidance which allows the user to easily learn braille and read texts at their own speed. A key component in Brailliant is its innovative use of sliders and stepper motors to replace pre-existing solenoid/actuator architecture which reduces the pricing as well as the energy consumption of displaying braille patterns. Thus, it is Brailliant’s goal to provide a highly accessible way to learn and read braille anywhere, at anytime.

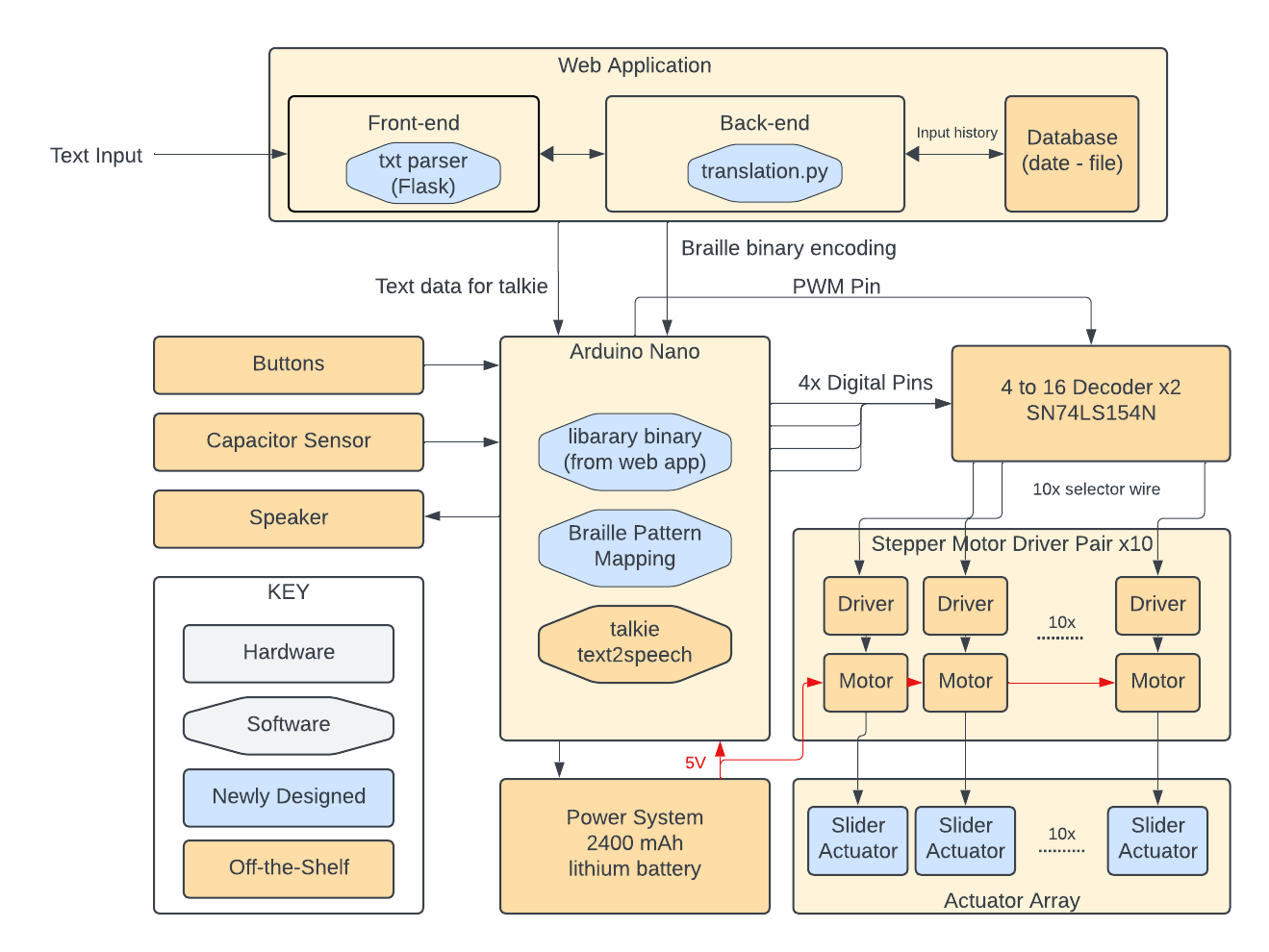
# Use-Case Requirements

Our product aims to provide a learning tool for the visually impaired that can help bridge this gap in braille literacy, at an accessible price. As such, the device must be able to dynamically refresh an array of braille pins to sequentially display words from any text input. It must be portable, easy-to-use for visually impaired users, and have battery capacity to last at least a day of learning. These use-case requirements were developed with our users–visually impaired students–especially in mind.

Regarding the physical device, we identified a form factor of 30cm x 20cm, resembling that of an e-reader, for the device to accordingly function as a portable and accessible educational device. Additionally, any physical power, volume, and I/O buttons must have braille embedded into the 3D-printed model. The device should be able to display one word at a time, and thus include 10 braille cells (where each cell is 3pins x 2pins) to fit most English words. Finally, since the product should function as a reader, we identified that an acceptable learning reading speed is >10wpm. This is based on the average beginner braille reading speed of 6 wpm [5]. We aim to be able to actuate patterns of words quickly enough to achieve this speed.

