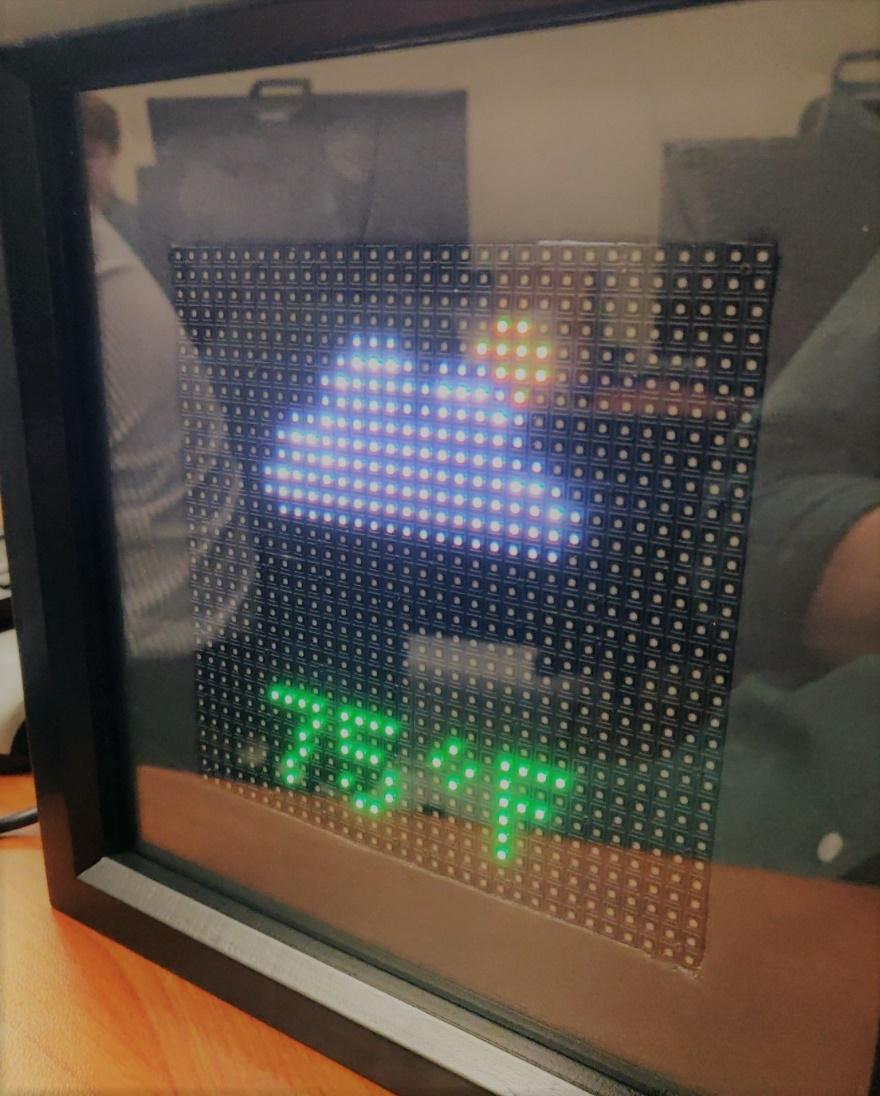
# Wisplay Weather Display



12/8/2018

Embedded Systems

Final Project Report

Matthew Carrano, Gabriel Hahn, Valentin Medina

# Project Summary

This device is an LED weather display that draws data from a sensor and a website to display indoor and outdoor weather data. Weather related data is generally collected from satellites and land, air, and water-based weather stations. The device can be adjusted to receive and display web data from anywhere in the United States. The device would most likely serve as a table-side display with its aesthetic appeal and frame-like casing, but it is also possible to mount the display on a wall. Users would be able to glance at the display to quickly get a reading of the desired weather information and choose which data is displayed via an app on their phone.

