UC Santa Cruz, S-Lab

Embedded Systems Prototyping

Creating a remote monitoring and control system using a Raspberry Pi

November 13, 2016

Contents

1 Introduction

1.1 Summary

This manual will detail the process of designing and implementing a remote monitoring and control system using a Raspberry Pi (RPi). This includes initial config and installation of the OS, communicating with the RPi over a TTL Serial connection, configuring SSH and a public IP to remotely connect, communicating with several sensors over SPI and I2C protocols, sending emails and HTTP requests, and using a Finite State Machine (FSM) to define the autonomous behavior of the RPi. The RPi is a very useful instrument for Internet of Things (IoT) type applications, given that it is basically fully featured computer with a small form factor. As a result, many designs are much easier and quicker to implement on an RPi than on a microcontroller. RPis are ideal for embedded applications in which there is not much constraint on the size and power consumption of the system. Upon completion of this module, you should be able to implement not only the particular system covered in this manual, but also another system of your own design.

Prerequisites: It is recommended that you have at least introductory-level programming experience prior to taking this module

1.2 Definitions and Terminology

1. RPi (Raspberry Pi): An awesome prototyping board that features a plethora advances hardware and utilities made incredibly easy to interface with
2. NOOBS (New Out Of Box Software): A software package for the RPi that makes installing the OS on an unconfigured RPi "much, much easier."
3. SSH (Secure SHell): A protocol commonly used for remotely authenticating into a computer
4. TTL (Transistor-Transistor Logic): A serial protocol used for communication between two systems using only a RX (receive) and TX (transmit) signal
5. SPI (Serial Peripheral Interface): A communications protocol that uses a master-slave architecture. More details can be found.
6. I2C (Inter-Integrated Circuit): A communications protocol that allows for multiple master and slave nodes in the network. More details can be found.
7. FSM (Finite State Machine): A programming design that uses inputs to determine transitions between states, and particular outputs during the process. They are very useful for defining the lifetime of a system and how it will react to all possible inputs. More details can be found.
8. HTTP (Hypertext Transfer Protocol) Request: The protocol used across the web for the majority of all communications. This manual will detail the process of using a POST request type to send data from the RPi to a remote destination.
9. Shell CLI (Command-line Interface): An instance of a program that allows a user to directly interface with services provided by the OS via a suite of commands.
10. LAN (Local Area Network): A computer network that, among other things, allows computers in the same vicinity to communicate with one another. Wireless printers in your home use the LAN set up by your router to communicate with your personal computer.
11. WAN (Wide Area Network): A computer network that, among other things, allows computers to communicate with one another, regarldess of geographic proximity. The internet is a form of WAN.

1.3 Materials

Note: If you are taking this module via the S-Lab at UC Santa Cruz, all of the essential materials will be provided during the first lab session.

1. Essential:
   (a) Raspberry Pi 3 Model B (Most of this manual will apply to other versions and models as well, however it is tailored to this particular version and model of the RPi.)
   (b) Micro SD Card (at least 8GB in size)
   (c) USB to TTL Serial Cable (aka Console Cable)
   (d) USB to Micro USB Cable (you most likely already use one for a phone or other device)
   (e) Humidity and Termperature Sensor
   (f) 7-Segment Display
   (g) Servo Motor
   (h) Breadboard
   (i) Jumper Wires
   (j) 4xAA Battery Holder
   (k) HDMI to HDMI cable

(l) Keyboard
(m) Mouse
(n) Monitor
(o) A Mac OSX, Windows, or Linux computer

2. Useful:
   (a) ...

2 Setting up the Raspberry Pi (RPi)

Note: Any `CLI output text` colored in `red` indicates that it will likely be different from your.

2.1 Using NOOBS to install the OS (Raspbian)

The RPi comes without any installed or configured OS. Follow the tutorial here to perform the following steps in order to get the RPi up and running:

1. Download NOOBS
2. Format your micro SD card
3. Copy NOOBS onto the micro SD card
4. Hook the RPi up to the monitor (via HDMI), keyboard , and mouse
5. Finally install the Raspian OS through the NOOBS GUI

Note: Be sure to choose Raspbian as the OS to install on your RPi, as it will be the OS used for the rest of this manual.

2.2 Some RPi basics

2.2.1 Connecting to a wireless network

Through the Raspbian GUI, there is a wifi icon in the upper right of the screen with and clicking on this will take you through a straight-forward prompts to connect to a wireless network. If you are currently using your RPi in "Headless" mode (without a monitor..and GUI), this guide details the process of connecting to a wireless network via the command line (command line access is described in the section 2.3 and section 2.4)

2.2.2 Shutting down

It is important that the RPi is always properly shutdown prior to being disconnected from a power source. If it is not, there is a chance that the OS and other persisted data on the micro SD card will be corrupted. You can properly shutdown the RPi via command line or the Raspbian GUI (Graphical User Interface):

1. Command Line: Run `sudo shutdown now` and wait until you receive an output of `[ 1837.852002] reboot: Power down` in the shell before you disconnect the RPi from its power source.
2. GUI: navigate to ***Menu » Shutdown*** and then click on the *Shutdown* option

2.3 Wired connection over TTL Serial

It is often convenient to be able to interface with the RPi with only your personal computer, instead of with the monitor, keyboard, and mouse that were required when installing the OS. A TTL Serial connection between your RPi and computer will allow you to do everything you would normally be able to do through the Raspbian GUI, through the Raspbian CLI via a serial terminal on your computer.

2.3.1 Enable the interfaces

To connect over TTL Serial, we must first enable the Serial interface on the RPi. Later on we will also need the SSH, SPI, and I2C interfaces so we'll enable those now as well. Navigate to ***Menu » Preferences » Raspberry Pi Configuration*** as shown in the Figure 1 below.

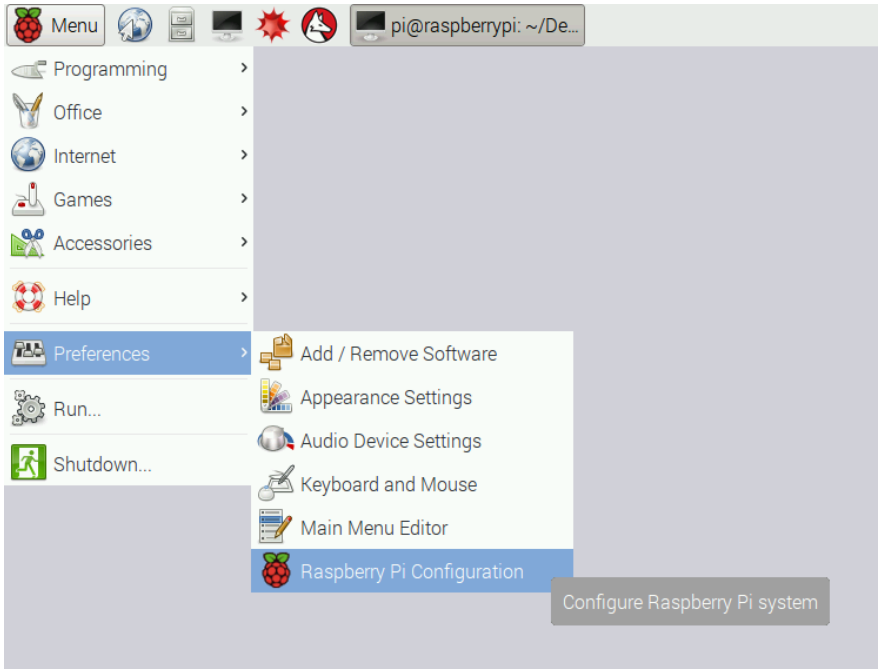

Figure 1: RPi Configuration at ***Menu » Preferences » Raspberry Pi Configuration***

Next, navigate to the ***Interface*** tab and enable all of the necessary interfaces (SSH, SPI, I2C, and Serial) as shown in Figure 2 below.

