Product: BraillEd

Team: UpliftEd

Abstract

BraillEd is a tool to help blind and partially sighted children in mainstream primary schools to learn braille[1], by allowing them to enter text via a keyboard or voice recognition, and read the corresponding braille on a refreshable braille display. There will also be a text-to-speech function to allow them to check they typed the correct text before they have learned braille and a camera which will allow the device to translate printed material into braille.

Since the last demo, we have implemented a keyboard-based interface for the user to interact with the software, as well as text-to-speech using the Python library pyttsx3[2]. We have produced a display of 10 pairs of octagonal wheels, each of which can be rotated independently by a stepper motor fixed onto a moving carriage. We have created a control system allowing the hardware to keep track of the location of the carriage and rotation of the wheels, and turn the motors the correct amount to line the carriage up with the wheels and spin them to the correct rotation. This causes the wheels to line up, displaying the desired braille dots. We have also improved the text-to-braille translation by using the library Liblouis[3], meaning our system can now switch between contracted and uncontracted braille [4].

1. Project management update

1.1. Achievement of goals

- Implement text-to-speech [achieved].
- Update text-to-braille to be able to handle both uncontracted and contracted braille [achieved].
- Create software to keep track of braille cell positions and control the motors accordingly [achieved].
- Integrate software components so they are all user controllable using the keyboard [achieved].
- Produce 10 braille cells which can be activated to the desired position [partly achieved].
- Integrate the software with the hardware so the braille produced by the software can be displayed on the hardware [achieved].

Unfortunately, the gear on the motor sometimes gets stuck while moving to line up with the gears of the braille wheels. This means that the device sometimes fails to spin the wheels to the correct location, and occasionally needs to be manually reset. Because of this, we consider the building of the braille cells to only be partially complete, as we need more time to work on preventing this issue, which we unfortunately did not have before the demo.

1.2. Coordination of progress

We coordinated our work towards these goals primarily through the use of GitHub issues and GitHub projects, as together these allow us to specify start and finish dates for each task, as well as assigning them to individuals so we know who is responsible for getting a particular task done.

To ensure quality of code we decided upon a policy of implementing changes in branches and submitting pull requests for other members of the team to review and approve.[5]

Whenever integration between the hardware and software was required, work on this was primarily done by the hardware and software leaders, while other members of the teams carried on with developing other parts of the system. This was made possible through the use of different branches on GitHub, enabling changes to subcomponents to be made without risk of breaking ongoing integration work.

1.3. Allocation of time

The following is a summary of the areas of the project worked on by each team member, and the estimated total time each person has spent so far on the project.

Nikodem Raspberry Pi setup, speech-to-text, text-to-braille, software component integration, software-hardware integration (121 hours total).

Daniel braille display prototyping, testing with motors, AutoCAD 3D modelling of gears and octagons, braille display, modelling and configuration of motor mounts, 3D modelling of casing (110 hours total).

Nikita Raspberry Pi setup, speech-to-text, text-to-braille, research and experimentation for image-to-text, demo work (88 hours total).

Ripley rail construction, motor mounting, moving rail interaction (118 hours total).

Ol project management, ethics application for user testing, user needs research, marketing research, keyboard interface, software component integration and debugging (92 hours total).

[1] We write braille with a lowercase ‘b’, to follow the practise of the Royal National Society of Blind People.

[2] https://pypi.org/project/pyttsx3/

[3] https://liblouis.io/

[4] Uncontracted braille has a one-to-one mapping from letters to braille characters, and generally learned first. Contracted braille adds contractions for commonly used words and letter sequences (Royal National Institute of Blind People).

[5] Text duplicated from our previous report is printed in blue.

Balint stepper motor drive, motor calibration and integration (95 hours total).

Florian market research, text-to-speech (42 hours total).

Poppy text-to-braille, software-hardware integration (71 hours total).

Souparna braille display engineering, Solidworks CAD designing of the new rail system, formulation and plan of overall hardware, error correction design for pips, Rasberry Pi setup, stepper motor circuit design and integration, pip motor circuit design and integration, hardware-software integration, manufacturing and assembling hardware components (130 hours total).

1.4. Budget and technician time

Table 1 provides a breakdown of how we have spent our allocated budget and technician time so far.

1.5. Outlook

Our goals for demo 2 include an optional stretch goal of adding a camera which uses computer vision to convert text on a physical page into braille on the display. Following research and experimentation, we have decided to proceed with this goal.