The device uses sensors to measure temperature, pressure, and humidity. The data is updated every 2 seconds. The display shows the app selected data as well as an image correlating to that displayed data. The app can select between indoor or outdoor temperature, pressure, or humidity.

All electronic components are housed within a framed casing allowing for a clear, glass covered display. The casing has a slit in the side of it providing a route for power access as well as for allowing the internal sensor to get more accurate local data. The device is powered using a 12V 10A power supply that plugs into an outlet.

# Project Objectives

1. Collect Data: Collect region specific weather data from a weather API and measure temperature, pressure, and humidity locally via a sensor.
2. Display: Shows current weather data as selected by the user and provides a correlating image.
3. App Control: Allow the user to choose which data is shown on the display from the ‘Blynk’ app and update the display accordingly.
4. Aesthetic: The device would be a household consumer item and should therefore be visually appealing should easily integrate into any in home location without being too distracting.

# Selected Hardware Platform

A Raspberry 3 Model B+ was chosen as the hardware platform for the project for several reasons. The Raspberry Pi 3 has built-in WIFI capabilities which allows it to collect data from an API and a phone app. Due to its popularity, there is also an ample amount of helpful information online for any problems encountered.

It also can easily run python which many products including the sensor, LED-matrix, and Blynk app are designed to run seamlessly with. Further, there is an accessible HAT for the Pi that allows centralized power dispersion rather than having to power each component individually. It preconfigured I2C SDA and SCL lines, allowing for easy communication with the I2C sensor used. It also has enough GPIO pins for the LED display which used all GPIO pins.

# Hardware Design Description

Figure 1 below shows the hardware design of our entire device.