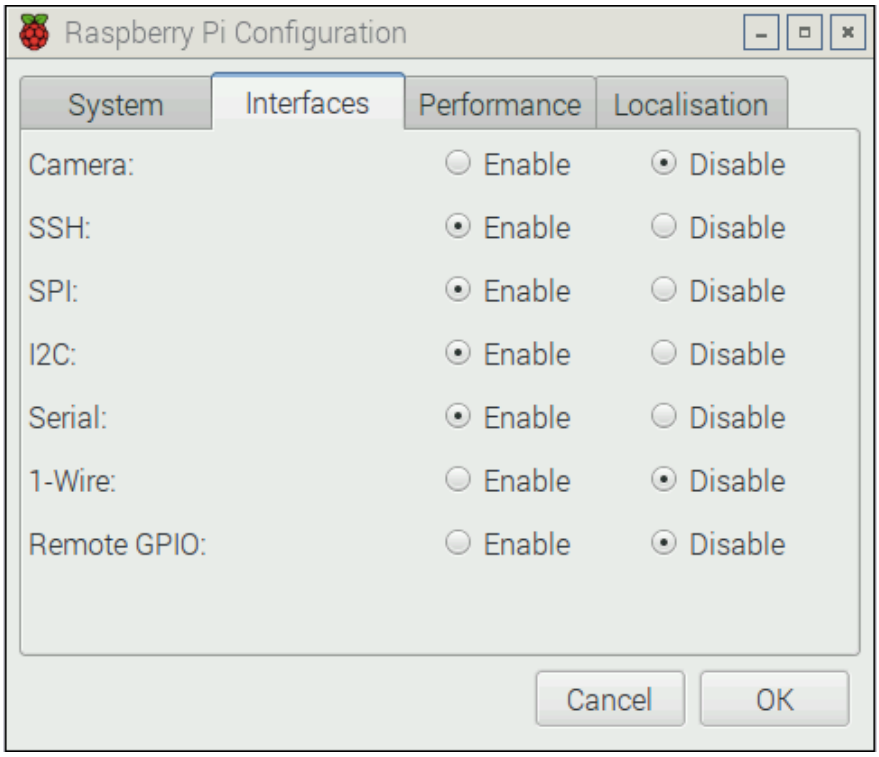

Figure 2: Necessary interface configuration for the RPi

Now press *OK*, and reboot the RPi by going to ***Menu » Shutdown*** and clicking *Reboot*.

2.3.2 Disable bluetooth to free up GPIO 14 & 15 (TX, RX)

On the Pi 3 Model B, it so happens that the built-in Bluetooth chip consumes GPIO (General-Purpose Input/Output) 14 & 15 (pins 8 & 10) that correspond to TX and RX, respectively, that are required for the serial connection. We must now disable this in order to free up these pins. First, open the *Terminal* application to get a CLI Shell into Raspbian. The application can be navigated to by ***Menu » Accessories » Terminal***. You should now see a window that looks like the following in Figure 3.

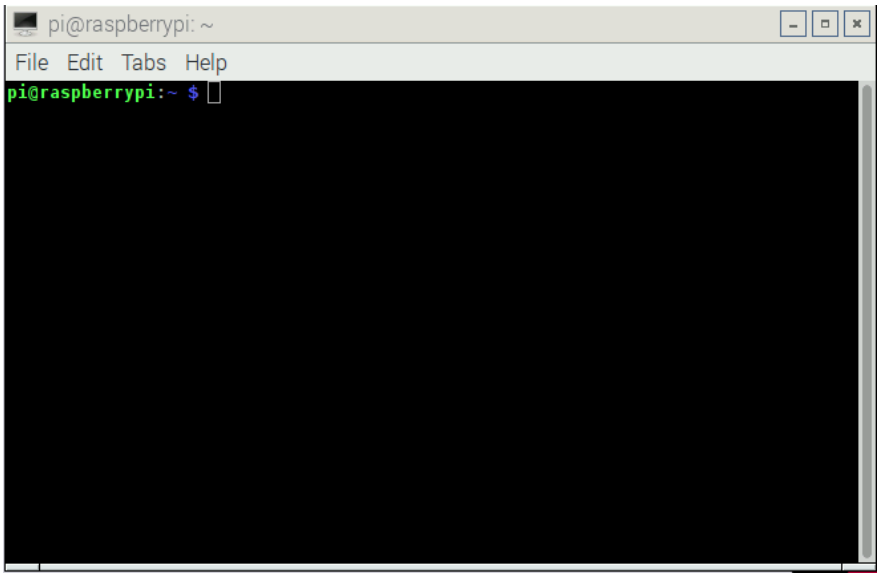

Figure 3: Raspbian Terminal App (Shell CLI)

In the shell, you can enter unix style commands indicated in this manual by `this styling`. Run the following commands to ensure that your RPi is up to date.

Update environment

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
sudo rpi-update
```

Now to disable bluetooth, run the following lines which append the distable-bt setting to the RPi's config file and stop the BT modem from trying to use UART, respectively.

Disable bluetooth

