**Morse Pi Project**

From: Andrew M. & Gozie A.

Discipline: Computer Engineering Technology  
Date: 2/26/2018

# Declaration of Joint Authorship

We Chigozie Aham and Andrew McGuire, hereby declare that this thesis and the work presented in it is entirely our own. Where we have consulted the work of others, this is always clearly stated.

Signed:

Date:

### Approved Proposal

## Executive Summary

As students in the Computer Engineering Technology program, we will be working together to integrate the android software and hardware component of the Morse-Pi project. This proposal requests the approval to integrate the hardware portion that will connect to a database as well as to a mobile device application. The database will store Morse messages and its equivalent translations. The mobile device functionality will provide a more interactive user interface to view data from the hardware functionality.

## Background

The problem solved by this project is to enable ease in the use of Morse code by radio communicators to transmit messages back and forth. This project will facilitate the protection of information by encoding it as well as ensuring ease in decoding it by its listener. Morse code is a predated communication tool used to represent text information in a series of electronic pulses, usually represented as a short pulse (called a "dot") and a long pulse (a "dash"). Only skilled listeners are able to translate and understand Morse codes.

Existing android software products on the market include [1] Morse Transmitters.

In the Computer Engineering Technology program, we have learned about the following topics from the respective relevant courses:

* Java Docs from CENG 212 Programming Techniques In Java,
* Construction of circuits from CENG 215 Digital And Interfacing Systems,
* Rapid application development and Gantt charts from CENG 216 Intro to Software Engineering,
* Micro computing from CENG 252 Embedded Systems,
* SQL from CENG 254 Database With Java,
* Web access of databases from CENG 256 Internet Scripting; and,
* Wireless protocols such as 802.11 from TECH152 Telecom Networks.

This knowledge and skill set will enable us to integrate the hardware and software components.

**Methodology**

This proposal is assigned in the first week of class and is due at the beginning of class in the second week of the fall semester. My coursework will focus on the last two of the 3 phases of this project:  
 Phase 1 Hardware/Software build.  
 Phase 2 System integration.  
 Phase 3 Demonstration to future employers.

*Phase 1 Hardware/Software build*

The hardware and software build has been 80% completed in the past fall term.

*Phase 2 System integration*

The system integration will be completed this winter term.

*Phase 3 Demonstration to future employers*

This project will showcase the knowledge and skills that we have learned to potential employers.

## Concluding remarks

This proposal presents a plan for exploring outdated communication technologies to improve the contemporary technologies. We request approval of this project.

# Abstract

In this project, outdated communication medium was implemented on newer technologies; the solution of encoding messages using Morse code was explored. The Android and Raspberry-pi devices were platforms used to implement this functionality. Exchange of messages between two different parties are possible in 2 different ways:

· From one raspberry-pi to another via direct connection using IP address

· From one raspberry-pi to an Android device via the cloud using hostgator database.

Using two raspberry-Pi devices, the receiving party is only able to view sent messages. Using the raspberry-pi and the Android device, messages could be viewed and sent between the receiving and sending party. The Morse-pi project combines the use of obsolete and current technologies so it promises an even better and secure communication.

**Table of Contents**

[Declaration of Joint Authorship 2](#_Toc512067133)

[Approved Proposal 3](#_Toc512067134)

[Executive Summary 3](#_Toc512067135)

[Background 3](#_Toc512067136)

[Concluding remarks 4](#_Toc512067137)

[Abstract 5](#_Toc512067138)

[Illustration List 8](#_Toc512067139)

[1. Introduction 9](#_Toc512067140)

[2. Project Description 9](#_Toc512067141)

[2.1 Problem 9](#_Toc512067142)

[2.2 Rationale Behind Project 10](#_Toc512067143)

[2.3 Project Scope 10](#_Toc512067144)

[2.4 Software Requirement Specifications 11](#_Toc512067145)

[2.5 Project Overview 14](#_Toc512067146)

[2.5.1 Bill of Materials 14](#_Toc512067147)

[2.5.2 Time Commitment 15](#_Toc512067148)

[2.5.3 Mechanical Assembly 15](#_Toc512067149)

[2.6 Problems Encountered 15](#_Toc512067150)

[2.6.1 Inconsistent Input 15](#_Toc512067151)