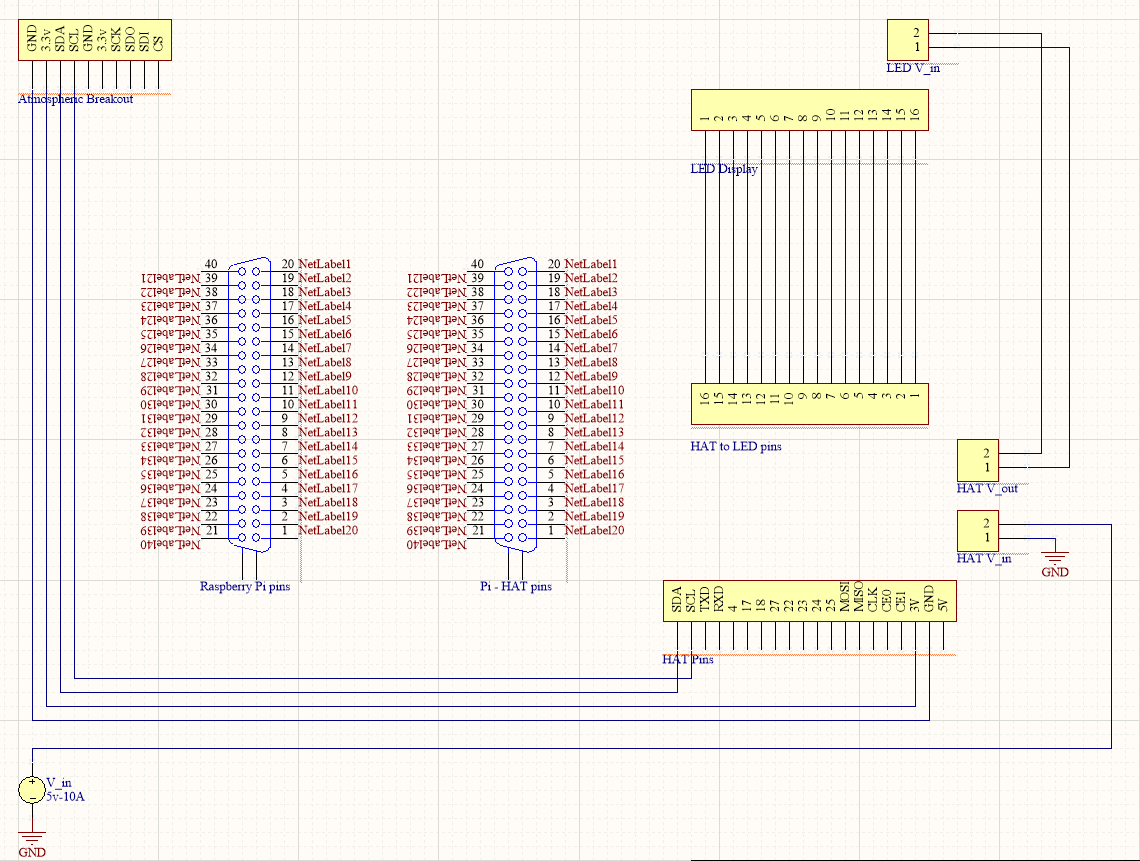


Figure 1. Hardware design schematic.

This project used the BME280 Atmospheric Sensor that measures temperature, pressure, and humidity and has I2C communication. The 3.3V power out pin on the Raspberry Pi HAT was used to power the sensor. These sensors were also connected to pins SCL (clock line) and SDA (data line) on the Pi to enable I2C communication. It used the 0x77 slave address.

The LED-matrix display was powered by the 5V source voltage through the HAT and grounded by the Raspberry Pi. The GPIO pins were connected to pins 1-16 on the HAT. The primary power source is a 5V 10A wall plug-in power supply that powers the Raspberry Pi and HAT through the on-board barrel jack. This provides more than enough power for the device to run and allows the possibility of expansions (adding more LED displays) in the future.

These hardware connections were placed together and the three connections for the sensor to the HAT were soldered . All circuitry was fully enclosed inside the casing, and the physical implementation of the hardware design can be seen in Figure 2 below.

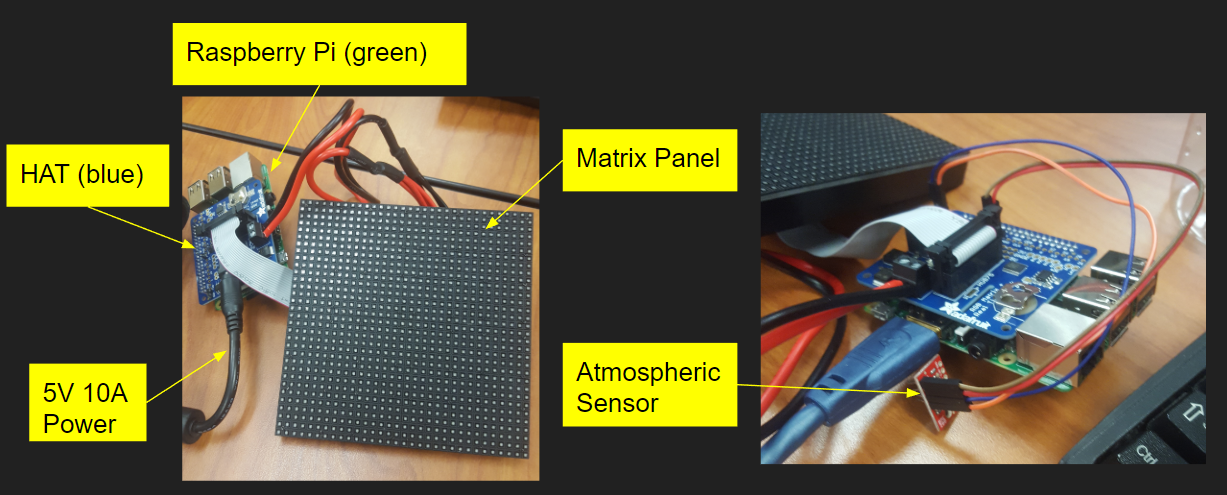


Figure 2. Hardware implementation.