Finally, the device must have greater than 90% accuracy for our text-to-braille (T2B) algorithm, including error handling for any unrecognizable characters in the text. After we produce the pattern encodings, the device must have greater than 90% accuracy in physically actuating the correct braille pin patterns. The battery life should last a day of learning, so we aim for 6 hours running time. This ensures that the device is a viable learning tool for our users.

# Architecture and/or Principle of Operation

Figure 1 outlines our full system diagram. Our device can be divided into two major components: the hardware braille pattern actuator array, and an accompanying software for user customization of the array. The braille pattern actuator will be enclosed in a 3D printed 12'' x 8'' box. As illustrated by the CAD rendering in Figure 2a, we split up each individual 3x2 braille cell by column, and the resulting three braille dots can be actuated at the same time by a slider. The curvature changes along the slider allows it to push a different subset of the dot pins upwards at different given discrete locations. A detailed design of the slider is shown in Figure 2b, and since there are only eight unique configurations for a group of three dots, the reader can verify for themselves that this design indeed can cover all of them. The linear actuation of the slider is achieved by driving a stepper motor through a gear and rack mechanism. Further, a single braille cell can be formed by joining two of these sliding actuator face-to-face, and a alternating motor mounting position is designed to allow a linear layout of ten of these braille cells to be mounted inside the box enclosure, as shown in Figure 2a. This actuation scheme is inspired by previous project by Ulmas Zoirov [6]. Contrary to most off-the-shelf solutions, where each braille dot is actuated by a separate piezo or solenoid component, this design is much cheaper to manufacture, while retaining reasonable actuation speed as outlined in section II.

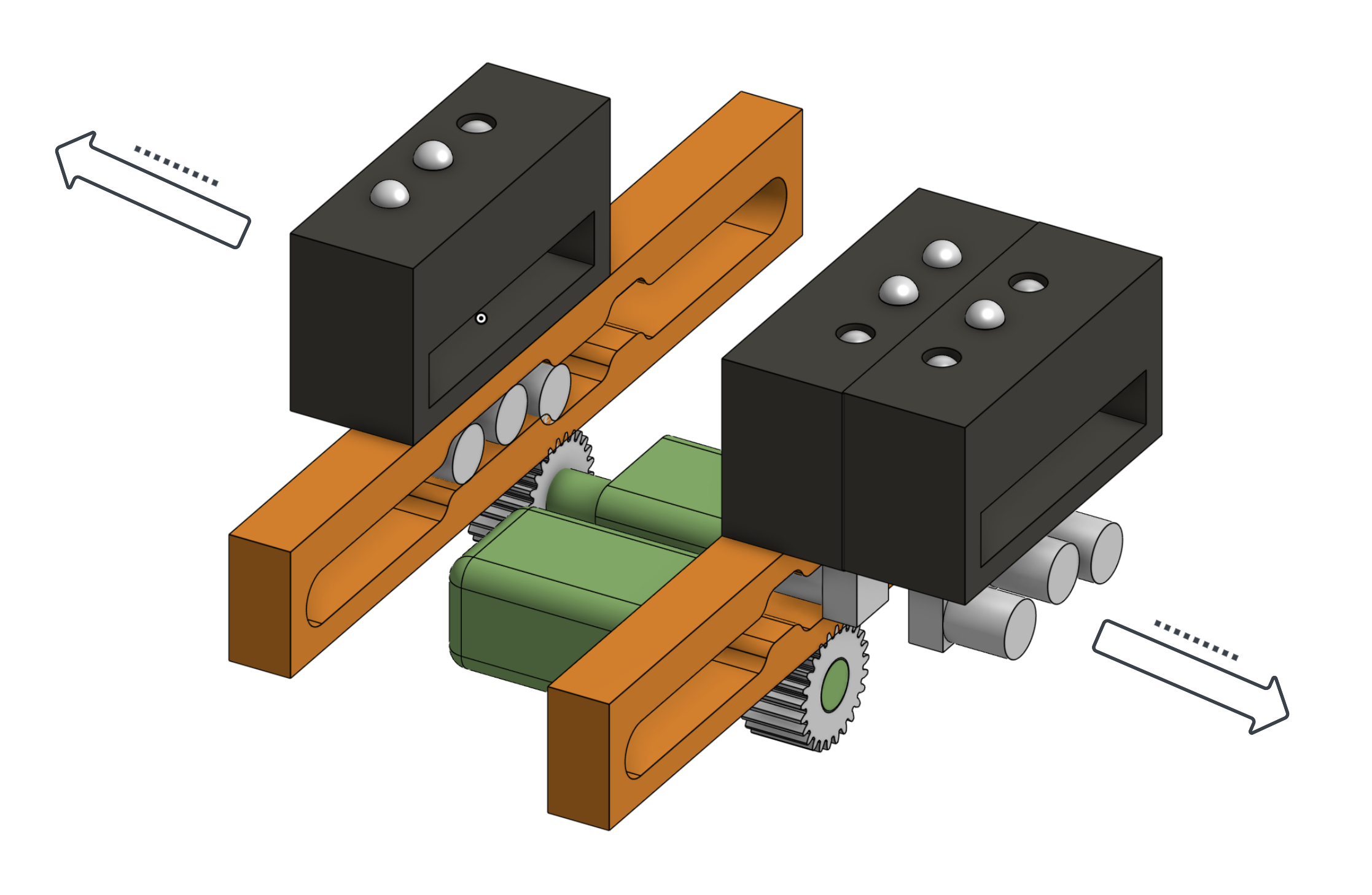
Each motor is controlled by a driver, which in turn receives its signal from a PWM pin from the Arduino Nano controller. The Arduino will only control one pair of drivers that correspond to a single braille cell at any given time. The PWM signal is sent to two 4-to-16 decoders, which reroute the signal to one of ten driver pairs. Four digital pins of the Arduino also connect to the decoder as selector wires.

