



Monitor PiCam and temperature on a PiTFT via adafruit.io

Created by Jeremy Blythe



<https://learn.adafruit.com/monitor-picam-and-temperature-on-a-pitft-via-adafruit-dot-io>

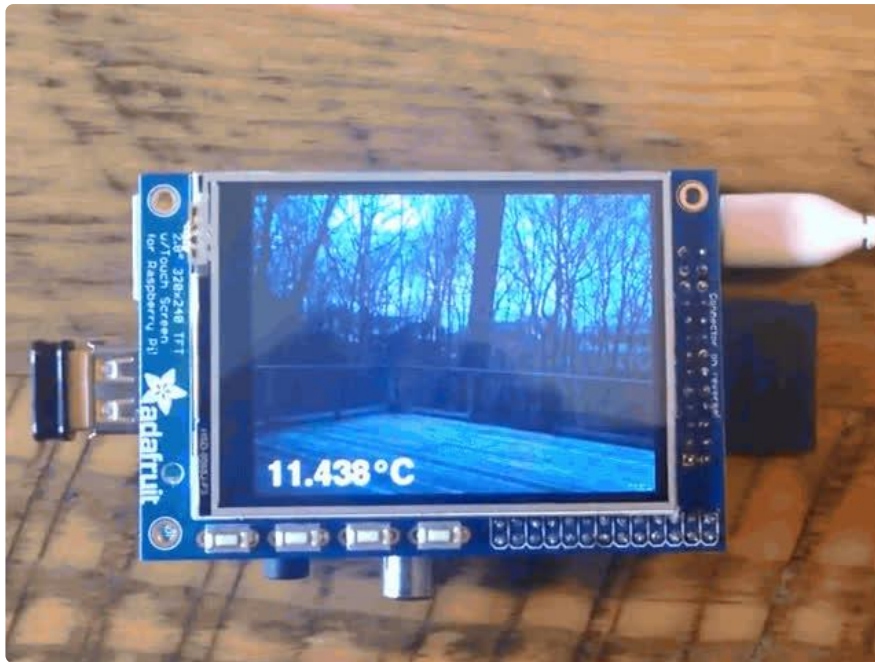
Last updated on 2022-12-01 02:42:54 PM EST

Table of Contents

Overview	3
<ul style="list-style-type: none">• Shopping list	
Sender	4
<ul style="list-style-type: none">• Hardware setup• Sending snapshots to Adafruit.io• Sending temperature readings	
Receiver	10
<ul style="list-style-type: none">• Setup• MQTT events• Drawing and Switching pages• Connecting and looping• Temperature Graph	

Overview

Did you know you can save and retrieve images from adafruit.io? You can! That means you can have a Raspberry Pi with a camera upload images to the service, along with other sensor data. Then, another Pi on the other side of the world can view the images and data, to create a custom remote-viewing tracker. It's like your very own Nest but you can add any sort of data overlay you like.



This project uses two Raspberry Pis - a sender and a receiver. The sender has a Raspberry Pi Camera and an MCP9808 temperature sensor to publish data to adafruit.io. The receiver, a dashboard somewhere else in the world, subscribes to this data feed and displays it.

This dashboard Raspberry Pi has a PiTFT and displays the image whenever it's sent to the feed (every 5 minutes), the current temperature is overlaid on the image using pygame. The final cherry on the cake here is that if you tap the screen you flip to the graph view. This takes the data from the feed using the `io-client-python` data method, pulls out the last 24 hours and uses matplotlib to draw a graph of temp/time. Of course, you can see the feeds in the adafruit.io online dashboard too!

Shopping list

To follow this guide exactly you'll need:

- 2x [Raspberry Pi - you can use anything from a Pi 1 Model B to a Pi 3 or beyond, but a Pi 2 or 3 works best](#) ()
- 1x [PiTFT 2.8", 3.2" or HAT, \(\)](#) get one that matches your Pi
- 1x [MCP9808 temperature sensor board](http://adafru.it/1782) (<http://adafru.it/1782>)
- 1x [Raspberry Pi Camera](#) ()
- 2x [Wifi dongles or ethernet cables \(or built in WiFi on a Pi 3\)](#) (<http://adafru.it/814>)
- 4x [Female to female jumper cables](http://adafru.it/266) (<http://adafru.it/266>)
- A plastic container for the sender
- Some duct tape

Sender

Hardware setup



I've just used an old plastic food carton and cut holes in the side and front for the cabling and camera. As this box is outside I've kept the number of holes to a minimum. It's worth having long enough jumper cables so the temperature sensor can hang outside the box. I was getting slightly high readings when I had it in the box with the lid on due to the small amount of heat produced by the Pi.

Some duct tape holds the camera in place behind the hole.