A brief summary of each hardware component is included below, along with part specifications:

1. **Raspberry Pi 3 Model B+**

* Built-in WIFI capabilities and ability to communicate with an I2C sensor

1. **Atmospheric Sensor**—SparkFun Atmospheric Sensor Breakout – BME280

* Measures temperature, humidity, and pressure at sensor location and communicates with I2C

1. **Matrix Panel**—32x32 RGB LED Matrix Panel – 4mm Pitch (ID: 607)

* Displays atmospheric data as well as an image correlating with the data

1. **5V 10A Power**—5V 10A switching power supply (ID: 658)

* Provides sufficient voltage to power HAT and all connected devices
* Extra Capacity
* Plugs directly into Raspberry Pi

1. **HAT**—Adafruit RGB Matrix HAT + RTC for Raspberry Pi – Mini Kit

* Allows centralized power location
* Provides GPIO access to panel
* Protects against negative, over or under-voltages

1. **Cell Phone (Not Shown)**—Android or Apple

* Provides access to Blynk app for remote control of display

**SOFTWARE PLATFORM**

We chose to use Python as our software platform. This was largely because we used a Raspberry Pi as our main controller. Python is commonly used in conjunction with Raspberry Pi’s, so we knew there would be good Python support for whatever we wanted to accomplish using the Pi.

**SOFTWARE DESIGN**

**Overview**

There are five main components the software must handle: driving the LED matrix, fetching weather data from the Internet (OpenWeatherMapAPI), fetching atmospheric data from the sensor, communicating through the Blynk, and icon logic. Those components will be explained in detail in their own sections. This overview will mostly describe the code that connects those components and the overall structure of the code.

Most of the important code runs in a while loop. Inside the while loop the first thing that happens is the weather data using the weather API and from the sensor are loaded into respective variables. Next, there is logic to decide which source is displayed (the internet or sensor data) and which reading is displayed. This uses the Blynk code which will be explained later. Once that is decided, the data can be put into a string and concatenated with a unit (e.g. °F) for display purposes. Next there is logic for centering the displayed string and for deciding what color is used to display the temperature on the matrix (assuming temperature is chosen by the user). The color for humidity and pressure are set at cyan and orange, respectively. Next, the string containing the data and its units is displayed on the matrix. Finally, there is the logic for deciding which icon to show. This while loop is on a two second delay to reduce the number of calls to the weather API. Lastly, there is a main function which instantiates the RunMatrix class, thus running the code.

**Driving the LED Matrix**

To drive the LED matrix, the “rpi-rgb-led-matrix” library was used. It can be found [here](https://github.com/hzeller/rpi-rgb-led-matrix). The structure in our code of a Python class with \_\_init\_\_ and run methods comes from that library. It is required for using the graphics functions. The main function also comes from that library. There are four main methods from the graphics class that are used in this code: DrawText, DrawLine, DrawCircle, and Color. DrawText is used to display the data string to the matrix. It receives a font, an x position, a y position, a color and the string to display. DrawLine receives an x/y coordinate for 2 end-points and a color. DrawCircle receives an x and y position, a radius, and a color. Color receives RGB values and is a convenient way to save a color as a variable which can be passed into draw functions (this can be seen at the top of the code just under the library imports). Another notable library method is LoadFont (seen at the top of the run method), it loads a font from a file in the library. Fonts mostly affect the size of the text displayed.

**OpenWeatherMap API**

This API was used to receive weather data wirelessly using the Internet. To use it one must [sign up](https://home.openweathermap.org/users/sign_up) for it on their website. Once signed up they will email you a unique code. This code will allow you to call their API. To call the API one simply sends a get request using this URL: https://api.openweathermap.org/data/2.5/weather? APPID=[yourCodeHere]. In our code a city ID and units was added to the query string. The city ID specifies that we want the data from Waco and the units specifies that we want our data to use imperial units. After the request is made the API returns the requested data in JSON format. To make a get request in our code, the “requests” library was used. The URL was passed into the get function. This gave us the JSON data. JSON data isn’t great to work with using python so the loads function from the “json” library was used to convert the JSON data to a Python dictionary named “weather”. Now that the data is in a dictionary it is easy to save certain values into variables using the dictionary’s indexes. This data is of course used to give a reading when the Internet is selected as the source, but it is also used for displaying the correct icon.

