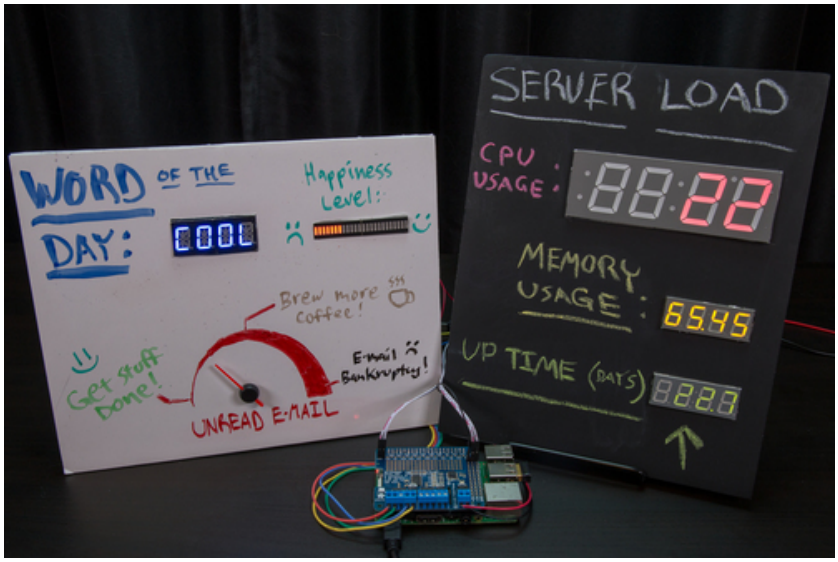
adafruit learning system

Raspberry Pi Physical Dashboard

Created by Tony DiCola

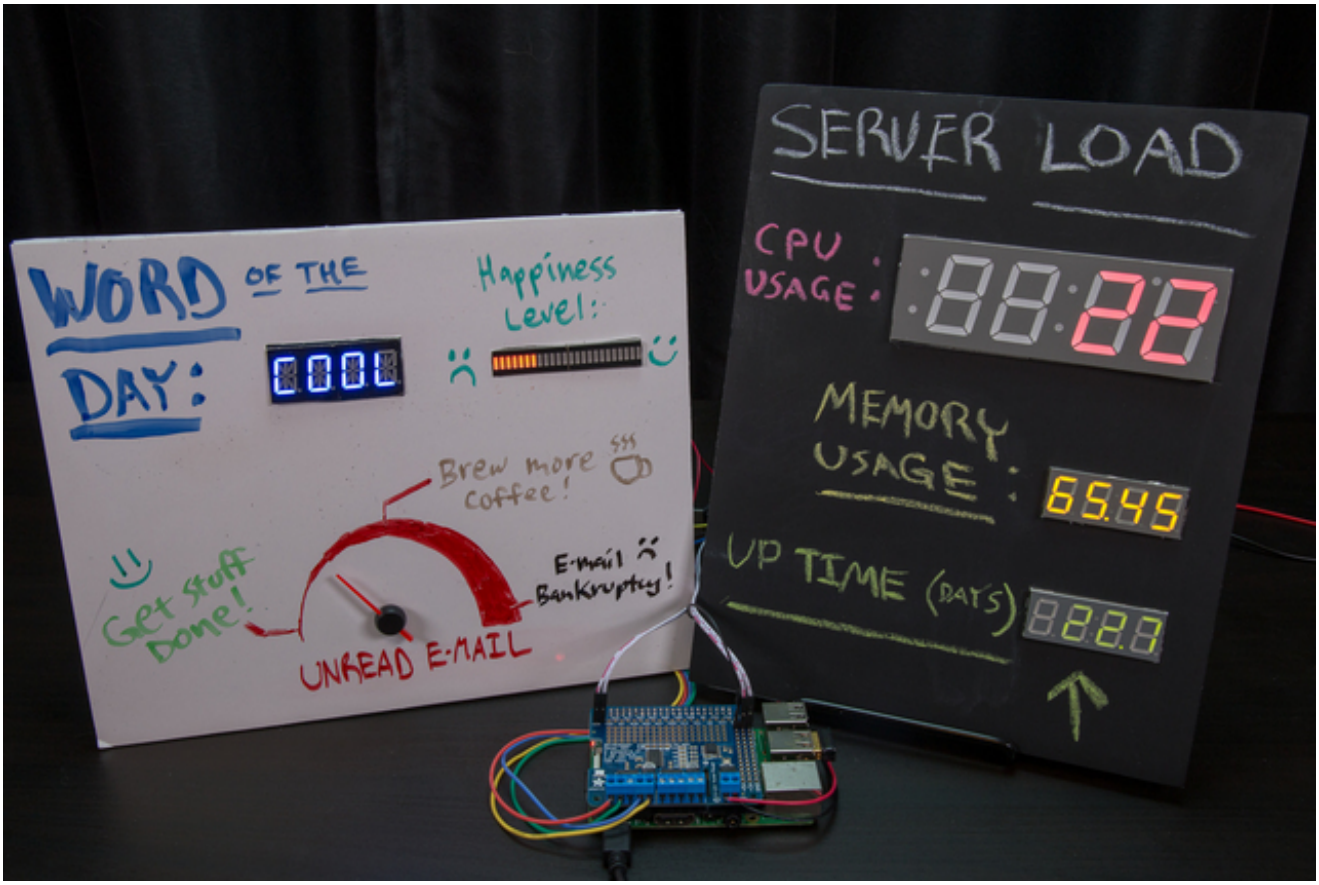


Last updated on 2016-01-25 09:40:12 PM EST

Guide Contents

Overview

This project will show you how to use a Raspberry Pi to build a physical dashboard that displays any kind of data. Use beautiful LED displays and automotive dial gauges to build an exciting dashboard that tracks important metrics. For example keep tabs on the health of a web serivce by displaying server health data with dials and bright seven segment LED displays. Or build a dashboard for your home that displays sensor data--if you can measure it, you can put it on this physical dashboard by just calling a simple REST API. You can even build your dashboard into a whiteboard or chalkboard so you can reuse it and change the type of data being displayed!



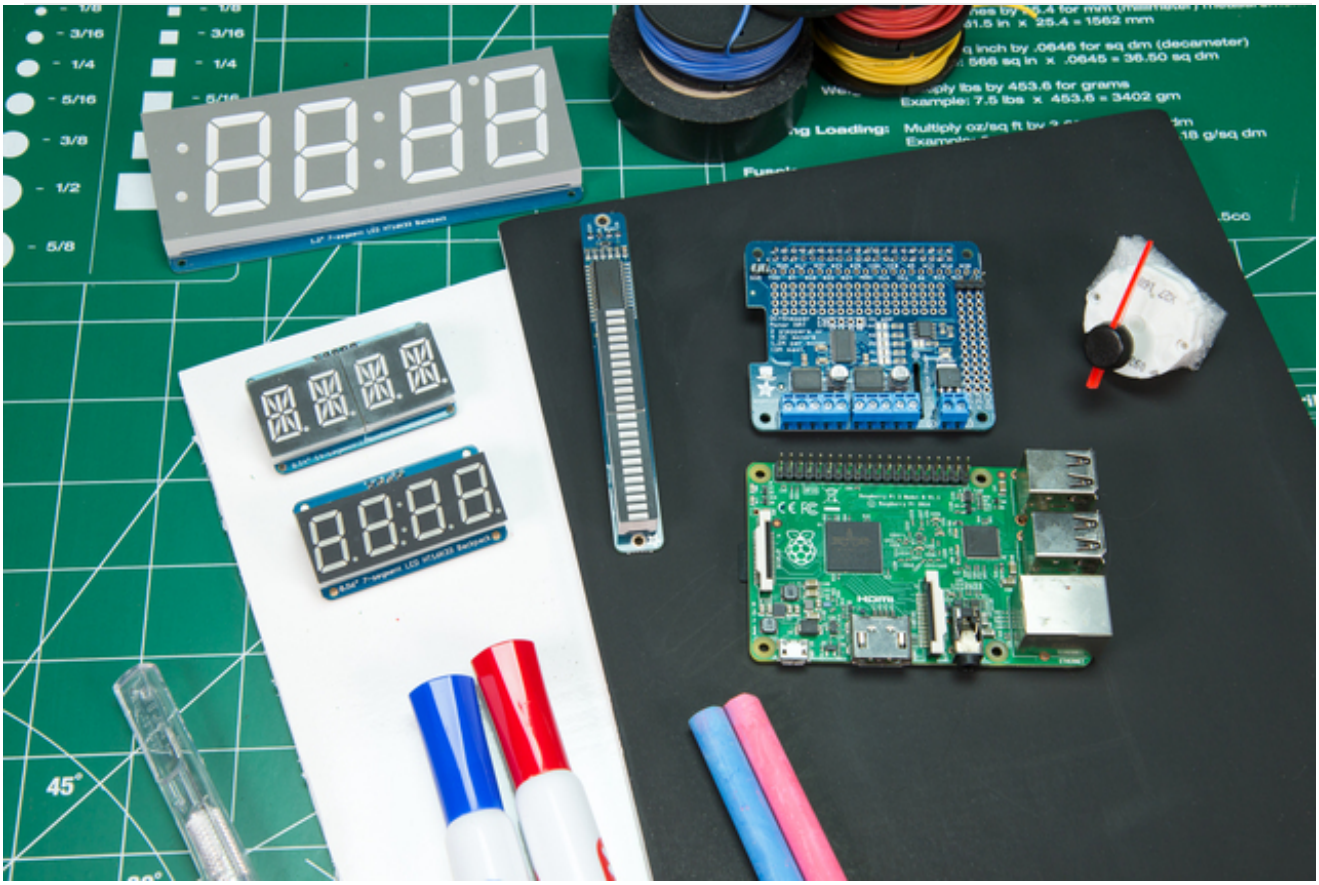
Before you follow this project it will help to familiarize yourself with the following guides:

- Adafruit LED Backpacks (http://adafru.it/dEM)
- Adafruit Stepper & DC Motor HAT (http://adafru.it/kCC)
- All About Stepper Motors (http://adafru.it/kMd)

In addition you will want to be familiar with the basics of using the Raspberry Pi, like how to burn an operating system to SD card (http://adafru.it/kMe), connect the Pi to your network (http://adafru.it/jd2), and access the Pi's command terminal with SSH (http://adafru.it/jsE).

Continue on to learn about the hardware and parts used in this project.

Hardware



Parts

You'll need the following parts to build a Raspberry Pi dashboard:

- **Raspberry Pi (http://adafru.it/eCB)** - I recommend the Pi 2 since it has a bit more power and memory compared to a normal Pi, but the B+ or A+ would work great too. A Pi Zero might work but will be tricky to connect to the motor HAT. The older model B and A Pi with 26 pin GPIO adapter won't work with the motor HAT.
- **LED Backpacks (http://adafru.it/dEM)** - This project is made to work with the following LED backpacks. Note that a Pi can control up to 7 LED backpacks (each with their own unique I2C address).
    - 0.56" 4-digit 7 segment display (http://adafru.it/dxd)
    - 1.2" 4-digit 7 segment display (http://adafru.it/dxe)
    - Quad alphanumeric diplay (http://adafru.it/dxf)
    - Bicolor 24 segment bar graph (http://adafru.it/1721)
- **Stepper & DC Motor HAT (http://adafru.it/2348)** and **automotive stepper gauge (http://adafru.it/2424)** - You can control up to two automotive stepper gauges using the motor HAT. Grab a few pairs of Pi HAT standoffs (http://adafru.it/2336) to securely

mount the HAT to the Pi.

- **USB WiFi Dongle** (http://adafru.it/814) - Use a Pi-compatible WiFi dongle to easily get your Pi on a network. The dashboard can be controlled by any machine on the network that makes web requests to the Raspberry Pi.
- **Raspberry Pi power supply (http://adafru.it/1995)** - You need a 5V supply with micro USB connector that can source around 2 amps of current to power the Pi.
- **5V 2amp power supply** (http://adafru.it/276) - You'll want a separate power supply to drive the motors. This 2 amp supply should have enough current to handle a couple automotive steppers and the LED backpacks (see below). Grab the **2.1mm barrel jack to screw terminal adapter (http://adafru.it/368)** to make wiring to the supply easy.
- **3.3V regulator (http://adafru.it/2165)** or **buck converter (http://adafru.it/2745)** - This is optional, but if you're driving more than about 3 LED backpacks you'll likely find the Pi can't supply enough current to them all. Use a 3.3V regulator like the LD1117-3.3 or a fancier 3.3V buck converter to drop the motor's 5V supply down to 3.3V for powering the LED backpacks. Grab a couple 10uF electrolytic capacitors (http://adafru.it/2195) if using the LD1117-3.3 regulator too.
- **Female/Female jumper wires (http://adafru.it/793)** - If you solder male headers onto the LED backpacks then you'll want female/female wires to connect to each backpack. You could skip these and solder wires directly to the LED backpack terminals as an alternative.
- **Breadboard (http://adafru.it/64)** and **male/male jumper wires (http://adafru.it/153)** - For prototyping and connecting components together a breadboard and jumper wires will be very handy. Once everything is working you can solder components and wires to a perma-proto (http://adafru.it/1609) for a cleaner and more rugged setup.
- **Soldering tools (http://adafru.it/fE3)** - You'll need to do a little bit of soldering to build this project. In particular you'll solder wires to terminals on the auto gauges, and solder LED displays and headers to the LED backpacks. If you're new to soldering be sure to follow the guide to excellent soldering (http://adafru.it/dxy) and it will help to practice a bit on spare parts or other projects ahead of time.
- **Wire** (http://adafru.it/egK)- You need a bit of wire to solder to the terminals on the automotive gauges. I recommend the 26AWG silicone stranded wiring (http://adafru.it/egK).
- **Dry-erase foamboard (http://adafru.it/kMf)** or **chalkboard foamboard (http://adafru.it/kMA)** - You can use almost anything as the base for your dashboard but dry erase or chalkboard foamboard is easy to cut and can be erased and reused easily.
- **Sharp knife (http://adafru.it/kMB)** for cutting the foamboard. Be careful and use adult supervision when necessary.
- **Electrical tape**

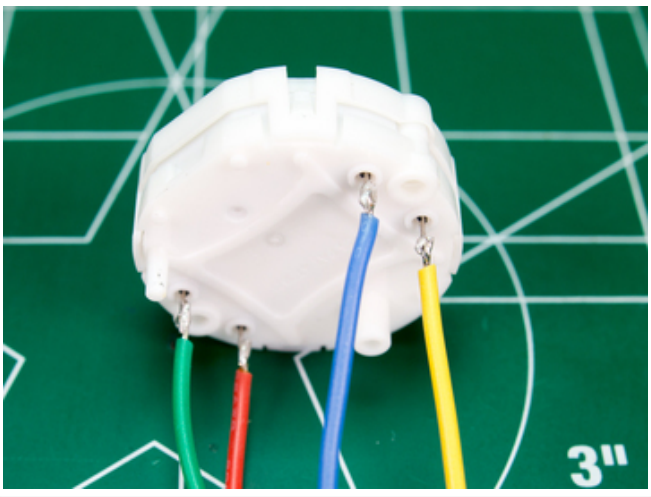
Assembly & Testing

Before you get started with the dashboard you'll need to first assemble and test the motor HAT & LED backpacks. Carefully follow each product's guide to assemble and test them:

- Adafruit Stepper & DC Motor HAT Guide (http://adafru.it/kMC)
- Adafruit LED Backpacks Guide (http://adafru.it/dEM)

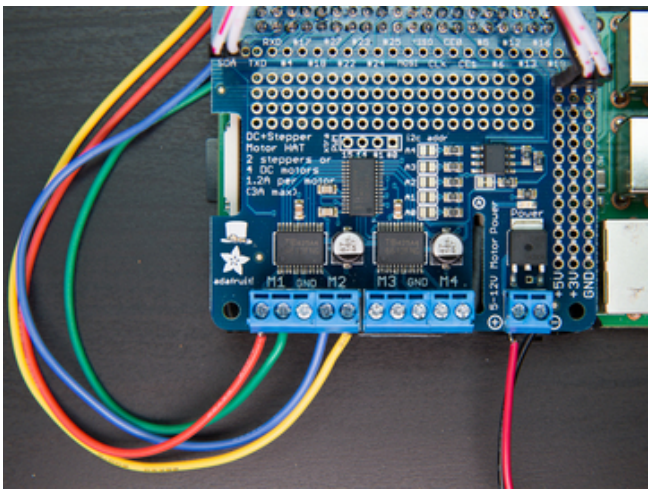
In addition go through the following guide to install the Python LED backpack library and test using the backpacks from the Raspberry Pi:

- LED Backpack Displays on Raspberry Pi & BeagleBone Black (http://adafru.it/kMD)

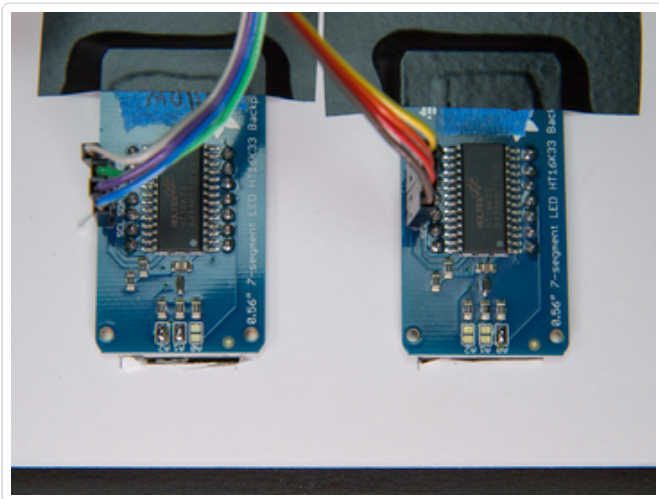


For testing the auto gauge stepper motors with the motor HAT I found it easiest to solder wires directly to the four terminals on the back of the motor. Tin the terminals and the wires separately, then hold them against each other and flow the solder with the iron to make a firm connection.

Each 'column' of terminals is a separate coil of the stepper. Connect one column to a motor controller position like M1 and the other column to the other position on the terminal block like M2. The center GND pin of each stepper terminal will be unused.



Each LED backpack should be assigned a unique address so they can be connected to the Pi's I2C bus. Notice the 3 address solder pads on the bottom of the boards in the photo to the left. If you bridge a pad with solder it will enable that address bit. The address starts at 0x70, so the board on the left has address 0x76 (bits A1 and A2 are set, or 2+4=6) and the board on the right has address



0x71 (bit A0 is set, or 1).

It's very important that each LED backpack has a unique I2C address assigned by setting their address bits! If two devices happen to have the same address you'll see errors and unpredictable behavior.

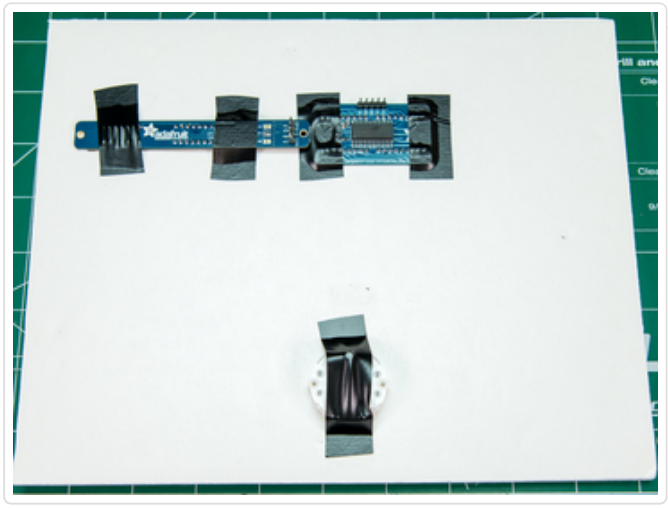
Before moving on make sure you've tested each component individually by following it guide mentioned above! It's much easier to troubleshoot components when they're outside the assembled dashboard.

Dashboard Assembly

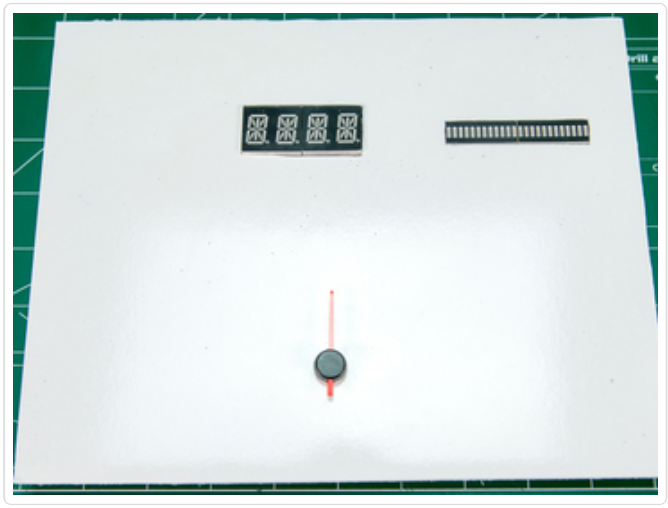
Use dry erase marker or chalk foamboard to build the dashboard. This will allow you to update and erase the labels to change what the dashboard displays.

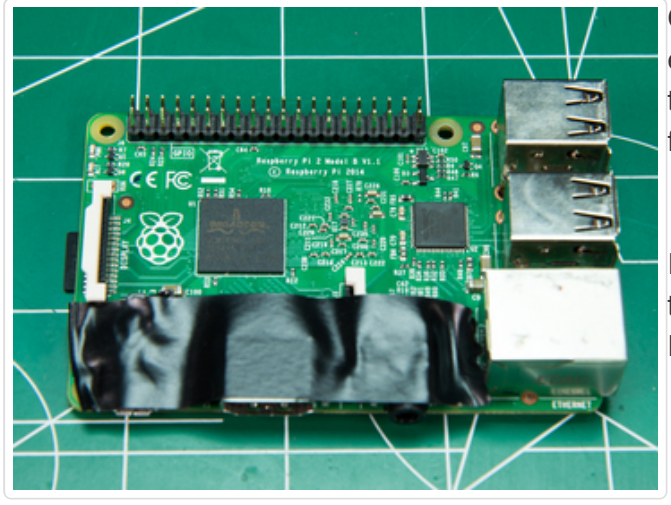
Carefully trace the shape of each display and cut out an appropriate hole in the foamboard. Tape each display and motor to the dashboard.

For the automotive stepper gauges if your foamboard is thin (~1/4" thick) you can just poke a hole for the gauge shaft to stick through instead of



cutting out a hole for the entire gauge.

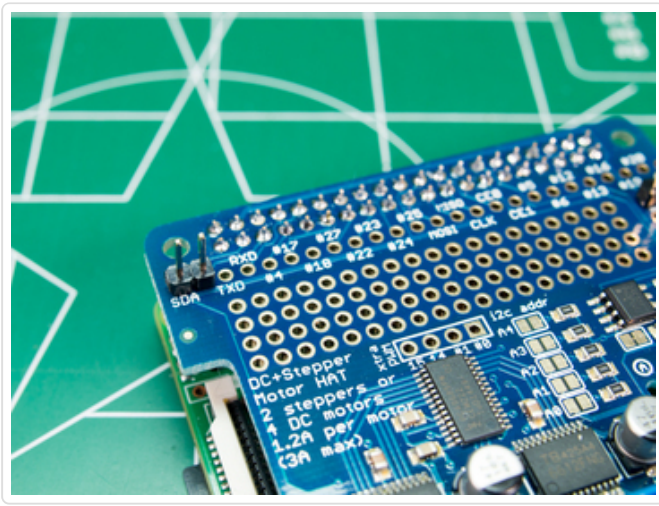




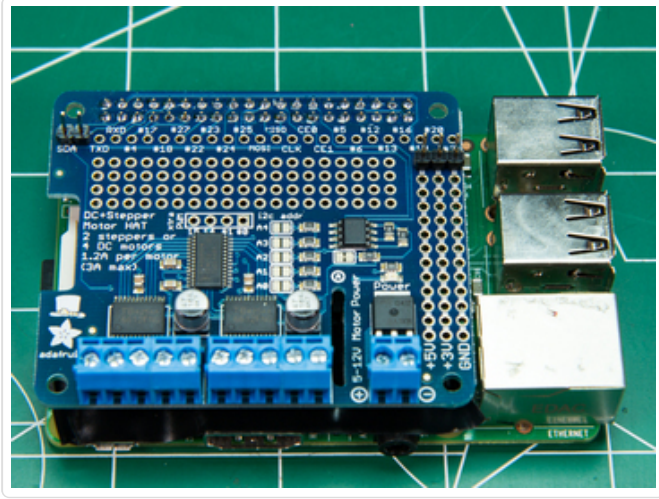
Cover the HDMI port half of the Raspberry Pi with electrical tape if you don't have standoffs to hold the motor HAT. This will prevent HAT connectors from touching the Pi and shorting out.