We planned on carrying out user testing of our product with people who read braille by the final demo. We are unfortunately still waiting for ethics approval to do so, so although we would still like to do this, we may not be able to. Therefore, we plan on carrying out more user research online to substitiute for this in case approval is not granted in time.

2. Quantitative analysis and testing

2.1. Software testing

We undertook testing to ascertain the accuracy of speech processing on the new Raspberry Pi 5 with VOSK[6] a toolkit which we have previously used for its lightweightness that allowed us to test it without much worry for the Pi's limited storage capacity. We decided to temporarily stick with the lightest model available that we have used previously for a more seamless transition to the Raspberry Pi 5. This evaluation serves a dual purpose, for testing as well as comparing the performance against the previous Pi 3 and future models.

Testing was done by each of the software subteam speaking 10 sentences, 10 times. We used VOSK's built in confidence label as a valuable bit of context for the software's outcome, that allowed us to interpret and analyse patterns that came up within the used phrases, although the exact reasoning for the disparity between the exact classifications and the average confidence rate is something that can be rather difficult to pin point.

[6]https://alphacephei.com/vosk/

The average confidence rate and exact classification rate for each phrase are depicted in table 2 Notably, VOSK struggled with the phrase 'maths is hard': exhibiting lower confidence levels in transcribing them. From direct observation of the data of the phrase, VOSK did not hear the word 'maths' specifically due to its minimal syllables count and lesser stress in sounds. The issue is potentially originating from the user being unaware of when the microphone is recording and when it is not.

Due to these findings and upgrade to the Raspberry Pi 5, we have opted to explore alternative options for Demo 3, by pairing the system with a more robust and multilingual model. We aim to enhance the comprehensiveness of our comparisons by replicating our tests. Additionally, we plan to extend our analysis to include a variety of spoken languages, surpassing the current scope for a more extensive evaluation. Additionally, we will be measuring the duration of processing in our evaluation.

Interpreting the results, the level of accuracy for certain phrases is concerningly low. It may be necessary to explore ways of improving the accuracy, such as better microphones and alternative speech recognition models. That being said, 100% accuracy is not necessary; problems with inaccuracy can be mitigated by recommending users to use the text-to-speech function to verify the text has been recognised correctly before reading the braille.

It has to be said that the testing isn't very exhaustive however. There are bound to be several more phrases that the speech recognition system would have a hard time with, but it would be unfeasible to test every single word combination to ensure the best results. The results we gained are good enough for our purposes, however, since they provide sufficient data to be aware of these changes we have to make. More thorough testing will be undertaken once changes are made.

2.2. Hardware testing

Testing of the current hardware is split into testing the accuracy of setting a single wheel of a braille cell (half-cell), setting both wheels of a cell (full-cell) and setting both wheels of 2 braille cells (multi-cell). Combined, these tests encompass the precision of the display system, which is crucial for usability. Each test is done over 20 repetitions and times are recorded using a stopwatch measured from the initial motion of the wheel. The half-cell test isolates the cell-actuating stepper motor from the full carriage system. For the test to be considered a success, the motor should turn the braille wheel by a single position so that the next character on the octagon is moved into the place of the previous one facing up. In the case of the full-cell test, success is only granted if the left wheel is correctly actuated, the carriage correctly moves across to the right wheel and then the right wheel is correctly actuated. Finally, a successful multi-cell test will do what the full-cell test demands for 2 side-by-side cells.

Currently, implementation solutions are inherently prone to

Item	Cost	Technician time
1/2 sheet of 5.5mm plywood	£10.27	–
Laser cutting of box	–	0:30
3D printing of various components	£5	1:30
TOTAL	£15.27	2:00

Table 1. The usage of our allocated budget and technician time.

Expected Phrase	Exact Classifications	Exact Classification Rate	Average Confidence Rate
the colour is red	21/40	0.525	0.94
we are going on a field trip	10/40	0.25	0.77
the quick brown fox	26/40	0.65	0.91
jumps over the lazy dog	19/40	0.475	0.83
maths is hard	1/40	0.025	0.61
the cat is black	13/40	0.325	0.82
humpty dumpty sat on a wall	1/40	0.025	0.91
This is unfair to your peers	16/40	0.4	0.88
no eating or drinking in the lab	20/40	0.5	0.92
draw me a beautiful snowman	13/40	0.325	0.67

Table 2. Classification Rate and Average Confidence Rate

Test	Average time (s)	Success rate
Half-cell	0.2	8/20
Full-cell	0.3	8/20
Multi-cell	1.2	2/20

Table 3. Display system precision

cumulative errors since there are no locking mechanisms which would hold the braille wheels in exact positions and no calibration mechanisms for the rail and carriage systems. As such, the tests inherently have a progressively increasing fail rate as the errors creep up.