[2.6.2 Debounce Fix 16](#_Toc512067152)

[2.6.3 LCD Touchscreen Failure 16](#_Toc512067153)

[2.6.4 Data Communication 16](#_Toc512067154)

[2.6.5 App Data Fetching 17](#_Toc512067155)

[2.8 Walkthrough of System 17](#_Toc512067156)

[2.8.1 Microcontroller 17](#_Toc512067157)

[2.8.2 Server 17](#_Toc512067158)

[2.8.3 Phone Application 17](#_Toc512067159)

[3. Progress Reports as per emails to Austin Tian 17](#_Toc512067160)

[3.1 Report 1 17](#_Toc512067161)

[3.2 Report 2 20](#_Toc512067162)

[3.3 Report 3 21](#_Toc512067163)

[4. Build Instructions 23](#_Toc512067164)

[Cost 23](#_Toc512067165)

[Time 24](#_Toc512067166)

[Assembly 24](#_Toc512067167)

[Configuration 25](#_Toc512067168)

[Run 27](#_Toc512067169)

[Database build 27](#_Toc512067170)

[5. Recommendations 28](#_Toc512067171)

[6. Technical References 29](#_Toc512067172)

[7. Appendices 29](#_Toc512067173)

[7.2 Website Code 7.2.1 Main Page 7.2.2 Data Page 7.2.3 Conversation Page 7.2.4 Login Page 7.2.5 Account Page 7.3 Android Phone Application 7.3.1 Main Menu 7.3.2 Current Data 7.3.3 Morse To Text 7.3.4 Text to Morse 7.3.5 Morse Letters 7.3.6 Morse Graph 7.3.7 Channel Page 7.3.8 Selected Channel Page 7.3.9 Pi Morse History 7.3.10 Style 7.3.11 Text 7.3.12 Login Layout 7.3.13 Morse To Text Layout 7.3.14 Text To Morse Layout 7.3.15 Channel Selection Layout 7.3.16 Channel Layout 7.3.17 Morse Instructions Layout 7.3.18 Settings Layout 30](#_Toc512067174)

# Illustration List

2.5.3a Circuit layout for supplying power to pump. p.5

2.5.3b Circuit layout for sensor connections to microcontroller board. p.6

2.5.3c Finished prototype in acrylic case. p.7

2.5.7a Components in acrylic case before PCB and dividers. p.9

# 

# 1. Introduction

The purpose of the Morse-pi project is to allow for a secure communication between two end-users. The problem solved by this project is to enable ease in the use of Morse code by radio communicators to transmit messages back and forth. This project will facilitate the protection of information by encoding it as well as ensuring ease in decoding it by its listener. Morse code is a predated communication tool used to represent text information in a series of electronic pulses, usually represented as a short pulse (called a "dot") and a long pulse (a "dash"). Only skilled listeners are able to translate and understand Morse codes. The theory behind the project is to apply the rigidity of older technologies and the flexibility of newer technologies to achieve a next-level style of communication.

# 2. Project Description

The Project will incorporate the use of two raspberry pi 3.0, a contact key and a Adafruit TTPT 3.5” and a 2.8” touch screen display. The pi’s will connect via the internet and will use the contact keys as input into the system and be recognized as Morse that will displayed for the users in text.

## 2.1 Problem

The problem the project solves is the lack of a current physical learning tool for Morse code learners. This project provides a solution that those interested in learning Morse code can set-up and use to assist them in learning Morse with an environment that allows them to send actually messages in Morse and check that the messages they are sending are correctly interpreted.

## 2.2 Rationale Behind Project

The rationale behind the project is that it incorporates many of the skills learned throughout the Computer Engineering Technology course at Humber. This project is a testament to what has been learned throughout the course and displays our skills in: troubleshooting, coding, project management, system design, and working in a team environment. The project also displays our knowledge of: programming, databases, network programming, control systems/microcontrollers, analog to digital systems, and system configuration.

## 2.3 Project Scope

The scope of this project involves the connectivity of two raspberry Pis, a website, a database and a app. The raspberry Pis have different code in order to establish a network connection between them and so they are able to connect over the internet, and the Pis then connect to the database in order to send data to it. The database will have two tables that will be used to store user information and information that has been sent between the Pis. The app will have both an offline and online function. While online the app will be able to access the data from the database in order to be displayed upon the app in both text and in mores in order to help improve the users understanding. The app interfaces with the database via a PHP server that fetches information from the database in order for the app to be able to easily access it. The website will connect to the database and be able to fetch data and modify and remove messages in order to grant user control.