```
sudo echo "dtoverlay=pi3-disable-bt" » /boot/config.txt
sudo systemctl disable hciuart
```

Finally, reboot the RPi to allow the changes to take effect.

2.3.3 Download and install the drivers for the TTL Serial cable

There are specific drivers required for this protocol that are available for Mac OSX, Windows, and Linux. Follow the appropriate guide below to download and install the drivers for your computer.

- ***Mac OSX*** Download/Installation Guide for TTL Serial Driver
- ***Windows*** Download/Installation Guide for TTL Serial Driver
- ***Linux*** Download/Installation Guide for TTL Serial Driver

2.3.4 Connect the leads and USB port

Now, connect each of the four colored-coded leads on the TTL Serial cable to their corresponding pins on the RPi's GPIO header. VCC and GND have multiple options for pins, but TX and RX must be placed on GPIO 14 & 15. The following is a recommended configuration of the TTL Serial cable's leads:

- **VCC**: Pin **4** (5V)
- **GND**: Pin **6** (Ground)
- **TX** : Pin **8** (GPIO 14)
- **RX** : Pin **10** (GPIO 15)

Note: Depending on where you purchased your cable, the leads are likely colored for **TX either white or orange** and **RX either green or yellow**. VCC is always colored red and GND always black.

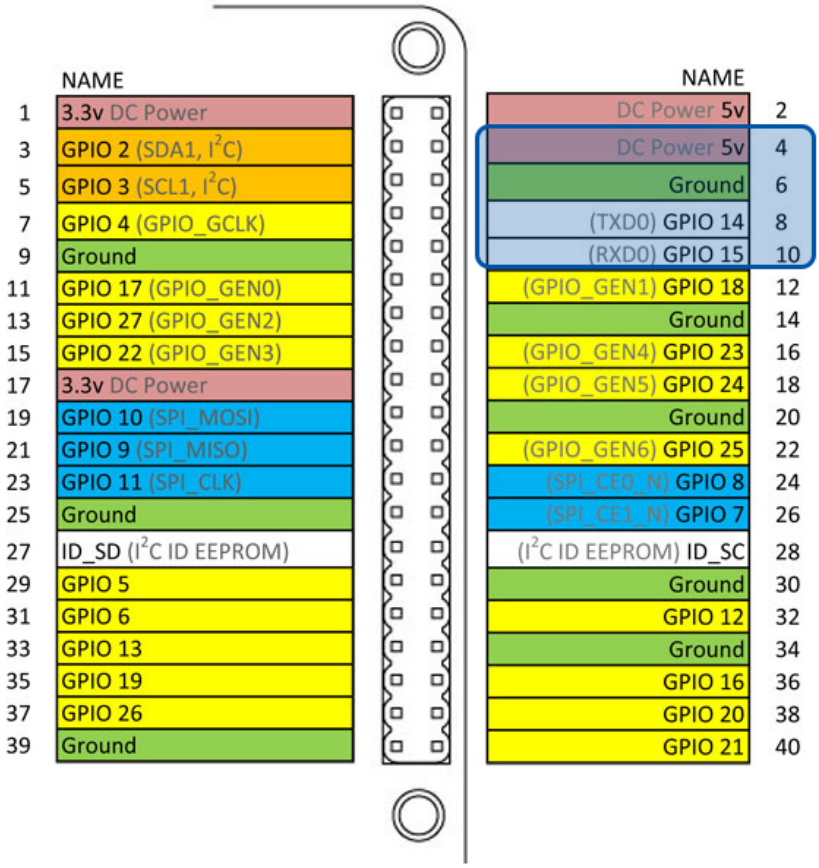

Figure 4: Raspberry Pi 3 Model B, GPIO header. Photo credit: mcmelectronics.com

DO NOT SUPPLY POWER OVER BOTH MICRO USB AND THE SERIAL CABLE and double check that the leads of the serial cable are connected properly; the RPi is not especially tolerant when it comes to power and can be fried somewhat easily. Once you're certain the leads of the cable are properly connected, plug the USB into your computer and you should see the RPi's power LED indicator turn on as it did when supplied power over micro USB.

2.3.5 Testing the connection

Now, all that's neccessary to connect over the serial connection is to open a serial terminal and connect to the right port using **115200 baud**. *Screen* is a command line serial interface and comes with recent releases of *Mac OSX*. It does not come with all distributions of *Linux*, but can easily be installed using `sudo apt-get install screen`. *Windows* requires an application called *PuTTY* and the process

takes slightly longer. This is a guide detailing the complete steps for each operating system.

Note: As indicated in the aformentioned guide, the login credentials for a user with root privileges on the RPi is by default:

- username: **pi**
- password: **raspberry**

2.4 Wireless connection over LAN via local IP & SSH

Sometimes it is more convenient to connect to the RPi over SSH because of the extra features it provides. Included in these features is a protocol called SFTP (SSH File Transfer Protocol) that makes it quick and easy to transfer files between your computer and the RPi. To connec to your RPi in this manner requires the use of a LAN (Local Area Network) on which both your personal computer and the RPi must be one.

Note: This method only works for local networks and cannot be used to connect to your RPi if both your computer and the RPi are not connected to the same network. See the following section if you wish to connect to your RPi from a different network than what the RPi is on.

2.4.1 Connect the RPi to a local network

To connect over SSH or SFTP, the RPi must first be connected to a local network (which was done earlier in this manual)

2.4.2 Determine the RPi's IP address

After connecting to your local network, the RPi will be allocated an IP address. To find this, connect to the RPi over serial (as described in section 2.2). Then, in the command line run `hostname -I`. This should output something of the form **`10.0.1.26`** `2601:1c0:5101:1ea8:e2af:7dc1:9da6:2025`, where the bolded portion is the RPi's IP address. Copy this address to your clipboard.

2.4.3 Making the SSH connection

In the command line on your computer, SSH into the RPi by running `ssh pi@<IP-ADDRESS>` where `<IP-ADDRESS>` is replaced by the IP address you copied earlier. After running the command, you will be prompted for the password which, unless you changed it, will be its default of **raspberry**. Once authenticating, you have access command line access to the RPi (same as was achieved over the TTL Serial cable). This connection is, however, dependent on a the existence of a network, wheras the serial connection is not. A complete guide for this method can be found here.

2.4.4 Using SFTP to transfer files

In a similar way, you can remotely access the RPi over SFTP, which makes file transfer between your computer and the RPi much easier. SFTP into the RPi by running `sftp pi@<IP-ADDRESS>` in the command line of your computer. Then, you can use some of the standard UNIX commands to navigate through both the file system of the RPi and your computer (except that all commands need to be prefaced with **l** (letter "ell") indicating the local file system). The commands `get` and `put` can be used to transfer files from the RPi and to the RPi, respectively. A complete guide to using this method can be found here.

»»

TODO : - Add "Wired connection over direct-ethernet via private IP & SSH"

««

»»

TODO : - Add "Wireless connection over global network via public IP & SSH"

«««

2.5 Wireless connection over WAN via Weaved & SSH

Often times you are not in range of the network your RPi is on but still want to connect to it. In this case, where you need to connect to your RPi through a WAN (Wide Area Network), you can use a service called Weaved (https://www.weaved.com). Weaved handles all the potential security risks involved with opening your RPi to the public, and allows you to SSH into it just as is possible through a LAN.

2.5.1 Setup an account at Weaved.com

First, setup a free account at https://developer.weaved.com/portal/index.php.

2.5.2 Installed Weaved on the RPi

Next, download and install Weaved onto the RPi but running the following code in the command line (after accessing the command line of the RPi through TTL Serial or LAN SSH):

Download and install Weaved to the RPi

```
sudo apt-get update
sudo apt-get install weavedconnectd
sudo weavedinstaller
```

After running the last command, you will be taken through a series of prompts in order to configure Weaved on your RPi. Perform the following, in the following order, in order to configure Weaved for SSH on your RPi:

1. Select #1 *Sign in to existing account*
2. Enter your Weaved login credentials
3. Select #1: *Attach/reinstall Weaved to a Service*
4. Select #1: *SSH on default port 22*
5. Enter: y (for "yes continue")
6. Enter name: SSH-Pi (and then wait for installation to finish)
7. Select #3: *Exit*

Weaved should now be properly configured on your RPi.

2.5.3 Connecting to the RPi over SSH with Weaved

Follow this procedure to SSH into your RPi with Weaved:

1. Login to your Weaved account here: https://developer.weaved.com/portal/login.php.
2. Click on your service named "SSH-Pi"
3. Copy the text in the field labeled "For pi username"
4. Paste this text into the command line of your personal computer and run it

You should now be prompted for the password of your RPi's *pi* user, after which you will have a command line into your RPi. *Note*: The host and port copied from Weaved.com are dynamically generated and only valid for 30 minutes (with the free version). To reconnect after 30 minutes, you will have to repeat steps 2 & 3 above. The paid versions of Weaved extend the lifetime of SSH credentials. Also, they have a free mobile app that in which you can also issue SSH credentials for your RPi.

3 Sensor inputs

3.1 Communicating with the si7021 humidity and temperature sensor (over I2C)

The RPi is nothing more than your personal computer if not given the chance to interact with its environment. The 40 pins in the GPIO header are what allow the RPi to do this. They provide a number of interfaces, one of which (Serial) was already implemented earlier in this manual, that can be used to communicate with a variety of sensors. I2C is the protocol that the humidity and temperature sensor (si7021) uses, and so that is how we will implement it in this section.

Note: This section is tailored to connecting to the RPi over TTL Serial, however the same procedure can be carried out over SSH. It is also tailored the *GNU nano* editor. You can find out more about how to use nano in this guide.

3.1.1 Configuring I2C

I2C must be configured, as by default, it is neither enabled nor is its kernel module loaded at boot time. As a result of some prior steps in this manual, the steps in this section are likely unnecessary, but it's a good idea to go through them anyway. Make sure that you have a serial connection with the RPi. If you're unsure of this, refer back to section 2.3.

First, make sure that I2C is not blacklisted by inspecting the contents of the file at ***/etc/modprobe.d/raspi-blacklist.conf*** by running `sudo nano /etc/modprobe.d/raspi-blacklist.conf`. If there is a line of the form `blacklist i2c-bcm2708`, comment it out by writing a hash (#) at the beginning of the line. If there is no line of this form, then I2C is already enabled. You can save and exit *nano* by typing **`control-O <enter> control-X`**.

Next, make sure that the I2C kernel module is loaded at boot time by running `sudo nano /etc/modules` to open the file and append to it the line `i2c-dev` if it does not already exist.

3.1.2 Install packages and probe the I2C interface

Run the following:

Update environment and install necessary packages

```
sudo apt-get update
sudo apt-get install i2c-tools
sudo apt-get install python-smbus
```

Now, in case the *pi* user is not by default a member of the I2C group, run `sudo adduser pi I2C`. Finally, probe the I2C interface to make sure that everything has been configured properly by running `i2cdetect -y 1`. This should output something of the following form:

I2C interface detection output (si7021 **NOT** wired)

