Technical Report

for

Trippie

Version 1.0 approved

Prepared by Ravneet Singh, Gaganpreet Singh Grewal and Bojan Lazic

February 9, 2018

Declaration of Joint Authorship

We, Ravneet Singh, Gaganpreet Grewal and Bojan Lazic, hereby declare that all the content in this technical report titled “Software Requirements Specification for Trippie” is our own joint work.

* All content from external sources are clearly stated and cited.
* The following report is a joint work done by the team members.
* All contributions made by others are acknowledged. It is clearly shown what has been done by the team, and anything used externally is quoted and sourced.
* This report is intended for academic purposes for Humber college.

Approved Proposal

2018-02-05

***Proposal for the development of Trippie***

Prepared by Gaganpreet Singh Grewal, Ravneet Singh, and Bojan Lazic  
*Computer Engineering Technology Students*

https://github.com/princess97/TripPlanner\_V1

**Executive Summary**

As a student in the Computer Engineering Technology program, I will be integrating the knowledge and skills I have learned from our program into this Internet of Things themed capstone project. This proposal requests the approval to build the hardware portion that will connect to a database as well as to a mobile device application. The internet connected hardware will include a custom printed circuit board with the following sensors and actuators Temperature Sensor, Gps Sensor and Stepper Motor. The database will store The database will store the different temperature readings, locations already visited by the user and locations which user wants to visit in future.. The mobile device functionality will include Mobile application will easily analyze the data collected by the sensor and it will also show the temperature of the particular location which user wants to visit. For example, If the temperature will be above or below certain range mobile application will alert the user, not to visit that place. and will be further detailed in the mobile application proposal. I will be collaborating with the following company/department Humber Tech Group, Humber Parts Crib, Prototype Lab. In the winter semester I plan to form a group with the following students, who are also building similar hardware this term and working on the mobile application with me Gaganpreet Grewal(N01139945), Ravneet Singh(N01148757) and Bojan Lezic(N01109108). The hardware will be completed in CENG 317 Hardware Production Techniques independently and the application will be completed in CENG 319 Software Project. These will be integrated together in the subsequent term in CENG 355 Computer Systems Project as a member of a 2 or 3 student group.

**Background**

The problem solved by this project is Most of the people in this world like to travel different places. Sometimes they visit a place which is totally unknown to them, so they need a guide who helps them to explore that place. In return, they have to pay the guide. This increases the buget of the person.. A bit of background about this topic is In order to save money we are working on a project which will help people to explore different places in the world easily and free of cost. Gaganpreet Singh is working on a temperature sensor which will give the reading of temperature of the environment. Bojan Lezic is working on a stepper motor which will be attached with a plastic shelter to protect the delicate hardware of the project. Since this machine is going to be at many different places, the GPS sensor will come in handy to distinguish where each raspberry pi will be located. Ravneet Singh is working on the GPS sensor..

Existing products on the market include [1]. I have searched for prior art via Humber’s IEEE subscription selecting “My Subscribed Content”[2] and have found and read [3] which provides insight into similar efforts.

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 me to build the subsystems and integrate them together as my capstone project.

**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 first two of the 3 phases of this project:  
 Phase 1 Hardware build.  
 Phase 2 System integration.  
 Phase 3 Demonstration to future employers.

*Phase 1 Hardware build*

The hardware build will be completed in the fall term. It will fit within the CENG Project maximum dimensions of 12 13/16" x 6" x 2 7/8" (32.5cm x 15.25cm x 7.25cm) which represents the space below the tray in the parts kit. The highest AC voltage that will be used is 16Vrms from a wall adaptor from which +/- 15V or as high as 45 VDC can be obtained. Maximum power consumption will be 20 Watts.

*Phase 2 System integration*

The system integration will be completed in the fall term.

*Phase 3 Demonstration to future employers*

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

The brief description below provides rough effort and non-labour estimates respectively for each phase. A Gantt chart will be added by week 3 to provide more project schedule details and a more complete budget will be added by week 4. It is important to start tasks as soon as possible to be able to meet deadlines.

Raspberry Pi 3 starter kit, Temprature Sensor(DS18B20), GPS sensor, Stepper Motor, Jumper wires(Male - Female, Female - Female, Male - Male)

**Concluding remarks**

This proposal presents a plan for providing an IoT solution for This is an opportunity for us to integrate many things to create a collaborative project which will help users to plan and visit the different places in the world without the help of any guide and in bonus it will tell the temperature of that place which user wants to visit. This will reduce the budget of the user and save lots of time.. This is an opportunity to integrate the knowledge and skills developed in our program to create a collaborative IoT capstone project demonstrating my ability to learn how to support projects such as the initiative described by [3]. I request approval of this project.

**References**

[1] N/A

[2] Institute of Electrical and Electronics Engineers. (2015, August 28). IEEE Xplore Digital Library [Online]. Available: https://ieeexplore.ieee.org/search/advsearch.jsp

[3] N/A

Executive summary

The following technical report will be on the software requirements for the *TripPi*. This device is a multi-sensor which is stationary at a set location, which works with our app TripPlanners. The report will cover the requirements for the device as well as the app, and will go over the specifications while providing diagrams for the application. This report is significant as it provides insight on all the needs for the project.

Table of Contents

Executive summary v

Table of Contents vii

Revision History viii

1. Introduction 1

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 System Overview 1

2. Overall Description 3

2.1 Product Perspective 3

2.2 Product Functions 3

2.3 Operating Environment 3

2.4 User Documentation 3

2.5 Mobile Application Requirements 3

2.6 Database Requirements 5

6

2.7 Hardware Requirements 6

2.8 Programming Knowledge 6

3. Developmental Software 7

3.1 Designing Applications 7

3.2 Coding Applications 7

4. Raspberry Pi Preparations 8

4.1 Raspberry Pi Image 8

4.2 WiringPi 8

4.3 Enabling the One Wire Interface 9

4.4 Installing GPSD on the Raspberry Pi 9

5. Build Instructions 10

5.1 Introduction 10

5.2 System Diagram 10

5.3 Budget 11

5.4 Time Commitment 11

5.5 Mechanical Assembly 12

5.6 Unit Testing 13

5.7 Production Testing 15

5.8 Printed Circuit Boards 15

5.9 Creating the TripPi Case 17

6. Trippie Application (Software) 18

6.1 Greeting Page 18

6.2 Login 18

6.3 Toolbar 18

6.4 Bucket List 18

6.5 Map 19

6.6 History 20

6.7 My Location 20

6.8 Settings 20

6.9 TripPi Display 21

7. Testing Firmware 22

7.1 Stepper Motor Testing 22

7.2 Ds18b20 Sensor Testing 25

8. Finalized Firmware 27

8.1 Python Code for Temperature, GPS readings and Motor Operations 27

8.2 PHP Script to Read Data 30

8.3 PHP Script to Write Data 31

8.4 PHP Code to Read Dummy Data 32

9. Progress Reports 33

9.1 Report 1 33

9.2 Report 2 33

9.3 Report 3 34

References 36

10. Conclusion and Recommendations 37

References 38

Appendix A: Glossary 39

Appendix B: Analysis Models 39

Appendix C: To Be Determined List 39

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

The purpose of our project is to work as a personal travel guide. Our android app will work anywhere in the world. We have used google places APIs to accomplish this. We will upload co-ordinates and temperature of a particular location and user can decide whether or not to visit that place based on the temperature. Our app will alert the user if temperature is below or above certain degree Celsius. The coordinates and temperature will be recorded by our device **TripPi** which is the hardware portion of our joint product.

## Document Conventions

This document uses generic technical document conventions such as numbered sections. In terms of font sizes. The section headings are larger as they indicate the start of every new section making them easier to find. All the headings are bolded to indicate each separate section branch, and each new section is shown on an odd page. Images are also added to show a more in-depth view of whatever topic they are associated with. Certain text is also bolded to stand out. These are mainly commands that we used, which should be easy for a contributor to find so that they can replicate what we did. There will also be some small annotations in the form of small roman numerals (As an example: (xvi)) to indicate that there is a reference provided that belongs with that context.

## Intended Audience and Reading Suggestions

The intended audience for this SRS document is our mentor Professor Kristian Medri and our team of three would like to improve this project in the future. This document can also be used as a guide by future users of the app and even other developers that might want to contribute and modify or improve the project for any reasons. Since, this software is open source. This technical report will not be in order for steps that require completion, and that is due to the fact that many different things can be worked on simultaneously. As such, the recommended reading order for this technical report will differ depending on who the reader may be. If it is an individual or group that would like to replicate or modify the project, then the reading order will not matter as each section is dedicated to a different component that ties with the Trip Planners project.