This highly error-prone system is not acceptable for the final product since braille needs to be repeatedly displayed correctly and should be resilient against prolonged daily usage.

This testing is not completely exhaustive because it does not consider a long period of use or the effects of touching and moving the device. However, it provides us with the necessary data we need to know we need to work on improving the accuracy of the device. After this is resolved, we will undertake more comprehensive testing, the results of which will be included in our final demo report.

3. Budget

3.1. Final prototype budget

We estimate that our final prototype will cost about £256 to manufacture. The breakdown of this cost is shown in table 4.

3.2. Commercial product budget

We estimate that even producing a commercially viable product, we would be able to keep the cost around the

Item	Cost
Plywood for casing	£11
3D printed components	£20
Raspberry Pi 5	£60
Aluminium struts for rail	£10
2 large stepper motors	£50
1 small stepper motor	£2
Keyboard	£20
Camera	£60
PCB manufacturing	£3
Misc. fixings	£10
Misc. wires and electronics	£10
TOTAL	£256

Table 4. The estimated cost of items for our final prototype.

£250 mark. A breakdown of this estimate is provided in table 5. This is well below the cost of existing braille learning products on the market, such as the Taptilo, which costs over £1000 (Sight and Sound Technology).

4. Miscellaneous

4.1. Text-to-braille implementation

In order to implement the text to braille element of the system, we initially planned to use the Liblouis library. However, after running into issues with implementing the library on our system, we came to the conclusion that the best path forward was to take the tables used by the system and write our own software to compile them. Although the full Liblouis table language is not relevant to our particular use case, being able to use these crowd-sourced tables even in a reduced capacity massively increases both the accuracy of our translation and the number of languages we are able to support. Algorithm 1 below describes the method by

Item	Cost
Casing	£20
Cogs	£15
Rail	£10
Raspberry Pi style microcomputer	£60
Motors	£50
Keyboard	£20
Camera	£60
PCB manufacturing	£3
Misc. fixings	£10
Misc. wires and electronics	£10
TOTAL	£258

Table 5. The estimated cost of items for our hypothetical commercial product.

which a Liblouis table is parsed.

Algorithm 1 Parse Liblouis Table

```
Input: string $tablePath$
Initialise stack $tables$ and push $tablePath$ onto it
repeat
    POP an item from $tables$ and open the table at the indicated filepath as $file$
    for each line in $file$ do
        if the first character of the line is # (indicating a comment) or the line is blank then
            Ignore the line
        else if the first word in the line is "include" then
            Concatenate the string ”$tables/$” and the second word in the line
            PUSH the result onto $tables$
        else
            Translate any liblouis escape sequences in the line to the implementation equivalents
            Split the line into a list of tokens on whitespace characters
            Lookup the first token in the list of opcodes
            Create a new rule using the opcode and its expected number of operands from the subsequent tokens
            Store the rule for use in translation
        end if
    end for
until $tables$ is empty
```

In addition, the extensive online documentation of the format of a Liblouis table helps to ensure the system will be able to remain current with any future Braille standard. This is key in ensuring our system will stand the test of time. Although only a majority of the specified opcodes have been implemented so far, work continues on ensuring the system is able to accept any Liblouis table and fully utilise the translation rules encoded therein. This work also opens up the possibility of as-yet unconsidered extension features, such as the use of a braille engraver or 8-dot braille cells for computer braille or complex mathematical braille. This focus on extensibility and modularity is a key focus not only of the braille translation but of the system as a whole.

A. System diagram

Figure 1 provides a diagram detailing the operation of the overall system.

References

Royal National Institute of Blind People. Contracted (Grade 2) braille explained. https://www.rnib.org.uk/living-with-sight-loss/education-and-learning/braille-tactile-codes/contracted-grade-2-braille-explained/. Accessed: 2024-02-29.

Sight and Sound Technology. Taptilo. https://www.sightandsound.co.uk/product/taptilo/. Accessed: 2024-02-29.