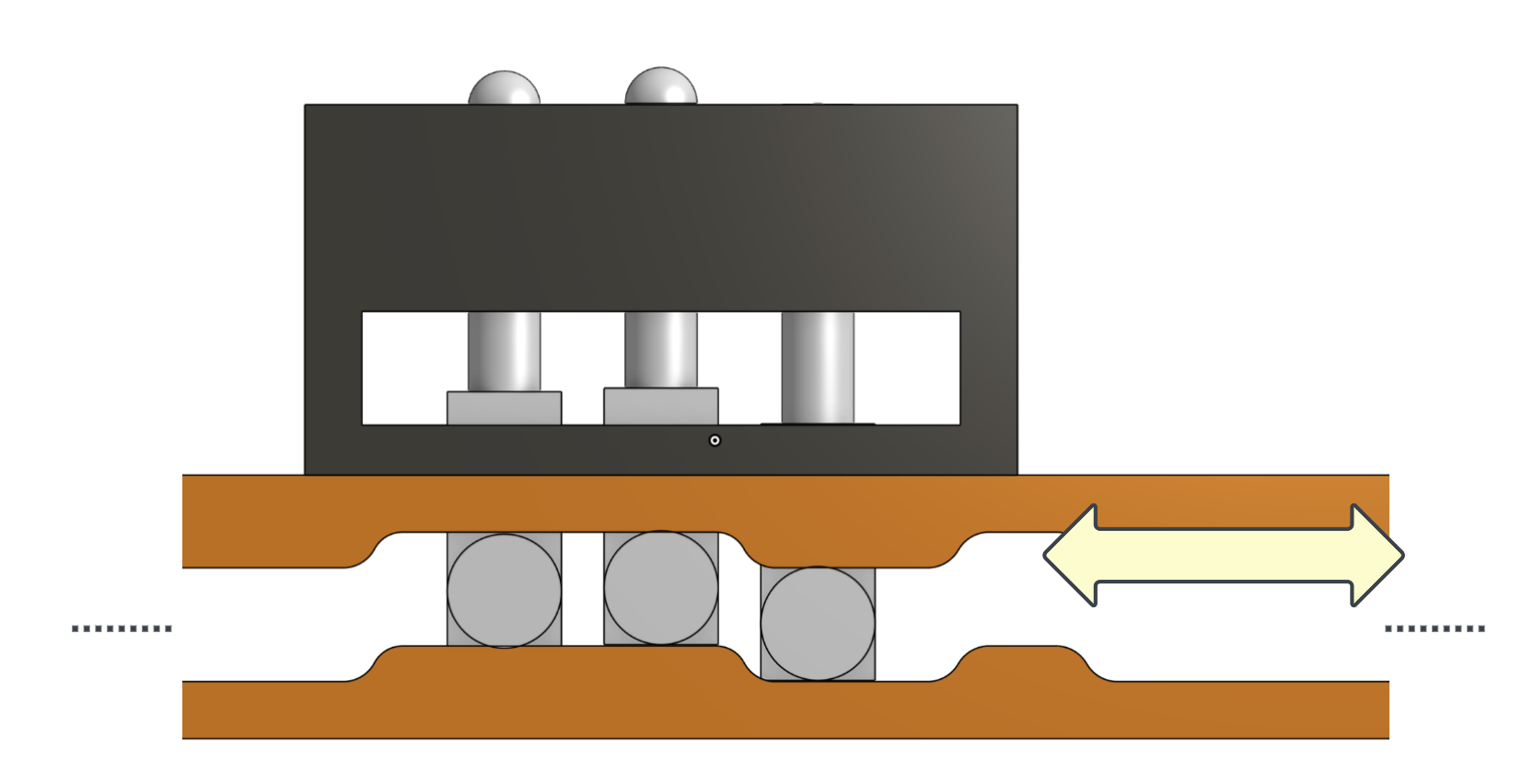
On the software side, our translation process and text input will be handled by the web app. The user can either input the texts manually through the application or input/drop a .txt file. The web app will contain a hash map of character key to braille pattern value where each braille pattern is a 7-bit binary value encoding with each 3 bits representing the 8 possible patterns a

column (2 total) of braille cell could display. The one remaining bit would be used as a stop bit to mark the end of a word. Once this is done the original text array and the encoding data would be sent to the Arduino to save via a serial port connecting the web accessing device and the braille pad.