## System Overview

This document will go all of the information regarding the Trip Planner project. The main idea is that our device will be able to send accurate information regarding the temperature and coordinates depending on where the device is.

# Overall Description

## Product Perspective

The purpose of our device is to provide a GPS coordinates, as well as a temperature in an area where the device is set to read from. The device has the ability to protect itself from harsh environments using a timed stepper motor connected to a small trapdoor. This device is meant to provide a reliable, native reading on the characteristics for the location which the user chooses. An important thing to note, which may resolve any confusion, is that our company name is **Trip Planners**, the mobile application is called **Trippie (Pronounced trip-pee)**, and the device that has all the sensors and the raspberry pi integrated is called the **TripPi (Pronounced trip-pie)**.

## Product Functions

Both the device and application need an internet connection to function and will connect to an SQL database hosted on host monster. While our device is connected to a network, it will be able to do the following:

* Read the temperature at its location, and send it to the server so the Trip Planner app can read it
* Read the coordinates at its location, and send it to the server so the Trip Planner app can read it
* Close and open its own trapdoor based on a 5 minute timer to read the temperature and protect itself from harsh temperatures

## Operating Environment

The device will use a 5V bipolar stepper motor, Raspberry Pi 3, active internet connection to send the location and temperature readings to the server, temperature sensor and a GPS sensor.

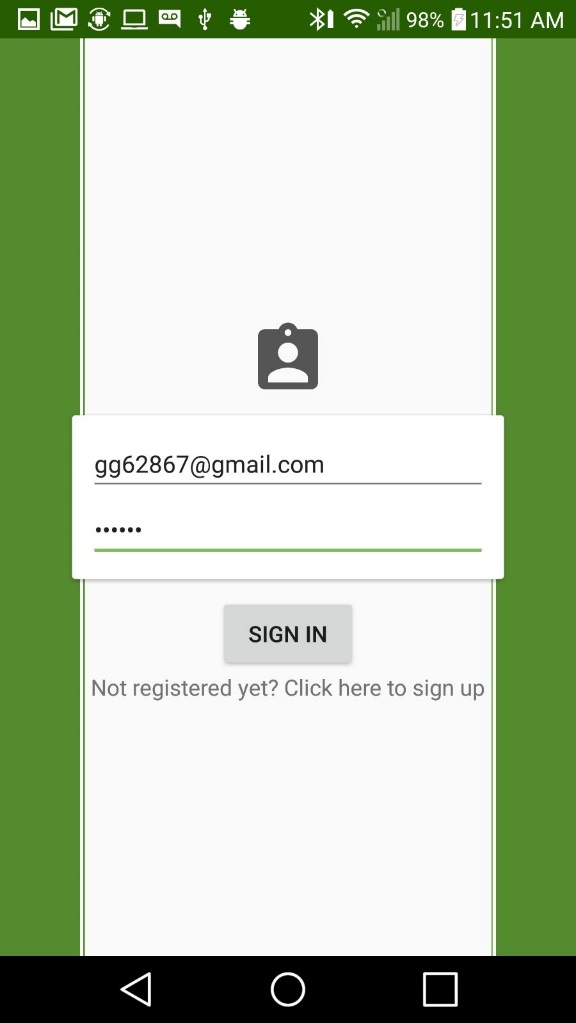
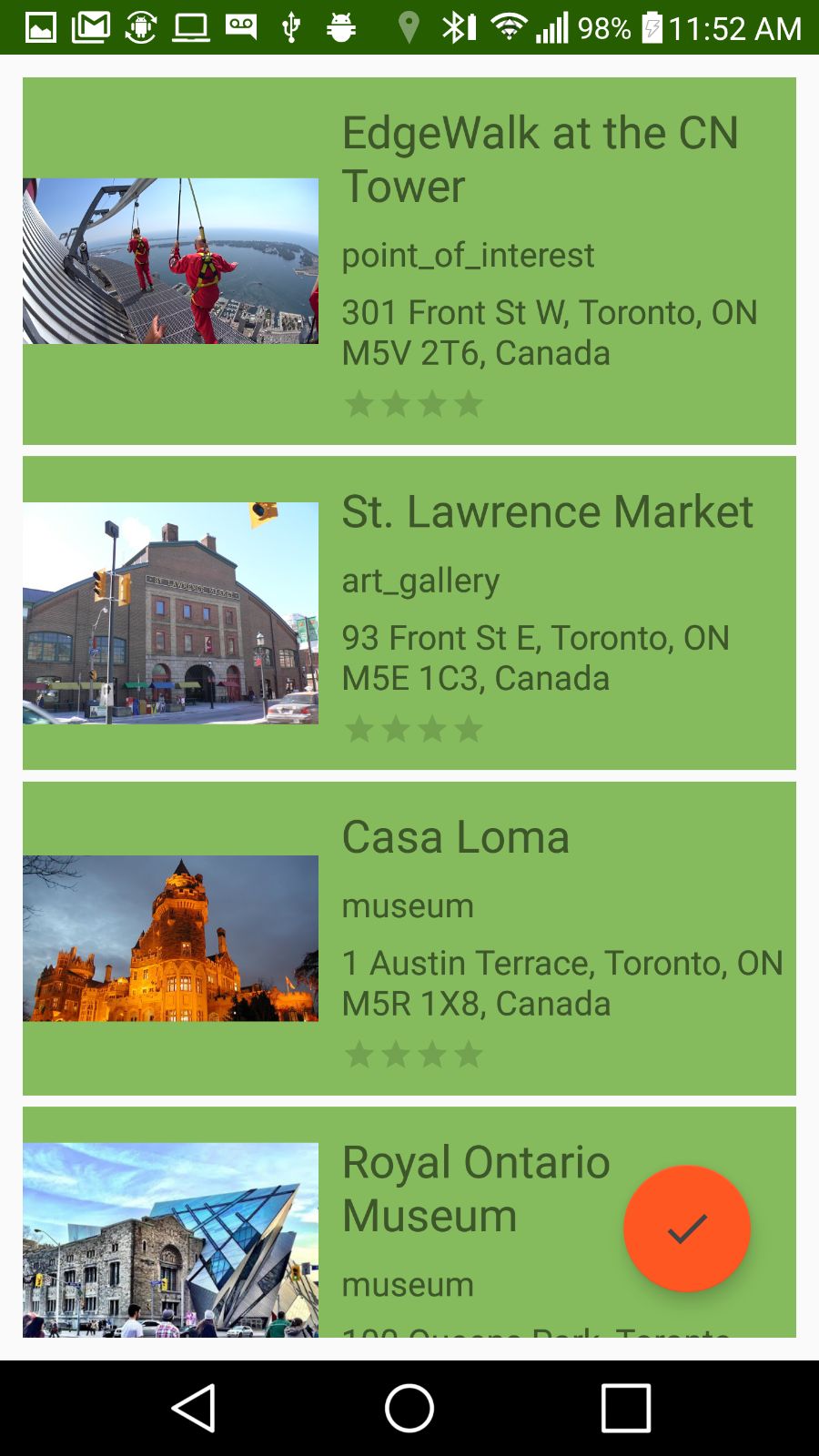
For the mobile app, the user needs to have a minimum SDK version 21 and an active internet connection.

The user will **not** be required to actually own the device, as our company will be providing these to the different locations around the globe.

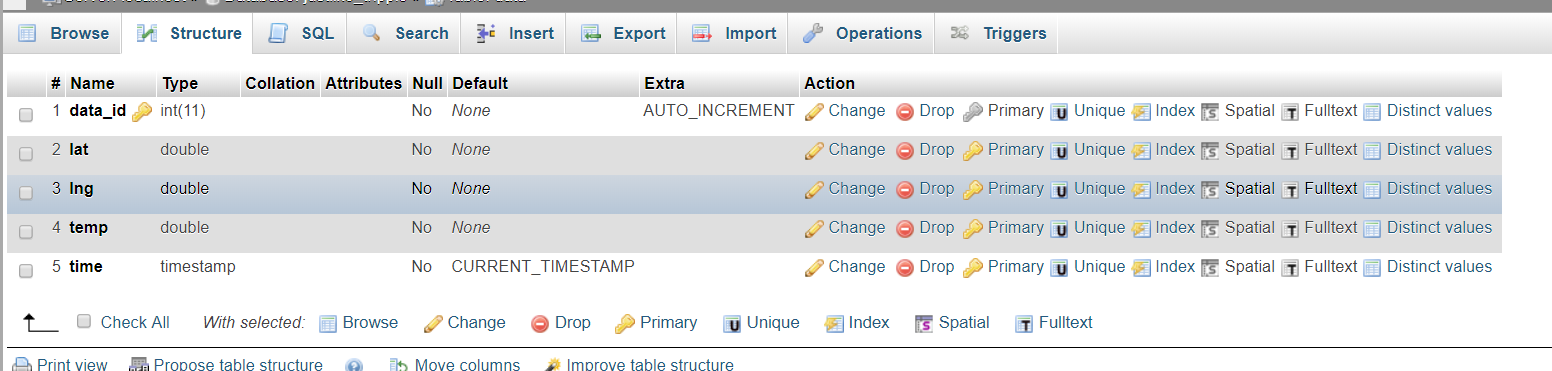
## User Documentation

The user documentation will be available on the Trip Planners official website. This documentation will come with instructions on how to use the app. The hardware will not have any documentation as the user will not be using the device.