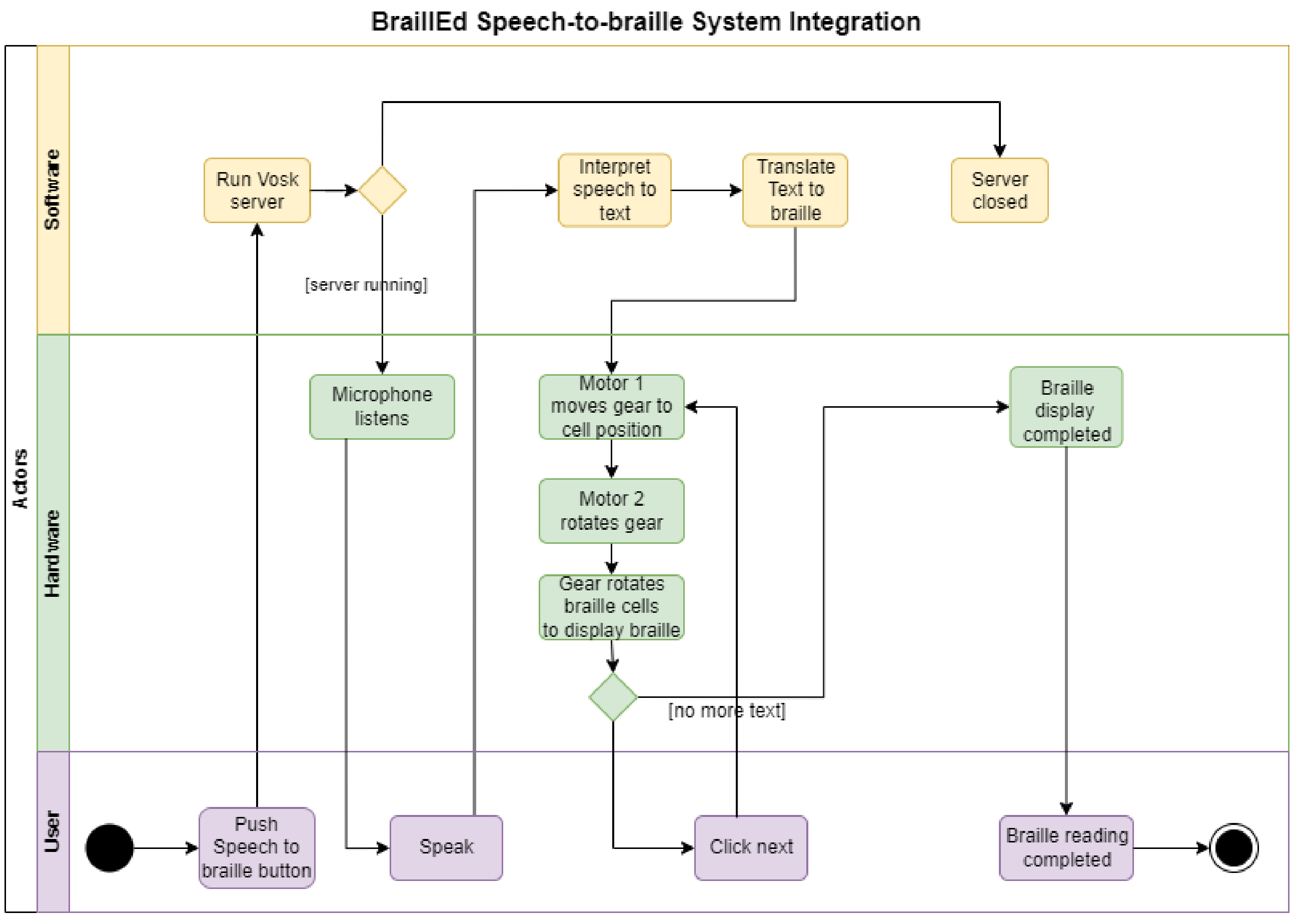
BraillEd Speech-to-braille System Integration
Actors
Software
Hardware
User
Run Vosk server
[server running]
Interpret speech to text
Translate Text to braille
Server closed
Microphone listens
Motor 1 moves gear to cell position
Motor 2 rotates gear
Gear rotates braille cells to display braille
[no more text]
Braille display completed
Push Speech to braille button
Speak
Click next
Braille reading completed

Figure 1. A diagram detailing the operation of the whole system.