UC Santa Cruz, S-Lab

Prototyping a Remote Monitoring and Control System using a Raspberry Pi

April 14, 2017

Contents

1 Introduction **2**
- 1.1 Summary . . . 2
- 1.2 Prerequisites . . . 2
- 1.3 Definitions and terminology . . . 2
- 1.4 Materials . . . 3
- 1.5 Learning outcomes . . . 3

2 Pre-Lab **3**
- 2.1 Basics of Unix shell navigation and interaction . . . 3
- 2.2 Basics of programming in Python . . . 4

3 Day 1: Configuration and Input **4**
- 3.1 Setting up the Raspberry Pi (RPi) . . . 4
  - 3.1.1 Using NOOBS to install the OS (Raspbian) . . . 4
  - 3.1.2 Some RPi basics . . . 4
  - 3.1.3 Wired connection over TTL Serial . . . 5
  - 3.1.4 Wireless connection over LAN via local IP & SSH . . . 9
  - 3.1.5 Wireless connection over WAN via Weaved & SSH . . . 10
- 3.2 Sensor inputs . . . 11
  - 3.2.1 Communicating with the si7021 humidity and temperature sensor (over I2C) . . . 11

4 Day 2: Local and Remote Output **17**
- 4.1 Sensor outputs . . . 17
  - 4.1.1 Communicating with the 7-segment display (over I2C) . . . 17
  - 4.1.2 Using the servo motor . . . 21
- 4.2 Remote outputs . . . 24
  - 4.2.1 Sending an email with Gmail and Python . . . 24

5 Day 3: Improved Autonomy **26**
- 5.1 The state machine . . . 26
  - 5.1.1 Designing the state machine . . . 26
  - 5.1.2 Implementing the state machine . . . 28
  - 5.1.3 Daemonizing the FSM (main.py) . . . 29
- 5.2 Data Analysis . . . 30
  - 5.2.1 Saving & Emailing CSV sensor w/ Python . . . 30
- 5.3 Conclusion and Troubleshooting . . . 31

Appendices **31**
.1 Suggested Material Vendors 31

1 Introduction

1.1 Summary

This manual details 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 a 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.

1.2 Prerequisites

It is recommended that you have introductory-level programming experience prior to taking this module.

1.3 Definitions and terminology

1. RPi (Raspberry Pi): A small but powerful linux machine with great I/O capabilities.
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 securely connecting to a remote computer
4. TTL (Transistor-Transistor Logic): A serial protocol used for communication between two systems using only RX (receive) and TX (transmit) signals
5. SPI (Serial Peripheral Interface): A communications protocol that uses a master-slave architecture
6. I2C (Inter-Integrated Circuit): A communications protocol with greater latency than SPI, but greater node addressibility
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
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): A mechanism for interfacing with the OS, through which commands can be executed
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 a 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, regardless of physical proximity. The internet is considered a form of WAN.

1.4 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 w/ NOOBS Pre-Loaded
   (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) ...

1.5 Learning outcomes

After completing this tutorial you should be able to do the following:

1. Configure a Raspberry Pi for headless mode and control it over a serial connection to your computer
2. Connect to and run operations on your Raspberry Pi over LAN and WAN via SSH and SFTP
3. Communicate over SPI and I2C with sensors and store their inputs
4. Drive a servo to actuate mechanical systems
5. Program intelligent, autonomous logic with Finite State Machines (Note that this implementation uses Python's blocking sleep() function for timing events and is therefore not immediately responsive to sudden changes in inputs. Non-blocking style can be implemented using Python's multithreading library. An example can be found here: https://github.com/mmwebster/rpi-attendance-tracker)
6. Collect and remotely transmit (email) sensor data for later analysis

2 Pre-Lab

2.1 Basics of Unix shell navigation and interaction

Before attending your first lab session, please complete items 1-6 of this tutorial (http://linuxcommand.org/lc3_learning_the_shell.php) on the basics of navigation in and interaction with the Unix shell.

2.2 Basics of programming in Python

It is recommended, if you do not have prior experience with Python, that you complete the "Learn the Basics" portion of this tutorial (https://www.learnpython.org/en/Welcome) to gain a working knowledge of Python.

3 Day 1: Configuration and Input

3.1 Setting up the Raspberry Pi (RPi)

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

3.1.1 Using NOOBS to install the OS (Raspbian)

The RPi comes without any installed or configured OS. Follow the instructions below to use the micro SD card with NOOBS pre-installed to install and configure Raspbian on the RPi. If you'd prefer formatting your own micro SD card, you can follow this tutorial (https://www.raspberrypi.org/help/noobs-setup/).

1. Place the micro SD card in the slot on the bottom of the RPi
2. Plug in all peripherals to your RPi over the HDMI and the USB Ports (Monitor, Keyboard, and Mouse)
3. Plug in the micro USB cable into the RPi's micro USB port
4. Select and install Raspbian 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.