## Mobile Application Requirements

If the user is already registered in our app then the first page which to be opened will be sign in page and it will allow our user to sign in into the app. The first time users of our app will have to sign up. When the user sign up then the app will redirect the user to a page which is called as a bucket list in which user add different places according to his plan to visit. After signing in to the app user is able to see their bucket list in which they can add different places which they want to visit in future. This page will be linked with the firebase database and it will show the readings of the temperature at different places which user wants to visit. Ravneet Singh is the lead developer for this

## Database Requirements

Ravneet Singh is the lead developer for the database functionality of this system. The hosting space is controlled on Host monster and the database is powered by SQL. Following is the screenshot of the table:

## Hardware Requirements

Originally, the design required a protector for the TripPi device. The following show who was meant to do this.

Bojan Lazic is working on the stepper motor and the contraption that has to be connected with it. This contraption will be used to protect the delicate temperature sensor.

Gaganpreet Singh Grewal is working on the temperature sensor and the design of the 3D contraption for the sensor. The tools used for the design is OpenSCAD and also windows 3D paint.

Ravneet Singh is working on getting the GPS coordinates and writing the python code to get the temperature sensor simultaneously. He is also working on the code to push that data to the table inside the database.

The hardware requirements changed, as the protector idea was scrapped, and instead a gauge was implemented. The gauge has a native design by the Trip Planners Company.

## Programming Knowledge

In order to complete this project, the user must have knowledge in C, Java, PHP and Python, as well as JSON. These are all used in creating the TripPi device as well as the Trippie application.

# Developmental Software

For each of the parts in the TripPi device, different software was needed for designing and creating. There are many different kinds of software that can be used to accomplish what is meant to be accomplished in this project, but the software we used is most likely the most optimal that can be used. In this section, the software used will be mentioned and a very brief tutorial for each one will follow. This section is meant to allow the user to customize our product if they so choose.

## Designing Applications

### Corel Draw

Corel Draw is an application which allows the user to draw unique shapes with line tools and have them exported as a file for laser cutting. This is the application used to cut out our devices case. The way that this specific application works when exporting for laser cutting, is how the lines are color coded. If the lines drawn are red, those will be cut first, if the lines are green, they will be cut last. If the lines are black however, this means that the laser cutter will instead engrave the lines. This is useful if the user would like images or text on the laser cut object. Some alternatives to this would be AutoCAD or PDF exportation.

### Fritzing

Fritzing is an application which is designed to assist engineers in create circuits. This application is what we used to create our printed circuit boards. To create or modify a printed circuit board, the user has a choice to directly edit a virtual printed circuit board within the software or create the breadboard representation which will generate a virtual printed circuit board that can also be edited if it is not exactly correct.

## Coding Applications

There will be no steps provided on how to use these applications as the user should be able to know how to write in different languages such as C, Java, PHP and Python, and that is mainly what is required for these applications. The applications used by us include:

**Android studio: Used to create the android application. Knowledge in java required**

**Nano editor in Raspberry Pi: Used to create the python code for the devices**

Any coding application can be used, but these we found were the most optimal for what we needed.

# Raspberry Pi Preparations

The raspberry Pi does not come prepared with everything ready. In order to do this project, a proper image for the raspberry Pi must be acquired and installed, and in order to use the raspberry Pi to its fullest, and all the devices associated with this project, many different libraries and software must be enabled and installed as well.

## Micro SD Storage

Similar to computers people use every day, the raspberry Pi requires a location to store all of its memory, like a hard drive. For its memory, instead of a hard drive, the raspberry pi uses a micro SD card to store the chosen raspberry pi operating system. Without the micro SD card, the raspberry pi is unusable. If the micro SD card is inserted into the raspberry pi but the micro SD card is blank, the raspberry pi is still unusable. In order for it to work, the user must install a raspberry pi image to the micro SD card through another device.

## Raspberry Pi Image

The raspberry pi image is what lets the user operate the raspberry Pi. Without it, there is no project. There are many different images provided by different sources, but the one that we used is a standard image provided by Canakit. This is a general image which allows access to things in the raspberry Pi such as a code editor, the terminal, all the files in the raspberry Pi, a web browser, and a VNC server-side application.

### VNC Server

This is an important software as it allows the user to connect remotely to a client device. VNC server is already installed on the Canakit image by default, except it is disabled. To enable it is simply locating the preferences in the settings and enabling it. It is important to note that the raspberry Pi’s IP address must be set as a static IP rather then a dynamic IP which it will be set to by default. Once the VNC setting is enabled, the user will then need to install the client application on a separate device, most preferable another computer entirely. VNC is supported by Windows, Apple, Linux, and some others. Once it is installed there, the user will just need to enter the raspberry Pi’s IP address into the client, and they will have a secure connection between the two devices. There is a workaround to this by using a regular HDMI cable, mouse and keyboard, however it is not optimal and not recommended.

## WiringPi

In order to actively use the GPIO pins on the raspberry Pi, it is required that the user must install Gordon’s wiringPi. This will give the user access to the pins while coding their programs. The steps in order to do this are:

**$ sudo apt-get install git-core**

**$ sudo apt-get update**

**$ sudo apt-get upgrade**

**$ cd**

**$ git clone git://git.drogon.net/wiringPi**

**$ cd ~/wiringPi**

**$ ./build**

## Enabling the One Wire Interface

In order to use the temperature sensor, the user will need to enable the one wire interface. The one wire interface (1-Wire) is a device communications bus system designed by Dallas Semiconductor Corp. It is able to provide signaling, power and low-speed data over a single conductor. It is very similar to I2C with a longer range and lower data rates. Without it, the temperature sensor cannot operate. The steps in order to enable the One Wire interface are as follows(i):

1. At the command prompt, enter: **sudo nano /boot/config.txt** and add **dtoverlay=w1-gpio** to the bottom of the file.
2. Reboot the raspberry Pi by using the command **sudo reboot.**
3. Log back into the pi and enter **sudo modprobe w1-gpio** in the command prompt.
4. Afterwards, enter **sudo modprobe w1-therm**
5. Change the directories to the **/sys/bus/w1/devices** directory by entering: **cd /sys/bus/w1/devices**
6. You can use the command **ls** to list the devices
7. Then, enter **cd 28-xxxxxxxxxxxx** (replace x’s with own numbers)
8. Now, enter **cat w1-slave** which will show the raw temperature reading output by the sensor.
9. Lastly, you can enter the command **cd** to return to the root directory.

Afterwards, once these steps are completed, the user will be free to implement the temperature sensor in any of the programs they choose to create, and it allows the TripPi device to read the temperature to send to the mobile application.