If you do have Pi HAT standoffs use those instead to hold the Pi HAT and prevent it from touching the Pi.

Solder headers or wires onto the SDA, SCL, 3.3V, 5V, and GND pins (on the far right of the HAT, not



shown) to allow connecting LED displays to the I2C bus.



Push the motor HAT onto the GPIO connector of the Pi.

Now wire up the auto gauge steppers to the motor HAT terminals as you did when testing them earlier.

Connect the 5V motor power supply to the positive (+) and negative (-) motor power terminals on the motor HAT.

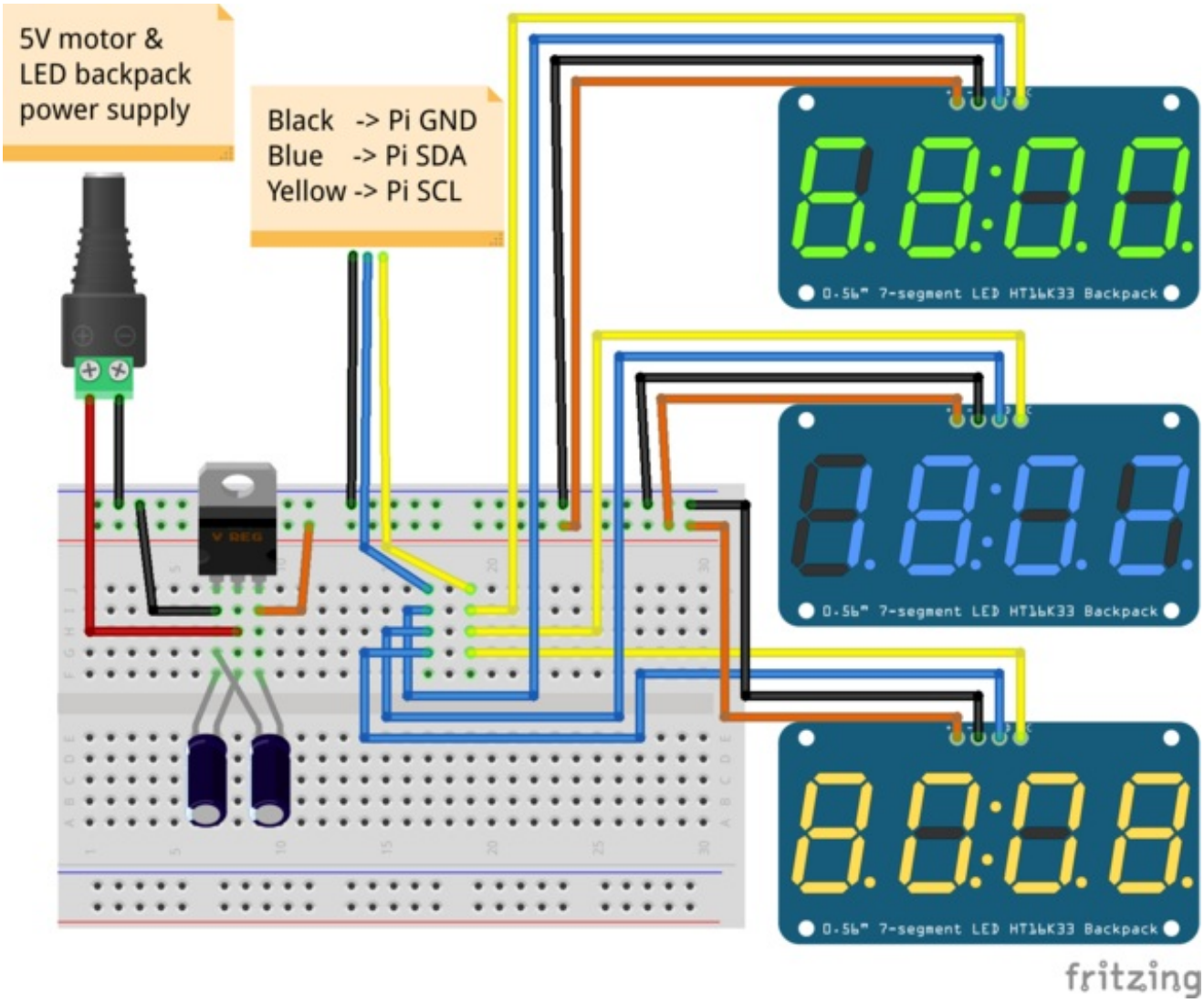
For each LED backpack connect its **clock (C)** / **SCL pin** to the **Pi's SCL header**, and its **data (D)** /

SDA pin to the **Pi's SDA header**.

If you're just using ~1-3 LED backpacks then you can power them directly from the Pi. Connect each backpack's negative (-) power header to the Pi's GND, and positive power (+) header to the Pi **3.3V** power (**NOT** the 5V power!). However for the large 1.2" 7-segment backpack it works best with 5V power instead of 3.3V so connect its positive (+) header to the Pi 5V power. For the large 1.2" 7-segment and quad alphanumeric backpack connect their extra **V_IO** / **Vi2c** line to the **Pi 3.3V** power (**NOT** the 5V power!).

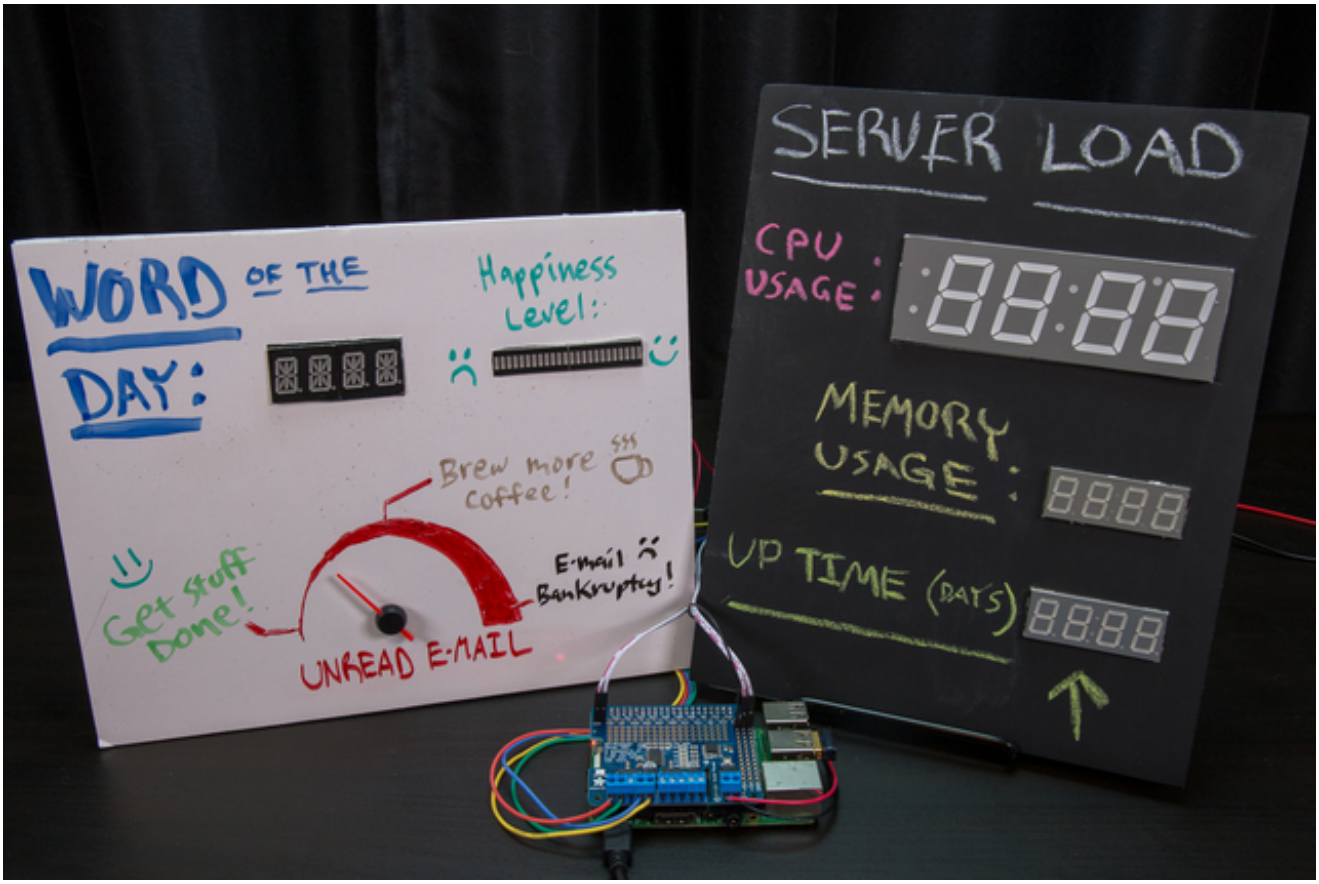
NOTE: If your 1.2" display doesn't seem to work when connected to 5V power there might be slightly too much difference between the 5V power and Pi's 3.3V I2C signal. Put a power diode in series before the 5V enters the display positive terminal (remember the striped side of the diode is the cathode/negative side and should face away from the positive 5V power) to slightly drop the 5V down to a level the display can better work with.