3.1.2 Some RPi basics

3.1.2.1 Connecting to a wireless network

Through the Raspbian GUI, there is a wifi icon in the upper right corner of the screen; clicking on this will take you through 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 (https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md) details the process of connecting to a wireless network via the command line (command line access is described in section 2.3 and section 2.4)

3.1.2.2 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

Note: To turn the RPi back on after safely shutting it down, just unplug and then plug the power source back in (micro USB).

3.1.3 Wired connection over TTL Serial

It is often convenient to be able to interface with the RPi from 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.

3.1.3.1 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.

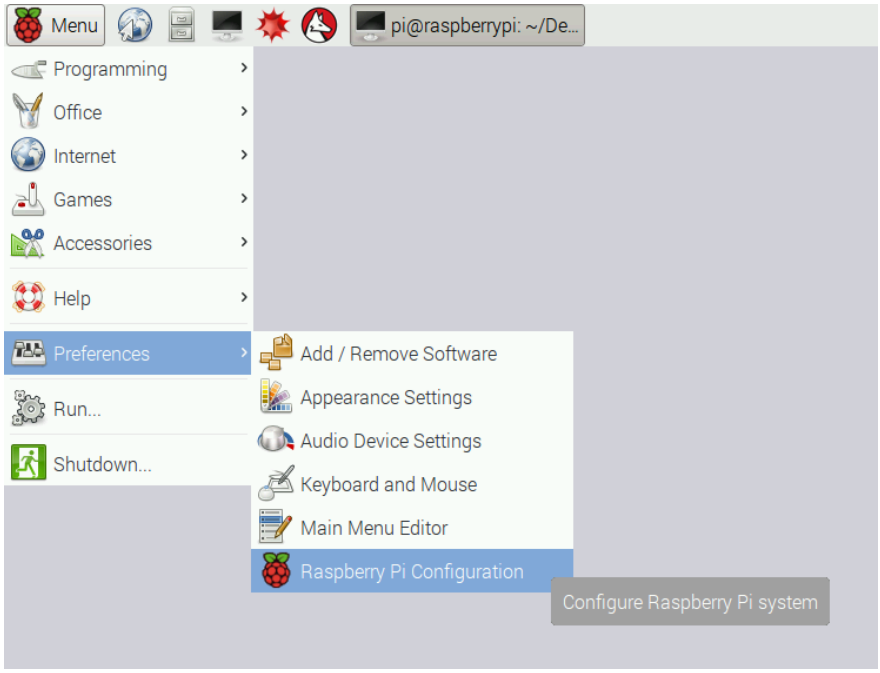

Figure 1: RPi Configuration at ***Menu » Preferences » Raspberry Pi Configuration***

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

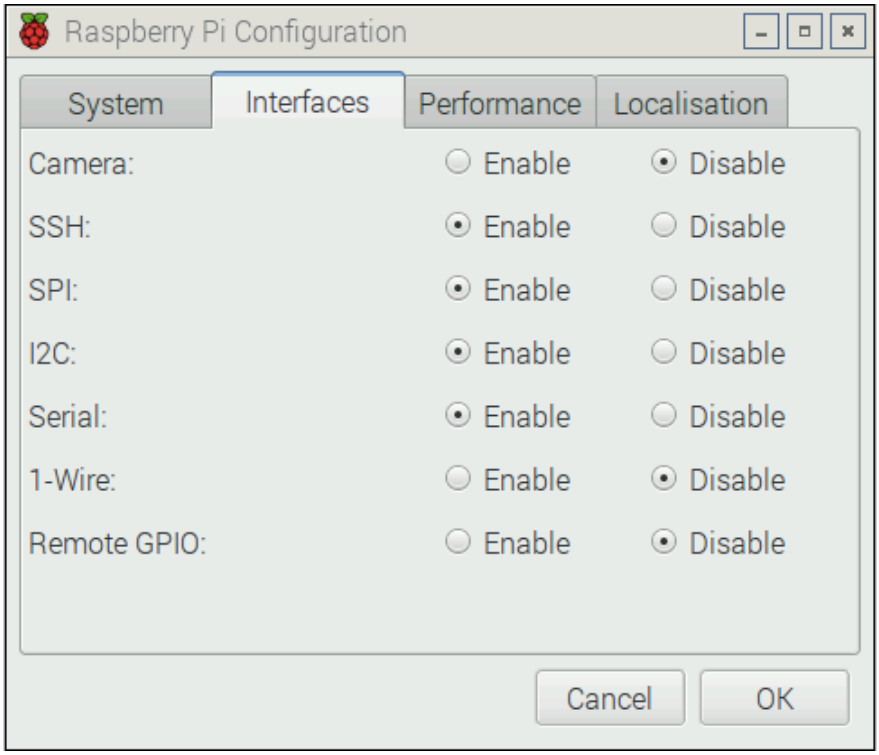

Figure 2: Necessary interface configuration for the RPi

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

3.1.3.2 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; these are required for a serial connection with the RPi. We must now disable bluetooth 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.