The web app will also be connected to a relational database containing the past text files that have been inputted to the device. This is to allow easier accessibility to the users in case they want to edit previously studied words or add new words to a pre-existing input. This history data would be displayed through the web-app for the users to easily retrieve. The table of data would map each text data to the added date and unique name given to the input.

The braille pad will handle the incoming data through its Arduino where it will step through the chunk of encoding(7 bits at a time) to map each binary data to a stepper motor activation until the stop bit is read. Then, the Ardunio would input the corresponding text to the audio component of the device, reading the word in speech. Once the next button is pressed, the same process would repeat with the stored text and braille encoding data saved on the Arduino.

  
(a)



(b)

Fig 2. Braille pin actuation mechanic. (a) Top view of design. (b) Side-view of design.

# Design Requirements

In order to satisfy the above use-case requirements and considering our working implementation plan, we have identified the following technical design requirements.

To achieve the identified use-case requirement for battery life, we need to consider the power consumption of our entire system. The micro stepper motors we have selected consume 0.6W and one motor per column per braille cell. As a result, the 20 motors running for 6 hours will at most require the following:

Thus, the above equation shows us that our device must have a rechargeable battery capacity of at least 14400 mAh in order to operate the required number of motors for the required length of time.

Next, our device’s usability and accessibility as a learning device depends greatly on the physical layout. Specifically, each braille pin must adhere to standard braille sizing requirements for physical braille. According to the Braille Authority of North America, the distance between two braille dots within the same cell may be between 2-3mm [4]. Further, the distance between corresponding cells can be between 6-8mm. The layout of our 3D-printed braille cells and sliders must adhere to these spacing requirements. Finally, each braille pin should protrude 1mm up from our device’s “screen” in order to be properly recognized by our users’ hands.

The accuracy requirements beget functionality to reset and correct any mechanical jams we are bound to experience given the small scale and mechanical stress. As such, the stepper motors may lose tracking, especially considering that stepper motors do not have a feedback system to control and remember position. The device must recognize and re-try pattern actuation in these situations.

The beginner reading speed of 12wpm requires our design to be able to actuate the correct pins for any given word quickly enough to be able to display words on the device to users at this speed.

Thus, the derivation above shows us the timing requirements for our motor-slider actuation subsystem is 5s per word; in other words, the design requirements is 0.5s per cell. This signifies that each motor should spin to the correct position within this amount of time.

# Design Trade Studies

## Software Platform

The choice of Arduino as the hardware/software electronic platform was made in consideration of the low power usage, and low price range for the overall product. Table 1 details the difference between the two products. In terms of its alternatives, a valid alternative would be a Raspberry Pi. An advantage of using an RPi would be it’s higher processing power (1.6GHz vs. Arduino’s 16MHz). However, the processing speed only proves to be beneficial in the case of complex software computations which would not be necessary as Brailliant’s translation process would be done on the web app prior to inputting the text data into the hardware device. In other words, our project deals with sending the predetermined instructions to the motors and hardware components to the device through the application and thus would not require the processing power and flexibility that RPi provides. Furthermore, considering the portability of the product, RPi’s power usage stands around 1.7W which is comparable to Arduino’s 0.14W which much suitable to our battery driven product.

Table I. Comparison of Board Specs

| Table Head | Type of Board | |
| --- | --- | --- |
| Arduino Uno | Raspberry Pi |
| CPU Type | 8-bit Microcontroller | 64-bit Microcontroller |
| Operating System | None | Linux |
| Storage | 32KB Flash | Depends on SD Card |
| Memory | 2KB | 1GB RAM |
| GPU | None | Built-in |
| Networking | None | Wifi, Bluetooth, Ethernet |
| Price | $20-30 | Up to $80 |
| Power Consumption | Less than 0.25W | Up to 1.8 W |

## User Interface

The trade-offs that we considered when choosing the user interface included the choice between a web-app, mobile application, and a design where all the data management and translation are all incorporated into the braille display device. Ultimately we decided a mobile application would not be suitable due to the higher accessibility of web-based applications as well as the fact that more text files are accessible through the web and computers rather than mobile text editors. Additionally, we decided to separate to user interface into the physical implementation of the device and the web application. To step through the words in the text file, we decided to use a button on the actual device considering the fact that the device is to be portable and to be used in outside environment. For inputting the list of words, we decided to use the web app since the texts are likely to be generated by a third party, and as a result we needed to incorporate the input process using the most widely accessible method of interface which is the web. Furthermore, by using a web application we could handle the translation process and dictionary data within the web prior to transferring the data to the device which could alleviate the memory usage and processing done in the Arduino. This is more suitable for the project since it would be unreasonable for the Arduino’s limited 2KB RAM and 32KB flash to hold the translation algorithm as well as the dictionary data which is to be accessed continuously throughout the translation process.