```
pi@raspberrypi:~/Desktop/slab/tmp $ i2cdetect -y 1
...  0 1 2 3 4 5 6 7 8 9 a b c d e f
00:  - - - - - - - - - - - - - - - -
10:  - - - - - - - - - - - - - - - -
20:  - - - - - - - - - - - - - - - -
30:  - - - - - - - - - - - - - - - -
40:  - - - - - - - - - - - - - - - -
50:  - - - - - - - - - - - - - - - -
60:  - - - - - - - - - - - - - - - -
70:  - - - - - - - -
```

This output indicates that none of the 128 (hexadecimal 0x0 to 0x7f) I2C addresses are being used (the si7021 sensor is not connected) but the I2C interface has been properly configured.

3.1.3 Wiring the si7021 sensor to the RPi

We must now wire the temperature and humidity sensor to the RPi. To do this, we'll need the breadboard and jumper cables included in the materials list at the beginning of this manual. Using these materials, connect the si7021 to the RPi according to the wiring diagram below. But first, make sure that you are aware of the following:

1. **DO NOT** change the wiring of your system while the RPi is still powered on. Instead, shutdown the pi, then unplug it from the power source. This is done to prevent short circuits and other dangers while re-wiring your system.

2. **DO NOT** connect the si7021 or any other 5V intolerant device to a 5V output on the RPi. Many sensors are designed to work using 3.3V, and can be easily fried if this voltage is exceeded.

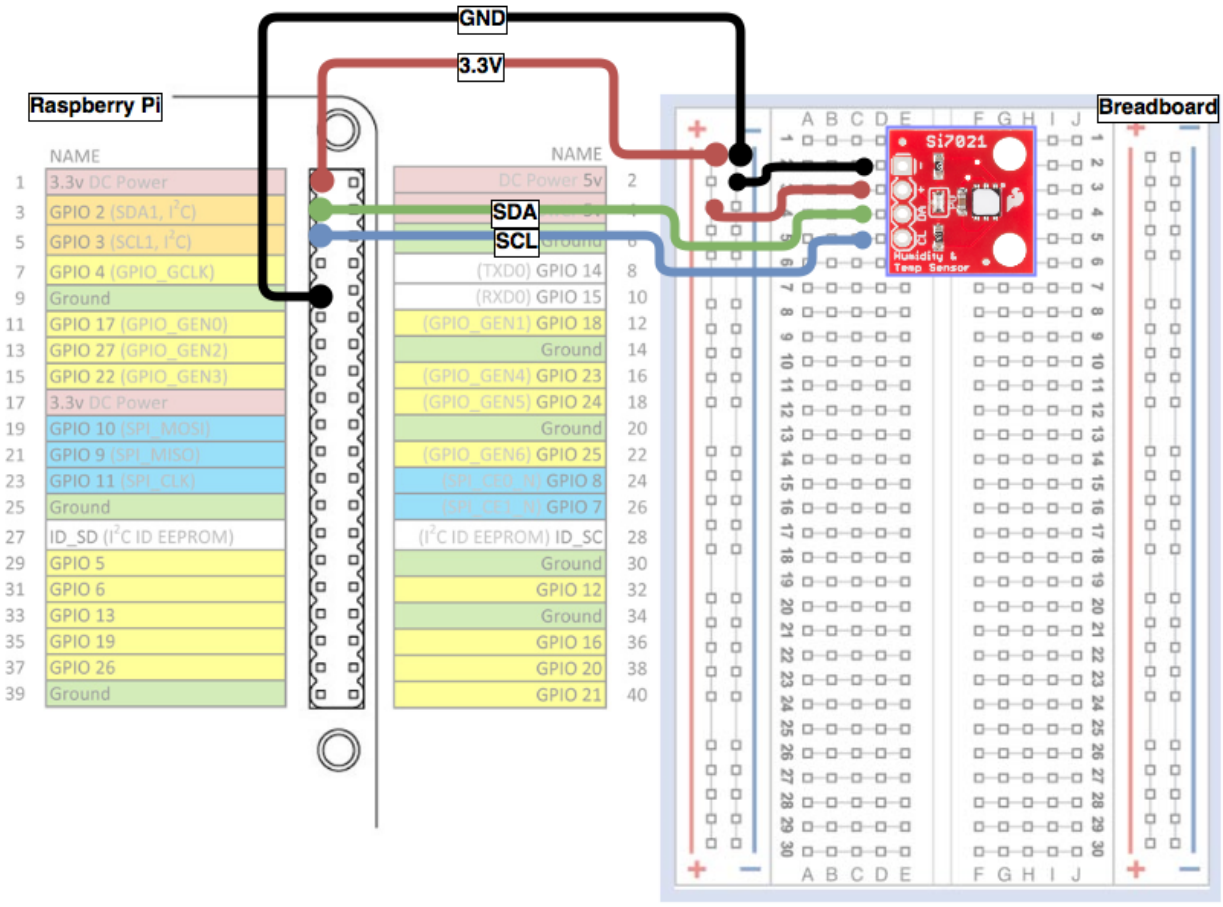

Figure 5: Wiring diagram for connecting the si7021 to the RPi over I2C

3.1.4 Testing the wiring

There are many ways to test for faulty wiring, however in this case, it is easiest to just the I2C interface detection tool to confirm that the si7021 is properly wired. First, plug the Pi back into your computer and get the serial connection running again. Then, run the same command from before `i2cdetect -y 1`. The output should now indicate that there is an I2C device connected using address 0x40, having somethign similar to the following form:

I2C detection output (si7021 wired)

```
pi@raspberrypi:~/Desktop/slab/tmp $ i2cdetect -y 1
...   0 1 2 3 4 5 6 7 8 9 a b c d e f
00:   - - - - - - - - - - - - - - - -
10:   - - - - - - - - - - - - - - - -
20:   - - - - - - - - - - - - - - - -
30:   - - - - - - - - - - - - - - - -
40:   40 - - - - - - - - - - - - - - -
50:   - - - - - - - - - - - - - - - -
60:   - - - - - - - - - - - - - - - -
70:   - - - - - - - -
```

If the detection utility does not indicate that I2C address 0x40 is active, double check that you have properly wired the sensor to the RPi. If the problem persists (*i2cdetect* does not indicate address 0x40), it

is possible that you have fried the sensor and/or the RPi, but while troubleshooting this should be a last resort.

3.1.5 Daemonizing the *pigpiod* program

Before we can talk with the si7021 from the context of Python (the programming language used in this manual), we must first daemonize the program that coordinates with the Python I2C module to make it easy to communciate with sensors over I2C via Python. In the command line, run the following commands:

Prep the environment and locate *pigpiod*

```
sudo rpi-update
sudo pkill pigpiod
whereis pigpiod
```

This will output something of the form:

`whereis pigpiod` command output

```
pi@raspberrypi:~$ whereis pigpiod
pigpiod:  /usr/bin/pigpiod /usr/man/man1/pigpiod.1.gz
```

Now, copy the path this outputs that has a similar form as the portion highlighted above (the path is likely, but not necessarily, defaulted to ***/usr/bin/pigpiod***). Then run the following comand:

Open the root crontab

```
sudo crontab -e
```

This will now open the root crontab file in which you can indicate that the pigpiod program should be run on reboot. Do this by appending the following line to this file (where you paste in the path you copied where it says <your-path-to-pigpiod>):

Daemonize *pigpiod* command

```
@reboot <your-path-to-pigpiod>
```

Now save the file and exit the editor. To allow the changes to take effect, reboot the RPi by running `sudo reboot` and login again once the prompt appears. Test that the *pigpiod* program was successfully daemonized by running `pgrep pigpiod`. If this outputs a single number (its process id) to the screen, then the program was sucessfully daemonized. If not, review the previous steps to make sure that nothing was missed.

3.1.6 Talking to the si7021 with Python

The next natural step is to talk to the newly installed sensor from the context of a Python script to get back readings for humidity and temperature. In the command line, create and navigate to a workspace for your python code on the desktop by running the follow:

Create and navigate to your workspace

```
mkdir -p ~/Desktop/python-code/test-scripts
cd ~/Desktop/python-code/test-scripts
```

Now create and edit a file using the same *nano* editor used before:

Create si7021 Python testing script file