If you're using more than about 3 backpacks at once you might find the Pi can't supply enough power to drive them all at the same time. This might manifest as flakey dislays that don't show up with the i2cdetect command, or dimly lit LEDs that pulse and strobe visibly. In this case its best to use a dedicated 3.3V power supply to drive all the backpacks instead of the Pi's 3.3V supply. As mentioned in the parts you can use a 3.3V regulator to drop down the 5V motor power supply voltage to the 3.3V the LED backpacks need to operate. See the diagram below for an example of wiring a LD1117-3.3 regulator (and two 10uF electrolytic capacitors) to multiple LED backpacks:



Once assembled go through and carefully check each display and auto gauge can be controlled just as your tried in testing earlier. This will make sure you don't have any problems with connections, power, wiring, etc.

Finally label your dashboard with whatever metrics you'd like to display! A couple examples of dashboards are below:



Continue on to learn how to install and setup the software for the dashboard.

Software

Before you install the software for this project make sure your Raspberry Pi is running the latest Raspbian Jessie or Jessie Lite (http://adafru.it/fQi) operating system. If you're new to using the Raspberry Pi be sure to read the first few Learn Raspberry Pi guides (http://adafru.it/dpe) to learn how to burn an operating system to SD card (http://adafru.it/kMe), connect the Pi to your network (http://adafru.it/jd2), and access the Pi's command terminal with SSH (http://adafru.it/jsE).

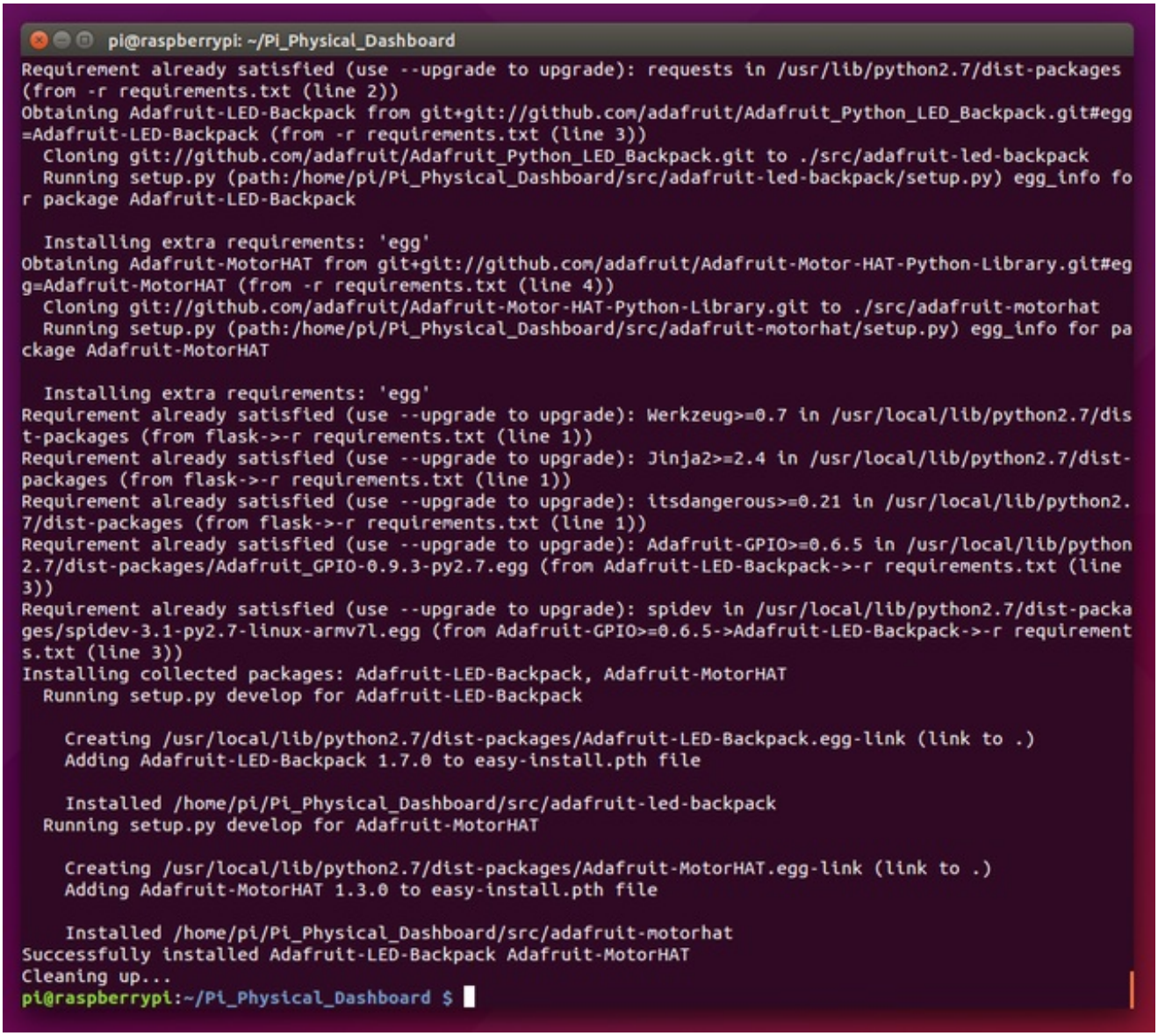
Also make sure your Raspberry Pi is connected to the internet with a wired or wireless network connection so software dependencies can be installed before continuing.

Installation

To install the dashboard software from its home on Github (http://adafru.it/kME) connect to your Pi's command terminal and run the following commands:

```
sudo apt-get update
sudo apt-get install -y git build-essential curl python-pip python-smbus python-dev
git clone https://github.com/adafruit/Pi_Physical_Dashboard.git
cd Pi_Physical_Dashboard
sudo pip install -r requirements.txt
```

After running the last command you should see pip print that it successfully installed dependencies and is cleaning up:



The software will be installed in the /**home**/**pi**/**Pi_Physical_Dashboard** folder and can be run with its **main.py** Python script. However before running the dashboard software you'll first need to configure it by following the next section.

Configuration

To configure the dashboard software you need to edit its config.ini (http://adafru.it/kMF) file in a text editor. For example you can use the nano text editor on the Pi by running this command in the **Pi_Physical_Dashboard** folder:

```
nano config.ini
```

Inside the file you'll see comments (lines that begin with #) that describe the configuration.

At a high level the configuration file defines a list of widgets which are items on the dashboard that can be manipulated. For example a seven segment LED display or automotive stepper gauge is a widget that be added to a dashboard. The software for this project will run a little REST API server that allows any program to control the value displayed by a widget on the dashboard.

Widgets are defined by sections in the config.ini file. A section starts with the name of the widget in square brackets. On the following lines are configuration options for the widget. In particular every widget must have a type option that defines the type of widget, like SevenSegmentWidget for a seven segment display. Depending on the type of widget there might be other options provided.

For example the configuration shows a commented out seven segment widget:

```
# Example of a seven segment display called 'cpu'.  The type is SeveSegmentWidget,
# the I2C address of the device is specified as 0x74, and the number of digits
# to show after the decimal point is set to 0.
#[cpu]
#type = SevenSegmentWidget
#address = 0x74
#decimal_digits = 0
```

If the comments are removed then the widget would be enabled:

```
# Example of a seven segment display called 'cpu'.  The type is SeveSegmentWidget,
# the I2C address of the device is specified as 0x74, and the number of digits
# to show after the decimal point is set to 0.
[cpu]
type = SevenSegmentWidget
address = 0x74
decimal_digits = 0
```

Notice this widget has the name **cpu** (the value inside the square brackets), has a type of **SevenSegmentWidget** (i.e. is a seven segment LED display), and sets a few seven segment display specific options like the I2C address to 0x74 and the number of digits to display after to decimal point to zero.