## Translation Algorithm

Pre-existing translation algorithms include a basic hash map dictionary based algorithm and a machine learning classification algorithm. While contemporary ML is extremely powerful, we decided that it would not be necessary as there is no variability in the translation for braille and no grammatical changes depending on the context of the word. Thus, we decided to use hash maps organized in key-value pairs in C++. Most braille translation algorithm in the market uses one map that maps each alphabetic pattern to a braille arrangement. However, in our project we decided to use two maps (one for whole words, and one for prefixes, suffixes, and braille contractible alphabet patterns such as “aa”, “ment”, or “pre”. The difference in the number of dictionary maps used creates a large difference in the time complexity of the algorithm. With each access of the dictionary being O(1), the usage of one hash map would lead to a complexity of O() as for each of the examined characters, the rest of the characters need to be checked for a match in the dictionary. On the other hand, the two map dictionary model we are using take advantage of the fact that all contractible character sequences are capped at a length of 4, and the rest are whole word abbreviations. This leads to a time complexity of O(4n) since each character needs to be checked with its 4 following characters at most to determine their contractibility.

## Actuation Method

The choice to utilize micro stepper motors to actuate braille pins was developed based on our requirements for size, speed, and cost. We first considered more straightforward actuation with solenoids, planning to lay out an array of solenoids with one solenoid per braille pin. However, this led to significant design challenges as the available solenoid components are not nearly small enough to fit our sizing design requirements. Furthermore, we found out that solenoids require power to remain in the actuated position. This led to significant power consumption concerns since our requirements are to actuate each pin and remain in this state until the user has finished reading the entire word. We also looked into piezoelectric actuators, however this led to cost challenges as we required a great number of actuators and the cost per part exceeded our budget.

We finally reached a solution with an entirely distinct approach to actuating pins—actuate patterns of pins using a 3D printed sliding component and stepper motors. Each slider with engravings will actuate all 8 different patterns that each column of 3 pins can achieve. This design has significant advantages over solenoids or piezoelectric actuators due to the low power consumption, smaller component size, and reduced number of components. We were able to find small stepper motors that operated at 0.5W-1.2W. Further, this reduced the number of “actuators” by a factor of three—2 motors per cell compared to the previous 6 actuators per cell.

# System Implementation

We will start the section by discussing fabrication method. This section is then split up by hardware and software implementation. The hardware portion is further split into actuator mechanics (motor, driver, and decoders), and miscellaneous (power system, speaker, capacitor sensor, etc.). At the very end an integration plan is also given.

## Fabrication

We plan to use off-the-shelf products for all electronic components. Critically, we will fabricate the device enclosing and all individual slider actuators ourselves using resin 3D printers. This choice is primarily motivated by our design goal of making this device as easy to manufacture by DIYers in a garage as possible, and other fabrication equipment, like laser cutters and CNC routers, are not as readily available to the general public as 3D printers. The printer of our choice is Formlabs Form 3+, since this printer is available on Carnegie Mellon University Campus, and can achieve a precision of 0.1mm.

## Web-application

The web application will be built using Flask, and the front-end will be written in CSS and HTML. Figure 3 displays the flow of the web app. The web application will be linked to a SQL database to store the history of user input with the date as the key and the value as the .txt file. Once the user clicks done, the translation.py function in the backend would translate each word to a Grade-2 standard braille, mapping each sequence of characters to a pattern from left to right.

The braille dictionary would be incorporated into two different key-value hash maps: one for whole word contractions and one for 4 letter max character sequence contractions which are the two types of word trimming used in grade 2 standard braille. The translation process starts from left to right with each letter and its 3 following letters being checked with the dictionary hash map to output a matching braille encoding. After the translation, the data would be transferred to the Arduino and the user inputted text data would be stored to the SQL database for reuse in the future.

## Device Software Implementation

The programming for the braille display would be done in the Arduino in C++. With the input from the web app, the Arduino would map each braille encoding to a stepper motor through a 4 to 16 decoder. In terms of the audio output, we will use Arduino’s built in Talkie library which contain functions that are able to take in a string argument and output a speaker readable output which is transferred directly to the mounted speaker in our device.