```
nano si7021-test.py
```

Now copy and paste the following code into this file:

Python Code 1 Script for testing si7021

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#####################################################################
# Import libraries
#####################################################################
import time
import pigpio


#####################################################################
# Perform initializations
#####################################################################
i2c_bus_ID = 1
i2c_address = 0x40
degree_sign= u \N{DEGREE SIGN}  # unicode for the degree symbol

# instantiate the local RPi s GPIO
pi = pigpio.pi()
# humidity and temperature sensor - I2C interface detailed on pg. 18
# of si7021 datasheet
si7021 = pi.i2c_open(i2c_bus_ID, i2c_address)


#####################################################################
# Main logic
#####################################################################
while True:

    # Request rel. humidity w/ command:  Measure Relative Humidity, No
    # Hold Master Mode
    pi.i2c_write_device(si7021, [0xf5])
    time.sleep(0.1) # wait to ensure proper delay

    # Read the word written back by the si7021; returns (numBytesRead,
    # [msByte, lsByte])
    numBytesRead, word = pi.i2c_read_device(si7021, 3)

    # Format the word s rel. hum. payload
    rh_code = (word[0] << 8) + word[1]

    # Convert the payload to relative (%) humidity (pg. 21 of si7021
    # datasheet)
    rh_value = ((125.0 * rh_code) / 65536.0) - 6.0
```

```
    # -----------------------------------------------------------------

    # Request temperature w/ command:  Measure Temperature, No Hold
    # Master Mode
    pi.i2c_write_device(si7021, [0xf3])
    time.sleep(0.1) # wait to ensure proper delay

    # Read the word written back by the si7021; returns (numBytesRead,
    # [msByte, lsByte])
    numBytesRead, word = pi.i2c_read_device(si7021, 3)

    # Format the word s temperature payload
    t_code = (word[0] << 8) + word[1]

    # Convert the payload to temperature in Celcuis (pg. 22 of si7021
    # datasheet)
    t_value = ((175.72 * t_code) / (65536.0)) - 46.85

    # -----------------------------------------------------------------

    # Print and delay before the next iteration
    print ("Humidity: {:.2f}%, Temperature: {:.2f}" + degree_sign
           + "C").format(rh_value, t_value)
    time.sleep(1)

# house keeping
pi.i2c_close(si7021)
pi.stop()
```

Note: When copying the above code, make sure that you do not copy the page number in between the two pages.
Now save the file (**control-O <enter> control-X**) and run the script by executing the following in the command line:

Run the test script for si7021 sensor

```
python si7021-test.py
```

This should then continually output relative humdity and temperature readings like the following:

Test script output for si7021 sensor

```
pi@raspberrypi:~/Desktop/python-code/test-scripts$ python si7021-test.py
Humidity:  60.91%, Temperature:  19.33°C
Humidity:  60.91%, Temperature:  19.32°C
Humidity:  60.90%, Temperature:  19.35°C
...
```

»»

TODO : - Detail installation process for *pigpio*

««

»»

TODO: - Confirm that the *python-smbus* module is necessary
«««««««««««««««««««««««««««««««««««««««

4 Sensor outputs

In this section, it will be assumed that I2C is already fully configured on your RPi. If this is not the case, refer back to section 3.1.1. It is now necessary to set up some outputs for this system to get feedback from the sensor and interact with the environment.

4.1 Communicating with the 7-segment display (over I2C)

The 7-segment display (refferred from now on as the seg7) supports a variety of communications protocols, however for convenience, we will use I2C just as for the si7021. If you would like more information about the seg7 used in this manual, its documentation can be found here
https://github.com/sparkfun/Serial7SegmentDisplay/wiki/Serial-7-Segment-Display-Datasheet.

4.1.1 Wiring the seg7 display to the RPi

First, the seg7 must be wired to the RPi. As with the si7021, we'll need the breadboard and jumper cables included in the materials list at the beginning of this manual. Using these materials, connect the seg7 to the RPi, next to the si7021, according to the wiring diagram below. And again, make sure that you are aware of the following points prior to wiring this system:

1. **DO NOT** change the wiring of your system while the RPi is still powered on. Instead, shutdown the pi, then unplug it from the power source. This is done to prevent short circuits and other dangers while re-wiring your system.

2. **DO NOT** connect the si7021 or any other 5V intolerant device to a 5V output on the RPi. Many sensors are designed to work using 3.3V, and can be easily fried if this voltage is exceeded.

Some notes for wiring the seg7 display:

1. The seg7 is actually rated to a range of 2.4V to 6V and therefore works with a supply of either 3.3V or 5V from the RPi. We will use 5V, as it results in a brighter display.

2. The SDA and SCL I2C pins on the seg7 will be connected to the same I2C bus used for the si7021. The design of the 7-bit ID, I2C protocol allows for 2^7 (128) devices with unique IDs. The seg7 display defaults to an I2C address of 0x71, which does not conflict with the 0x40 address of the si7021.

3. Make sure that you do not place the seg7 in an upside down orientation on the breadboard. This would result in the the 5V/GND pins being switched with the SCL/SDA pins.

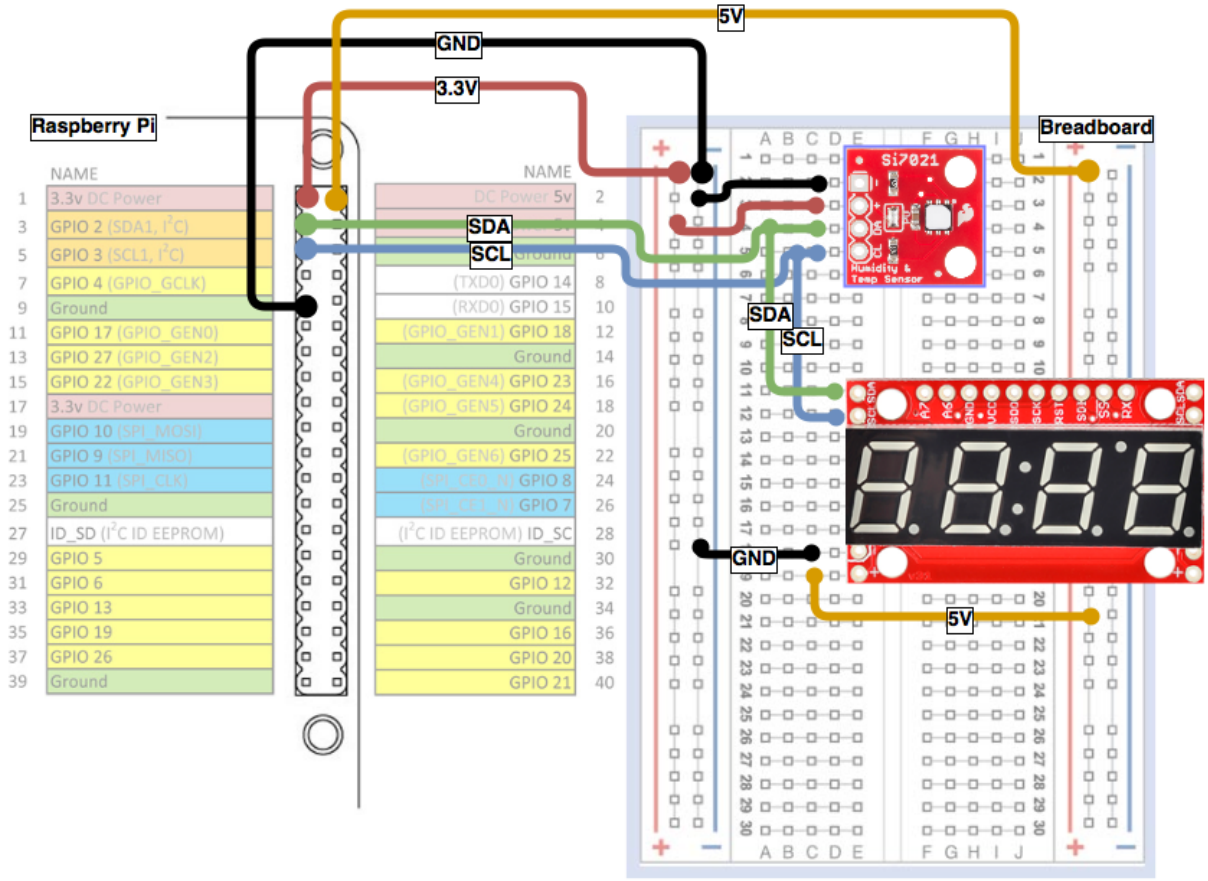

Figure 6: Wiring diagram for connecting the seg7 (along with si7021) to the RPi over I2C

4.1.2 Testing the wiring

As with the si7021, we must now check that the seg7 was wired properly to the RPi using the same *i2cdetect* tool in the command line on the RPi. Plug in the RPi back into your computer over the TTL Serial connection and login with the same default Raspbian login credentials for the *pi* user before. Then run the command `i2cdetect -y 1`. The output should now, if the seg7 was wired properly, indicate that there are two I2C devices connected over addresses 0x40 and 0x71, having a similar form as the following:

I2C detection output (seg7 and si7021 both wired)

```
pi@raspberrypi:~/Desktop/slab/tmp $ i2cdetect -y 1
...  0 1 2 3 4 5 6 7 8 9 a b c d e f
00:  - - - - - - - - - - - - - -
10:  - - - - - - - - - - - - - - - -
20:  - - - - - - - - - - - - - - - -
30:  - - - - - - - - - - - - - - - -
40:  40 - - - - - - - - - - - - - - -
50:  - - - - - - - - - - - - - - - -
60:  - - - - - - - - - - - - - - - -
70:  - 71 - - - - - -
```

If this command does not indicate that address 0x71 is active, double check that the seg7 was wired to the RPi properly.

