Software and Microcontroller Implementation for a Braille Teaching Device

By

Jonathan A. Hernandez

Electrical and Biomedical Engineering Design Project (4BI6)

Department of Electrical and Computer Engineering

McMaster University

Hamilton, Ontario, Canada
Software and Microcontroller Implementation for a Braille Teaching Device

By

Jonathan A. Hernandez

Electrical and Biomedical Engineering

Faculty Advisor: Prof. Doyle

Electrical and Biomedical Engineering Project Report

submitted in partial fulfillment of the degree of

Bachelor of Engineering

McMaster University

Hamilton, Ontario, Canada
Abstract

Despite all the technological advancements, the accessibility of Braille has declined in recent years. Less than 10% of the United States population is capable of reading Braille and less than 10% of children are currently learning it. The Braille Teaching Device aims to create a practical and affordable solution that allows a person to learn with an assistant or on their own. Development of a computer application that integrates numerous computer hardware and software components into a single package is necessary for efficiency and accessibility. Written in C#, the program interacts with an Arduino microcontroller, MATLAB and a Braille Teaching Device to help teach a person how to read Braille. The microcontroller must also be programmed to communicate between the software and hardware; determining which letters are to be displayed on the device and allowing an external form of GUI navigation. The program provides several ways of teaching Braille, allowing a person to continuously challenge themselves while learning. Working with a partner, a person can choose words from a pre-defined database or use their own words if they desire. Alternatively, a person can be tested using a random selection of words. Should a person desire to independently learn, the program allows interaction with Speech Recognition software that takes in a vocal response and determines if the correct word is identified. The program also records statistics for each test, allowing the reader to track and identify their progress over time.

Key words: assistive device, Braille, computer input, microcontroller programming, program interfacing,
Acknowledgements

The author would like to acknowledge the contributions of certain people who aided in the completion of this project. Dr. Doyle served as the course coordinator and the group’s supervisor; providing help and ensuring progress throughout the course.

Other acknowledgements go out to all the students in the 4BI6 Biomedical Capstone Course. The insight and perspective of other peers provided valuable feedback in developing and creating a better final product and better prepared the group for possible difficulties and obstacles.
Contents

Abstract ........................................................................................................ iii
Acknowledgements ....................................................................................... iv
Contents ......................................................................................................... v
List of Figures ............................................................................................... ix

1 Introduction ................................................................................................ 10
  1.1 Background .......................................................................................... 10
  1.2 Objectives ............................................................................................ 11
  1.3 General Approach to the Problem ....................................................... 12
  1.4 Scope of the Project .............................................................................. 12

2 Literature Review ....................................................................................... 14
  2.1 AutOMathic Blocks .............................................................................. 14
  2.2 Touchable Online Braille Generator ................................................... 15
  2.3 Braille Labelmaker .............................................................................. 15

3 Statement of Problem and Methodology of Solution ............................ 17
  3.1 Overview ............................................................................................... 17
  3.2 Software Methodology of Solution ....................................................... 17
    3.2.1 Programming Language ................................................................. 17
    3.2.2 GUI Layout ..................................................................................... 18
    3.2.3 Capital Letters and Numbers ......................................................... 18
3.2.4 Word Lists and File Reading. ........................................... 18
3.2.5 Editing Word Lists and File Writing. ............................... 19
3.2.6 Editing Word Lists – Special Cases. ................................. 21
3.2.7 Sorting Words. ............................................................ 21
3.2.8 Sorting Numbers. ......................................................... 22
3.2.9 PictureBoxes. ............................................................. 24
3.2.10 Picture Boxes – Special Cases. ....................................... 25
3.2.11 PictureBoxes and Advanced Display. .............................. 26
3.2.12 Random Generation of a Sub-List. ................................. 27
3.2.13 Timers. ..................................................................... 28
3.2.14 Countdowns in Random Mode. ...................................... 29
3.2.15 Statistic Tracking. ......................................................... 30
3.2.16 Deleting Statistics. ....................................................... 32
3.2.17 Tracking Correct and Skipped Words. ............................. 32
3.2.18 MATLAB and C# Communication. ............................... 34
3.2.19 Speech Recognition Problems and Design Considerations. .. 35
3.2.20 Speech Recognition within C#. ................................... 36
3.2.21 Sending Serial Data. ..................................................... 37
3.2.22 Pushbuttons and Receiving Serial Data. ......................... 37
3.2.23 Multi-threading and Playing Sounds. .............................. 38
3.3 Microcontroller Methodology of Solution ........................................... 39

3.3.1 Pushbuttons and Interrupts .............................................................. 39
3.3.2 Displaying Letters on Hardware ...................................................... 39
3.2.3 Capital Letters and Numbers ......................................................... 40
3.3.4 Handling Punctuation ................................................................. 41

4 Design Procedures ............................................................................. 43

4.1 Menu Navigation ............................................................................. 43
4.2 Learning Application ......................................................................... 43
4.3 Test Application – Timed ................................................................. 44
4.4 Test Application – Random ............................................................. 47
4.5 Speech Recognition ........................................................................... 48
4.6 Microcontroller ................................................................................. 49

5 Results and Discussion ........................................................................ 50

5.1 User Input Words ............................................................................. 50
5.2 GUI Control and Non-User Inputs Buttons ......................................... 51
5.3 Microcontroller Buttons ................................................................... 51
5.4 Other Exceptions and Considerations ............................................... 52

6 Conclusions and Recommendations .................................................. 53

Appendix A: English and Braille Letters ................................................... 55

References ............................................................................................ 57
List of Figures