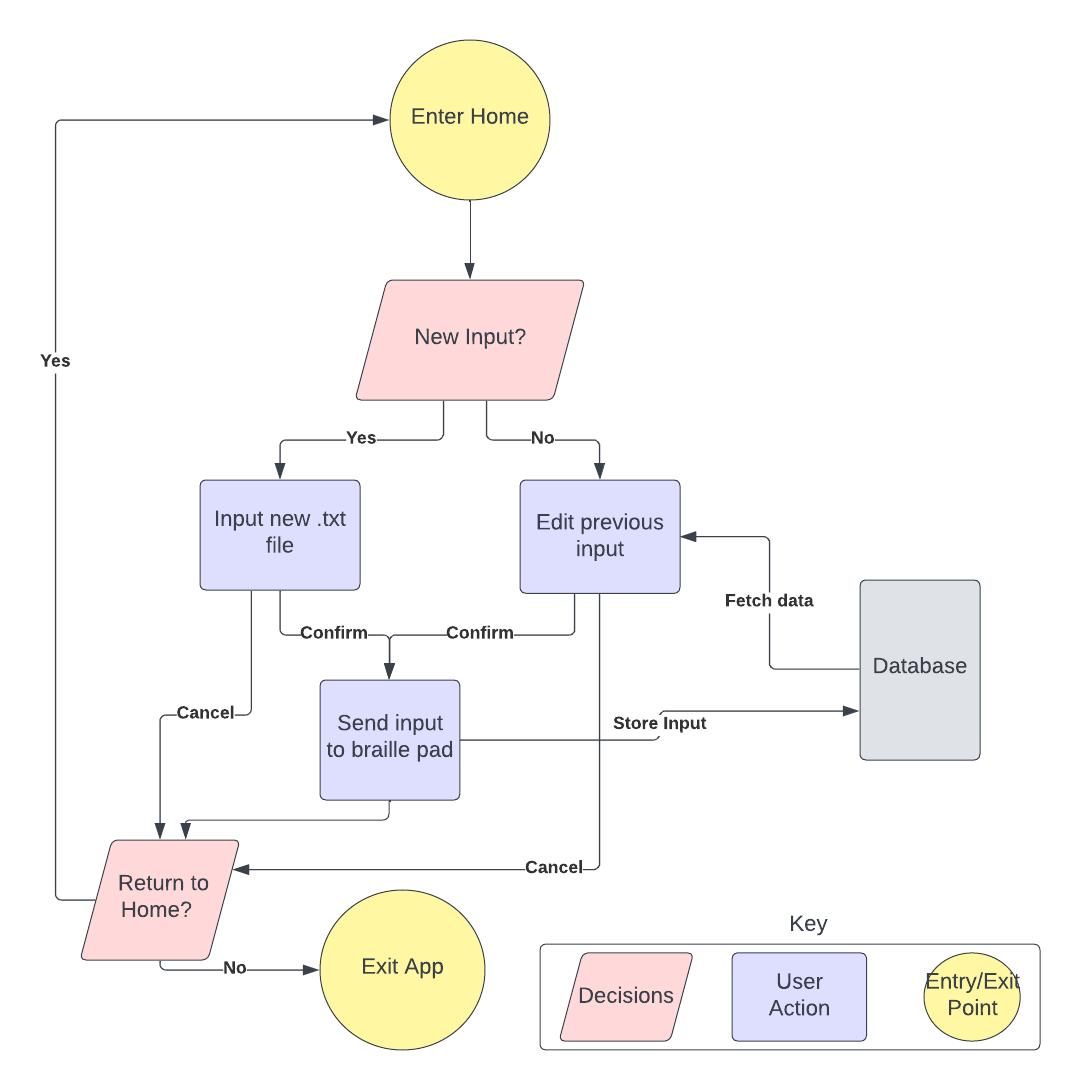


Fig 3. Web-app user-flow diagram

# Test, Verification and Validation

## Tests for Web Application

## We will be testing the web-application mainly through user-studies and Apple’s Accessibility Inspector Tool. We will collect more than 10 subjects to perform user-testing on the difficulty of the text-inputting process, data transfer between the web-app and the braille display, retrieval of input history, and the basic end-to-end performance of the site. The user-testing will be further augmented with Apple’ Accessibility Inspector Tool for debugging, allowing us to test the app’s performance against built in accessibility feature tests. We will test the latency of the web-application by timing how long it takes the site to interpret the user-input as well as to retrieve previous inputted text data from the database. As aforementioned in sections II and III, accessibility is a crucial requirement for this device, so we will be extensively taking feedbacks from our user-testing to ensure our ease of navigation through the site.

## Tests for Translation Algorithm

As mentioned in section IV, the correctness of our

translation is crucial to our project. In order to allow the previously stated 90% accuracy in text to braille translation we will be testing the translation algorithm implemented in our web-app with numerous sequences of words. In order to do this, we will collect more than 100 words of varying length and complexity to achieve at least 90% accuracy. As speed is also an important use-case requirement in our project we would be using Python’s built in timer to time the translation time of 100 words to reach a speed that is appropriate for educational use.

## Tests for Hardware Pin Actuation

Other than the algorithm itself, it is also significant for the array of stepper motors to be accurate by itself. On more than 100 different braille patterns ranging from length of 1 character to 10 characters, we will be testing accuracy of braille actuation visually to achieve a 90% accuracy which we derived on section IV. As readability of the pins is not completely a quantitative qualification, we will be testing the readability through user-testing. By contacting CMU’s Office of Disability of Resources and Pittsburgh’s School for the Blind Children, we will be testing our braille pins on more than 3 visually impaired subjects and analyze feedbacks on the usability of our product as an accurate educational device.