Scroll through the rest of the configuration file and you can see examples of all the possible widget types and their options:

- **SevenSegmentWidget** - seven segment LED display
- **AlphaNum4Widget** - quad alphanumeric LED display
- **BicolorBargraph24Widget** - bicolor 24 segment LED bar graph display
- **AutoGaugeWidget** - automotive stepper motor gauge

Add to the config.ini your own widget configuration. For example if you wanted to define a seven segment display called **foo** (with I2C address 0x72) and an automotive gauge called **bar** (on motor HAT port M1) your config would look like:

```
[foo]
type = SevenSegmentWidget
address = 0x72

[bar]
type = AutoGaugeWidget
motor_id = 1
```

Save the configuration file and exit nano by pressing **Ctrl-o**, **enter** and then **Ctrl-x**.

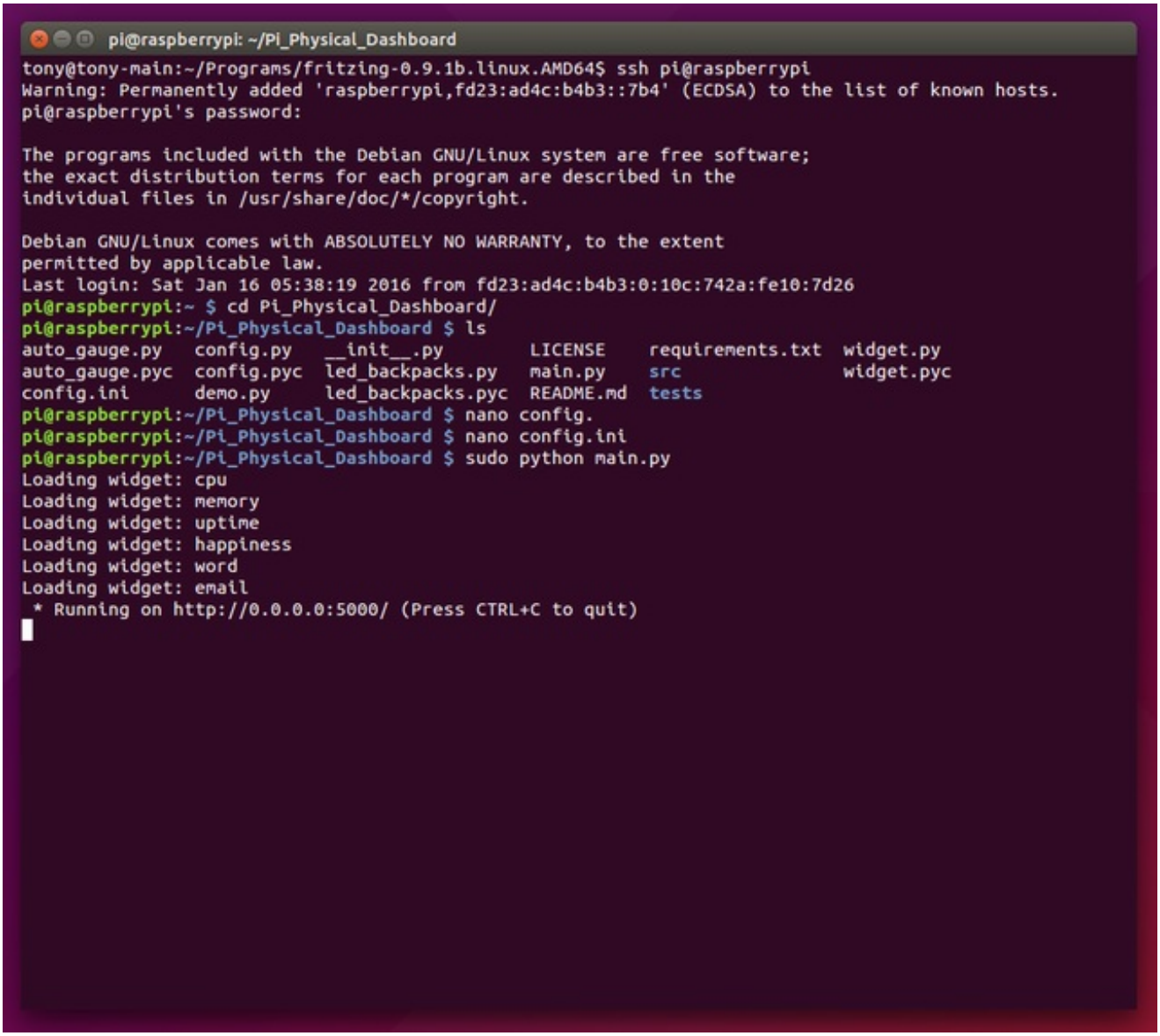
Usage

Once the dashboard configuration has been set you're ready to run the program. In the **Pi_Physical_Dashboard** folder execute the following command to start the server:

```
sudo python main.py
```

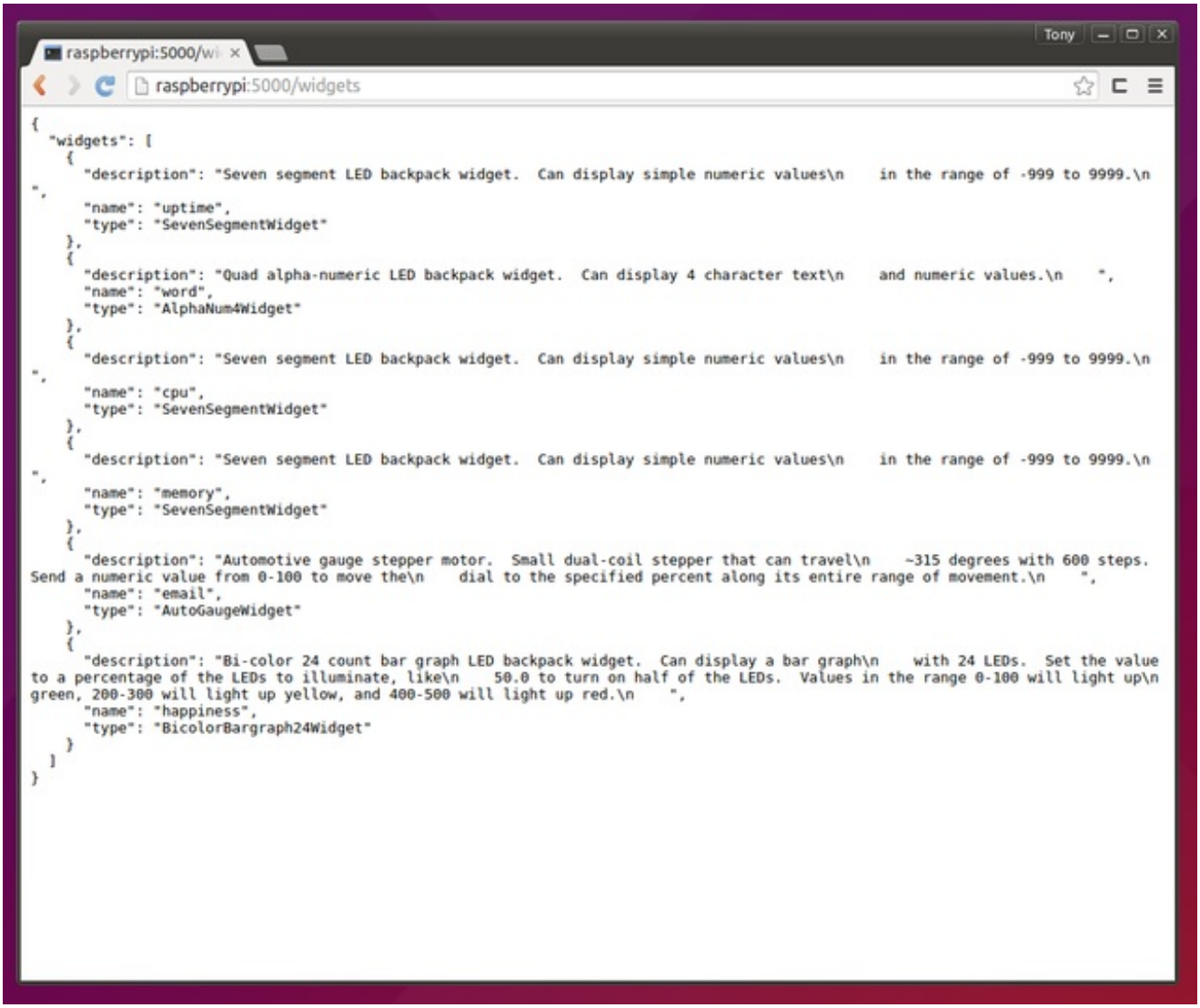
NOTE: When you start the dashboard server it will move all automotive gauges as far left as possible to put them in an initial 'home' position. You might see the gauge vibrate as it hits the stops which prevent it from moving completely around. This is OK and not something that will damage the gauge. Once the gauge stops moving you should remove the dial from the shaft and place it back on so it's pointing in the 0 position you want for your dashboard.