2.1 Braille Labelmaker .................................................. 16

3.1 Adding Words ....................................................... 20

3.2 Removing Words .................................................... 20

3.3 ListBox.Sorted .......................................................... 22

3.4 Sorting Numbers Alphabetically ..................................... 23

3.5 Shell Sorting ............................................................. 23

3.6 Displaying Letters in PictureBoxes .................................... 24

3.7 Comparison of English and Braille Letters .......................... 25

3.8 Special Braille Letters ................................................. 26

3.9 Loading Braille and English Pictures .................................. 26

3.10 Displaying Letters with engWord ..................................... 27

3.11 Randomly Selecting Words ........................................... 28

3.12 Keeping Track of Time ................................................ 29

3.13 Tracking Statistics .................................................... 31

3.14 Recording Statistics .................................................. 32

3.15 Tracking Correct and Skipped Words ................................. 33

3.16 Completing Timed Test ............................................... 34

3.17 Referencing with MATLAB through C# .......................... 34

3.18 C# and MATLAB Communication .................................... 35

3.19 Correctly Guessed Words ............................................. 36
3.20 Pushbuttons. ......................................................... 38
3.21 Displaying Letters on Hardware. .................................. 40
3.22 Handling Capital Letters ............................................. 41
3.23 Handling Numbers ..................................................... 41
3.24 Handling Punctuation. ................................................ 42
4.1 Computer Program Main Menu. ....................................... 43
4.2 Learning Application .................................................... 44
4.3 Timed Test Application. ............................................... 45
4.4 Completing the Testing Application. ................................. 46
4.5 Cancelling the Testing Application. ................................. 46
4.6 Random Test Application .............................................. 47
4.7 Speech Recognition ..................................................... 48
A.1 English and Braille Letters. .......................................... 55
A.2 English and Braille Numbers ......................................... 55
A.3 English and Braille Punctuation .................................... 56
Chapter 1
Introduction

1.1 Background

One of the biggest problems regarding visually impaired people in our time is the inability to read Braille. Fewer than 10% of Americans read Braille and only 10% of blind children are learning the language [1]. Research on the subject shows a lot of work has been placed on further advancing the use of Braille and making it more applicable to daily usage for those who understand how to read it. Unfortunately little has been done to help make Braille more accessible and easier to learn for people in general.

There were two initial challenges to consider in the creation of this project. First, it was important to create an effective method of teaching, so that a visually impaired person would be able to learn the language over time. The other problem was to consider how the person would be able to relate to the device; what types of sounds could be used and what kind of feedback should the device provide so that the user would be able to identify if they are reading the right or wrong words.

The Braille Teaching Device was designed to cover two very different approaches. When connected to the computer, the user is able to work independently using the software’s implemented speech recognition, or can work with another person who can assist and even help teach the language without knowing how to read Braille themselves.

The entire scope of the project requires developments in both hardware and software. The hardware portion consists of the creation of an electronic device driven by a microcontroller that is connected to the computer. The software portion is split into two distinct parts. The first part consists of an application with a Graphic User Interface (GUI) that outputs commands to the microcontroller, which drives the proper pins to create a readable output in Braille. The intention behind this is to allow a second person to act as an aid or assistant in teaching Braille and to provide a more interactive and engaging experience in learning how to read. The second portion involves the
implementation of speech recognition software in MATLAB; allowing a person to use the device without the necessity of seeing what is on a computer screen.

1.2 Objectives

For this portion of the project, the main objectives are: to develop a computer application to help teach Braille and to program a microcontroller that would take serial data from the computer application and output it to some hardware which would create a legible word in Braille.

The computer application is designed under the context that there will be two people who will be operating both the computer program and the hardware. One person, who is the one trying to learn Braille, will be working specifically with the hardware. The other person will be operating the computer application and will be able to test their partner. The computer application is designed to show the word in Braille (as to be seen on the hardware device), as well as English. This allows the person working on the computer to be able to hint towards answers and provide feedback on what the person may be doing wrong. At the same time, the program allows an interactive method for an assistant to learn to read Braille, even if they do not have any previous knowledge of the language.

There are two different modes on the application. “Learning App” allows the computer user to choose words from a list and send them to the hardware device. The chosen word is displayed on the program in both English and Braille. The user can also add or remove words from the list.

“Test App.” tests the Braille reader based on either a time limit or a certain number of words. Words will be randomly chosen based on the dictionary of words compiled by the user. In “Test App.”, answers are to be typed in by the computer user. While the word in Braille is always displayed on the screen, the English language equivalent can be set to display only the correctly guessed words or can be completely hidden based on how the computer user wants to assist or give hints to the person reading.

For the hardware device to properly operate, the microcontroller must be programmed to read in the data sent from the program and translate it to specific pin outputs to tell the device which letters are to be displayed. It is also important for the microcontroller to be
able to detect inputs from push buttons, for the additional speech recognition feature. The push buttons are intended to replace the use of the keyboard and mouse, allowing somebody to use a microphone and a series of buttons to navigate through the different menus and commands. This part of the project encompasses all the interfacing required for the speech recognition to work with the device. All the calculations, computations and speech processing are covered by another group member.

1.3 General Approach to the Problem
The entire project is separated into three distinct parts: development of hardware, computer software and microcontroller implementation, and speech recognition. A general approach to this portion of the project would be to create computer software that sends a word out from the program to the hardware using a computer port to transmit data. The microcontroller inside the hardware would then read the data sent to it and drive certain output pins high to signify which pins in each Braille letter are to be raised. The device would then raise those pins, allowing the user to read the word from the hardware.

The speech recognition part could be accomplished by sending in a word to some algorithm or comparator. The program would then ask the reader to speak into a recorder and their speech would be compared with a stock answer. The end result would either say if the reader got the answer correct or incorrect.

1.4 Scope of the Project
The design will focus on a computer program capable of three distinct modes of teaching and the implementation of a microcontroller to command the hardware to output the respective Braille letters. The main computer software component for this project was written in C#, while the Boarduino microcontroller used a variant of C/C++. Speech recognition was programmed in MATLAB. The project is display the proof of concept and is limited to working with words that have a maximum length of four letters in Braille, as opposed to a four letter length in the English language. Braille notation can sometimes use an extra character preceding the word to note special cases such as a Capital letter, punctuation, or a numerical number.
Two of the modes are to be implemented in this portion of the project. One mode will allow a person on the computer to modify the database of words and choose the words output to the hardware. The other mode will choose a randomized set of words and simulate a basic test format based on either a limited amount of time or number of words the reader must attempt to identify. Both modes are implemented assuming that one person will be operating and running the computer application while the second person will be using the hardware to attempt to identify the given words in Braille. The answer can then be verified by having the computer user type the guess into the program.

The third mode, Speech Recognition, will be implemented by another member of the group. However, the computer software created will simply act as the interface and bridge between the speech recognition software and the hardware. In this respect, the C# program will select the necessary word or letter to MATLAB, where all the speech recording and processing will occur. MATLAB will then send a result back to C#, which will tell the user if their answer is right or wrong. Although a GUI has been implemented for this mode to work with a keyboard and mouse, it was also designed with the intention of only using the hardware to read the Braille output, a microphone to receive the signal and possible navigation from the available buttons.