## 2.4 Software Requirement Specifications

* **Overall Description**
* **Product Perspective**

This project is a self-contained system that allows users to learn and use Morse code to communicate over a network connection. The system requires a remote database that will allow users of the android app to view messages that they have sent via the raspberry pi.

* **Product Functions**
* Reads signal from Morse key
* Translate signal to text
* Creates sound for user feedback
* Transmits messages to pis and cloud
* Android app listens to messages for learning experience
* **User Classes and Characteristics**

Pi users, will have the Morse key and will be using the Morse Pi, the Pi users will need to be able to send messages via Morse code and be able to hear what they are typing and make sure it is correct based on that as they do not see their own messages. This is the most important user experience as they will actually be learning how to Morse with a Morse key rather than with the app that lets then click dahs and dits to make their messages.

App users, will be using the app and some will be able to log in and see the messages transmitted from the Pis. This user will experience a simpler version of Morse as they won’t have to be skilled to create a Morse messages and then use the translator to get the text.

The Web users, will be like moderators they will see all the messages sent from the Pis in the database, and be able to edit and modify as needed.

* **Operating Environment**

The system will operate on three hardware platforms, two raspberry pis 3.0, and an Android phone that supports API 26 or newer, and a database which will be hosed on the cloud. The raspberry Pi will be operating in Python code and the Android App will be created with Java and xml in android studios. Both the Python code and the Java code will have connected to the database on the cloud, the Pi will send data to the database and to the second Pi and the Android App will fetch data from the database to display to the users, as well as use the database to insure login credentials, as not all users will be able to login and see the messages from the Pis.

* **Design and Implementation Constraints**
* User must have an internet connection for most of the functionality to work
* Low project budget so damaging part is NOT AN OPTION!
* Time constraint of 10 weeks to get system fully functional with all components working
* **User Documentation**

User manual, as well as build instructions will be provided (Link not yet available)

* **Assumptions and Dependencies**

API level 26 is assumed to be what the mobile phone will be running however the app does support a minimum of API level 23. The python compiler is also assumed to be 3.6.1.

* **External Interface Requirements**
* **User Interfaces**

The user interface that will need work/tweaking is the interface for the app. The app will display messages for the User in both Morse and in the text translation of the mores. The app will display the message one word at a time with the Morse above the text, this is done for readability so that the user can help to understand which bit of Morse represents what words.

* **Hardware Interfaces**

The hardware will connect to the Pi via its GPIO header which the Pi will be taking input from for the GPIO header.

* **Software Interfaces**

The Python’s code server/client code for the Pi will interface with a secondary python file that handles the translations from the dots/dahs to text. The App has classes dedicated for the Text to Morse translations, Morse to Text translations and a Morse to Sound.

* **Communications Interfaces**

The Pi and the app will connect to the database over the internet via a TCP connection in order to fetch and push data. The primary and secondary Pi will connect via TCP in order to transmit their messages directly. The Web interface will operate over http and allow the user in order to fetch and modify data within the database.

* **System Features**

The main functionality of the project is the translated text from the Morse that is read from the signal from the Morse key.

* **Reading the Morse Signal**
* Description and Priority

This is the highest priority feature. This feature is the translation of the signal from the GPIO header to Morse, without this feature the project can’t be done.

* Stimulus/Response Sequences

User begins to enters in a Morse letter on the first pi, user feedback should be the audio from the Pi letting the user know what they are inputting.

* Functional Requirements

User keys in Morse letter on the contact key and hears instantaneous response from the Raspberry Pi. Any none Morse keys enters are ignored and after a brief moment a space in made of the second display and the primary Pi is able to start typing in the next Morse letter.

* **Transmitting the Message**
* Description and Priority

This feature is the second highest priority feature, without this feature the system will not be able to use the database or connect to the secondary Pi. This feature allows the Pi to transmit the messages to the secondary Pi and the database.