4.1.3 Writing to the seg7 with Python

The seg7 can display most characters logically displayable on 7 segments. This includes characters such as **0-9** and **A-F**, but does not includes characters such as **v**. There are three basic operations that can be performed on the seg7 display:

1. Reset: Turn off all LED segments and return the cursor to the leftmost digit by writing the byte **0x76**.
2. Move Cursor: Move the cursor (where the next written character will be placed) to a particular position by writing the byte **0x79**, followed by another byte with a value of **0-3**. The data (second) byte, indicates the position to move the cursor to, where position 0 is the leftmost digit, and position 3 is the rightmost digit.
3. Write Character: Write a character to the digit at which the cursor is currently at by writing a byte of 0-15 (hex values 0-9,A-F) or a byte corresponding to a character's ASCII value (for example, 'u' or 'E'). As mentioned before, not all characters can be displayed.

Note: The cursor defaults to the leftmost position, however it is a good practice to ensure this position by setting it prior to writing characters to the display.

Now, move to the workspace you created earlier by running the following:

Navigate to your workspace

```
cd ~/Desktop/python-code/test-scripts
```

Then, create and edit a file using the same *nano* editor used before:

Create seg7 Python testing script file

```
nano seg7-test.py
```

Now, copy and paste the following code into this file:

Python Code 2 Script for testing seg7 display

```
#!/usr/bin/env python

########################################################################
# Import libraries
########################################################################
import time
import pigpio


########################################################################
# Perform initializations
########################################################################
i2c_bus_ID = 1
i2c_address = 0x71
enabled = True # boolean to control flashing
delay = 0.2 # period of alternating (1/2 total period)

# instantiate the local RPi s GPIO
pi = pigpio.pi()
```

```
# instantiate the seg7 I2C device
seg7 = pi.i2c_open(i2c_bus_ID, i2c_address)


#############################################################################
# Main logic
#############################################################################

# Set cursor position to leftmost digit (position 0)
pi.i2c_write_device(seg7, [0x79]) # cursor control byte
pi.i2c_write_device(seg7, [0x00]) # data byte specifying pos. 0

while True:
    # write "HELO" if is display enabled
    if (enabled):
        try: # I2C write can fail, this will catch the failure
            pi.i2c_write_device(seg7, [ H ])
            pi.i2c_write_device(seg7, [ E ])
            pi.i2c_write_device(seg7, [ L ])
            pi.i2c_write_device(seg7, [ O ])
            print("Hello")
        except pigpio.error as e:
            print("ERROR: failed to write to seg7 over I2C")

    # else clear the display
    else:
        try:
            pi.i2c_write_device(seg7, [0x76]) # reset byte
            print("Goodbye")
        except pigpio.error as e:
            print("ERROR: failed to write to seg7 over I2C")

    # toggle enabled
    enabled = not enabled

    # wait until next iteration
    time.sleep(delay)

# house keeping
pi.i2c_close(seg7)
pi.stop()
```

Note: When copying the above code, make sure that you do not copy the page number in between the two pages.
Now save the file (**control-O <enter> control-X**) and run the script by executing the following in the command line:

Run the test script for seg7 sensor

```
python seg7-test.py
```

This should then continually output the lines "Hello" and "Goodbye" like the following:

Test script output for seg7 sensor

```
pi@raspberrypi:~/Desktop/python-code/test-scripts$ python seg7-test.py
Hello
Goodbye
Hello
...
```

And you should see the text "HELO" flashing on the 7 segment display that you connected to your RPi.

4.2 Using the servo motor

This section will cover the implementation of a servo motor using PWM (Pulse Width Modulation) via the Python RPi.GPIO package. For more information on controlling servos with PWM, this wikipedia article has a very complete overview: https://en.wikipedia.org/wiki/Servo_control. And this contains a collection of helpful PWM resources: https://learn.sparkfun.com/tutorials/pulse-width-modulation/duty-cycle.

4.2.1 Overview of controlling the servo with PWM

PWM requires is a method of controlling the following properties of a signal in order to digitally vary its voltage or move a servo to a specific degree rotation.

1. Duty Cycle: This is the percentage of the signal that is high. A 50% duty cycle means that the signal will by high 50% of the time, and low the other 50%.
2. Frequency: This is the rate of oscillation of the signal from low to high. A common frequency for servos is 50Hz, but this can vary from servo to servo.

4.2.2 Wiring the servo motor to the RPi and battery pack

Most servo motors have the following pins on them:

1. VCC: The servo listed in Materials requires a positive supply of 4.5V to 6V
2. GND: This must be tied to GND on both the RPi and battery pack
3. Signal: This carries the PWM control signal with a specified frequency and duty cycle that tells the servo what degree rotation it should be at

To wire this system, first prepare the battery pack by placing four AA batteries into the harness. Since the servo used in this manual is rated for 4.5V to 6V, either 3 or 4 AA or AAAs will work, but the battery holder listed in the Materials section is designed for 4 AAs.

Next, place the male-male header onto the the female leads of the servo. Then, connect the servo, battery pack, and RPi together according to the following wiring diagram (this diagram was made on top of that for the si7021 sensor and seg7 display):