It is important to make note that the word database for the speech recognition portion has been restricted to the simple alphabet consisting of the letters “a” through “z” only.
Chapter 2

Literature Review

The following literature review is not exclusive to Braille teaching devices but includes various types of devices that in aid in incorporating Braille with other activities. It was important to analyze these designs in hopes of realizing possible obstacles and problems for our own project. One of the main concerns in the initial design process was to create something self-explanatory and easy to use without having to read an in-depth tutorial or manual that covers the basics.

2.1 AutOMathic Blocks [2]

Professor Arthur Karshmer from the University of San Francisco developed a system using manipulative Braille blocks to teach mathematical problems in conjunction with computer assistance. For the device to work, the student is required to place the mathematical blocks onto a device consisting of several buttons and a grid which the blocks are placed onto.

Before placing a block onto the grid, the block is scanned by a machine and dictates the scanned number. Once placed into the grid, the machine announces which column and row the block was placed into. After all the blocks have been placed for the desired problem, the numbers appear on the screen to show a teacher what numbers the student has entered. The student then chooses the type of mathematical operation (ie. addition, subtraction, multiplication and division) and is then required to enter the suspected answer. As the student scans in blocks and places them into the answer row, the machine is capable of telling the user if they are inputting the correct numerical digits or if they are incorrect. Alternatively, the device could also work for grammar and spelling by placing alphabetical Braille blocks into the grid and having the machine do various functions ased on what the student has written with their Braille blocks.

Professor Karshmer’s design is interactive and allows a person uneducated in teaching Braille to supervise and help the student if they need it. One concern regarding the use of
this device is that more time may be spent on the actual scanning and placement of the blocks as opposed to actual time spent learning to read the language.

2.2 Touchable Online Braille Generator [3]

The Touchable Online Braille Generator is a program that translates a set of user-inputted text and utilizes a force-feedback mouse to provide an output. After the text is translated, the equivalent Braille translation for each word is shown directly below its English counterpart. As the mouse hovers over each letter, a form of force-feedback is provided through the mouse to signify what letter is shown on the screen.

The issue with this process is that there is not a lot of flexibility between selecting the text to be translated and receiving the end result from the generator. A person with vision is needed to operate the computer, select the desired text from some form of media, place it into the online generator then translate it to Braille for someone else to read.

2.3 Braille Labelmaker [4]

A project initially started by several students at MIT, the Braille Labelmaker allows an easy method of labeling and marking items to avoid possible confusion. The label maker is a necessity of distinguishing similarly shaped objects from one another, such as marking certain types of canned foods after purchasing groceries. The device is portable and handheld, along with being very simple to use.

A person would insert a roll of label tape into the device and continuously press the space bar until a portion of tape appears on the output side. The length of tape can then be readjusted should there be an excess of tape sticking out. The person would then turn the device on and push the respective Braille keys for each letter they wish to type. After releasing a set of keys, the label maker would then quickly write the letter and push the tape outward so that the next letter can be written. Once finished typing, the user would move a switch that triggers an internal cutting mechanism to cut the label so it can be used.
The Braille Labelmaker is a great design that displays how simple interactions can make a product useful and applicable to everyday life. The only concern with the project was that the label maker specifically requires text to be typed in Braille and does not allow any other form of typing for visually-abled people who might want to use the device to label items for a friend or relative.
Chapter 3
Statement of Problem and Methodology of Solution

3.1 Overview
The project aims to develop computer software that sends data to the hardware and microcontroller to output a specific word or letter in Braille. The software is to allow the user to choose words, or have an algorithm randomly choose words from a database. The word database should be easily modifiable, allowing words to be added or removed and must account for preventing inappropriate words (i.e. inappropriate length, invalid mix of characters, improper capital letters, etc.) from entering the database.

The program, written in C#, must also interact with MATLAB to perform Speech Recognition. C# will tell MATLAB which letters to test and in turn will receive an output from MATLAB stating if the recorded answer is correct or incorrect. All processing, including the recording and comparison of the reader’s speech will occur in MATLAB and is not covered in this portion of the project. Finally, the microcontroller implementation will receive an output from the computer software and will be programmed to drive certain components of the hardware so that the chosen word in the software will be translated over to the hardware for somebody to read.

Although C# is an object oriented language, all code (except controlling the GUI) was programmed in a non-object oriented manner (i.e. void of classes, subclasses and inheritance). All algorithms are therefore presented as flow charts and not under UML 2.0 standards. For further details on the program code and how each mode works as a whole, please refer to Chapter 4 – Design Procedures.

3.2 Software Methodology of Solution
3.2.1 Programming Language
The program was written in C# because it allowed an easy method of creating and modifying a GUI without extensive knowledge or research required. Minor attempts
programming in Java and QT showed insufficient progress for the amount of time we were given to complete the project.

Further reasons for choosing C# involved the necessity to be able to interact with a USB connected microcontroller and the ability to interact with functions written in MATLAB. C# allows event handling for receiving data from a microcontroller; necessary to distinguish which buttons were pushed so the program can respond accordingly.

### 3.2.2 GUI Layout

The navigation of the GUI was handled by using panels as opposed to TabControl. Although TabControl is much simpler to manage, the use of panels allows the same objects to be reused in various functions, allowing code to be reused and requiring less computer resources; multiple instances of objects would need to be created in each TabPanel. Panels are a bit trickier to use and require the proper docking, hiding and showing of panels each time you want the GUI to change but are much more flexible to work with in general.

Simple functions were created to help manage navigating through the program and increase program efficiency by reducing the overall size of code. Each time a panel was changed, panelHideAll() would be called to hide every panel in the program before calling another panel function to make specific panels visible.

### 3.2.3 Capital Letters and Numbers

Braille places a special character before a phrase to denote if it is the start of a sentence or a number. Since the hardware is only capable of displaying four Braille letters, either scenario must be limited to a maximum three character length. Further elaboration of how the program handles these special cases is detailed in the following subsections.