* Stimulus/Response Sequences

After The user enters a full Morse letter, the second Pi (to display the letter) and the Database should receive a message (a full word) after it has been entered,

* Functional Requirements

The Pis must connect to each other and the database via TCP in order to transmit the messages.

* **Viewing the Message on the App**
* Stimulus/Response Sequence

The app will query the database for messages and refresh at an appropriate rate in order to display messages that have been sent by the Pi to the database. This is the lowest priority feature of the project but still one that is required for the end product.

* Stimulus/Response Sequence

The user will navigate to the Pi monitor section of the app and be prompted to login. Once proper login credentials have been entered the user will be able to view all of the messages that the Pi’s have sent into the database and it will appear like a text conversation between both Pis. The user will be able to scroll through the messages to view messages that have been sent recently or messages that have been sent in the past.

* Functional Requirements

The app must query the database and then either refresh manually or periodically in order to receive new messages

### 2.5 Project Overview

The goal of Morse-Pi v1.0 project allows for communication between two different parties using Morse code. Encoding of a message helps to ensure content protection, so only skilled users are able to do its translation. Messages could be transmitted directly between two raspberry pis or over the cloud between one raspberry pi and then viewed on the Morse-pi android app. The user is able to view the original Morse code as well as its equivalent translation from the android app.

## 2.5.1 Bill of Materials

The cost of the materials varies but the minimum requirements include a Raspberry Pi 3.0 and some wire and a contact key.

This brings the effective minimum bill of materials to ~$45 per communicating system, so ~$90 for two systems that can communicate.

The Build with an Adafruit Pi-TFT 2.8” is priced around ~150 per system for and effective ~$300 for two systems that can communicate.

## 2.5.2 Time Commitment

This project takes around 1 -5 hours to complete break down as follows   
  
~1-5 hour – Raspberry Pi initial configuration (this includes all required installations and burning the image onto an SD for the Pi.)

~ 30 Minutes – Soldering header onto an Adafruit 3.2” (if used)

~5 Minutes – Download required software from github.com/goziethelegion/MorsePi

~5 minutes – Hardware set up

## 2.5.3 Mechanical Assembly

Assembly may be required depending on which Adafruit Pi-TFT display is used. Instructions on how to assemble an Adafruit Pi-TFT kit can be found on Adafruit’s website, instructions may also be provided with purchase.

## 2.5 Problems Encountered

This project has not been without its own obstacles and challenges. This section will describe what problems have occurred and solutions that have been used to overcome issues.

### 2.5.1 Inconsistent Input

This problem occurred in the input signal from the Morse Key to the Raspberry Pi. The contact key is essentially a button that when presses the Raspberry Pi detects input otherwise when it isn’t pressed it doesn’t read a signal i.e button isn’t pressed. The problem occurred when the user pressed down the button unevenly and so it caused the Pi to read multiple inputs at once, even though the button could be held or merely pressed. This problem also came with another problem, which solution would be used. Two options were available either a hardware fix or a software fix. At first we decided to use a hardware fix to our issue, in order to demonstrate our ability to design a PCB and implement it into the system. This was not without its own issues, we prototyped the circuit design on a breadboard and it did slightly resolve the debounce issue, but it caused another issue of holding the signal longer than intended so the input was often mistaken for a dash when a dot was intended. Our next prototype we designed we had Austin help us address the issues that were persistent before and so we were able to get it working perfectly. The only reason we decided to avoid using this solution was cost, we felt as though adding to the cost of the project wasn’t in the best interest of the project and so we went with a software solution.

### 2.5.2 Debounce Fix

This Section here will go over the specifics of the solution and how they addressed the issue stated in the above section.

The software solution we ended up implementing was fairly simple when broken down. The issue was that on the input side when the button was pressed slight nano second breaks in signal would occur when the button was pressed and when the button was held. So the solution was fairly simplistic and required minor changes to the input function of the code. The solution involved testing if the input was long enough for a dash, in the event that it was, send dash to the buffer that would be used to decode, and in the even if it wasn’t, test if the input was long enough to be a short break, and so we tested if it was at least 0.01 seconds long or in other words at least 10 milliseconds long which is prefect as when the user just pressed the button quickly for a dot it averaged around 50 milliseconds on the input.