If the dashboard starts successfully you should see a line printed with the name of each configured widget, similar to the following:



If the dashboard fails to start check the output to see if it gives a hint as to what widget might be failing. In particular make sure the I2C address of each widget is set correctly. Use the i2cdetect tool (http://adafru.it/dEO) to check the Pi can see the widget at the address you expect.

To check the server is working you can access its widget list from your web browser. Open a browser and connect to http://raspberrypi:5000/widgets (http://adafru.it/kNa) (if you renamed your pi change raspberrypi to the name of your pi, or if that fails check your router to see what IP address is assigned to the Pi and use that IP in the URL). You should see a JSON formatted list of widgets, for example:



This widget list endpoint is useful for checking that all the configured widgets are available.

Now to set the value of a widget you can make an **HTTP POST** request against the http://raspberrypi:5000/widgets/widget_name (http://adafru.it/kNb) endpoint, where **widget_name** is the name assigned to a the widget. As either a query parameter or form parameter in the body of the request, add a parameter called **value** which is set to the value you'd like to display on the widget.

An easy way to make this request is with the curl (http://adafru.it/kNc) command. You can install curl on your computer (if using Windows you might need to install cygwin (http://adafru.it/kNd) to get curl) or you can even run curl from the Raspberry Pi running the dashboard.

For example to set the value of the widget named **cpu** to 55 you would run the following curl command:

```
curl -d value=55 http://raspberrypi:5000/widgets/cpu
```

Try using curl to set a value for a widget you have defined on your dashboard. You should see the widget update with the new value! If you don't see the widget update make sure you are using the right name for the widget. Also check the output of the main.py process to see if it prints a more useful error message to help you debug the issue.

One important thing to note is that each widget has a range of values they can have set:

- **SevenSegmentWidget** - Set any floating point numeric value in the range of -999 to 9999. Depending on the decimal digit precision the value might be rounded to fit on the display. If the value won't fit then ---- is displayed. You can also send the value **colon_on** to turn on the colon, and **colon_off** to turn off the colon.
- **AlphaNum4Widget** - Set any floating point numeric value just like the seven segment widget, or set any 4 character alphanumeric string. Long strings will be truncated to their first 4 characters.
- **BicolorBargraph24Widget** - Set any floating point percentage value in the range of 0 to 100 (don't include a % sign). This will illuminate the specified percent of LEDs green. You can also send a value in the range 200-300 to set that percent of LEDs yellow, or 400-500 to set that percent of LEDs red.
- **AutoGaugeWidget** - Set any floating point percentage value in the range of 0 to 100 (don't include a % sign). This will move the gauge's needle to the specified percent value within its range of movement. I.e. a value of 50 will move to the middle of its 315 degree movement arc.

Some more examples of setting values with curl look like:

```
# Set an alphanumeric widget called foo to have a value COOL:
curl -d value=COOL http://raspberrypi:5000/widgets/foo

# Set a bicolor bargraph widget called bar to be 35% illuminated:
curl -d value=35 http://raspberrypi:5000/widgets/bar

# Set a bicolor bargraph widget called bar to be 75% illuminated red:
curl -d value=475 http://raspberrypi:5000/widgets/bar

# Set an auto gauge widget called gauge to be at 35%:
curl -d value=35 http://raspberrypi:5000/widgets/gauge
```

NOTE: If your auto gauge widget moves the wrong way you can invert its movement with the invert option for the widget. See the comments in the config.ini file for details.

That's all there is to updating data on the dashboard! You can use any programming language which can make HTTP requests (which is pretty much every language!) to control a dashboard. For example with Python you can use the requests (http://adafru.it/kNe) library to make HTTP POST requests against a dashboard. See the demo.py (http://adafru.it/kNf) code included in the source for the project to see an example of Python controlling a dashboard with random values.

If you'd like to make the dashboard start every time the Pi boots, see this excellent guide on using systemd (http://adafru.it/jXa) to create a service that runs the dashboard's main.py script. The latest Raspbian Jessie operating system uses systemd just like the BeagleBone Black mentioned in the guide.

Woo hoo, you now have a cool physical dashboard you can control with web requests! Hook up the dashboard to display interesting metrics like load from a server, the status of sensors in your home, and much more.

Adafruit IO

Now that you have a cool physical dashboard built, how do you hook it up to Adafruit IO? Check out the video below that walks through creating a simple Python script to display Adafruit IO feeds on a physical dashboard. Below the video are the basic steps to follow and the code to use.

Setup

To hook up the Pi physical dashboard to Adafruit IO you'll first need to do a little bit of setup. Before you start make sure you have the physical dashboard project assembled and working (using curl or other tools to test it). In addition you'll want to have an account on Adafruit IO setup (visit http://io.adafruit.com (http://adafru.it/eIC) to join the beta), and it will help to familiarize yourself with Adafruit IO feeds and dashboards by checking out these AIO basics guides (http://adafru.it/iDX).

Next you'll need to connect to the Pi and run the following commands to install a Python Adafruit IO client library:

```
sudo pip install adafruit-io
sudo pip install requests
```

Now configure your physical dashboard with the gauges and widgets you'd like to display. For this example I'm using the following dashbord config.ini:

```
[slider]
type = SevenSegmentWidget
address = 0x74
decimal_digits = 0

[humidity]
type = SevenSegmentWidget
address = 0x76
decimal_digits = 2

[temp]
type = SevenSegmentWidget
address = 0x71
decimal_digits = 1
```

This configuration defines the following three 7-segment displays:

- **slider** - A 7-segment display at I2C address 0x74 with no digits after the decimal point

displayed.

- **humidity** - A 7-segment display at I2C adress 0x76 with 2 digits after the decimal point displayed.
- **temp** - A 7-segment display at I2C address 0x71 with 1 digit after the decimal point displayed.