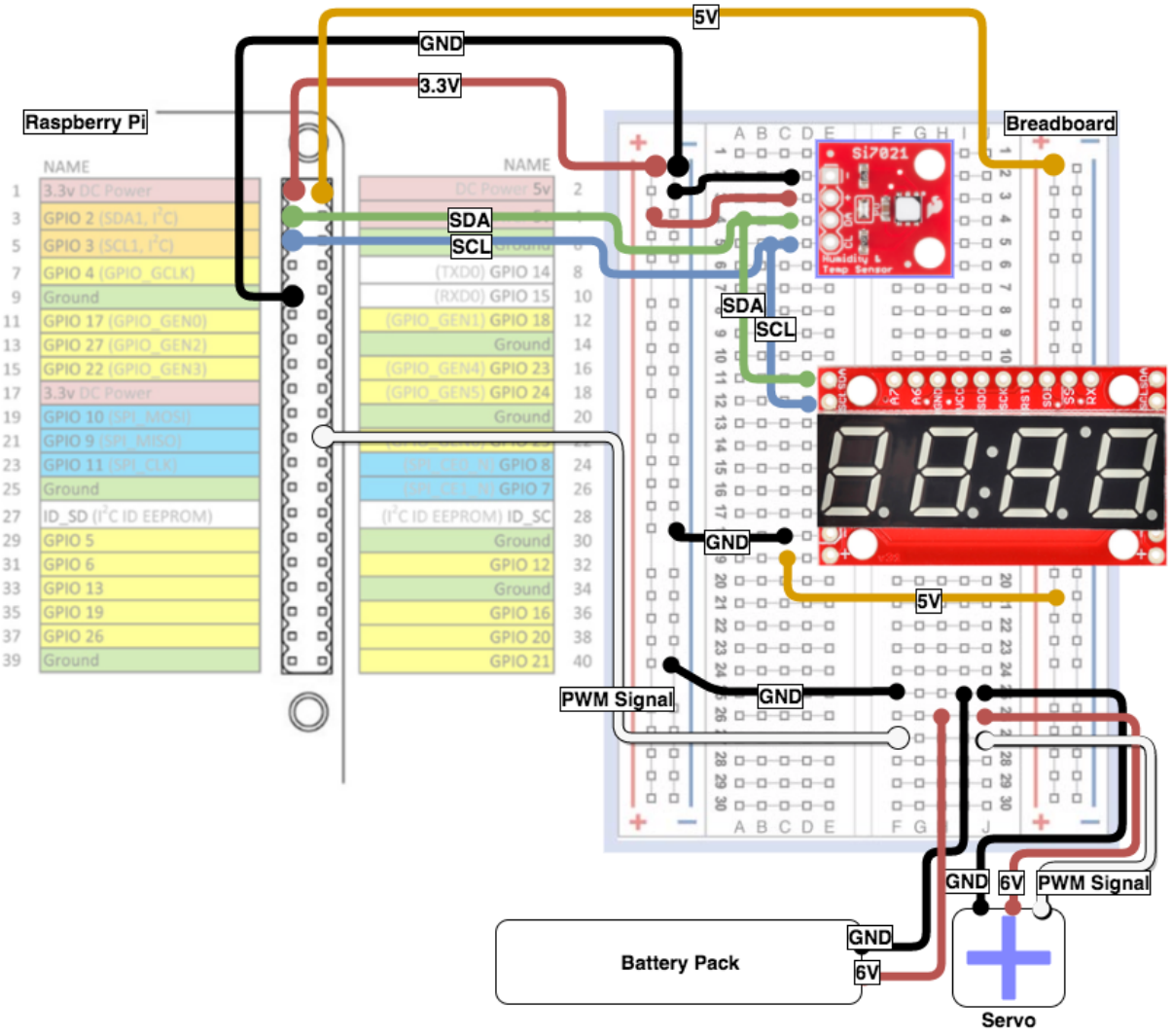

Figure 7: Wiring diagram for connecting the servo & battery pack (along with si7021 and seg7) to the RPi to control over PWM

4.2.3 Controlling the servo with Python

In Python, the servo can be moved to different angles by varying the duty cycle of the 50Hz (the frequency this servo requires) signal. It turns out that the range of duty cycles for this servo is ~2.1% (~210°) to ~12.4% (0°). To control the PWM signal from coming from the RPi using Python, we will actually be implementing software driven (as opposed to hardware driven) PWM. This means that we can map any of the RPi's GPIO to as many servos as there are GPIO, instead of only being able as many servos as there are PWM supported pins on a board. The draw back to software driven PWM is that it is only accurate within 100 µs, whereas hardware driven PWM is accurate within 1 µs. This innacuracy results in servo "jitter", but this can be mitigated with some techniques on the programming side. Now, as with the other Python sensor tests, get back into you RPi's command line and run the following to navigate back to your workspace:

Navigate to your workspace

```
cd /Desktop/python-code/test-scripts
```

The, create and edit a file using the same *nano* editor used before:

Create servo Python testing script file

```
nano servo-test.py
```

Now, copy and paste the following code into this file:

Python Code 3 Script for testing the servo motor

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#############################################################################
# Import libraries
#############################################################################
import time
import RPi.GPIO as GPIO

#############################################################################
# Perform initializations
#############################################################################
GPIO.setmode(GPIO.BCM) # pin numbering scheme that uses GPIO numbers
GPIO.setwarnings(False)
GPIO.setup(25, GPIO.OUT) # set GPIO 25 as output
servo = GPIO.PWM(25, 50) # instantiate PWM output to GPIO 25 @ 50Hz

degree_sign= u \N{DEGREE SIGN}  # unicode for the degree symbol
dc_min = 2.1 # the min duty cycle corresponding to 210deg rotation
dc_max = 12.3 # the max duty cycle corresponding to 0deg rotation

#############################################################################
# Main logic
#############################################################################
servo.start((dc_min + dc_max) / 2) # start at 105deg
time.sleep(1) # wait until rotation is finished
print("Rotated to 105" + degree_sign)

servo.ChangeDutyCycle(dc_min) # move to 210deg
time.sleep(1) # wait until rotation is finished
print("Rotated to 210" + degree_sign)

servo.ChangeDutyCycle((dc_min + dc_max) / 2) # move to 105deg
time.sleep(1) # wait until rotation is finished
print("Rotated to 105" + degree_sign)

servo.ChangeDutyCycle(dc_max) # move to 0deg
time.sleep(1) # wait until rotation is finished
print("Rotated to 0" + degree_sign)

# house keeping
servo.stop()
GPIO.cleanup()
```

Note: When copying the above code, make sure that you do not copy the page number in between the two pages.
Now save the file (**control-O <enter> control-X**) and run the script by executing the following in the command line:

Run the test script for the servo motor

```
python servo-test.py
```

This should then output text indicating that a series of rotations were performed like the following:

Test script output for servo motor

```
pi@raspberrypi:~/Desktop/python-code/test-scripts$ python servo-test.py
Rotated to 105°
Rotated to 210°
Rotated to 105°
Rotated to 0°
```

And the servo motor should rotate between these points.

5 Remote outputs

The RPi will need some way to to send you feedback of its state (and the state of its environment) when you are not in its general vicinity. This is the meaning of a remote output.

5.1 Sending an email with Gmail and Python

One method for remote output is sending an email. Python makes this process very easy, in fact, no additional packages need to be installed for this to work. Everything that is required is already installed and configured on your RPi.

5.1.1 Create a gmail account

This manual will be tailored to using a gmail smtp server, however a similar method can be used with other email providers. To authenticate into your gmail account from a Python script, you will need to change your security settings. This does change will make it easier for attackers to get into your account, so if you value your personal gmail account, it is suggested that you create a new, free one at https://www.gmail.com.

5.1.2 Enable "less secure apps"

To authenticate into your gmail account from a Python script, you need to turn on the "Access for less secure apps" setting in your gmail account here: https://www.google.com/settings/security/lesssecureapps. As mentioned in the previous section, this does less make your account more vulnerable to attacks, so it is advised that you create a new account. Make sure that when you enable this option via the above URL, you are logged in to the account you wish to use.

5.1.3 Write and execute the Python script

To send the email from your RPi, navigate back into your workspace after once again accessing your RPi's command line.

Navigate to your workspace

```
cd ~/Desktop/python-code/test-scripts
```

Then, create and edit a file using the same *nano* editor used before:

Create email Python testing script file

```
nano email-test.py
```

Now, copy and paste the following code into this file:

Python Code 4 Script for testing email

```
#!/usr/bin/env python

##########################################################################
# Import libraries
##########################################################################
import smtplib
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText

##########################################################################
# Perform initializations
##########################################################################
# connection fields
from_address = "XXXX@gmail.com" # the email account you created
from_pass = "XXXX" # the password for this new email account

# email fields
to_address = "XXXX@gmail.com" # the email where you will receive updates
email_subject = "RPi Update (test)" # the email s subject
email_body = "An update from your RPi" # the email s body

##########################################################################
# Main logic
##########################################################################
# populate the message
msg = MIMEMultipart()
msg[ From ] = from_address
msg[ To ] = to_address
msg[ Subject ] = email_subject
msg.attach(MIMEText(email_body,  plain ))

# Connect to the email server
server = smtplib.SMTP( smtp.gmail.com , 587) # hostname, port
server.starttls()

# Authenticate
server.login(from_address, from_pass)

# format and send the email
```

```
text = msg.as_string() # package the email into a single string
server.sendmail(from_address, to_address, text)
server.quit()
```

Note: When copying the above code, make sure that you do not copy the page number in between the two pages.
Now save the file (**control-O <enter> control-X**) and run the script by executing the following in the command line:

Run the test script for email