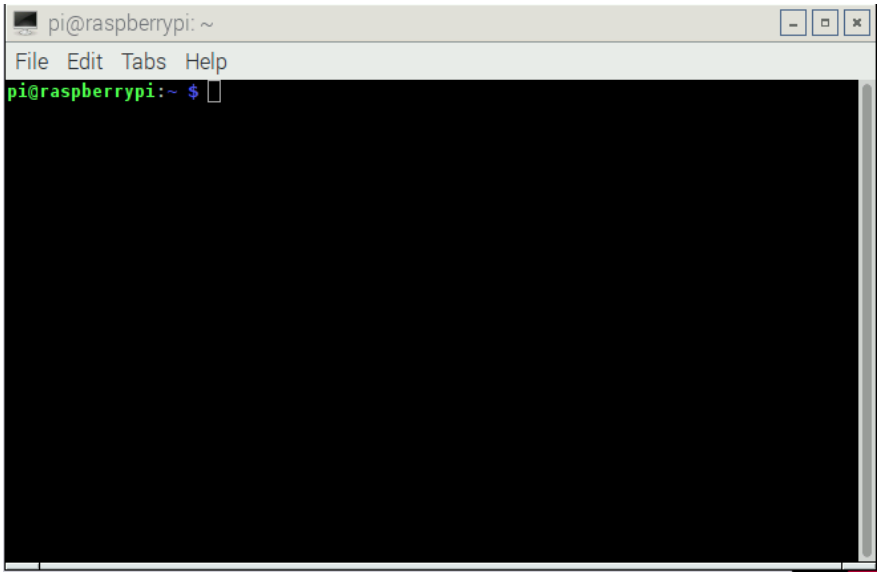

Figure 3: Raspbian Terminal App (Shell CLI)

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 command in order to edit the RPi's config file and stop the BT modem from trying to use UART:

Disable bluetooth: Edit boot config file

```
sudo nano /boot/config.txt
```

Then, add the following line to the end of the file:

Disable bluetooth: Text to append to config file

```
dtoverlay=pi3-disable-bt
```

Now, after exiting the nano editor, run the following command:

Disable bluetooth: Disable hciuart

```
sudo systemctl disable hciuart
```

Finally, reboot the RPi (to allow the changes to take effect) by running `sudo reboot`.

3.1.3.3 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

3.1.3.4 Connect the leads and USB port

Now, connect each of the four color-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.

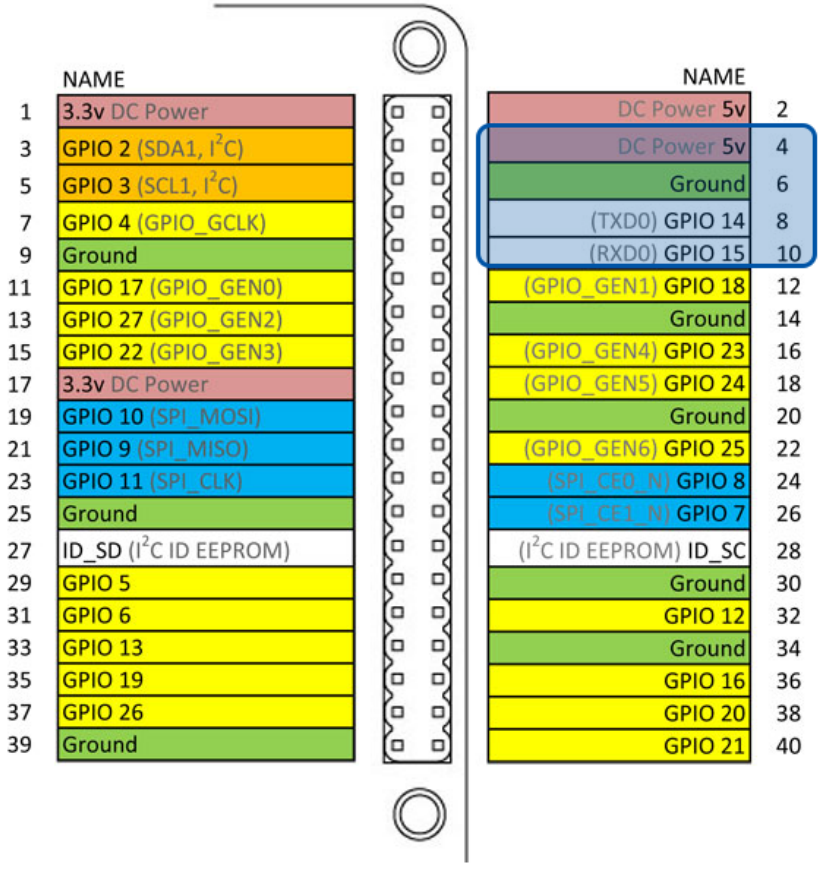

Figure 4: Raspberry Pi 3 Model B, GPIO header. Photo credit: mcmelectronics.com

CHECKPOINT: Before proceeding, please confirm with your lab instructor that you have correctly wired your TTL Serial Cable.