## Installing GPSD on the Raspberry Pi

In order to use the BU-353S4 USB GPS with the raspberry Pi, the user must install the GPSD libraries. GPSD is a service daemon that will monitor multiple GPSes or AIS receivers, which are attached to a host through USB ports. GPSD is supported by Linux, BSD, Mac OS X and Android. The steps to install GPSD will be described in the following steps:

1. First, open the command prompt and type **sudo apt-get upgrade**
2. Next, type **sudo apt-get install gpsd gpsd-clients python-gps**
3. And finally, type **sudo nano /lib/udev/gpsd.hotplug**

After installing the files, the GPSD daemon can be started with:

1. **sudo /etc/init.d/gpsd restart**
2. **sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock**
3. **cgps -s** – this will display the information provided by the USB GPS

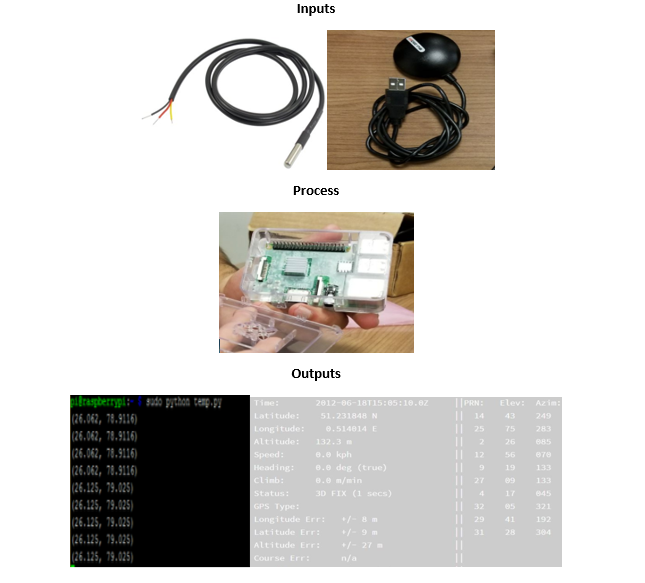
With the steps mentioned above completed, the user will then be able to view all the data received from the GPS and they will also be able to implement that retrieved data in the user’s programs by importing the gpsd libraries.

# Build Instructions

## Introduction

In our project we decided to use three sensors the Temperature Sensor, GPS sensor and a stepper motor. Our goal is make a hardware which will tell the location of the user and temperature at that particular location. The temperature sensor will detect the Temperature at various sites. GPS sensor will get the coordinates of the user’s particular location. We are connecting stepper motor to a contraption as a protection to our sensors. The database will store the different temperature readings, locations already visited by the user and locations which user wants to visit in future. The mobile device functionality will include Mobile application which will easily analyze the data collected by the sensor and it will also show the temperature of the particular location which user wants to visit.

## System Diagram



## Budget

The main components which are required for our project are a Raspberry pi, a temperature sensor, a GPS sensor and a stepper motor. These all components are available on Amazon, so you can easily get them. Some other components you will need to make this project are breadboard, jumper wires and a resistor. You can get these components from your parts kit.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Item | Source | Price  (Including Taxes) | Total |
| 1 | Raspberry Pi 3 | Amazon.com | $79.08 |  |
| 2 | GPS sensor, Temperature sensor and stepper motor | Amazon.com, RobotShop.com | $49.62, $16.93, $7.62 |  |
| 3 | PCB board | Prototype Lab | $25.50 |  |
| 4 | Jumper wires, Resistor | Parts Kit | $180(Partskit) |  |
|  |  |  |  | = $358.75 |

## Time Commitment

The project began with the modification to previous projects which we made last semester. We all spend some time check our individual projects whether they are working or not. Then we started integrating all the three sensors together which took about 2.5 hours. Then we started implementing the code by which both the temperature and the GPS sensor will work together and we spend 2 hours to complete it. We also designed a 3d printed contraption for our project which is connected to a stepper motor as a protection to our sensors which took about 1 day to get printed and 1 hour to connect it to the stepper motor.

## Mechanical Assembly

The assembly of our project is very simple. Firstly, put the SD card into the pi and set it up. Connect the 3d printed contraption to stepper motor. Attach a GPS sensor directly the raspberry pi through USB port and for temperature sensor, connect it to the raspberry pi with the help of breadboard, 4.7k ohm resistor and jumper wires as shown in the pictures and follow the steps given below to connect temperature sensor to the raspberry pi.

#1 - Connect GPIO GND [Pin 6] on the Pi to the negative rail on the breadboard and connect GPIO 3.3V [Pin 1] on the Pi to the Positive rail on the breadboard.

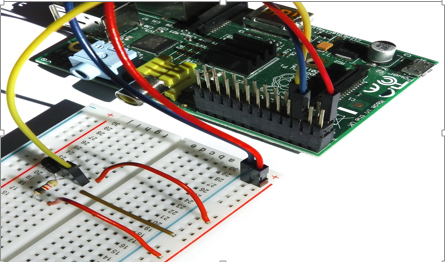
#2 - Plug the DS18B20+ into your breadboard, ensuring that all three pins are in different rows. Familiarise yourself with the pin layout, as it’s quite easy to hook it up backwards!

#3 - Connect DS18B20+ GND [Pin 1] to the negative rail of the breadboard.

#4 - Connect DS18B20+ VDD [Pin 3] to the positive rail of the breadboard.

#5 - Place your 4.7kΩ resistor between DS18B20+ DQ [Pin 2] and a free row on your breadboard.

#6 - Connect that free end of the 4.7kΩ resistor to the positive rail of the breadboard.

#7 - Finally, connect DS18B20+ DQ [Pin 2] to GPIO 4 [Pin 7] with a jumper wire.

Follow the steps given below to connect GPS sensor to the raspberry pi:

1. Place the Raspberry Pi on a non-conductive surface with GPIO pins facing up

2. Connect the raspberry-pi-3-model-b with the internet using a CAT-5 cable.

3. Connect the raspberry-pi-3-model-b with a mouse and a keyboard using USB.

4. Start terminal and input the following command to install the GPS drivers: "sudo apt-get install gpsd gpsd-clients python-gps"

5. Next we need to start the daemon. This is done using the following command: "sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock"

6. Ignore any messages from the console or in the log files, you may see it complaining about IPv6 but you can ignore that.

7. Connect the GPS receiver with the pi by USB.

8. Use the following command and the program will start running: cgps –s

Follow the steps given below to connect Stepper Motor to the raspberry pi:

1. Use the printed circuit board intended to drive the motor, rather than wiring the entire thing on the breadboard.

2. Attach all four wires (excluding the red common power wire, this will not be used) to the output pins of the L293D.

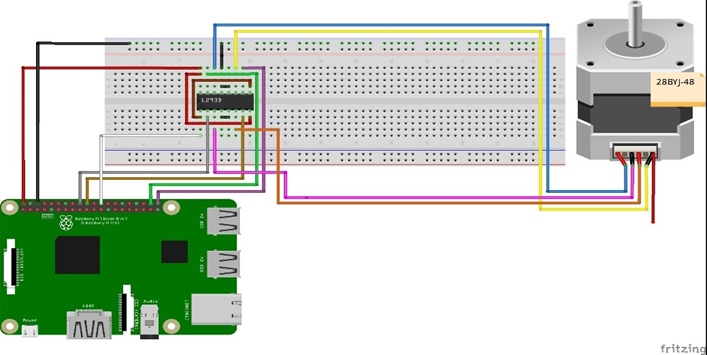
3. The input pins must be wired to the GPIO pins as per choice, but preferably pins 35 (GPIO19), 37 (GPIO26), 38 (GPIO20) and 40 (GPIO21).

4. Attach the enable pin from the chip (pin 1) to pin 33 of the raspberry pi (GPIO13) and attach the that enable pin from the chip to pin 9 of the chip, which is the enable pin for the other two input and output pins of the driver chip. Attach the ground pin of the driver chip to pin 39 of the raspberry pi as this is a ground pin.

5. The vss and vs pins of the chip must be wired to each other and wired to the 5V pin on the raspberry pi (pin 2).

6. Be sure that the modules are plugged into the sense hat. The BME280 will be sensing the temperature.

7. This will complete the wiring of the stepper motor. A diagram of the breadboarding can be seen below.



## Unit Testing

There are lot of build instructions in our project so unit testing should not be required because I did unit testing to get the project work. There is no Unit testing for the stepper motor because it is only used with the contraption that protect our sensors. For the GPS sensor the information of the user is received by connecting with multiple satellites orbiting the Earth. For this connection, the sensor must be under open sky. Unit testing is helpful if you face any problems in the mechanical assembly. Below is the unit testing which we did on the project. Testing our sensor.