**Atmospheric Sensor**

The sensor uses the I2C protocol for communication. The code for this is in the bme280\_v2.py file which uses the “smbus” library for I2C communication. It was imported for code cleanliness. The most important function from it is readBME280. It returns the values for temperature, pressure and humidity in Celsius. These values are saved into respective variables for use in our code.

**Blynk**

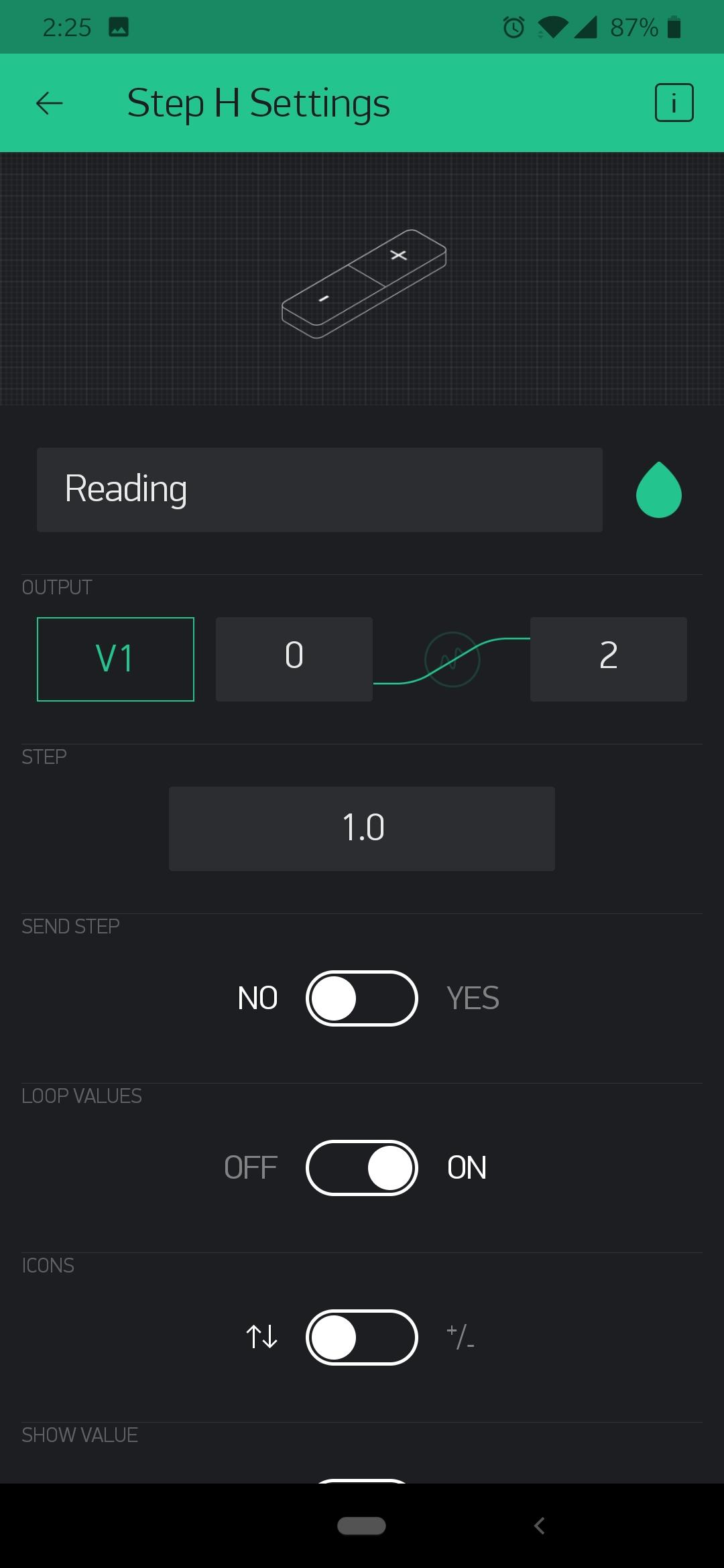
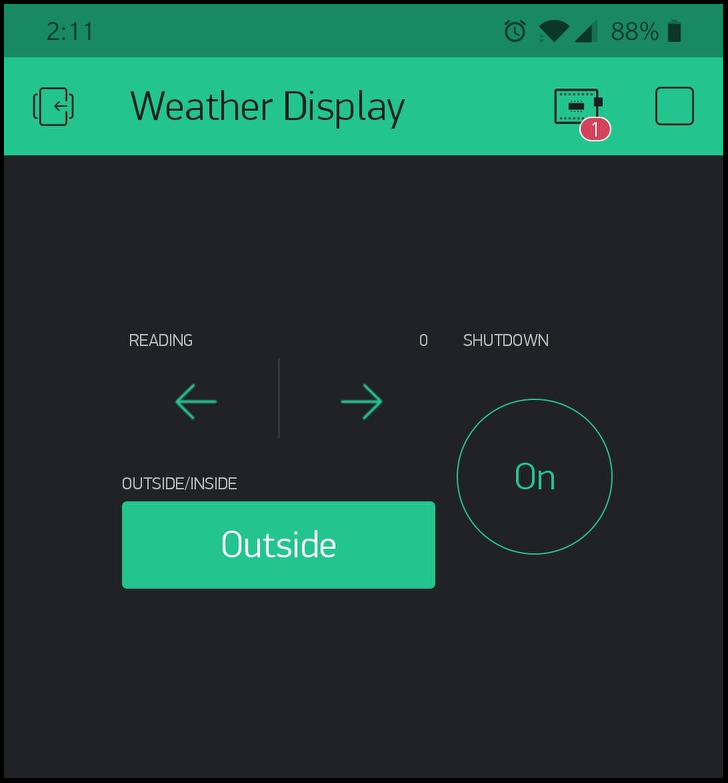
****