### 2.5.3 LCD Touchscreen Failure

The issue here is with the 3.2” Adafruit Pi-TFT because soldering was required. I found that the display often disconnected a ribbon cable at the back of the display, which was a simple fix of just reconnecting it, however a bit of trouble shooting was required as I originally thought it was an issue with the Pi and the Pi’s image. The second LCD touch screen failure issue occurred after soldering the headers onto the LCD board. The LCD board uses a capacitive touch screen and like all capacitive systems uses capacitors which can pop when exposed to high temperatures such as from a soldering iron. When heating the board so that the solder would adhere to it better, I managed to over heat a capacitor when the heat travelled up the copper trace and so the touch capabilities of the touch screen no longer function. This issue would of caused a lot of increased cost if we had decided to create a graphical user touch experience instead of the Morse contact key, however we decided to stay with the Morse contact key as it gave a more authentic Morse experience for the user.

### 2.5.4 Data Communication

We had minor data communications issues between the Pi’s python code and the database and between the app and the database (App’s issue will be discussed below). The issue we were having was a run time error that was saying our database couldn’t authenticate and wasn’t able to get any data from the database. This issue persisted for a few weeks as we tried to troubleshoot it, eventually we found out that the service that we were using did not allow for remote connections to the database, and so we decided to make a change to our database. We ended up using a remote VM service that allowed us to create our database on it and connect to it whenever we needed without limit. Once we ported our database over to the VM we were then able to connect to with minor tweaks in the Python code and minor updates to the Pi’s Python libraries.

### 2.5.5 App Data Fetching

The issue here occurred between the Database and the App. Our App was unable to correct fetch and display the database’s content for a while. We used PHP to parse the database data into a JSON object that the App would translate into the conversation. Our issue was with our PHP script. For two weeks we tried tweaking the app’s fetching of the JSON object and its manipulation. Once we went over the PHP script, we found out that there was a slight issue with our loop that fetched each row from the database, our issue was that we reinitialized an array instead of the loop so that instead of adding the new row into our array what we mistakenly did was create a new array each time with the next element from the array. Once we found this out we fixed the loop so that it no longer reinitialized the array and went the JSON object array correctly, after that we went back to our original code for fetching and manipulating the array and it worked without issue.

## 2.7 Walkthrough of System

This section here will provide and overview of the functionality of each component within our system and what each part is responsible for the purpose of data flow.

### 2.7.1 Microcontroller

The microcontroller, i.e our Raspberry Pi which for this project was used like a Microcontroller, was our data input to our system as well as data output for our system. The Raspberry Pi took an incoming signal from the Morse Contact key which is our data input into the system. The Pi translated the Signal into Morse dots and dashes which were passed to a thread that handled decoding the dots and dashes into characters. Once the Morse was decoded to a character, the decoded character was sent across the network, or internet if the Pi’s were not in the same network and then displayed on the Raspberry Pi’s display (data output). After the character was displayed the same character was then transmitted to the database for further use in the system.

### 2.7.2 Server

Our server hosted PHP scripts that read data from the database and parsed it into a JSON object array (data control). The script connected to our database which was hosted by a different service than our server and then inside of its loop queried our conversation table for each row, on completion of the query the PHP then displayed the entire JSON object array for the App to read from.

### 2.7.3 Phone Application

Our phone application has some functionality that is separate from our project. The part of the app that is a part of our system is the ability for it to read all of the messages that have been sent between the Raspberry Pis. It fetches the data from the PHP server