### 3.2.4 Word Lists and File Reading

One of the features in this program is the ability to create and modify the list of words however needed. The words lists are accessible in “Learning Mode” and any changes made to each list will also be applied to “Testing Mode”.
The dictionary of all words for the program is split up into several distinct categories: one letter words, two letter words, three letter words, four letter words, punctuation, capital letters and numerical orders. Any time a list is loaded, the object StreamReader is used to read through a .txt file line by line and add each word into a ListBox to display all necessary words. The “Learning Mode” ListBoxes are set to alphabetical order to allow easy and efficient sorting. Unfortunately the main tradeoff in this case is that the numerical list is loaded alphabetically and not numerically, causing numbers like 1005 to appear before 654. Should “All Words” be chosen for the database of words, an extra algorithm is created to read through each of the files (one letter words to four letter words) and the program makes note that all words were selected.

3.2.5 Editing Word Lists and File Writing
One of the features of the program is that it can store a database of words that the user has created. This means that any changes, including addition and removal of words must be implemented. Since the list of words is stored as a .txt file, anytime a change is mind to one a list its respective .txt file must be updated.

Before adding a word to the list, the program first checks if the word is already a part of the dictionary. It then checks if the length of the word matches the type of list it is to be added to. Finally the program checks if the word contains acceptable characters and adds it to the dictionary. When an accepted word is added, the program updates the ListBox with the new word. Since the ListBox that takes in the word is sorted alphabetically, the new word is automatically placed at the proper index. The program then determines which set of words was updated and writes the entire list to the respective file.
When a word is removed, the program first checks if the word exists in the respective database it would be found in. Once it is found to exist, the word is removed from the ListBox and all of its remaining contents are rewritten back to the .txt file. If the word does not exist, nothing happens and the program throws an error message to the computer user.
Although it may seem inefficient to immediately update the program’s files after a word is added or removed, it is important that the changes occur immediately should the person decide to switch to another mode and want to use their newly changed list of words. Each list is assumed to be relatively small (ideally less than 100 words per list) so reading and writing from each file should not be too heavy on computational resources.

3.2.6 Editing Word Lists – Special Cases

If the user tries to add a word containing a capital letter, the program first checks if the proper list is selected. After verifying the current set of words, the program then checks to make sure that the word is three letters or less. The rest of the steps taken to add the word are similar to the previous subsection.

If the user tries to add a number the program first checks if the proper list is selected. After verifying the current list is for numbers, the program then checks to make sure that the number is three digits or less. The rest of the steps taken to add the number are similar to the previous subsection.

For further information on how the program deals with special cases please see 3.2.10 PictureBoxes – Special Cases on page 25.

3.2.7 Sorting Words

C# allows efficient addition of words to an alphabetically sorted ListBox. To alphabetically sort in other programming languages, the user would have to write a method to determine where the word would fall into place. It would then move every following word down one spot before copying the word into the list. In C# the user can simply set a ListBox’s sorted property to true. Any words added to the list will automatically be sorted and placed into its corresponding spot.
While C# allows words to be automatically sorted by letter, it does not have any method to automatically sort numbers. When entering numbers into a ListBox and the Sorted property is true, the numbers will be sorted alphabetically and not numerically as shown in Figure 3.4. To present a numerically sorted ListBox, all the contents are copied into an integer array. The array is then sorted using the Shell sort algorithm.
Shell sorting works by repeatedly sorting an array’s indices. The algorithm first analyzes smaller segments of the array and numerically sorts them. The algorithm then analyzes bigger sized segments and performs a numerical sort. The algorithm continues and increases in segment size until everything is sorted. A simple example of shell sorting is provided below.

![Figure 3.4: List of numbers that are sorted alphabetically.](image)

After the array is numerically sorted the ListBox is cleared and all items inside it are deleted. The “Sorted” property is set to false and the elements inside the array are copied one at a time. The ListBox then displays the numbers in proper numerical order; increasing in quantity as the list goes down. Should the list be changed back to an alphabetical list, the “Sorted” property is set back to true to allow alphabetical sorting.

![Figure 3.5: Simple example of Shell Sorting](image)
3.2.9 PictureBoxes

When a word is chosen to be sent to the device, the software’s PictureBoxes automatically update to display the new word in both English and Braille. Regardless of what mode the computer user is in, the program will always display the word in Braille. The English language equivalent word can however be toggled several different options should “Testing Mode” be running. The English word can either be displayed in full, not displayed at all, or can display the correctly guessed letters while keeping the incorrect letters hidden.

When one of the PictureBox methods is called, the word is broken up and analyzed letter by letter from right to left. The last letter in the word is analyzed first so that all of the words appear right-aligned in the program and any unused (blank/white) PictureBoxes will be to the left of the word. Although left-aligning the letters to each PictureBox would be easier to program, right-aligning the word gives the program a more professional appearance.

![Diagram of how the program reads a word and displays it in the PictureBoxes](image)

Figure 3.6: How the program reads a word and displays it in the PictureBoxes

As a word is broken into its individual letters, the word is checked to determine its type (ie. numerical, punctuation, first letter is capitalized, lower-case only, etc.) and the
program then uses continuous if-statements to find what image should be loaded into each PictureBox. Special attention must be paid to numbers, words containing a capitalized first letter and punctuation. In any of these cases, the English PictureBox set uses one extra character (ie. the left-most character) to display that the following word should not be read as an alphabetical word. For numbers and words containing capital letters, the Braille language actually uses a character that precedes the word to denote it is not strictly lower-cased. This is because some English letters and numbers share the same Braille translation. Such an example would be how the number “1” would be read as “a” in Braille in the event that the number example did not precede it.

![Figure 3.7: Translation of a letter and number from English to Braille.](image)

### 3.2.10 Picture Boxes – Special Cases

Extra attention must be paid to words containing capital letters or numbers. The Braille equivalent uses one extra letter that precedes the word; denoting that the word should not be read as if it was a simple lower-case phrase. Once a word is chosen, the program checks if the phrase is strictly lower-case letters or if it contains unique properties. If the phrase contains other characters outside of the lower-case alphabet the program will call a specialized function to properly display the word on the PictureBoxes.