#!/usr/bin/python

import os

#from gps import \*

import glob

import pygame, sys

import time

from time import \*

import serial

#os.system('modprobe w1-gpio')

#os.system('modprobe w1-therm')

base\_dir = '/sys/bus/w1/devices/'

device\_folder = glob.glob(base\_dir + '28\*')[0]

device\_file = device\_folder + '/w1\_slave'

def read\_temp\_raw():

f = open(device\_file, 'r')

lines = f.readlines()

f.close()

return lines

def read\_temp():

lines = read\_temp\_raw()

while lines[0].strip()[-3:] != 'YES':

time.sleep(0.2)

lines = read\_temp\_raw()

equals\_pos = lines[1].find('t=')

if equals\_pos != -1:

temp\_string = lines[1][equals\_pos+2:]

temp\_c = float(temp\_string) / 1000.0

temp\_f = temp\_c \* 9.0 / 5.0 + 32.0

return temp\_c, temp\_f

base\_dir = '/sys/bus/w1/devices'

#device\_folder = glob.glob(base\_dir + '28\*')[0]

device\_file = device\_folder + '/w1\_slave'

def read\_temp\_raw():

f = open(device\_file, 'r')

lines = f.readlines()

f.close

return lines

def read\_temp():

lines = read\_temp\_raw()

while lines[0].strip()[-3:] != 'YES':

time.sleep(0.2)

lines = read\_temp\_raw()

equals\_pos = lines[1].find('t=')

if equals\_pos != -1:

temp\_string = lines[1][equals\_pos+2:]

temp\_c = float(temp\_string) / 1000.0

return temp\_c

#initialise serial port on /ttyUSB0

ser = serial.Serial('/dev/ttyUSB0',4800,timeout = None)

fix = 1

x = 0

while x == 0:

gps = ser.readline()

# print all NMEA strings

# print gps

# lat = "cfhbdf"

# print lat

# check gps fix status

if gps[1:6] == "GPGSA":

fix = int(gps[9:10])

# print time, lat and long from #GPGGA string

if gps[1 : 6] == "GPGGA":

# get time

#time = gps[7:9] + ":" + gps[9:11] + ":" + gps[11:13]

# if 2 or 3D fix get lat and long

if fix > 1:

lat = " " + gps[18:20] + "." + gps[20:22] + "." + gps[23:27] + gps[28:29]

#lat = " " + gpsd.fix.latitude

lon = " " + gps[30:33] + "." + gps[33:35] + "." + gps[36:40] + gps[41:42]

print "Latitude: " + lat + " | Longitude: " + lon + " | Temperature: " + str(read\_temp())

# if no fix

else:

lat = " No Valid Data "

lon = " No Valid Data"

print "Latitude: " + lat + " | Longitude: " + lon + " | Temperature: " + str(read\_temp())

## Production Testing

The production testing is very easy. First, run the program and it will show the temperature readings on the screen. If the raspberry-pi-3-model-b is purchased in bulk, i.e. more than 100 raspberrypi’s then cost of each pi drops down to CA$35 per pi. No such discount is offered by the GPS receiver. Since the code running on the pi is free and same for every scenario the total cost of production would reach CA$9,375 which is 10.7% cheaper than the original cost per unit.

## Printed Circuit Boards

The TripPi device must include two custom printed circuit boards. One is built to drive the stepper

motor, and the other is to have the temperature sensor connected neatly to the raspberry PI. The materials needed for the temperature printed circuit board include:

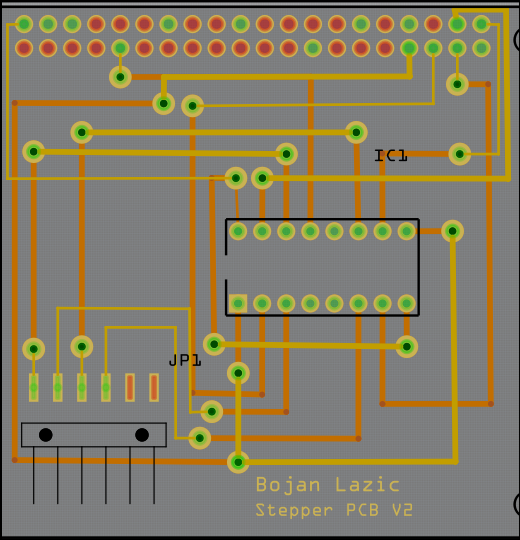
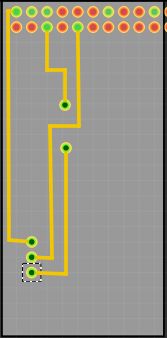
**Temperature sensor: Model Ds18b20**

**4700 ohm resistor**

**20 pin header**

To get started, the temperature sensor printed circuit board is simpler to create as it only has three traces. The first/red wire trace should connect the power wire to a 3.3V pin on the raspberry pi. The second/black wire trace should connect to one of the ground pins on the raspberry pi. The third/yellow wire trace should lead to a GPIO pin on the raspberry pi. Any pin will do as long as it does not interfere with another GPIO pin in use by another lead. The GPIO pin we used in this example is GPIO4. On this printed circuit board, a 4700 ohm resistor must be connected between the power trace and the GPIO trace, without touching the ground trace. The board itself only needs to be about half the size of a full size pi board which connects to all 40 pins, as this one does not need to use most of the pins. This 20 pin header must also be connected so that the pins face upwards, and the sockets face downwards to ensure the printed circuit board can fit nicely into the motor driving board.

The temperature sensor will most likely come as a long wire, around three feet long. This can be cut down to whatever the preferred size may be. For this example, we cut our wire down to approximately a foot in length. This simply makes the device more compact which is generally what we want for this design as it provides greater portability.



*Temperature PCB Stepper Motor PCB*

The second printed circuit board, which is used to drive the stepper motor, will be more complicated to create

due to many overlapping traces, and the use of double the pins that the temperature sensor printed circuit board uses. The materials needed for this printed circuit board are:

**L293D IC Chip**

**5 pin header**

**40 pin GPIO header**

**16 Pin IC socket**

What the L293D IC chip contains is the capacitors and smaller components needed to drive the motor, all contained into the one IC chip. The board can still be made without the IC chip but would need many more components and would be more time consuming to create. Generally, the printed circuit board will contain 7 traces connecting the raspberry pi pins to the different components on the printed circuit board. The traces will be connected as described in the following L293D IC pinout.

**[insert L293D pinout here]**

As shown, the pins on the chip are numbered as they are on common IC chips. To get the general idea, pin 16 on the IC must be connected to the 5V power pin on the raspberry pi, and pin 16 must be connected to pin 8, as both sides of the chip must be powered in order to work. There are four ground pins, but only one needs to be connected to the ground as all of the ground pins should be connected internally. Pins 2, 7, 10 and 15 are all input pins on the chip and need to be connected to different GPIO pins on the raspberry pi. After those are connected, pins 3, 6, 11 and 14 are to be connected to the four wires connected to the stepper motor. There is a certain order in which the pins should be connected, and the shown diagram of the stepper printed circuit board is the most optimal way to connect it. Since this is a printed circuit board, the motor will not be connected directly to it, for mainly for testing purposes and maintenance if needed. As such, this is where the five-pin header will be implemented. The pins that are to be connected to the motor from the L293D chip will instead be connected to this header so that the motor can simply be plugged into the header. Connecting the traces on the printed circuit board will seem difficult as there are many different components to connect, which would lead to some traces crossing over on one side. In these situations, there are holes surrounded by copper called "vias" which should be implemented in order to cross the traces to the other side of the printed circuit board. An important note is that the 40-pin header must be added to the printed circuit board in a way that the pins face upwards and the sockets face downwards. This is to ensure that the printed circuit board can "plug" into the raspberry Pi smoothly.

Connecting both of the printed circuit boards to the raspberry pi after they are completed is simple. First the motor driver circuit board with the 40 pin socket header must be connected to the raspberry pi, with each pin lined up with the pins on the raspberry pi. The second printed circuit board dedicated to sensing the temperature connects right on top of the first printed circuit board so that it will cover half of the pins on the raspberry pi starting from pin 1, or the power pin, on the raspberry pi.

## Creating the TripPi Case

To create the case for the TripPi, it is best to locate a laser cutting company, but for this prototype, we used the Humber prototype lab in order to cut out a case for free. There are different ways to create the case such as 3D printing and such, but laser cutting is the most optimal way to proceed.

The design for the TripPi case is a heavily modified design based off of a case made by user Elliot\_BXL on thingiverse (vi). The modifications include a removal of the lid, removal of the holes that keep the lid in place, elongating the sides of the case to double the height, a native temperature gauge design on one side of the case, and our company name engraved on the other side of the case. These files can be found in the repository, and images of the design are as shown as the following.

**[Case Design Images]**

The case is designed to clip together around the raspberry pi without the need of screws and bolts. This simplifies testing and creating the device, as it can be taken apart and put back together in seconds resulting in freedom of access at any time while making modifications. The temperature gauge also has two screw holes for the stepper motor to be attached to the case.

The software used by us is Corel Draw 2018 X7. Any version of Corel draw could work, but newer files cannot be opened on older versions. Other software can work, as long as it can export the file properly for laser cutting.

[Creative Commons License](http://creativecommons.org/licenses/by-sa/4.0/)  
raspberrypi-b-plus-case by <https://github.com/diy-electronics/raspberrypi-b-plus-case> is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-sa/4.0/).