DO NOT SUPPLY POWER OVER BOTH THE 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 that 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.

3.1.3.5 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**

3.1.4 Wireless connection over LAN via local IP & SSH

Sometimes it is more convenient to connect to the RPi over SSH because of the additional features that 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. Connecting to your RPi in this manner requires the use of a LAN (Local Area Network) to which both your personal computer and the RPi must be connected.

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 LAN. See the following section if you wish to connect to your RPi from a different network than what the RPi is on. *Note*: If on Windows, you will have to install an application such as PuTTY to SSH into the RPi.

3.1.4.1 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).

3.1.4.2 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 local IP address that was assigned by your router. Copy this address to your clipboard.

3.1.4.3 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 authenticated, you have command line access to the RPi (the same as was achieved over the TTL Serial cable). This

connection is, however, dependent on the existence of a local network, wheras the serial connection is not. A complete guide for this method can be found here: https://www.raspberrypi.org/documentation/remote-access/ssh/.

3.1.4.4 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. Remote file transfer would ideally be achieved through Git, however that is out of scope of this manual. 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.

3.1.5 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 over 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 over a LAN.

3.1.5.1 Setup an account at Weaved.com

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

3.1.5.2 Install 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.

3.1.5.3 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 command line access to 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 in which you can also issue SSH credentials for your RPi.

3.2 Sensor inputs

3.2.1 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.

3.2.1.1 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.

3.2.1.2 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 120 (hexadecimal 0x0 to 0x77) I2C addresses are being used (the si7021 sensor is not connected) but the I2C interface has been properly configured.

3.2.1.3 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.

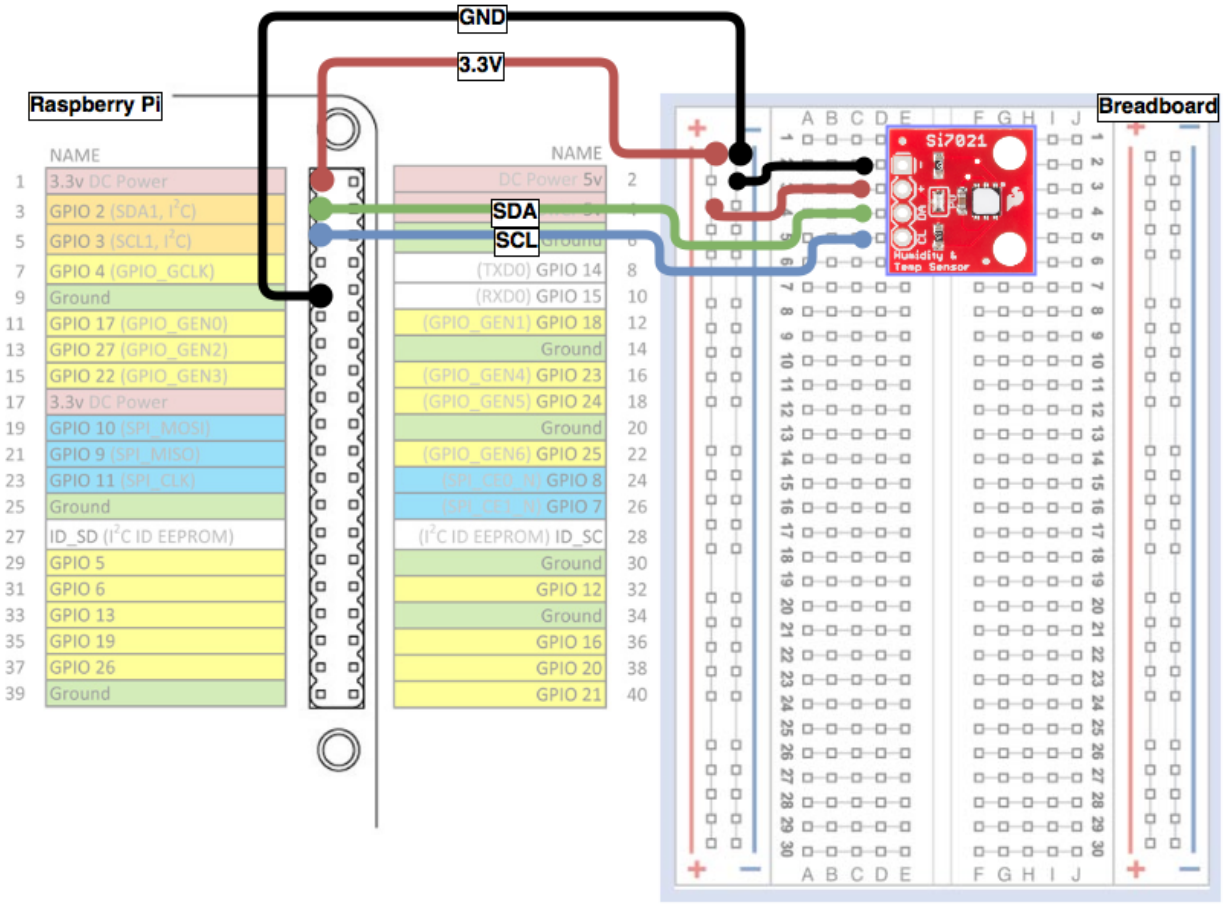

Figure 5: Wiring diagram for connecting the si7021 to the RPi over I2C

CHECKPOINT: Before proceeding, please confirm with your lab instructor that you have correctly wired your si7021 to the RPi.