### 3. Progress Reports

## 3.1 Report 1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Progress    We have successfully tested the PHP scripts on the database level for logging in an existing user. We manually created only two users for practical communication. There are two of PHP scripts dedicated to signing in each user (Gozie and Andrew).  So far Gozie and I have made progress on the Morse Pi project. Together, Gozie and I created a prototype for our debounce circuit that we will be using to fix an issue that the hardware has with data input. Separately I have been working in fritzing in, so that we can make the final debounce circuit that will work with our project and I have also been helping Gozie plan out the exact set up for the database. Gozie has been working on the database and how the python will interact with the database from the raspberry pi.  We have followed database designs to allow for communication between two users similar to Facebook’s. Some of the tables found in the link have been successfully created. The general idea behind the design is to create a database that is able to save both the Morse codes and its English translation based on the user’s session.  <https://www.9lessons.info/2013/05/message-conversation-database-design.html>   |  |  | | --- | --- | | [https://lh3.googleusercontent.com/-q7nP4lSJ4tg/Vim1CZJDWvI/AAAAAAAANIU/vE0RZ8-9rCE/s599-Ic42/9lessonsLogo.png](https://www.9lessons.info/2013/05/message-conversation-database-design.html) | [Facebook Style Messaging System Database Design.](https://www.9lessons.info/2013/05/message-conversation-database-design.html)  www.9lessons.info  This post explains you how to design the Facebook Style message conversation system using PHP and MySQL. I have been working with messaging system labs 9lessons |       Problems    One of the problems we've been experience in our project is our debounce issue.  <https://www.embedded.com/electronics-blogs/break-points/4024956/Solving-Switch-Bounce-Problems>   |  |  | | --- | --- | | [https://m.eet.com/images/edn/images/icons/contentitem-default.png](https://www.embedded.com/electronics-blogs/break-points/4024956/Solving-Switch-Bounce-Problems) | [Solving Switch Bounce Problems | Embedded](https://www.embedded.com/electronics-blogs/break-points/4024956/Solving-Switch-Bounce-Problems)  www.embedded.com  No chaste unblemished logic signals these; a scope will show the contacts torturously bouncing before settling into a stable state. |      |  |  |  |  | | --- | --- | --- | --- | |  | [Solving Switch Bounce Problems | Embedded](https://www.embedded.com/electronics-blogs/break-points/4024956/Solving-Switch-Bounce-Problems)  [www.embedded.com](http://www.embedded.com/)   |  |  | | --- | --- | | [https://m.eet.com/images/edn/images/icons/contentitem-default.png](http://www.embedded.com/) | [Embedded](http://www.embedded.com/)  www.embedded.com  Embedded.com is a leading source for reliable Embedded Systems development articles, tech papers, webinars, courses, products, and tools. Visit Embedded.com for the latest information on Embedded Development industries, insights, and educational resources. |   No chaste unblemished logic signals these; a scope will show the contacts torturously bouncing before settling into a stable state. |   Above is a link that we have used to help us create a circuit that will allow us to mitigate this issue and continue working on integration of the system.    The other problem we have been having is connecting our raspberry pi to our database. Our database is hosted by a website called hostgator and it is a SQL database. Our issue here is our lack of knowledge of how to connect to it via python, Gozie connected to this database via his app for Software project however with python the connection does not seem to be working. Gozie has been looking into this issue so that we can communicate to the database.    Here is a link we used to get a library on the raspberry pi to help connect to the Db, unfortunately it is still not connecting.  <https://thepythonguru.com/installing-mysqldb/>    Here is a links that were used to help troubleshoot  <https://www.youtube.com/watch?v=OlwPEHgg_bM> <https://dev/mysql.com/doc/mysql-windows-excerpt/5.7/en/resetting-permissions-windows.html> <https://stackoverflow.com/questions/21944936/error-1045-28000-access-denied-for-user-rootlocalhost-using-password-y>   |  | | --- | | [https://cdn.sstatic.net/Sites/stackoverflow/img/apple-touch-icon@2.png?v=73d79a89bded](https://stackoverflow.com/questions/21944936/error-1045-28000-access-denied-for-user-rootlocalhost-using-password-y) | |  |
|  | |
|  | |

## 3.2 Report 2

|  |  |
| --- | --- |
| Progress:  On the android app, work is currently ongoing on the design for displaying the database contents. The design layout is all ready, it just has to be implemented to be in sync with the java functionalities. Before the weekend, this should be completed and it would conclude work on the Android application.  Different attempts at debouncing have been tried in order to find the best usable fix as well as one that fixes the original issue. How data has been handled has also been tweaked in order to better communicate with the server.  Financial Update:  Please find the budget attached. So far, there has been no added costs. |  |
|  | |
|  | |

## 3.3 Report 3

|  |  |
| --- | --- |
| Progress  Connection to the database has been successfully established and data is being shared directly from one raspberry pi to another.  The previous error with the Android application has been fixed. The Android application works as expected.  Problems  Initially, a PCB was proposed to be designed to address the debouncing problem. After careful consideration, a software solution was preferred. The first attempt to the software solution has not proven successful. Another solution is in works.  Documentation is still not updated. The documents exist but hasn't been added to the overall technical report because of overloaded and mixed schedules. This would be addressed as soon as possible.  Financial update  $80 was saved from switching from designing the PCB to the software. Please find it attached. |  |
|  | |
|  | |

|  |  |
| --- | --- |
|  |  |
|  | |
|  | |

|  |  |
| --- | --- |
|  |  |
|  | |
|  | |

# 4. Build Instructions



### 4.1 Cost

The cost of this project varies, its possible to do this project with a single raspberry pi however the full functionality is designed for two.