Raspberry Pi is a trademark of the Raspberry Pi Foundation

Based on the [Adafruit Pi Box - Enclosure for Raspberry Pi Model A or B](https://www.adafruit.com/products/859)

# Trippie Application (Software)

The TripPi device is mainly an accessory to the Trippie application. This application is meant to provide a client with a simplified way to plan our future travel plans while managing old destinations. This application provides quite a few functions given the simplicity for the user. These functions will be explained in further detail throughout this section. Due to the code for the application being very large, it will not be provided in this report, however it will be available in the Trip Planners GitHub repository.

## Firebase Database

This entire project actually uses two different databases. The first one is HostMonster which maintains the sole purpose of communicating the information retrieved by the raspberry Pi to the mobile application. The second database used in this project is Firebase. Firebase is an extremely useful tool that can be utilized by android developers for free. It works differently then a normal database as the information can all be pushed and pulled from the application code itself, meaning there is no backend coding on the server. Our application of Firebase is used for each user separately to contain their login information as well as the locations they select, and the history of locations they have already visited.

## Greeting Page

When opening the application for the first time, the user will be greeted with a four-page design explaining the purpose of the application very briefly. After viewing this once, it will never show up again, even if the user exits the application completely and re-opens it.

## Login

Each user must register (registration is free), in order to use this application. This is required so that each user saved in our database will have their own private saved locations. If there was no registration/login, all the locations saved by each user would combine into one giant entry with thousands of saved locations. After registering, the user will be logged in, and will only be logged out if they choose to log out. This is the same if they log back in after ever logging out, they will stay logged into the application even if the app fully exits, unless the user chooses to log out.

## Toolbar

At the top of the screen on the toolbar, there are two hamburger buttons. The one on the left will open up a small menu for the user which contains the settings and the information received by the TripPi device. On the left of the toolbar is another hamburger which will list the options available to the user within the application. These contain the bucket list, a visual map using the google maps API, a history, the users current location, settings and a logout button.

## Bucket List

The Trippie application allows the user to plan multiple trips at the same time. These trips when selected through the location search option, will appear in a “bucket list” which is a list where the locations show up in order from what is selected. When the options are shown in this list, the user is able to select a location to view further details about that location.

## Map

This activity contains a map provided by the google maps API. The maps activity allows the user to view locations which they added to their bucket list, and these locations are indicated by a red flag at the separate locations. Locations that the user has added to the bucket list, but has visited, will appear as a green flag instead. This is meant to help the user keep track of locations they have visited and locations they have yet to visit. The user themselves is indicated by a small humanoid figure.

## History

In this activity, the locations already visited by the user will appear in a simple list. These are only here to keep track of those locations which the user has visited. This list has a maximum of 50 entries, meaning once the user starts visiting more then 50 trips, the oldest ones will disappear from the list, also effectively removing the green locations flags from the map.

## My Location

This activity simply will display the location where the user is currently, in the form of latitude and longitude, along with the location name. It is important to note that there are two different activities which display coordinates. This one in particular will display the coordinates based on where the user is, as it is showing the devices own GPS coordinates. The TripPi activity will show the coordinates based off of where our TripPi device is located.

## Settings

In the settings menu, the user is given three options. These include the language settings, location settings, and a logout option. So far, the application is only provided in English and French, so the language options will only provide those two options. The location settings button will only take the user to the location settings in the phone, which is a useful button for navigating while in the application. The logout button will log the user out of the device.

## TripPi Display

This activity is dedicated to display the information gathered and provided by the TripPi device. In this activity, the information is displayed in a text view format, with the data ID shown first, the latitude shown second, the longitude shown third, the temperature at the TripPi location shown fourth, and the date and time shown last. This information is what is meant to inform the user about all this information at the location of the given TripPi device.

# Testing Firmware

In this section, there will be different codes to allow the developer to test the different parts of the TripPi device. Most of these are not to be included in the final product and are here for testing purposes only. All the testing was done prior, so these may be unneeded, unless the developer chooses to modify the product. The GPS does not use python code, but instead uses libraries provided by its developers, and only needs to be defined by the with the USB port as it is a USB device.

## Stepper Motor Testing

The following will provide the necessary functions to move the motor in the different directions, provided with either user input to control the direction, or a timer to change the direction.

### With User input

|  |
| --- |
| import RPi.GPIO as GPIO |
|  | import time |
|  |  |
|  | GPIO.setmode(GPIO.BCM) |
|  |  |
|  | enable\_pin = 25 |
|  | coil\_A\_1\_pin = 20 |
|  | coil\_A\_2\_pin = 21 |
|  | coil\_B\_1\_pin = 19 |
|  | coil\_B\_2\_pin = 26 |
|  |  |
|  | GPIO.setup(enable\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_A\_1\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_A\_2\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_B\_1\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_B\_2\_pin, GPIO.OUT) |
|  |  |
|  | GPIO.output(enable\_pin, 1) |
|  |  |
|  | def forward(delay, steps): |
|  | for i in range(0, steps): |
|  | setStep(1, 0, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(1, 0, 0, 1) |
|  | time.sleep(delay) |
|  |  |
|  | def backwards(delay, steps): |
|  | for i in range(0, steps): |
|  | setStep(1, 0, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(1, 0, 1, 0) |
|  | time.sleep(delay) |
|  |  |
|  | def setStep(w1, w2, w3, w4): |
|  | GPIO.output(coil\_A\_1\_pin, w1) |
|  | GPIO.output(coil\_A\_2\_pin, w2) |
|  | GPIO.output(coil\_B\_1\_pin, w3) |
|  | GPIO.output(coil\_B\_2\_pin, w4) |
|  |  |
|  | while True: |
|  | delay = raw\_input("Delay between steps (milliseconds)? ") |
|  | steps = raw\_input("How many steps forward? ") |
|  | forward(int(delay) / 1000.0, int(steps)) |
|  | steps = raw\_input("How many steps backwards? ") |
|  | backwards(int(delay) / 1000.0, int(steps)) |
|  |  |
|  | GPIO.cleanup() |

### Timer implementation

|  |
| --- |
| import RPi.GPIO as GPIO |
|  | import time |
|  |  |
|  | GPIO.setmode(GPIO.BCM) |
|  |  |
|  | enable\_pin = 25 |
|  | coil\_A\_1\_pin = 20 |
|  | coil\_A\_2\_pin = 21 |
|  | coil\_B\_1\_pin = 19 |
|  | coil\_B\_2\_pin = 26 |
|  |  |
|  | GPIO.setup(enable\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_A\_1\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_A\_2\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_B\_1\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_B\_2\_pin, GPIO.OUT) |
|  |  |
|  | GPIO.output(enable\_pin, 1) |
|  |  |
|  | starttime=time.time() |
|  |  |
|  | def forward(delay, steps): |
|  | for i in range(0, steps): |
|  | setStep(1, 0, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(1, 0, 0, 1) |
|  | time.sleep(delay) |
|  |  |
|  | def backwards(delay, steps): |
|  | for i in range(0, steps): |
|  | setStep(1, 0, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(1, 0, 1, 0) |
|  | time.sleep(delay) |
|  |  |
|  | def setStep(w1, w2, w3, w4): |
|  | GPIO.output(coil\_A\_1\_pin, w1) |
|  | GPIO.output(coil\_A\_2\_pin, w2) |
|  | GPIO.output(coil\_B\_1\_pin, w3) |
|  | GPIO.output(coil\_B\_2\_pin, w4) |
|  |  |
|  | while True: |
|  | forward(int(2) / 1000.0, int(128)) |
|  | time.sleep(60) |
|  | backwards(int(2) / 1000.0, int(128)) |
|  | time.sleep(60) |
|  |  |
|  | GPIO.cleanup() |

## Ds18b20 Sensor Testing

The following is intended to assist in developing a program which can read and use the information provided by the temperature sensor.

import os

import glob

import time

os.system('modprobe w1-gpio')

os.system('modprobe w1-therm')

base\_dir = '/sys/bus/w1/devices/'

device\_folder = glob.glob(base\_dir + '28\*')[0]

device\_file = device\_folder + '/w1\_slave'

def read\_temp\_raw():

f = open(device\_file, 'r')

lines = f.readlines()

f.close()

return lines

def read\_temp():

lines = read\_temp\_raw()

while lines[0].strip()[-3:] != 'YES':

time.sleep(0.2)

lines = read\_temp\_raw()

equals\_pos = lines[1].find('t=')

if equals\_pos != -1:

temp\_string = lines[1][equals\_pos+2:]

temp\_c = float(temp\_string) / 1000.0

temp\_f = temp\_c \* 9.0 / 5.0 + 32.0

return temp\_c, temp\_f

while True:

print(read\_temp())

time.sleep(1)

# Finalized Firmware

This section provides the programs needed for the TripPi device to operate as we intend it to. This will include the server side PHP, and TripPi device side code.