These functions are programmed to account for the extra character needed at the start of the word. Since the word is right-aligned with the PictureBoxes, it is displayed at the same time the left-most character is analyzed.
3.2.11 PictureBoxes and Advanced Display

While this feature may not be useful to the person reading from the device, it provides the option for the assistant to see what mistakes the reader might be making and allows them to provide insight by telling the reader where they are going wrong. This feature is tracked on the GUI by three side buttons and is tracked in the code by using a global variable called “engWord” and a variable strings to compare the current word to the guessed word.
Changing between the buttons causes engWord to obtain a different value, ranging from 0 to 2. When the PictureBox method is called, the guessed word is compared to the actual word at each index. The correctly guessed indices are noted and will properly display when toggled to this option. When loadTestImage() or any other equivalent method is called, a new function similar to the one explained in 3.2.5 is used. The function loads all the words in both English and Braille, then makes the necessary changes should the program not be set to show the entire English word.

![Figure 3.10: Use of engWord to change PictureBox display](image)

**3.2.12 Random Generation of a Sub-List**

Before randomly choosing the words used in a test application, all the words available are copied and stored into a ListBox. A second ListBox is used to store all the words used for testing, while an integer variable stores the total number of words available.

The command “objRan.Next(0, max);” is used to generate a random number between zero and the total numbers found inside the first ListBox. Using that randomly generated
number, the word at that index is deleted from the first ListBox and copied into the second one. The word is removed from the first ListBox to ensure that it is not copied over multiple times. This process is then repeated until the desired amount of words is met.

![Algorithm diagram](image)

Figure 3.11: Algorithm used to randomly select words in Test Application

**3.2.13 Timers**

In C#, the timers are set to trigger after a specified number of millisecond. Two timers were used to correctly update the GUI in real-time. Although it is possible to use one timer to do all of the work, two timers were used to simplify the amount of programming needed. One of the timers was set to trigger once every second (1,000ms) while the other timer would trigger once every minute (60,000ms). The seconds timer is initialized to “60”, whereas the minutes timer is initialized to “9”. Every time one second is counted down, the seconds timer is updated. After 60 seconds, the minute timer is triggered. The minute timer then decrements itself and resets the seconds timer to start at 60 seconds. Since the minute timer is set to “9”, it ensures that the seconds timer works for the proper duration and does not get restarted for an extra minute.
Once the 10 minute duration has passed, a function is called to stop the test and prevent any more guesses from being entered. The stats for that session are then recorded and all the right and wrong words are shown. When the menu is exited or cancelled, the timers are all reset to their initial values to that they will properly work for the next timed session.

### 3.2.14 Countdowns in Random Mode

Similar to the timers, a variable was used to keep track of the number of words left in a random test session. When a random test is started, the program checks to see if there are a minimum of 20 words available. In the event that there are less than 20 words, the program finds the total number and updates the test to make the counter start decrementing from that value. Each time a word is skipped or correctly guessed, the variable is decremented and the label is updated to display the amount of words left. Once the counter hits zero, a function is called to update the necessary panels. All correctly guessed and skipped words are shown and the counter is reset for the next run.
3.2.15 Statistic Tracking

In designing the program, it was decided to encourage users to guess words as much as they wanted to without any form of penalty. Although the statistics track how many times a word was incorrectly guessed, it is only stored to allow reflection and self-improvement. Statistics are tracked using four arrays; two are used to track a current session, while the other two are used to track the overall statistics before a session is started.

When the program is initially started, it reads through statsRand.txt and statsTest.txt. The text is converted to numerical information and is stored into two separate integer arrays to allow the statistics to be updated when a test is completed. The data is then displayed in several text labels that are displayed when a test is occurring. If the statistics have been reset or the program is freshly run, the program avoids showing any statistical information in percentages.

Once a word is randomly chosen, it is converted to a string variable and stored. When a guess is attempted, the word is checked to ensure that it has the proper amount of letters as the answer. If the amount of letters does not match, an exception is thrown to the user and the guess is discarded without being recorded. Should the word length of the guessed answer match the right answer, the guess is then compared to the actual word. If the strings do not match, the statistics will record an incorrect guess for the current session and will wait for the user to either correctly identify the word or skip it. If the word is currently identified, the program will note a correct guess and move onto the next word. Similarly, if a word is skipped it will be noted and move onto the next word.
Once a test is completed, the array storing the current statistics is added to the array that keeps track of the overall statistics. The “current” array is then reset, while the “overall” array is written to a text file so that the statistics are updated. In the event a test is cancelled, all current statistics are reset and are not recorded.
3.2.16 Deleting Statistics
The statistics can be reset from the main menu. Should a user decide to clear the statistics, the program simply sets every index in every integer array to “0” and then calls the method to update each statistic file. The labels are then updated to display the newly cleared statistics.

3.2.17 Tracking Correct and Skipped Words
As stated in the previous subsection, it was desired that the program allow as many attempts as possible on a word without any form of a penalty incurring. When the user is being tested, the program will allow infinite attempts on a word until it is either correctly identified or skipped. Incorrect guesses are only statistically recorded to allow a user to gain some knowledge of their progression.
Figure 3.15: Algorithm used to track which words are correct or skipped

When a test is completed, all correctly guessed and skipped words are displayed to the user. ListBoxes are used to track each correctly identified and skipped word. When a test begins, a ListBox is used to store all the test words. Two other ListBoxes are used to track all the correct and skipped words. When the user correctly identifies the word, the word is copied over to the “Correct” ListBox. When the user decides to skip a word, the choice is noted and the word is copied over to the “Skipped” ListBox. At the end of a test, both ListBoxes are shown allowing the user to go over the entire test and see any words that they may have had trouble with. If a test is cancelled, neither ListBox will be shown.
3.2.18 MATLAB and C# Communication

For MATLAB to work with C#, MATLAB must correctly be identified as a reference object in C#. For this project a reference was created by “Right-Clicking” references and choosing “Matlab Application” under the “COM” tab.

Once a reference is established, the MATLAB application must be initialized in the Form. For further reference, please refer to the code under the method panelSpeech().
Once the user enters into the speech recognition application, the program will attempt to open a MATLAB command window that relays information between C# and MATLAB. Any M-functions must be placed in the directory of the MATLAB.exe or else they will not work when called.

It should be noted that both MATLAB and C# have separate protocols for interacting with one another. For a MATLAB function to work externally, C# must send the proper syntax to MATLAB. Text must be sent to MATLAB as if the code was being written into a normal MATLAB console. The output must then be appended and converted to the proper variable type to retain any useful information. A simple example is provided below:

![Diagram showing C# and MATLAB communication](image)

3.2.19 Speech Recognition Problems and Design Considerations

One of the major issues between C# and MATLAB integration is that all communication can only be handled through sending and receiving text. While simple sounds can be recorded and played under the MATLAB Command Window, it is a lot more difficult to call C# to find and play those same files. To simplify the design process both C# and MATLAB use their own set of sound files and only pass numerical numbers to each other.
In this program C# can call MATLAB to record speech but cannot access it once it is recorded. Simple sounds and phrases were recorded by using the voice “Microsoft Sam” in Windows. After MATLAB sends the result back, C# then plays pre-recorded sound files to tell the user what the result is.