```
python email-test.py
```

You should then receive an email at the *to_ address* you specified, after the script has finished running.

6 The state machine

A state machine is a programming model that discretely defines the behavior of a system over its lifetime. Simply put, it is where we define what our system does and when.

6.1 Designing the state machine

The are a number of types of state machines, but one of the more common, and the one we will use, is a Finite State Machine (FSM). Exactly as its name describes, an FSM has a finite number of states that can be transitioned to. Our state machine will also have the sub-classification of a Meally, where its current output(s) is/are determined by both the current input(s) and the current state. If you wish to find out more information on FSMs beyond the scope of this manual, this article has a very complete

6.1.1 Outlining the high-level behavior

At a high-level, the state machine will function such that it outputs the following under the corresponding conditions.

1. 7 segment display:
   (a) Will alternate between displaying the current temperature and humidity, displaying each for 2 seconds before switching to the other. Humidity will have a precision of 2 after the decimal, and temperature will have a precision of 1 after the decimal.
   (b) Upon a change of state (of the FSM) it will display the new state's 4-digit code for 3s, before returning back to displaying the humidity/temperature.
2. Servo motor:
   (a) Begins at 105°rotation as a neutral position
   (b) After entering the HOT or DRY state, it will rotate to 0°, hold for 3 seconds, then return to 105°
   (c) After entering the COLD or WET state, it will rotate to 210°, hold for 3 seconds, then return to 105°
3. Email:
   (a) Upon transitioning from the IDLE state to a WET/DRY/HOT/COLD state, it will email you such
   (b) Upon transitioning from a WET/DRY/HOT/COLD state to the IDLE state, it will email you such

6.1.2 The state diagram

The next step in the design is to synthesize these high-level behaviors into a finite set of states, transitions, inputs, and outputs that discretely define this behavior. The primary dangers in this step include **1) defining transitions based on combinations of inputs that aren't mutually exclusive** and **2) not spanning all possible combinations of inputs with the transitions defined**. If, for a state in which only n boolean inputs are of concern, a transition doesn't evaulate for each 2^n possible combination of inputs, the FSM will exhibit undefined behavior. On the other hand, if each defined transition is not a unique combination of inputs, then it is possible for multiple defined transitions to evaluate which will also cause undefined behavior. Keeping these two dangers in mind, the following is a state diagram for the FSM we will use, with a fully spanned set of unique transitions for each state.

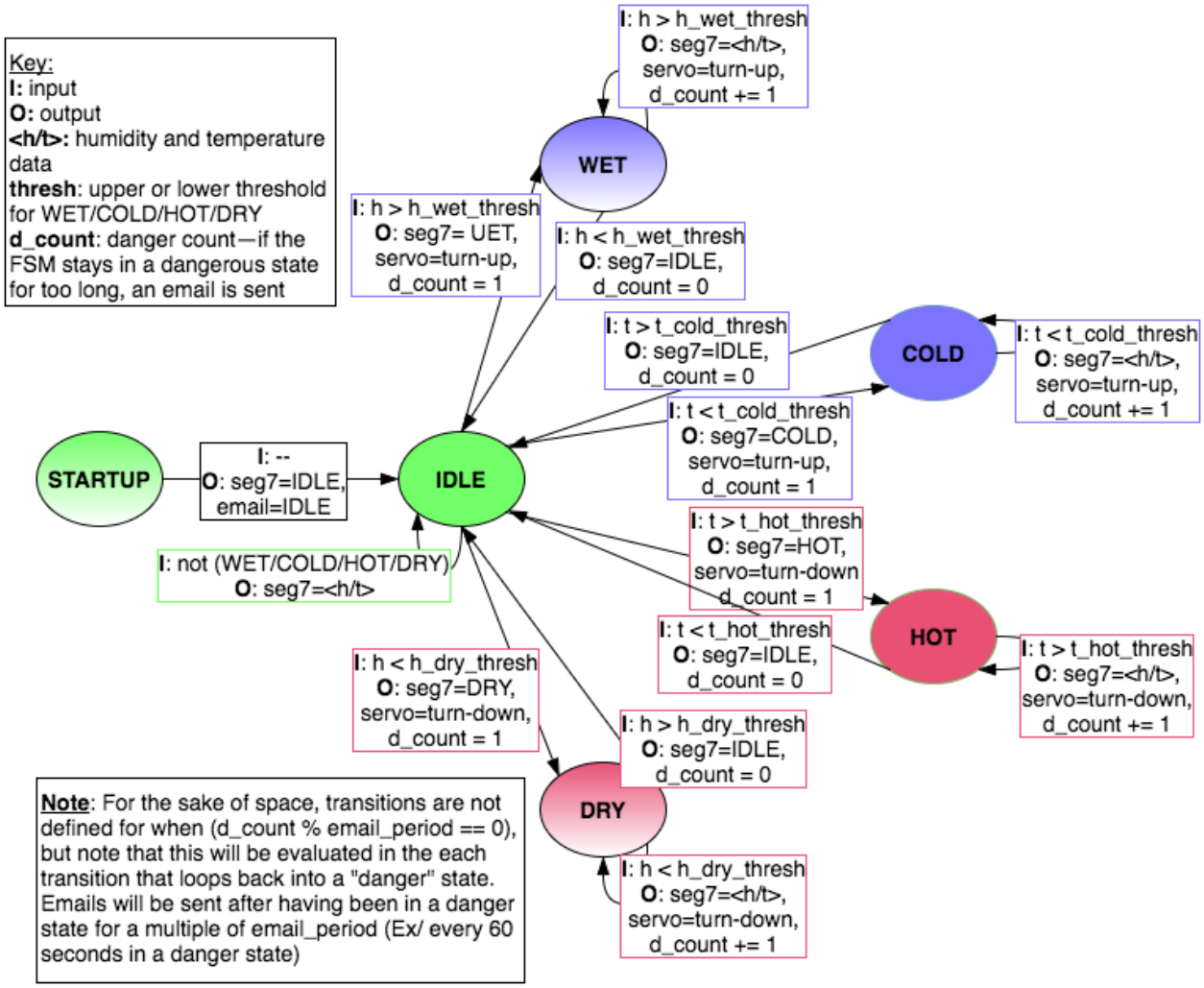

Figure 8: State diagram for Finite State Machine

Note: This system is in a fictional environment in which turning the servo clockwise (up) decreases humidity and increases temperature, and turning the servo counter-clockwise (down) increases humidity and decreases temperature. Also to note, inputs from the IDLE state corresponding to multiple of WET/-COLD/HOT/DRY are not explicitly spanned by the transition conditions, but allowed to default to the first evaluated. (e.g. if (h > h_wet_thresh) AND (t < t_cold_thresh), the FSM will enter the WET state and not the COLD state)

6.2 Implementing the state machine

Note: This section of the manual assume that you are no longer using the RPi in headless mode.

To implement this state machine, first, download a zip contain the module code here: hrefhttps://github.com/mmwebster/r remote-monitoring-and-controlhttps://github.com/mmwebster/rpi-remote-monitoring-and-control. Then, copy the "code" directory into your python workspace. Now the only step necessary to run the entire FSM is to execute `python main.py` in the command line.

»»

TODO: - Change every reference to "GPIO 25" to "GPIO 4", as this is the new wiring setup used and the GPIO number used in the codebase

««

»»

TODO: - Discuss implementing a secrets.py with a .gitignore file so that private credentials are not uploaded to the web.

««

6.3 Daemonizing the FSM (main.py)

6.3.1 Installing Supervisor

6.3.2 Configuring Supervisor

6.3.3 Daemonizing main.py with Supervisor

7 Data Analysis

Some simple data recording and analysis can be performed by outputting sensor data to a CSV file with Python. From there it is trivial to import into Excel or Google Sheets to perform analysis as desired. To do this, we will save sensor data into a separate file for every day, in which will be the fields time (timestamp), humidity (%), and temperature.(°C).

7.1 Save sensor data to CSV files with Python

We will modify the previous state machine code to include a few lines to append to a CSV file with the current sensor data every 8 seconds, which will result in about 10k entries in every day's CSV file. First, ...

»»

TODO: - Add stuff about saving data to csv files for every day with fields of (time,relHum,temp) that can import into Excel or Google Sheets to calculate some metrics on

««

8 Pre-Lab

Please have the following steps completed prior to coming to the first lab session for this module.

8.1 ...

...

Appendices

A Suggested Material Vendors

1. Raspberry Pi 3 Model B (Most of this manual will apply to other versions and models as well, however it is tailored to this particular version and model of the RPi.)
2. Micro SD Card (at least 8GB in size)
3. USB to TTL Serial Cable (aka Console Cable)
4. USB to Micro USB Cable
5. Humidity and Termperature Sensor
6. 7-Segment Display
7. Servo Motor
8. Breadboard
9. Jumper Wires (F/F, M/F, M/M)
10. 4xAA Battery Holder