3.2.1.4 Testing the wiring

There are many ways to test for faulty wiring, however in this case, it is easiest to simply use 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 something 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 a pin on the RPi. If this is the case, consult with your lab instructor before proceeding.

3.2.1.5 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 (do not run this, it is just example output from the previous commands):

`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 in yellow above (the path is likely, but not necessarily, a default of ***/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.

3.2.1.6 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= "deg" # 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
...
```

4 Day 2: Local and Remote Output

4.1 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.

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

The 7-segment display (referred to 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.

4.1.1.1 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 (data line) and SCL (clock line) 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.

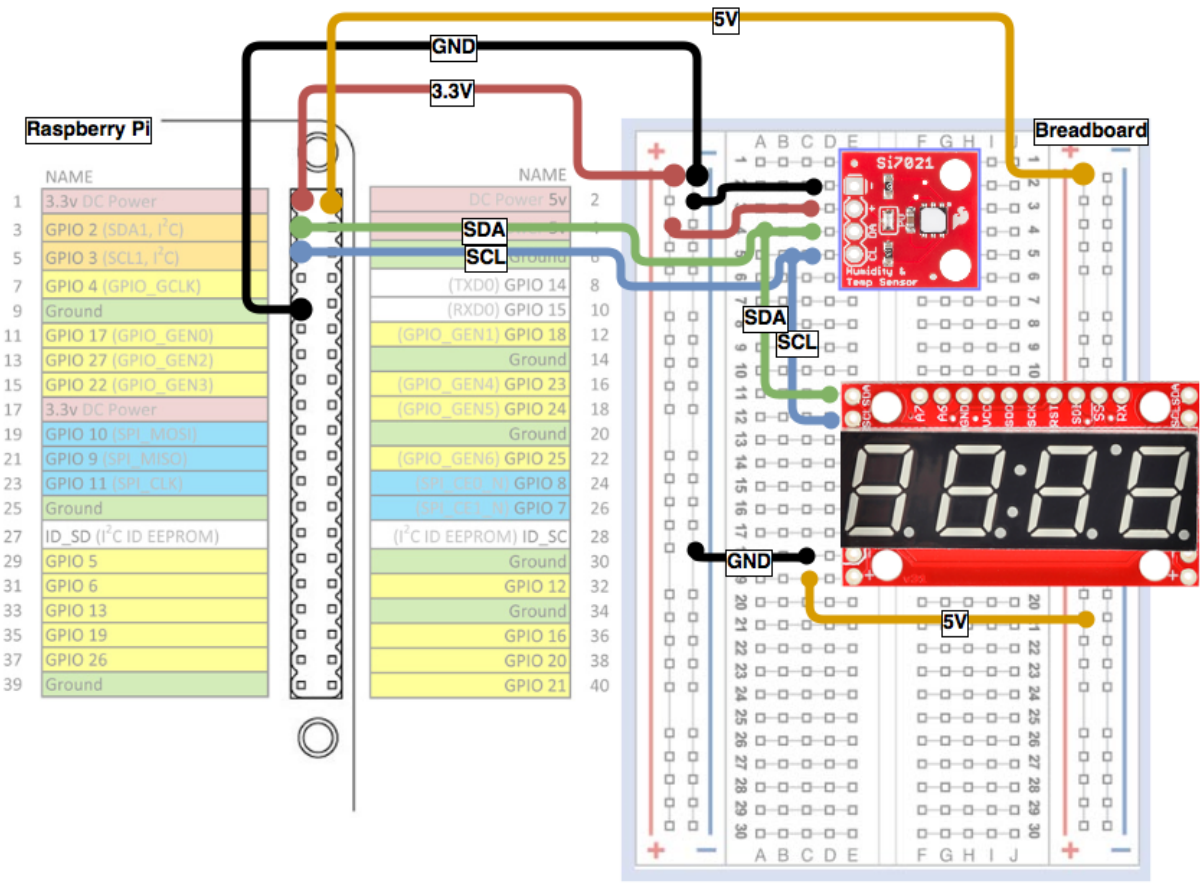

Figure 6: Wiring diagram for connecting the seg7 (along with si7021) to the RPi over I2C

CHECKPOINT: Before proceeding, please confirm with your lab instructor that you have correctly wired your si7021 and seg7 display to the RPi.

4.1.1.2 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 as 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.

4.1.1.3 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.

4.1.2 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.

4.1.2.1 Overview of controlling the servo with PWM