3.2.20 Speech Recognition within C#

For basic MATLAB and C# integration, please refer to 3.2._____. For details on how the MATLAB Speech Recognition works, please refer to ______ by Brett Lindsay. Once the program enters the Speech Recognition application, it attempts to open a MATLAB command window. If the computer does not have MATLAB installed, an exception is thrown and the user is sent back to the main menu.

![Algorithm diagram](image)

**Figure 3.19: Algorithm used to identify a correctly guessed word**

Upon initiation, C# will send a number to MATLAB that represents which value will be tested. The MATLAB function will always return a three digit number to C#. Once an answer is recorded, MATLAB inspects the speech and attempts to match it to the
database. If a match is not found, an output of 0XX will be received. Should MATLAB properly match the recorded speech to a letter, XX will range from 01 to 26; specifying the letter that was guessed by the user. If MATLAB cannot identify the guessed letter, it will output 050 and C# will notify the user that the speech was unreadable. Should the user correctly say the unknown letter, MATLAB will return a value of 100. The program will then tell the user that the letter was correctly identified and will move on to the next letter.

3.2.21 Sending Serial Data

When communicating with the microcontroller, data is sent and received. The functions microLearn(), microSpeech(), microTest() and microEndTest() are used to communicate with the microcontroller and send data. Each function uses the same protocol and sends a string “##xxxx”, consisting of a two digit number followed by the actual word. The first digit of the number denotes the type of word that is being sent: 0 – lower case, 1 – contains capital letter, 2 – numerical, 3 – punctuation. The second digit denotes how long the word is in English.

3.2.22 Pushbuttons / Receiving Serial Data

The program also receives serial data when buttons are pushed and is handled by the event sp_DataReceived() in C#. The buttons attached to the microcontroller act differently based on what the program is currently doing. The variable navButton is set to “0” when outside of Speech Recognition and is set to “1” when inside speech Recognition. When outside Speech Recognition, pushing Button #1 allows the user to enter the mode. All other buttons do not do anything. When inside Speech Recognition, all four buttons behave differently and are explained in the diagram below.
Multi-threading / Playing Sounds

A simple computer program runs on a single thread; it executes code and waits for it to finish before moving onto the next command. Multi-threading allows multiple commands to run simultaneously and is sometimes necessary depending on how a person wants to implement the functionality of a program. When using the serial port to communicate with the microcontroller, the program has to invoke a new thread to interact with itself.

When playing sound files within C#, the SoundPlayer.Play() function automatically calls a new thread on its own to play the file. This implies that the program is still running and is capable of calling functions while playing sound. In Speech Recognition, the application heavily relies on the use of sound files to notify the user of its current state. To prevent the user from overflowing the program while it is playing a sound file, a time delay is placed after playing each file. The time delay pauses the program and allows it to play the sound before moving on. The most important use of this is in recording an
answer with the microphone; the program plays a simple tone and waits for the tone to stop before it calls the MATLAB function to record and interpret the answer.

3.3 Microcontroller Methodology of Solution
The following sections only talk about how the microcontroller interacts with the hardware to show each Braille letter. It also covers all the microcontroller’s perspective of integrating with the C# application. For specifics on how the hardware works, please refer to Chris Agam’s report.

3.3.1 Pushbuttons / Interrupts
The microcontroller utilizes interrupts to detect when a button is pushed. The microcontroller is set to read in the voltages from the pins connected to the push buttons. The buttons feed in a LOW signal when idle and change to HIGH when pushed. The microcontroller recognizes the state change and calls an interrupt method that sends data to the program.

When a button is pushed, the microcontroller sends serial data for the program to receive. Each button sends different data to the program. The program then determines what was sent from the microcontroller and does the necessary action. A delay is placed after the data is sent to prevent the microcontroller from accidentally repeating itself.

3.3.2 Displaying Letters on Hardware
To display a letter on the hardware, the program sends a string “##xxxx”, consisting of letters and numbers. The string consists of a two digit number, followed by the actual word or text to be shown in Braille. The first digit represents the type of text to be shown: 0 – lowercase word, 1 – word starting with a Capital letter, 2 – number, 3 – punctuation. The second digit represents the length of the word in English. One should be reminded that capital letters and numbers use an extra letter in Braille and the specific algorithm used to handle these cases is explained in the following section.

Similar to displaying PictureBoxes, the microcontroller works by displaying the right-most letter first and then moves from right to left. The microcontroller first goes through each Braille letter and clears all the relays by setting each one to HIGH. Starting with the
right-most letter, the microcontroller uses setBraille() to determine which letter is to be displayed and sets the relays HIGH or LOW. Note that the relays lower when receiving a HIGH. After all the relays on a Braille letter are properly set, the microcontroller changes outputs to the decoder to signal that it is changing the next letter and the process repeats until all letters are displayed.

Figure 3.21: Algorithm used by microcontroller to display each letter on the Braille Teaching Device

### 3.3.3 Handling Capitals and Numbers

When working with capital letters or numbers, the microcontroller must account for the extra Braille letter shown at the start of the word. When the word sent to the microcontroller is detected to contain special characters, the microcontroller calls the functions capSet() and numSet(). capSet() works by adding a “+” to the start of the word, so that the function setBraille will use that letter to denote a capital letter. The capital letter is then replaced by its lowercase equivalent. Comparatively, numSet() adds a “#” to the start of the word and then converts each digit to its respective English letter. Recall that Braille uses the same layout for letters and numbers and instead places an extra letter at the start of the word to denote that it is numerical.
Figure 3.22: Example of how the microcontroller deals with capital letters and how the word is different in Braille

Figure 3.23: Example of how the microcontroller deals with numbers and the differences of displaying a word or number in Braille

3.3.4 Handling Punctuation

Both the computer program and the microcontroller handle punctuation differently due to difficulties with programming languages and comparing strings. Although the text can be
compared using their respective ASCII integer values, the program simply converts the punctuation to an alphabetical letter and tells the microcontroller that Braille punctuation is to be displayed on the hardware.