Here's what you'll need

PiTFT plus 320x240 3.2" cost $39.95CAD Canakit Raspberry PI 3 complete Starter Kit 32 GB edition cost $99.99CAD Eisco Labs Contact Key, Telegraphing/Morse Code, Single cost $9.77CAD

total cost after taxes $169.18CAD(338.34)

Other things you'll need

Some wire and cutters soldering iron and solder (Don't forget your safety glasses) jumper wires two cables with an alligator clip on one end and a banana clip on the other Keyboard, (No mouse required the touch screen handles that well) And last but not least if you are trying to connect the two raspberry pis you'll need a internet connection.

### 4.2 Time

This project shouldn't take more than a few days assuming every day half an hour to and hour is spent

### 4.3 Assembly

Assembly of the project is fairly straightforward however some soldering is required. WARNING purchasing the pitft as a kit will require some more soldering, here is a link that will assist in guiding you through that. <<https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/assembly>

1. So if you noticed the touch screen had 8 component holes, here you'll need to solder wire, or what I suggest clipping part of a resistor and soldering two of those into the component holes labeled with #27 on the bottom right. Note be careful with soldering these leads on as if you heat the component hole too much it is possible to damage the touch screen, speaking out of personal experience here.
2. after the component has cooled down connect jumper wires to the two pins, then connect the alligator clip to the one end of the jumper (some wire required first) and then the banana clip fits perfectly to the Eisco Contact Key.
3. Next part is simple you only need to connect the touch screen to the raspberry pi the screen should fit over top of the pi in a way that if covers the pi, If the screen looks like it hangs off the bottom of the pi you have the screen upside down so turn it around and plug it in again. If any of the pins of the touch screen is not connected it is not connected correctly make sure all pins are connected and the orientation of the screen is correct.
4. Here is what it should look like



### Configuration

Before plugging in the sd card that comes with the raspberry pi there are some steps that need to completed before testing things out.