PWM is a method for controlling the following properties of a digital signal, in order to digitally vary its average 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. Servos usually require a 50Hz signal, but this sometimes varies.

4.1.2.2 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 dictates the degree rotation of the servo

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):

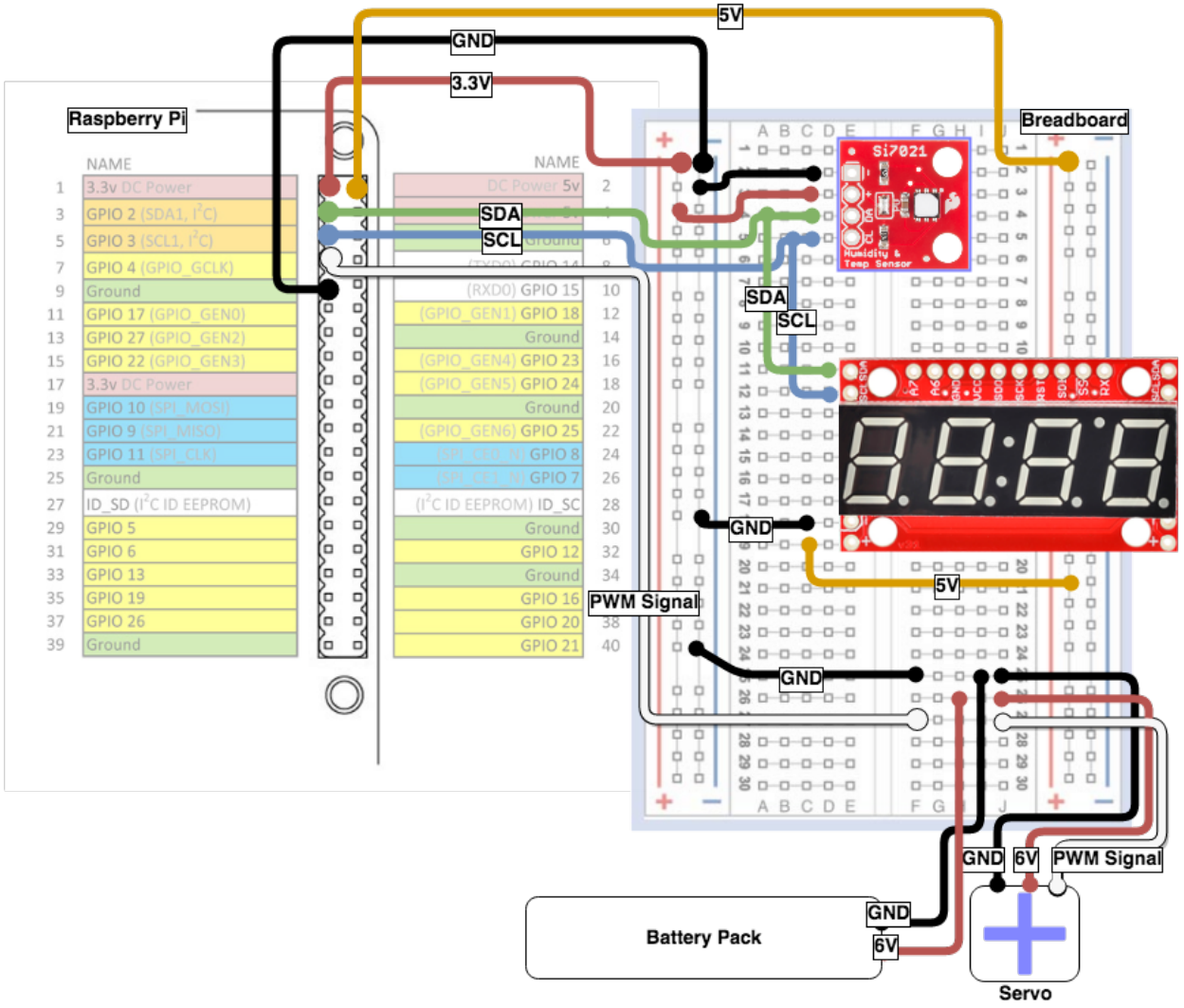

Figure 7: Wiring diagram for connecting the servo & battery pack (along with si7021 and seg7) to the RPi to control over PWM

CHECKPOINT: Before proceeding, please confirm with your lab instructor that you have correctly wired your si7021, seg7 display, and servo to the RPi.

4.1.2.3 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 the context of a Python script, we will 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 to connect as many servos to the RPi as there are hardware-driven PWM pins. The tradeoff with software driven PWM is that it is only accurate within 100 µs, whereas hardware-driven PWM can be accurate to 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, return to your RPi's command line and run the following to navigate back to your workspace:

Navigate to your workspace

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

Then, 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

########################################################################
# 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(4, GPIO.OUT) # set GPIO 25 as output
servo = GPIO.PWM(4, 50) # instantiate PWM output to GPIO 25 @ 50Hz

degree_sign= "deg" # 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.
Note: If using the RPi with peripherals, you can paste with SHIFT-CONTROL-C

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.