## Tests for Pin Refresh Speed

As stated in our use-case requirement, it is important for the

Braille cells in our device to display each braille cell at the average braille learner reading speed which is at 10wpm (1 word per 6 seconds). Thus, we will be testing the actuation speed for each word through a slow motion capture video, ensuring that the actuation of a word takes less than 6 seconds. Furthermore, the clearing of the braille cells should take less than 2 seconds to ensure that the user could move on to the next word as soon as possible.

## Tests for Battery Life

The reason why most braille devices are struggling with

implementation is due to the large amount of power consumption. In section II and IV we derived a necessary battery life of 6 hours for the use of our product during a normal school day. To test the battery consumption of our device we will be testing our device throughout the day, occasionally pressing the next word button to initiate the braille actuating process. Once the battery reaches its limit we will record the time passed and repeat the entire process 3 times to ensure our intended battery life.

## Test for Audio Accuracy

The audio output of our product must be

understandable to the general student and cannot be highly contaminated with noise and be inaudible. Therefore, we will be testing our audio over 100 different words to ensure a 80% accurate audio output for each text. To continue, the audio audibility will be part of our user-testing on blind students, to confirm the accessibility of our audio feature on our potential users.

## Test for Integration

To test the integration of our hardware and software

components, we will first test our Arduino output to make sure it is properly transmitting the correct binary voltage values to send to the stepper motors, and the position of the sliders. For integration between front-end and back-end, we will consistently ensure the data being sent is accurate as we add more components to our software. For backend integration testing, we will make sure the data retrieved by the Arduino corresponds to the data outputted by the translate.py file within the backend of the web-app such that there is no data loss in the data transferring process. Once we had made sure all the components are able to transfer data without loss, we will integrate the software and hardware together, and connect our device components to test the aforementioned test cases for braille readability, actuation speed and audio accuracy.

# Project Management

## Schedule

This subsection provides the schedule with milestones. You can use a full page at the end of your document to insert a detailed schedule, and refer to it in a short paragraph. This space limitation is only for the text portion of your write-up.

If your schedule was simple, you can insert it inline either as a wide figure like Fig. 3 or rotate it and fit it within a column as a typical two column figure as shown in Fig. 5. Please make sure your chart is readable (font size is at least 8 point, and the color choices are such that we can read the text. If you include your milestones in-line here, you can use **more than the half column** you have available for the schedule.

## Team Member Responsibilities

The project was divided into two primary components: Software and Hardware. Due to the mechanical challenges presented in the hardware implementations of the project, two members were allocated for hardware and one was assigned to deal with the software.

Ziyu is responsible for the CAD design of the project including the 3D printed sliders and their connection to the stepper motors. The pins and the casing design are also handled by Ziyu. He is also assisting the PCB schematic design handled by Samay to allow a smooth integration between the mechanical components and the electrical components.

Samay is primarily responsible for the schematic for the PCB. He is to deal with the power management, microcontrollers, as well as the speaker component of the product. It is his primary goal to seamlessly connect the hardware components all together and transfer the necessary signals.

Yujun’s is responsible for the software implementation. Aside from coding the Arduino to control data transfer between the processor and the hardware components, he is also dealing with the text-to-braille translation algorithm and create the braille dictionary hash map to encode braille patterns into encodings readable by the motors.

The testing for hardware will be done by Ziyu and Samay, while software testing is handled by Yujun. As 3D printing requires numerous trial and errors, everyone will be participating to print out the design and testing the printing parameters.

1. Schedule example with milestones and team responsibilities

## Bill of Materials and Budget

You can use an entire page for the table of parts and budget at the end of the report (see Table II). This is the easiest way to cut and paste any Excel based table you have been maintaining into this report. Include the description, model number, manufacturer, quantity and cost of each component purchased. (“Amazon” and “Digikey” are probably not the manufacturers of the parts you purchased.)

**This section may end up being one or two sentences referring to the page where the budget spreadsheet is located in your report.**

## Risk Mitigation Plans

We will continue to explore the design space of our slider braille actuator, experimenting with different configurations of material, lubricant, form factor, slope of slider, etc., and rapidly prototyping them to achieve the most reliable and fast-responding actuation this device can reach. Through the idling of the motors, we also plan to implement a software analyzer to automatically recognize the case where one or more sliders got jammed, and perform a preprogrammed back-and-fourth maneuver to attempt automatic unjam. In the worst case where our system cannot recover from a jammed slider on its own, we plan to make the front cover of our device easily removable and re-attachable, so that human intervention, e.g. pull down on the pin and slider, can be employed to help the device unjam and reset.