Punctuation noted.  
program sends:  
k  

microcontroller recognizes as: 
Semicolon (;)

Figure 3.24: Example of how the microcontroller deals with punctuation
Chapter 4
Design Procedures

The following design procedures explain the general process for each application and how the program functions as a whole. For descriptions of individual algorithms or how things work in general, please see Chapter 3 – Statement of Problem and Methodology of Solution.

4.1 Menu Navigation

Upon running the program, the user is taken the main menu. From here, they can either go into the Learning Application or Test Application. To change what is displayed on the screen, the program hides unnecessary panels while relocating and resizing the necessary panels to properly fit on the screen.

![Figure 4.1: Main menu](image)

4.2 Learning Application

Once the Learning Application is started, a simple notification is left on the right side of the screen to tell the user how to start using the program. To begin, the computer user must select the set of words they want to work with. Once loaded into the ListBox, the user must then choose a word they want. Once that word is sent to the microcontroller, the PictureBoxes are updated to show the word in both Braille and English. Clicking
“Random Word” will randomly choose a word from the ListBox and send it to the microcontroller.

If the user wants to modify a list, they can do so by clicking the “Modify Current List” button. A textbox appears below it, prompting the user to enter the word they wish to be added or removed. If the word is to be added, the list will first check if the word exists in the current list. Should the word not be found, it is then inspected to ensure that it is the proper length and contains the proper characters. If the word is to be removed, the program checks if the word exists in the ListBox and deletes it if found. After a successful change, the entire list is then rewritten to file to store any changes made.

4.3 Test Application – Timed

Once a timed test is initiated, the user is allowed to attempt as many words as possible in 10 minutes. The program will randomly choose up to 75 words for a list, as an upper limit for advanced users. If there are less than 75 words available, the program will simply take all the words in the list.
The person using the hardware is then to identify what word is being displayed. To allow cross referencing, the program displays both English and Braille words to the computer user. When the reader wants to make an attempt, they simply tell the computer user their answer and have it typed into the program. The program will only notify when a word is correctly guessed; it will not notify the user when they are incorrect. With each attempt, the statistics of skipped words, correct and incorrect guesses is updated and displayed onto the screen. When a word is correctly identified or skipped, the next word on the list is automatically chosen and sent to the Braille device.

The process will repeat until one of three things occur: the user goes through all the words in the list, the user runs out of time or the user decides to cancel the test before finishing. If a test is successfully completed, the statistics are recorded and the program will display all words correctly guessed or skipped in that session. If a test is skipped, the statistics are simply discarded and none of the words are shown. Once the application is finished and the user returns to the main menu, all current statistics are cleared and all timers are reset for the next session. Finally, the program clears all the lists of words used to randomly generate the test list.
Figure 4.4: User has just completed a test in the Testing Application

Figure 4.5: User has cancelled a test in the Testing Application
4.4 Test Application – Random

Figure 4.6: Using the Testing Application with 20 randomly chosen words

Once a random test is initiated, the user is given a set of 20 randomly chosen words to identify. If there are less than 20 words available, the program will simply take all the words in the list.

The person using the hardware is then to identify what word is being displayed. To allow cross referencing, the program displays both English and Braille words to the computer user. When the reader wants to make an attempt, they simply tell the computer user their answer and have it typed into the program. The program will only notify when a word is correctly guessed; it will not notify the user when they are incorrect. With each attempt, the statistics of skipped words, correct and incorrect guesses is updated and displayed onto the screen. When a word is correctly identified or skipped, the next word on the list is automatically chosen and sent to the Braille device.

The process will repeat until one of three two events occur: the user goes through all the words in the list, or the user decides to cancel the test before finishing. If a test is successfully completed, the statistics are recorded and the program will display all words correctly guessed or skipped in that session. If a test is skipped, the statistics are simply discarded and none of the words are shown. Once the application is finished and the user
returns to the main menu, all current statistics are cleared and the program clears all the lists of words used to randomly generate the test list.

4.5 Speech Recognition

![Figure 4.7: Inside the Speech Recognition Application](image)

The Speech Recognition application was designed under the idea that only one person would be operating the Braille Teaching Device. Although the application is functional with the GUI, the program is intended to be used for a person strictly reading words on the device. Navigation in this application can be achieved by using the buttons connected to the microcontroller. The use of the keyboard and mouse is optional.

Upon entering the Speech Recognition application, the program checks if MATLAB is installed. If MATLAB is not detected, the user will be notified and sent back to the main menu. When MATLAB is found, a command window is opened and runs in the background. The program then reads in the list of all single letter words and randomly organizes them, similar to a testing application. After scrambling the list the program then sends MATLAB a value of 01 to 26; signifying which letter is currently being displayed on the device.

The user must then guess the word being displayed by either pressing “Record Speech Input” or the Record button connected to the microcontroller. Upon calling the record
command, the user is to wait for a beep to occur before saying their answer into the microphone.

Once recorded, the MATLAB function then attempts to match the guess to a letter. If the speech is properly matched, the MATLAB function will output a value of “100”. If the speech is matched to an incorrect letter MATLAB will return a value of “0XX” where XX signifies what letter they guessed. If the speech cannot be interpreted, MATLAB will return a value of “050”.

The user can then check their answer using the “Verify Speech Input” button on the GUI or its respective Verify button on the hardware. If the user attempts to verify their answer without making any guesses, the program will alert them that they must guess before comparing their answer. If a user correctly guesses the word, the program will repeat the letter displayed on the device then move onto the next letter. If the user’s guess was incorrect, the program will tell them to try again. The user also has the option of moving to the next letter by pressing the Previous/Next GUI buttons or the Next button on the device.

When the user is done with the mode they can either return to the main menu by pressing the “Main Menu” GUI button or the Exit button connected to the device.

4.6 Microcontroller
The microcontroller is simply idle and waits to either receive or send data to the C# program. It is programmed to continuously loop and check for any data received from C#, denoting that a new word is to be displayed on the device. It then determines what type of word is to be displayed and goes through each Braille letter setting its relays HIGH or LOW. Setting a relay to HIGH will result in it lowering. After displaying the word the microcontroller turns to an idle state.