Figure 1 Figure 2

Blynk is a mobile app for IoT devices. It handles the backend for connecting a phone to hardware using the Internet or Bluetooth (in our case a Raspberry Pi to the Internet). To use, first you must download the app from the iOS or Play Store. Once downloaded you sign up with an email. You will then receive an authentication key which will be used in the code. The app is simple. It has a UI which allows you to drag and drop different items such as buttons onto a screen and then augment them with various settings. The Python code is also simple. First import the BlynkLib library. Then you simply instantiated a Blynk object with the key: (blynk = BlynkLib.Blynk(BLYNK\_AUTH)), create handler functions, and then call the run method (blynk.run()). To understand the function handlers, I will first describe more about the mobile app. The mobile app we made has two virtual buttons and a virtual +/- arrows button (Figure 1 and 2). Inside the settings of the buttons (Figure 2) a virtual pin is set to an integer value. For selecting the data source (Internet or sensor) this pin is V0. For the arrows it is V1 and for the shutdown button it is V2. The buttons are also set as switches in their respective settings. The function handlers start with this line: @blynk.VIRTUAL\_WRITE(VIRTUAL\_PIN\_NUMBER). They contain the function that is to be executed every time it’s called. For example, when I press the source button on my phone, I want the source value to toggle between 0 and 1 (Internet/Sensor). So, the function handler does exactly that. It also uses a global Boolean value so that the value can be accessed outside of the handler. The arrows handler works by receiving the selected value from the phone (0, 1, or 2) and then setting the “reading” value to ‘temp’, ‘humid’, or ‘pressure,’ respectively. The shutdown button handler simply tells the Raspberry Pi to turn off using the “subprocess” library when the button on the phone is pressed.