## Python Code for Temperature, GPS readings and Motor Operations

|  |
| --- |
| import os |
|  | import glob |
|  | import pygame, sys |
|  | import time |
|  | import serial |
|  | import urllib2, urllib |
|  | import RPi.GPIO as GPIO |
|  | import thread |
|  |  |
|  | base\_dir = '/sys/bus/w1/devices/' |
|  | device\_folder = glob.glob(base\_dir + '28\*')[0] |
|  | device\_file = device\_folder + '/w1\_slave' |
|  |  |
|  | GPIO.setmode(GPIO.BCM) |
|  |  |
|  | enable\_pin = 25 |
|  | coil\_A\_1\_pin = 20 |
|  | coil\_A\_2\_pin = 21 |
|  | coil\_B\_1\_pin = 19 |
|  | coil\_B\_2\_pin = 26 |
|  |  |
|  | GPIO.setup(enable\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_A\_1\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_A\_2\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_B\_1\_pin, GPIO.OUT) |
|  | GPIO.setup(coil\_B\_2\_pin, GPIO.OUT) |
|  |  |
|  | GPIO.output(enable\_pin, 1) |
|  |  |
|  | def forward(steps): |
|  | delay = 0.002 |
|  | print "\nRotating fowards\n" |
|  | for i in range(0, steps): |
|  | setStep(1, 0, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(1, 0, 0, 1) |
|  | time.sleep(delay) |
|  |  |
|  | def backwards(steps): |
|  | delay = 0.002 |
|  | print "\nRotating Backwards\n" |
|  | for i in range(0, steps): |
|  | setStep(1, 0, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 0, 1) |
|  | time.sleep(delay) |
|  | setStep(0, 1, 1, 0) |
|  | time.sleep(delay) |
|  | setStep(1, 0, 1, 0) |
|  | time.sleep(delay) |
|  |  |
|  | def setStep(w1, w2, w3, w4): |
|  | GPIO.output(coil\_A\_1\_pin, w1) |
|  | GPIO.output(coil\_A\_2\_pin, w2) |
|  | GPIO.output(coil\_B\_1\_pin, w3) |
|  | GPIO.output(coil\_B\_2\_pin, w4) |
|  |  |
|  | def my\_round(x): |
|  | return round(x\*4)/4 |
|  |  |
|  | def rotateMotor(temp\_init, temp\_final): |
|  | temp\_dif = my\_round(temp\_final - temp\_init) |
|  | if temp\_dif < 0: |
|  | temp\_dif \*= -1 |
|  | steps = int(temp\_dif \* 4) |
|  | backwards(steps) |
|  | else: |
|  | steps = int(temp\_dif \* 4) |
|  | forward(steps) |
|  |  |
|  | def save\_data(lat,lng,temp): |
|  | #Save To Database |
|  | mydata=[('lat',lat),('lon',lng),('temp',temp)] |
|  | mydata=urllib.urlencode(mydata) |
|  | path='http://www.justlikerav.com/trippie/write\_data.php' |
|  | req=urllib2.Request(path, mydata) |
|  | req.add\_header("Content-type", "application/x-www-form-urlencoded") |
|  | page=urllib2.urlopen(req).read() |
|  | print page |
|  |  |
|  | def CoordinateToDouble(hours, minutes, seconds, NEWS): |
|  | if NEWS == "W" or NEWS == "S": return (((minutes + ((seconds / 6000)%1))/60) + hours)\*-1 |
|  | else: return ((minutes + ((seconds / 6000)%1))/60) + hours |
|  |  |
|  | def read\_temp\_raw(): |
|  | f = open(device\_file, 'r') |
|  | lines = f.readlines() |
|  | f.close() |
|  | return lines |
|  |  |
|  | def read\_temp\_raw(): |
|  | f = open(device\_file, 'r') |
|  | lines = f.readlines() |
|  | f.close |
|  | return lines |
|  |  |
|  | def read\_temp(): |
|  | lines = read\_temp\_raw() |
|  | while lines[0].strip()[-3:] != 'YES': |
|  | time.sleep(0.2) |
|  | lines = read\_temp\_raw() |
|  | equals\_pos = lines[1].find('t=') |
|  | if equals\_pos != -1: |
|  | temp\_string = lines[1][equals\_pos+2:] |
|  | temp\_c = float(temp\_string) / 1000.0 |
|  | return temp\_c |
|  |  |
|  | ser = serial.Serial('/dev/ttyUSB0',4800,timeout = None) |
|  | fix = 1 |
|  | x = 0 |
|  |  |
|  | #set the scale |
|  | rotateMotor(0, read\_temp()) |
|  | oldTemp = read\_temp() |
|  | while x == 0: |
|  | gps = ser.readline() |
|  |  |
|  | if gps[1:6] == "GPGSA": |
|  | fix = int(gps[9:10]) |
|  | if gps[1 : 6] == "GPGGA": |
|  | newTemp = read\_temp() |
|  | if fix > 1: |
|  | lat = str(CoordinateToDouble(float(gps[18:20]), float(gps[20:22]), float(gps[23:27]), gps[28:29])) |
|  | lon = str(CoordinateToDouble(float(gps[30:33]), float(gps[33:35]), float(gps[36:40]), gps[41:42])) |
|  | save\_data(lat,lon,str(newTemp)) |
|  | else: |
|  | lat = "NULL" |
|  | lon = "NULL" |
|  |  |
|  | print "Latitude: " + lat + " | Longitude: " + lon + " | Temperature: " + str(newTemp) |
|  | if my\_round(newTemp-oldTemp) != 0: |
|  | print "\n\t | Tempdiff: " + str(my\_round(newTemp-oldTemp)) |
|  | rotateMotor(oldTemp, newTemp) |
|  | oldTemp = newTemp; |

## PHP Script to Read Data

|  |
| --- |
| <?php |
|  | $servername = "localhost"; |
|  | $username = "justlik6\_travel"; |
|  | $password = " "; |
|  | $dbname = "justlik6\_trippie"; |
|  |  |
|  | $sql = "SELECT \* FROM data ORDER BY data\_id DESC LIMIT 1"; |
|  |  |
|  | $con = mysqli\_connect($servername, $username, $password, $dbname); |
|  |  |
|  | $result = mysqli\_query($con, $sql); |
|  |  |
|  | $response = array(); |
|  |  |
|  | while($row = mysqli\_fetch\_array($result)){ |
|  | array\_push($response, array("id"=>$row[0], "lat"=>$row[1], "lon"=>$row[2], "temp"=>$row[3], "time\_stamp"=>$row[4])); |
|  | } |
|  |  |
|  | echo json\_encode(array("TripPi\_response"=>$response)); |
|  | ?> |

## PHP Script to Write Data

|  |
| --- |
| <?php |
|  | $servername = "localhost"; |
|  | $username = "justlik6\_travel"; |
|  | $password = " "; |
|  | $dbname = "justlik6\_trippie"; |
|  |  |
|  | // Create connection |
|  | $conn = new mysqli($servername, $username, $password, $dbname); |
|  | // Check connection |
|  | if ($conn->connect\_error) { |
|  | die("Connection failed: " . $conn->connect\_error); |
|  | } |
|  |  |
|  | $lat = 0; |
|  | $lon = 0; |
|  |  |
|  | if($\_POST['lat'] == 'NULL' || $\_POST['lon'] == 'NULL'){ |
|  | $sql = "SELECT \* FROM data ORDER BY data\_id DESC LIMIT 1"; |
|  | $result = mysqli\_query($conn, $sql); |
|  |  |
|  | if ($result->num\_rows > 0) { |
|  | while($row = $result->fetch\_assoc()) { |
|  | $lat = $row["lat"]; |
|  | $lon = $row["lng"]; |
|  | } |
|  | } else { |
|  | echo "0 results"; |
|  | } |
|  | }else{ |
|  | $lat = $\_POST['lat']; |
|  | $lon = $\_POST['lon']; |
|  | } |
|  |  |
|  | $sql = "INSERT INTO data (lat, lng, temp) VALUES ('".$lat."', '".$lon."', '".$\_POST['temp']."')"; |
|  | echo $sql; |
|  |  |
|  | if ($conn->query($sql) === TRUE) { |
|  | echo "New record created successfully"; |
|  | } else { |
|  | echo "Error: " . $sql . "<br>" . $conn->error; |
|  | } |
|  |  |
|  | $conn->close(); |
|  | ?> |

## PHP Code to Read Dummy Data

|  |
| --- |
| With open('GPS\_temp\_data','r') as the\_file:  print the\_file.read()  the\_file.close() |
|  |  |
|  |  |

## Trippie Application Code

As this is a very large application, the code will not be added to this document, but instead described briefly. The code can still be found in the Trip Planners GitHub repository. The Trippie application consists of fifteen different classes along with 20 different test case classes for the sole purpose of testing. Each class generally corresponds to a different activity or application function such as connecting to our firebase server. Along with these classes, there are thirty one different XML files that correspond to the different layouts. All together our application would take over 70 pages of code and for that reason will not be included in this technical report.