If the durability of the slider does not meet our design requirement, we will manufacture it from more rigid materials like use CNC machining on stainless steel. If manufacturing parts in such a small physical scale poses significant challenge, either for us or for future prospecting DIYers, we will scale up the device. We know this is doable since real life braille also varies in size.

This sub-section should identify the critical risk factors in your design and plans for how you will manage that risk. This is where you will list the known unknowns. For example, no one in the team has ever previously designed an UART for this FPGA, so our primary risk is getting sensor data to the FPGA to take advantage of its rapid computing capability.

# Related Work

What other projects or products are similar to what you are proposing to implement. You can use up to a full column in the two column format for this.

**Use no more than one page for both Related Work and Summary.**

# Summary

Briefly summarize your design, with emphasis on the impact of your design to stakeholders who care about the use case. Describe what you think will be upcoming challenges in implementation and in meeting the use-case and design requirements.

Glossary of Acronyms

CAD – Computed Aided Design

DIY – Do It Yourself

MQTT – Message Queuing Telemetry Transport

OBD – On-Board Diagnostics

PCB – Printed Circuit Board

PWM – Pulse Width Modulation

RPi – Raspberry Pi

WPM – Words Per Minute

References

1. Ackland, Peter, et al. “World Blindness and Visual Impairment: Despite Many Successes, the Problem Is Growing.” *Community Eye Health*, U.S. National Library of Medicine, 2017, www.ncbi.nlm.nih.gov/pmc/articles/PMC5820628/
2. Amato, Sheila. “Standards for Competence in Braille Literacy Skills in Teacher Preparation Programs.” Journal of Visual Impairment & Blindness 96, no. 3 (2002): 143-153.
3. “Human Interface Guidelines.” Apple Developer Documentation, developer.apple.com/design/human-interface-guidelines/. Accessed 12 Oct. 2023.
4. National Library Service for the Blind and Physically Handicapped, Library of Congress. Specification 800: Braille Books and Pamphlets. ([www.loc.gov/nls/specs/800\_march5\_2008.pdf](http://www.loc.gov/nls/specs/800_march5_2008.pdf))
5. Perkins School for the Blind, *Perkins School for the Blind: Reading Rates*, Accessed on Feb 24, 2024, [Online]. Available: <https://www.perkins.org/resource/reading-rates/>
6. Zoirov, Ulmas. Development of the low-cost solution for Braille display based on linear actuators with 3D printed mechanisms and servomotors. Rel. Marcello Chiaberge. Politecnico di Torino, Corso di laurea magistrale in Mechatronic Engineering, 2021

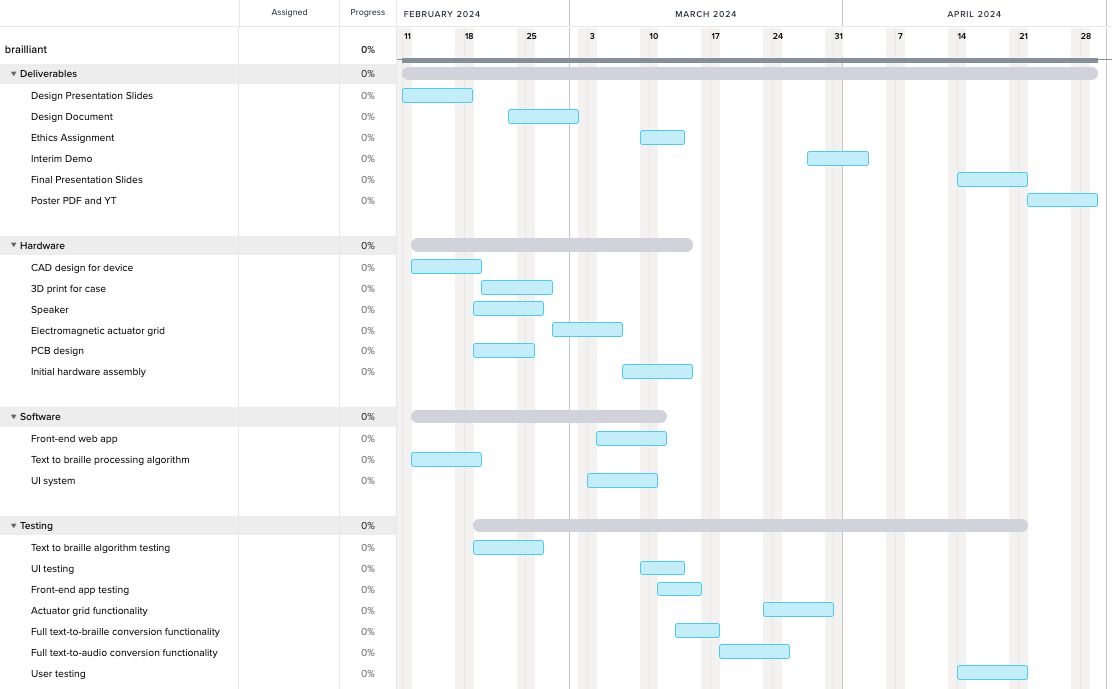


Figure 3. Schedule