1. first step is image the raspberry pi, in order to do this download the image here <https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/easy-install>
2. If you have software to burn an image on the SD card you can skip this step. After you have downloaded the image you'll need to be able to burn it onto the SD card some software is required. Etcher is the software that I used to burn the image onto my SD card, however if you have other software you can use that but for those who don't here is the link to download Etcher, after downloading go ahead and install <https://etcher.io/>
3. If your SD card is formatted to exFat or you can format it yourself to exFat skip this skep. Etcher is fortunately fairly simple to use however before just running it and burning the image you'll need to make sure the SD is formatted to exFAT there is software somewhere out there in the internet here is a link to what I used, download and install and format your SD card to exFat <<https://www.sdcard.org/downloads/formatter_4/>
4. Now after the SD formatter is installed, connect the SD card with the SD card reader that comes in the raspberry pi kit. Make sure that when using the formatter to format the SD card to verify that is formatted to exFat or the card may can fail to have the imaged burned to it.
5. After formatting the SD card next is to use Etcher to burn the image to the card, and back to the point I made two steps ago, Etcher is fairly straightforward to use. First you select the image that you will burn, this will be the file unzipped that was obtained in step 1 after selecting the image, next make sure you select your SD card. The last thing to do here is to click Flash!, this will take a while so you might as well go make some tea or whatever.
6. Now you can plug in the raspberry pi and start it up. If the Pi-TFT display displays bright white and remains in that state for longer than 5 minutes its possible something has gone wrong with the installation, if that is the case verify that you have installed the correct image and restart the process.
7. Now that the Raspberry pi has started there are a few things you'll need to do. First you need to go the Raspberry pi configuration. To navigate to it you can click on the start menu -> Preferences -> Raspberry Pi Configuration, while in here you'll want to change a few things set up for the pi. In the system tab you can change your password and host name for your raspberry pi(This is a safety option).Next go to the Interfaces tab, in here you will want to enable SSH , VNC , PSI, and I2C. SSH and VNC are just to allow you to use other tools in order to access the PI remotely, I will be going over the SSH use but the VNC use isn't necessary.
8. Now that your Pi is configured you need to get the software to run everything, here are a few options of how you can get that done. After getting an internet connection on the raspberry Pi you can navigate to its internet browser and go to <<https://github.com/Joemiddle/TouchPi-CENG317/tree/master/software> here you will see three important python files(They are the ones that end in .py) download them all on the raspberry pi. After downloading them to the raspberry pi you can move them from the downloads folder to another folder at your own discretion However for ease of use I suggest making a folder in the root of the user and then running the python code in that folder. The other way of getting these files onto the raspberry pi is the putty way on your computer not the pi. first download and run putty link below <<http://www.putty.org/> after you download putty you'll still need to establish an internet connection on the pi. Now on the Pi you need to go to the a terminal window by clicking on the start menu-> Accessories -> Terminal. Once in the terminal type in ifconfig which will let you find the ip address of the raspberry pi, if you are connecting to a local private network it will probably be something along the lines of 192.168.0. 1 to 255 after that you will type that into putty as the Host Name (or IP address) and click open at the bottom it will prompt you for username / password so make sure you know it if you left it as the default it will be pi / raspberry. Now on your computer you will want to open the python files in notepad or notepad++. Copy the content of the file and paste it into the same file name, you can make a file and start editing it with the command sudo nano (filename). so and example of it is to copy the content of the file clientCode.py and then on the pi create that file and start editing it with "sudo nano clientCode.py" all in one line. Now you just paste what you copied from clientCode.py to save you can press CRL + X and then y and then enter. Repeat this process for all the files codeServer.py and tcpmorse\_lookup.py. Now the Pi should be ready.
9. This step is the longest process... You've got to configure the second raspberry pi, the fortunate part to this part is you've done it all before, after you've finished the second pi you're done setting everything up and its onto testing.

### 4.5 Run

To run you need to the ifconfig on both of the Pis in order to get their IP addresses, make sure to make a note of it. Now you need to make one pi the server and the other the client, it doesn't matter which one is which but you need to type in different commands for each

The server needs to run its command first sudo python3 ./codeServer.py

The client can then run its code with this command sudo python3 ./clientCode.py

After that both screen should display ready and if both have correctly connected you will be able to send a message to the other pi by typing in Morse. The pi you send the message on will be able to hear the mores however the won't see the message, the message will display on the other pi. And with that it's done!

I hope you've may have learned a bit about the Raspberry Pi running through this process or maybe a bit if you've looked at the python code however that's all folks!

### 4.6 Database build

The database functions with two tables. The first table contains information about users, that has three fields: Username, UserId (Primary Key, auto\_increment), and password(hashed). The other table is the conversation table and it has 4 fields: MessageId(primary key, auto\_increment), sender(foreign key), receiver(foreign key), and data (stored in text, not Morse). The database is on its own private SQL server and is accessed by the raspberry pi and the android app in order to fetch and upload data.

# Recommendations

We recommend for when building this project to enable SSH on the raspberry Pi in order to easily configure it. The use of putty is also recommended as it’s a good tool that allows for using SSH into the Pi. VNC view can also be used when the Pi is completely configured to verify that the code can run on the Pi. Another recommendation is to use the 2.8” Adafruit Pi-TFT display as its build does not require soldering.

# 6. Technical References

# 7. Appendices

# 7.1 Android Phone Application 7.1.1 Main Menu 7.1.2 Current Data 7.1.3 Morse To Text 7.1.4 Text to Morse 7.1.5 Morse Letters 7.1.6 Morse Graph 7.1.7 Channel Page 7.1.8 Selected Channel Page 7.1.9 Pi Morse History 7.1.10 Style 7.1.11 Text 7.1.12 Login Layout 7.1.13 Morse To Text Layout 7.1.14 Text To Morse Layout 7.1.15 Channel Selection Layout 7.1.16 Channel Layout 7.1.17 Morse Instructions Layout 7.1.18 Settings Layout

**7.2 Python Code**

# 