4.2 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.

4.2.1 Sending an email with Gmail and Python

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

4.2.1.1 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 change will make it much 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.

4.2.1.2 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 make your account more vulnerable to attacks, so it is highly 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.

4.2.1.3 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.

»»»

TODO: - left off

«««

5 Day 3: Improved Autonomy

5.1 The state machine

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

5.1.1 Designing the state machine

The are multiple 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 into. 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 overview: https://en.wikipedia.org/wiki/Finite-state_machine.

5.1.1.1 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

5.1.1.2 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.

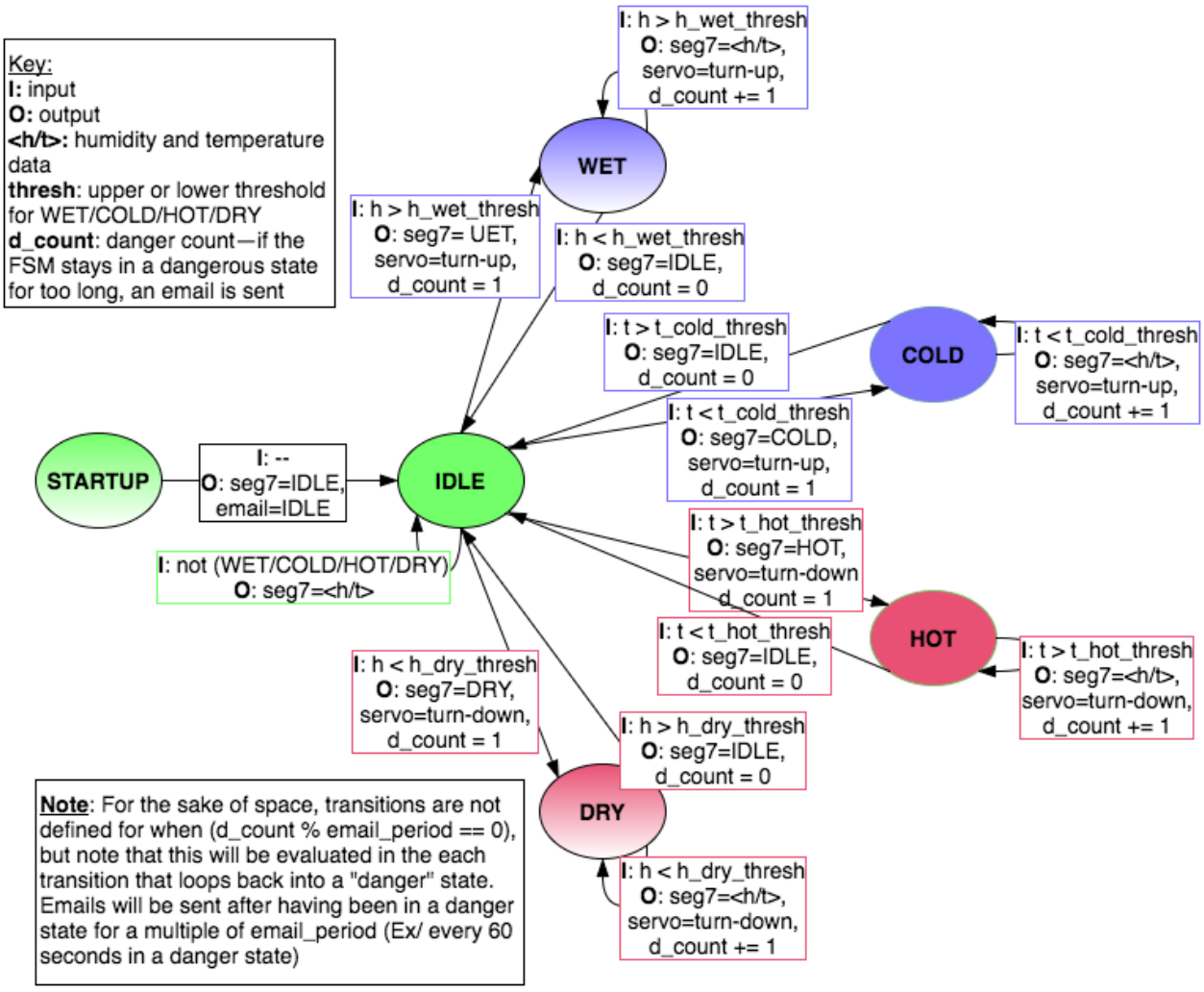

Figure 8: State diagram for Finite State Machine

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)

5.1.2 Implementing the state machine

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

To implement this state machine, first, download a zip containing the module code here: https://github.com/mmwebster/rpi-remote-monitoring-and-control. Then, copy the ***code*** directory into your python workspace at */Desktop/python-code/code/*. Now, the only step necessary to run the entire FSM is to execute ***main.py*** from the command line while inside the ***code*** directory. To do this, open the terminal application in from the top nav bar in the Raspbian GUI, then run the following commands:

Run the FSM