**Icon Logic**

This section will describe how the code decides which icon to display. The code has been designed to be able to display seven different icons (Cloud, Rain, Thunderstorm, Partly Cloudy, Clear, Clear at night, and Snow). Which icon displays largely depends on how to OpenWeatherMap describes different weather conditions. The documentation can be found [here](https://openweathermap.org/weather-conditions). It uses a “main” descriptor such as Rain or Clouds, and then has a sub descriptor “description.” It has quite a few different descriptors, so the code groups certain ones together. The code works as follows. First, it checks if the main descriptor is ‘Thunderstorms’, ‘Clouds’, ‘Atmosphere’, ‘Rain’, or ‘Drizzle.’ If it is then it will display the cloud icon drawn with graphics class. It will then check the main descriptor again, but this time only for ‘Rain’ or ‘Drizzle.’ If found it will display raindrops with the cloud icon, if not found then it will check for ‘Thunderstorm’ and display a lightning bolt with the cloud icon if found. Finally, if ‘Thunderstorm’ is not found it will check for the sub descriptor ‘few clouds’. If found it will display a partly cloudy which uses the main cloud icon. If a cloud icon should not be show it will check if main is ‘Clear.’ If it is it will decide on whether to show a sun or moon. If not, then it will check if it is ‘Snow’ and display it if found. That covers all “main” descriptors.

# Lessons Learned

1. You can power a board and other devices simultaneously from a HAT rather than having to power them separately. We spent a lot of time and a little bit of money trying to find the best ways to power all components in a clean manner only to realize later that the HAT could direct power to everything.
2. The LED Matrix code interferes with all GPIO pins. Not knowing this caused us to have to find a solution to switch weather modes without using a button as not GPIO pins were available. This also showed us how many alternatives there are to reading user inputs other than reading buttons such as using an app.
3. Different libraries are usually best suited for a specific board version. We spent a lot of time trying to implement libraries that weren’t made to run with the Raspberry Pi model we were using before realizing that there were more up to date libraries elsewhere.
4. Focusing on a concrete simplified design initially reduces the possibility of having to cut features in the future. Initially our project ideas were far too ambitious, but learning from past project experiences, we realized we wouldn’t have as much time as we felt we did. Because of this we spent a day or two just simplifying the design to something we were confident we could produce in time and then if we felt like adding extra features we could do that with extra time. This gave us incentive to nail the preliminary design, so we could add extra functionality like weather API data, app control, and a an aesthetic product.

# Final Notes

To recreate the portable weather display:

1. Construct the device as described in the hardware section
2. Download the rpi-rgb-led-matrix directory from this [GitHub Repo](https://github.com/BaylorComputerEngineering/Embedded_S2_Team2_Fall2018) to your Raspberry Pi
3. After connecting I2C sensors, check I2C connections
   1. sudo i2cdetect –y 1
   2. Verify slave addresses match expected values (0x77)
      1. If it different, edit line 27 of bme280\_v2.py so that DEVICE matches the value just detected
4. Make sure you have Python 2.7
   1. Put the following in the Linux Command Line to update and install it
   2. sudo apt-get update && sudo apt-get install python2.7-dev python-pillow -y  
      make build-python  
      sudo make install-python
5. Set the Pi up to automatically connect to your preferred WiFi network
   1. Follow [this](https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md) tutorial to set up Wifi
   2. Set the network priority in the WiFi config to 1 (e.g. write Priority=1 under psk)
6. Download the Blynk app and follow the instructions in the Blynk section of the software description above to setup the app
7. Make sure your phone is connected to the same WiFi network as the RPi
8. To get the Raspberry Pi to run boot, add and save the following lines to the end of pi/.bashrc
   1. cd /home/pi/rpi-rgb-led-matrix/bindings/python/samples
   2. sudo ./graphics.py --led-rows=32 --led-gpio-mapping=adafruit-hat
9. Reboot the Pi
10. Press the Play Button in the Blynk app while Pi is booting up
11. The display should be running and connected to your phone
12. To edit the code type
    1. idle graphics.py
    2. While in the /home/pi/rpi-rgb-led-matrix/bindings/python/samples directory