Alternatively, the microcontroller sends data to the program when buttons are pushed. Theses buttons trigger a change in pin voltage and cause the microcontroller to run interrupt functions. These functions check to make sure that the voltage is properly read in then send specific commands to the program, as if it was simulating simplified mouse or keyboard clicks.
CHAPTER 5

Results and Discussion

This section will cover the considerations of user input and exception handling for the software program and microcontroller. It is difficult to provide any graphical analysis of results due to the scope of this project. Since this project focuses on the development of a computer program to interact as an interface with hardware and other software, there is little to no data that requires analysis. The user input is very restricted and limited; the software only interprets commands from buttons being pressed within the GUI, hardware buttons being pushed or taking word inputs for all of the various applications.

5.1 User Input Words

User input in this project is defined as actual text containing of letters and numbers that is input into the program to either modify a list of words or compare as an answer to the hardware. All other buttons on the GUI act as either navigational tools or control the program in a specified matter. These types of buttons rely on the user to activate them. They do not take in any form of data to analyze and are considered to be “non-user input” buttons for this project.

A user can modify a list in “Learning App.” by entering a string of text and choosing to add or remove the word. If a string of text is to be added, the program first checks if it is already contained in the list. If it is not found inside the list, the program checks to see which list is currently being referenced. It then filters the string to see if it contains strictly characters or numbers. This result is cross referenced with the selected word list. Should the two match the word is successfully added. An exception is thrown for all other cases. Since the addition of words filters out all unnecessary and improper words, the removal process simply checks if the word exists and removes it if found.

When in “Testing App.” the computer user is required to type into the program what the Braille reader believes is the correct answer. The inputted string is first checked to see if it contains the same length as the word on the device. It is then compared character by character, in the event the user wants to know which letters were correctly guessed. The
program then compares the two strings and if equal, notifies that the word was correctly identified and moves on. The program will remain on the same word until a word is correctly identified or skipped.

5.2 GUI Control and Non-User Inputs Buttons
All non-user input buttons either act to navigate through the program or change the way the program is performing. Since it is difficult to go through and detail each scenario in the program, a general protocol will be provided that explains any precautions and considerations taken. A lot of functions are reused in managing each application so it is recommended to refer to the code for further understanding.

When navigating through the GUI, the program always hides all of the unnecessary panels. When entering an application, the GUI only shows relevant panels. To simplify coding, the program will hide all panels, visible or not, when exiting an application. In addition to that the program clears out any lists referenced, variables used and also resets all PictureBoxes so that they display a blank image.

When the Braille reader correctly guesses a word in “Testing App.” or chooses to move to another word in “Speech Recognition”, the program uses specific methods to display the necessary PictureBoxes and also sends the word to the microcontroller to display on the hardware.

5.3 Microcontroller Buttons
Pressing any of the microcontroller buttons outside of “Speech Recognition” will automatically set the program to run it. Once in “Speech Recognition” mode each button behaves differently. Upon pressing a button, it sends a unique string of data back to the program. The program then reads in the string and identifies which command is being called. Since the microcontroller can only send or receive data at a single incident, a time delay is placed on all microcontroller incidents where data is sent or received from the program to the hardware. Any buttons presses are ignored while the microcontroller is paused.
5.4 Other Exceptions and Considerations

It is assumed that any changes to files or other programs will only be done through the software and not externally. Manipulating any of the text files containing words or statistics can result in errors that the program will not properly handle. The program also requires the MATLAB command window to be open for “Speech Recognition” to properly work. Upon entering the application the program checks if MATLAB is properly installed and can be accessed. If MATLAB is not working the program will notify the user and effectively locks the user out of the application. In the scenario where the command window is successfully opened by the program and then closed by the user, calling any commands to MATLAB will result in the program crashing.
Chapter 6

Conclusion and Recommendations

The greatest difficulty in developing this computer program was the freedom of choice granted by the design project. There are an infinite number of features that can be incorporated as well as an infinite number of ways to program each one. At times it was difficult to choose between implementing a fully functional feature versus a simple proof of concept design. An outline of realistic goals and a strict development schedule is necessary to ensure progress towards a completed product. Although more features and options could have been implemented, a majority of the original goals were achieved to satisfactory level of quality.

Additionally, several extra ideas were incorporated into the project while other ideas were scrapped due to poor functionality or time constraints. Although the program was designed to be as user friendly as possible, more detailed explanations or tutorials on how to use each application could have been implemented. The GUI could have also been designed to specific standards or outlines to increase efficiency of navigation and overall accessibility. ISO-9241 and Fitt’s law are two examples of GUI principles that can be used to improve a layout. Without any previous experience in designing an actual GUI, it was difficult to take a design on paper and translate it to something fairly intuitive.

User feedback proved to be extremely useful in testing functionality and determining any bugs or errors the program had. Although some actions were programmed with the intent of being simple and apparent, the general population was found to disagree on occasion and often provided ways to improve upon the initial design. While a teaching program may make sense from a design perspective, it will not be a commercial success if its learning curve is too high. Had additional time been provided, it would have also been desired to test how well the program can improve a person’s ability to read Braille over weeks or months.

Most of the code was also written with a focus on functionality over efficiency due to the amount of different things that the program wanted to allow a person to perform. In
reality, all code should be efficient as possible to reduce runtime and decrease the opportunity for error to occur. With educational software, it should be assumed that the target market will be running dated equipment and will rarely be running state of the art machinery. It is important to create efficient software that is compatible with older computers to increase availability to a wider group of people.

The development of the software is also restricted by any accompanying hardware or software that it must be packaged with. Without MATLAB installed, the Speech Recognition application will not work. The program was limited to displaying four Braille letters due to hardware limitations; a more advanced program can be done to display sentences or even paragraphs. Although it may not be the most efficient means of teaching Braille to newer generations, this project should hopefully increase the awareness and importance for accessibility of media for people with disabilities.
Appendix A: English and Braille Letters

Figure A.1: English and Braille lowercase letters

Figure A.2: English and Braille numbers
Figure A.3: English and Braille Punctuation

- Apostrophe ‘
- Exclamation !
- Close Quotations “
- Brackets ( or ) *Placement dependent
- Hyphen -
- Semicolon ;
- Capital letter follows
- Period .
- Comma ,
- Question Mark ?
- Open Quotations “

*Question mark and open quotations use same character but depend on placement in sentence.
References