```
cd ~/Desktop/python-code/code/
python main.py
```

Note: You can stop the FSM from running by typing <***control***> ***c***

5.1.2.1 Implementing a .gitignore'd secrets.py

There is one last step required to implement the state machine such that you receive email alerts. You must create a `secrets.py` file that contains your passwords and other credentials for your SMTP email. It should be in the code's top-level directory with the following form:

Python Code 5 Example secrets.py

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

################################################################################
# Class definitions
################################################################################
class Secret:
    def __init__(self):
        self.secrets = {"email_password": "xxxxxxxxx", "email_from": "your.
    new.email@gmail.com", "email_to": "your.existing.email@gmail.com"}

    def fetch(self, key):
        if key in self.secrets.keys():
            return self.secrets[key]
        else:
            raise "ERROR: passed invalid key to fetch_secret"
            return None
```

Note: Replace the x's with the email account's password, leave "email_password" as-is.

5.1.3 Daemonizing the FSM (main.py)

5.1.3.1 Installing PM2

Run the following to update the RPi's packages and system, then install NPM and PM2:

Update RPi and install NPM and PM2

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install nodejs npm
sudo npm install -g pm2
```

5.1.3.2 Configuring PM2

You should now be able to view all of PM2's active processes by running `pm2 list`. Now, to configure PM2 to automatically start the FSM when it boots up, generate PM2's startup script command:

Generate PM2's startup script command

```
pm2 startup
```

Then, copy the outputted setup command and run it. It should have a form similar to the following:

Generate PM2's startup script (this is output, DO NOT EXECUTE IT)

```
sudo env PATH=$PATH:/usr/bin /usr/local/lib/node_modules/pm2/bin/pm2
startup systemd -u pi -hp /home/pi
```

5.1.3.3 Daemonizing main.py with PM2

Run the following commands to startup the FSM (main.py) and save it to the processes restarted on reboot:

Start FSM process; save it to reboot processes

```
pm2 start ~/Desktop/python-code/code/main.py -n fsm --interpreter=python
pm2 save
```

Note: The file path for main.py (in red, above) might be different for you depending on where you saved your code. You can run `pwd` to see what your current system path is in the command line.

5.1.3.4 General PM2 Commands

The following are useful PM2 commands for monitoring the processes it's currently running:

1. `pm2 list`: Lists all current processes
2. `pm2 stop <process-name>`: Stops the process <process-name>
3. `pm2 start <process/file-name>`: Restarts the process <process/file-name> if it's the name of a currently stopped process in PM2, otherwise it attempts to start a process for the given path.
4. `pm2 delete <process-name>`: Deletes the process (not the code, just the process) <process-name>

5.2 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 fields for time (timestamp), humidity (%), and temperature.(°C).

5.2.1 Saving & Emailing CSV sensor w/ 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, follow this tutorial to learn how the CSV module works in python: http://www.pythonforbeginners.com/systems-programming/using-the-csv-module-in-python/. Now, modify main.py with the following additions:

1. Import the CSV module at the top of the file
2. Write temperature and humidity data to the end of the file during every iteration of the state machine
3. In the main FSM runloop, email yourself with the sensor CSV data file attached using the provided Mailer class.

Note: You will have to write the CSV code yourself—it is not provided.

5.3 Conclusion and Troubleshooting

You should now be able to remotely monitor and control a system using a Raspberry Pi 3 and the additional materials detailed in this manual. If you have any questions, do your best to format them to only require yes or no answers, and then google them! `http://www.StackOverflow.com` and other online coding forums will likely hold answers to all of your questions. More often than not, the problem you're experiencing has already been solved and documented online; don't rely on your lab instructor!

Appendices

.1 Suggested Material Vendors

Figure 9: 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.) https://www.sparkfun.com/products/13825

Figure 10: Micro SD Card (at least 8GB in size) https://www.adafruit.com/product/1583

Figure 11: USB to TTL Serial Cable (aka Console Cable) https://www.sparkfun.com/products/12977

Figure 12: USB to Micro USB Cable https://www.sparkfun.com/products/10767

Figure 13: Humidity and Termperature Sensor https://www.sparkfun.com/products/13763

Figure 14: 7-Segment Display https://www.sparkfun.com/products/11442

Figure 15: Servo Motor https://www.sparkfun.com/products/9065

Figure 16: Breadboard https://www.sparkfun.com/products/12002

Figure 17: Jumper Wires: Female/Female https://www.sparkfun.com/products/12796

Figure 18: Jumper Wires: Male/Female https://www.sparkfun.com/products/12794

Figure 19: Jumper Wires: Male/Male https://www.sparkfun.com/products/12795

Figure 20: 4xAA Battery Holder https://www.sparkfun.com/products/552