Code

To hook up the dashboard to display Adafruit IO feeds you can create a simple Python script similar to the one shown below. For example create a script called **aio_dashboard.py (http://adafru.it/kME)** on the Raspberry Pi and fill in:

```
# Raspberry Pi Physical Dashboard & Adafruit IO Example
# Author: Tony DiCola
# License: Public domain
#
# This example will display three feeds from Adafruit IO on a dashboard.  The
# feeds are:
#  - pi-dashboard-slider: Will display a slider with values from 0-100 on a
#                         dashboard 7-segment display called 'slider'.
#  - pi-dashboard-humidity:  Humidity sensor feed displayed on a 7-segment display
#                            called 'humidity'.
#  - pi-dashboard-temp: Temperature sensor feed displayed on a 7-segment display
#                       called 'temp'.
#
# To modify this example to use your own feeds you'll want to change the following
# code:
#  - connected function: Add a client.subscribe() call for each feed you want to
#                        display on the dashboard.
#  - message function:  Add an if/elif check to check for each feed you want
#                       to display and have it make an HTTP POST against the
#                       dashboard using the feed payload as a value.
#  - DASHBOARD_URL variable: Change this to the URL of your Pi dashboard if you
#                            aren't running this script on the same Pi as the
#                            dashboard (otherwise keep it the same localhost:5000
#                            value).  Make sure NOT to end this with a slash!
#
# Be sure to modify the ADAFRUIT_IO_KEY and ADAFRUIT_IO_USERNAME variables to
# set your AIO key and username!

# Import standard python modules.
import random
import sys
import time
```

```
# Import Adafruit IO MQTT client.
from Adafruit_IO import MQTTClient

# Import requests library used for making HTTP calls to the dashboard server.
import requests


# Set to your Adafruit IO key & username below.
ADAFRUIT_IO_KEY      = 'YOUR ADAFRUIT IO KEY'      # Set to your Adafruit IO key.
ADAFRUIT_IO_USERNAME = 'YOUR ADAFRUIT IO USERNAME'  # See https://accounts.adafruit.com
                                                    # to find your username.
# Set the URL of the physical dashboard to use.  If running on the same Pi as
# the dashboard server then keep this the default localhost:5000 value.  If
# modified make sure not to end in a slash!
DASHBOARD_URL = 'http://localhost:5000'  # URL of the physical dashboard.
                                        # Don't end with a slash!


# Define callback functions which will be called when certain events happen.
def connected(client):
    # Connected function will be called when the client is connected to Adafruit IO.
    # This is a good place to subscribe to feed changes.  The client parameter
    # passed to this function is the Adafruit IO MQTT client so you can make
    # calls against it easily.
    print 'Connected to Adafruit IO!  Listening for feed changes...'
    # Subscribe to the three pi-dashboard feeds that will be displayed on the
    # dashboard.  Modify this to subscribe to all the feeds you want to display.
    client.subscribe('pi-dashboard-slider')
    client.subscribe('pi-dashboard-humidity')
    client.subscribe('pi-dashboard-temp')

def disconnected(client):
    # Disconnected function will be called when the client disconnects.
    print 'Disconnected from Adafruit IO!'
    sys.exit(1)

def message(client, feed_id, payload):
    # Message function will be called when a subscribed feed has a new value.
    # The feed_id parameter identifies the feed, and the payload parameter has
    # the new value.
    print('Feed {0} received new value: {1}'.format(feed_id, payload))
    # Update physical dashboard depending on the changed feed.
    # Notice the feed_id is checked to find out which feed changed, then the
    # appropriate physical dashboard widget is changed.
    if feed_id == 'pi-dashboard-slider':
        # The requests.post function will make an HTTP request against the
        # dashboard.  See the requests documentation for more information:
```

```
        # dashboard.  See the requests documentation for more information:
        #   http://docs.python-requests.org/en/latest/
        requests.post('{0}/widgets/slider'.format(DASHBOARD_URL), data={'value': payload})
    elif feed_id == 'pi-dashboard-humidity':
        requests.post('{0}/widgets/humidity'.format(DASHBOARD_URL), data={'value': payload})
    elif feed_id == 'pi-dashboard-temp':
        requests.post('{0}/widgets/temp'.format(DASHBOARD_URL), data={'value': payload})


# Create an MQTT client instance.
client = MQTTClient(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

# Setup the callback functions defined above.
client.on_connect    = connected
client.on_disconnect = disconnected
client.on_message    = message

# Connect to the Adafruit IO server.
client.connect()

# Use the loop_blocking function to run the message loop for processing Adafruit
# IO events.  Since this script doesn't do any other processing this blocking
# version of the message loop is fine.  All the program logic will occur in the
# callback functions above when Adafruit IO feeds are changed.
client.loop_blocking()
```

This script is based on the **mqtt_client.py (http://adafru.it/l7e)** example from the Adafruit IO Python library, but with a few small changes to make it send feed data to a dashboard. You'll want to modify a few parts of the script as described below.

First set your Adafruit IO key and username in this section near the top of the script:

```
# Set to your Adafruit IO key & username below.
ADAFRUIT_IO_KEY      = 'YOUR ADAFRUIT IO KEY'      # Set to your Adafruit IO key.
ADAFRUIT_IO_USERNAME = 'YOUR ADAFRUIT IO USERNAME'  # See https://accounts.adafruit.com
                                                    # to find your username.
```

(you might have noticed in the video above I load the AIO key from a local file, however it's not really necessary to load from a file unless you'd like to keep the key out of the script code)

Next if you're running this script outside of the Pi that's running your dashboard you'll need to change the **DASHBOARD_URL** variable below:

```
# Set the URL of the physical dashboard to use.  If running on the same Pi as
# the dashboard server then keep this the default localhost:5000 value.  If
# modified make sure not to end in a slash!
DASHBOARD_URL = 'http://localhost:5000'  # URL of the physical dashboard.
                                        # Don't end with a slash!
```

Make sure the **DASHBOARD_URL** value is set to the IP address or hostname of your Pi that's running the dashboard. If you're running this script on the same Pi as the dashboard then leave it as http://localhost:5000 (). Make sure this URL does **not** end in a slash!

Now modify the **connected** function code to subscribe to all the feeds you wish to display on the dashboard. This function will be called when a successful connection is made to the Adafruit IO service and it serves as a good point to subscribe to feeds or do other initialization.

```
# Define callback functions which will be called when certain events happen.
def connected(client):
    # Connected function will be called when the client is connected to Adafruit IO.
    # This is a good place to subscribe to feed changes.  The client parameter
    # passed to this function is the Adafruit IO MQTT client so you can make
    # calls against it easily.
    print 'Connected to Adafruit IO!  Listening for feed changes...'
    # Subscribe to the three pi-dashboard feeds that will be displayed on the
    # dashboard.  Modify this to subscribe to all the feeds you want to display.
    client.subscribe('pi-dashboard-slider')
    client.subscribe('pi-dashboard-humidity')
    client.subscribe('pi-dashboard-temp')
```

Notice I'm subscribing to three feeds in the above code:

- **pi-dashboard-slider**: This will be displayed on the **slider** dashboard widget.
- **pi-dashboard-humidity**: This will be displayed on the **humidity** dashboard widget.
- **pi-dashboard-temp**: This will be displayed on the **temp** dashboard widget.

Change the code to subscribe to each of the feeds you're using.

Finally modify the **message** function code to look for your feeds and push their payload to the dashboard:

```
def message(client, feed_id, payload):
    # Message function will be called when a subscribed feed has a new value.
    # The feed_id parameter identifies the feed, and the payload parameter has
    # the new value.
    print('Feed {0} received new value: {1}'.format(feed_id, payload))
    # Update physical dashboard depending on the changed feed.
    # Notice the feed_id is checked to find out which feed changed, then the
    # appropriate physical dashboard widget is changed.
    if feed_id == 'pi-dashboard-slider':
        # The requests.post function will make an HTTP request against the
        # dashboard.  See the requests documentation for more information:
        #   http://docs.python-requests.org/en/latest/
        requests.post('{0}/widgets/slider'.format(DASHBOARD_URL), data={'value': payload})
    elif feed_id == 'pi-dashboard-humidity':
        requests.post('{0}/widgets/humidity'.format(DASHBOARD_URL), data={'value': payload})
    elif feed_id == 'pi-dashboard-temp':
        requests.post('{0}/widgets/temp'.format(DASHBOARD_URL), data={'value': payload})
```

Notice the message function checks the **feed_id** parameter for each of the feeds that were subscribed to earlier. If one of those feeds changes then the requests library (http://adafru.it/kNe) is used to make a HTTP POST request that updates the appropriate dashboard widget. The **payload** parameter will hold the updated value of the feed and is used by the requests.post function to send the updated feed value to the dashboard.

For example in the code above changes to the **pi-dashboard-humidity** feed will update the dashboard widget called **humidity** (which is exposed at the http://raspberrypi:5000/widgets/humidity (http://adafru.it/l7f) URL with the dashboard).

You'll want to modify the if/elif statements to check for your feeds and push the value to the appropriate dashboard widget.

That's all you need to change to use this script! Make sure your physical dashboard is running on the Pi, then connect to the Pi and run the script:

```
python aio_dashboard.py
```

The script should print out a message when it's connected to Adafruit IO, and when it receives a subscribed feed change. Build a dashboard on Adafruit IO or use another script to update your subscribed feeds and you should see those feeds appear on your physical dashboard widgets--cool!

If you see an error that the script can't connect to Adafruit IO, make sure you have the AIO key and

username correctly set for your account.

If you don't see your feed updated on the physical dashboard when it changes on Adafruit IO check a few things:

- Make sure the script is successfully connecting to Adafruit IO (i.e. your username and key are correct).
- Make sure you're subscribing to the feeds that you're changing.
- Make sure the if/elif checks in your message function are looking for the feeds you've subscribed to and are changing.
- Check the DASHBOARD_URL points to the URL of the Pi running your dashboard (or the default localhost:5000 if running the script on the same Pi as the dashboard).