# Progress Reports

Throughout the course of creating the TripPi, there were times we were required to update our current progress at certain times. This was generally divided into three different days spread across four months. These reports will be provided here to give an understanding on how the developer should manage the time creating the TripPi.

## Report 1

03-05 2018

*“Currently, Ravneet is still working on the database and mobile application.*

*Bojan is working on the hardware stepper motor mechanical functionality and still needs the 3D printed materials. We are still figuring out how to retrieve the values from the database to display in the mobile application. Gaganpreet is working on the temperature sensor and the design of the 3D contraption for the sensor. We will proceed with making our device protector. All the sensors are working and just need to be integrated together.*

*-Team TripPlanners*

*Believe in an Innovative Future”*

## Report 2

03-19 2018

*“This is our status update on the project Trippie. Last week, we demonstrated the hardware of our project. Our GPS and temperature sensors were working well but our stepper motor was not connected to it and our database was not ready. We would get the readings for GPS sensor only if we went outside the classroom i.e we were not able to get the readings of GPS sensor inside the class. There was no specific design for our project.*

*This week for the hardware implementation, I made a special frame for our project in which I connect a stepper motor inside the frame which will help in opening and closing of the frame. First I tested the individual sensors on pi and then connected them together on one pi. All the three sensors were connected and worked well.*

*Ravneet Singh was working on the database and mobile application and he has successfully completed the database. He used a PHP script on the server as an intermediary to send data from a raspberry pi and the remote MySQL server on* [*hostmonster.com*](http://hostmonster.com/)*. Now we are able to push data to the server, receive data from the server and display it in the android app. He also wrote a code to put readings of the GPS sensor in a file and read them from that file.*

*Bojan designed an attachment which is connected in front of his stepper motor and will help open and close the frame. He also wrote a special code to rotate his motor in forwarding and backward direction up to a specific point. For the temperature sensor, we are using a breadboard but Bojan made a PCB to replace breadboard. Next week I will use the printed PCB instead of the breadboard.*

*While working on database Ravneet found some problems like reading database from the android app is not that straightforward. The reason for this is that Android applications can be decompiled, and the client will have credentials to access your database. The solution to this problem is a Service provider application. This application will create and publish web services (preferably restful) using a PHP script on the server. This PHP script will read the latest data and generate a JSON output which our android application will parse to get the values. We are using the same raspberry pi which we used in last semester and all the other jumper wires and breadboard is available in parts kit so there is no increase in the budget of our project.*

*Here are some helpful links:*

*Using firebase with python:*[*https://pypi.python.org/pypi/python-firebase/1.2*](https://pypi.python.org/pypi/python-firebase/1.2)

*Github:*[*https://github.com/princess97/TripPlanner\_V1*](https://github.com/princess97/TripPlanner_V1)*”*

## Report 3

04-02 2018

*“This is our status update on the project Trippie. Last time, we demonstrated the hardware of our project. All the three sensors were working well. The GPS sensor was getting the coordinates, the temperature sensor displays the temperature and stepper motor was designed to open and close the contraption of the pi. As you said there is no need to open and close the contraption you recommend us to work on the indicator which will be connected to the stepper motor and will change its direction with the change in temperature.*

*Student A: Bojan Lazic*

*As all the functionality of the stepper motor is already completed, so Bojan is working on a design for the Gage for the temperature sensor. Hopefully, it will be implemented into our project by next week.*

*Student B: Gaganpreet Singh Grewal*

*This week Gaganpreet worked on PCB board. Earlier he was using the breadboard to connect his temperature sensor to Raspberry Pi. But now he soldered all his components like resistor, jumper wires on a PCB and connect it to Pi. Now we are using 2 PCB boards one for the temperature sensor and other for the stepper motor.*

*Student C: Ravneet Singh*

*So far, all the three sensors are working together simultaneously but the GPS sensor does not work indoors. The earlier program that we wrote, stopped updating the database table the moment our Pi stops receiving any signal from the GPS sensor. I did not notice earlier, that the temperature sensor works anywhere just fine but inside a building. However, it is important to update the temperature readings regularly even when GPS signals receive nothing. For this reason, our goal was to have a system design that updates the temperature regardless of the GPS value.*

*To solve this issue we had to write a new PHP script that first saves the last GPS coordinates and then updates the temperature received from the temperature sensor. This happens only in the case if the GPS sensor does not receive the coordinates otherwise the GPS coordinates are updated.*

*As we are using the same Raspberry Pi which we used in last semester and all the other components like jumper wires are available in the parts kit. We designed a PCB and printed it out from Humber prototype lab at free of cost. So there is no increase in the budget of our project. But if you have to buy all these components form the market, then the cost for a raspberry pi is $79.08, the cost for jumper wires is $7.11, and the cost for PCB board is approximately $25.50. All the prices are including taxes.*

*Here are some helpful links:*

*Github:*[*https://github.com/princess97/TripPlanner\_V1*](https://github.com/princess97/TripPlanner_V1)

*Using firebase with python:*[*https://pypi.python.org/pypi/python-firebase/1.2*](https://pypi.python.org/pypi/python-firebase/1.2)*”*

# Conclusion and Recommendations

This Technical report presents a plan for providing an IoT solution for This is an opportunity for us to integrate many things to create a collaborative project which will help users to plan and visit the different places in the world without the help of any guide and in bonus it will tell the temperature of that place which user wants to visit. This will reduce the budget of the user and save lots of time. This is an opportunity to integrate the knowledge and skills developed in our program to create a collaborative IoT capstone project demonstrating my ability to learn how to support projects.

References

I Raspberry Pi DS18B20 Temperature Sensor Tutorial. (2017, April 10). Retrieved from

<http://www.circuitbasics.com/raspberry-pi-ds18b20-temperature-sensor-tutorial/>

ii DS18B20 One Wire Digital Temperature Sensor and the Raspberry Pi. (2015, September 11). \

Retrieved from

<https://www.modmypi.com/blog/ds18b20-one-wire-digital-temperature-sensor-and-the-raspberry-pi>

iii BU-353-S4. (n.d.). Retrieved from

<http://usglobalsat.com/p-688-bu-353-s4.aspx#images/product/large/688.jpg>

iv (n.d.). Retrieved from

<https://www.raspberrypi.org/forums/viewtopic.php?f=28&t=128028>

v Lazic, B. (n.d.). Bojan-sensei/CENG317. Retrieved from

<https://github.com/Bojan-sensei/CENG317/blob/master/firmware/BME280.py>

vi T. Elliot (n.d.). Raspberry pi 2 lasercut case by Elliot\_BXL. Retrieved from

<https://www.thingiverse.com/thing:1415311>

vii Raspberry Pi Stepper Motor Control with L293D / ULN2003A. (n.d.). Retrieved from

<https://tutorials-raspberrypi.com/how-to-control-a-stepper-motor-with-raspberry-pi-and-l293d-uln2003a/>

viii Wiegers, K. E. (1999). IEEE Software Requirements Specification Template. Retrieved February

12, 2018, from

<https://cs.gmu.edu/~dfleck/classes/cs421/spring08/srs_template.doc>

Appendix A: Glossary

The Following are the different words defined that may not be known the reader:

**TripPi-** This is the device name

**Trippie-** The name of the mobile application

**Trip Planners-** Company name for this project

**Java-** Object Oriented Programming Language

**C-** Most basic programming language used as a basis for other languages

**PHP-** Scripting language

**JSON-** Scripting language

**GPS-** Global Positioning System, determines a devices location

**Stepper Motor-** a motor driven by electric poles within the motor

**Software-** Applications and digital interfaces

**Firmware-** Codes and programs

**Hardware-** Physical devices

Appendix B: Analysis Models