Four female to female jumper cables are all you need to wire up the MCP9808. This guide has a nice diagram explaining the connections: [MCP9808 temperature sensor raspberry pi \(\)](#)

Sending snapshots to Adafruit.io

First create your adafruit.io account, here's the guide if you're stuck: [Adafruit.io getting started \(\)](#)

Now follow this guide, [Motion setup \(\)](#) but don't edit any of the settings in motion.conf we're going to use a different set of configuration in this project to simply send a snapshot every 5 minutes. When you get more familiar with Motion you can change the configuration to do motion detection to [upload videos to the cloud \(\)](#) and all sorts as well as the 5 minute snapshot if you wish.

You will likely also want to have Motion running on boot up so follow [this section of the guide \(\)](#) too.

Next we need to install the uploader code for this project. First, install the required packages:

```
cd ~
sudo pip install adafruit-io
```

```
sudo apt-get install -y libjpeg-dev python-dev
sudo pip install Pillow --upgrade
```

Now get the project code. The adaiot project includes all the code for the sender and the receiver but at this stage we're interested in the image uploader script.

```
pi@raspberrypi:~ $ git clone https://github.com/jerbly/adaiot.git
Cloning into 'adaiot'...
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 13 (delta 0), reused 0 (delta 0), pack-reused 8
Unpacking objects: 100% (13/13), done.
Checking connectivity... done.
```

Get your secret AIO key and user name from Adafruit.io and edit the code in `adaiot-uploader.py`. Replace the word `SECRET` with your secret AIO key.

```
#!/usr/bin/python

from Adafruit_IO import Client
from PIL import Image
import base64

aio = Client('SECRET')
BASE_DIR='/var/lib/motion/'
SRC_FILE=BASE_DIR+'lastsnap.jpg'
DST_FILE=BASE_DIR+'lastsmall.jpg'

fd_img = open(SRC_FILE, 'r')
img = Image.open(fd_img)
size = 320, 240
img.thumbnail(size)
img.save(DST_FILE, img.format)
fd_img.close()

with open(DST_FILE, "rb") as imageFile:
    str = base64.b64encode(imageFile.read())

aio.send('pic', str )
```

The script is fairly simple. First of all it logs into adafruit.io using the REST client. We don't need MQTT here because we're simply sending a value in to a feed. We then look for the `lastsnap.jpg` file and use the python image library (PIL from the Pillow package) to create a `smaller image () lastsmall.jpg`. In the next part of this project I'm using a 320x240 screen to display the feed so that's why we're resizing in this script. Of course, we could set Motion to 320x240 and then we wouldn't need the resize but, as I said earlier, you're likely going to want to use Motion for a few different purposes and 320x240 is pretty small! Finally the code converts the output jpeg to base64 and sends it to the pic feed.

Now we need to set up Motion to take a snapshot every 5 minutes and call this script when it does. So change some settings in `/etc/motion/motion.conf`

```
sudo pico /etc/motion/motion.conf
```

```
width 1024
height 768
output_pictures off
snapshot_interval 300
snapshot_filename lastsnap
on_picture_save /home/pi/adaiot/adaiot-uploader.py
```

Restart the motion service so the changes take effect:

```
sudo service motion restart
```

Now we can set up the adafruit.io dashboard to display the feed:

1. Log in and select "Your Dashboards"
2. Click "Create Dashboard"
3. Give it a name and create it
4. Click on the + icon to add a block
5. Choose the image block
6. Choose the "pic" feed

CREATE A NEW BLOCK ×

STEP 1: CHOOSE BLOCK TYPE EDIT

STEP 2: CHOOSE FEEDS

ⓘ Add up to 1 feed

SEARCH NEW FEED NAME CREATE

FEED/GROUP	LAST VALUE	RECORDED	ACTION
☐ My Feeds			
pic	/9j/4AAQSkZJRgAB..	3 minutes ago	CHOOSE
deck-temp	3.250	half a minute ago	CHOOSE

NEXT STEP >

You should now see the snapshots from the camera sent every 5 minutes. At the time of writing the image block cycles through all the images received during the browser session, don't be alarmed!

OK, next we need to send the temperature too...

Sending temperature readings

This guide, [MCP9808 Temperature sensor python library \(\)](#), has instructions on how to set up the python library but it boils down to these few commands below:

```
cd ~
sudo apt-get install -y python-smbus
git clone https://github.com/adafruit/Adafruit_Python_MCP9808.git
cd Adafruit_Python_MCP9808/
sudo python setup.py install
```

Edit `~/adaiot/adaiot-temp.py` and put your AIO key instead of `SECRET`.

```
import time
import Adafruit_MCP9808.MCP9808 as MCP9808
from Adafruit_IO import Client

aio = Client('SECRET')

sensor = MCP9808.MCP9808()
sensor.begin()

def get_temperature():
    temp = sensor.readTempC()
    return '{0:0.3F}'.format(temp)

while True:
    try:
        aio.send('deck-temp', get_temperature() )
    except:
        print "Failed to send"
        time.sleep(30)
```

This script uses the REST client again as we're only sending. In a forever loop it simply reads the current temperature from the MCP9808 and formats it to 3 decimal places. The value is then sent to the `deck-temp` feed. You can change this feed name if you wish. I have this sensor out on my deck.

To set the temperature sender to run automatically when you reboot the Pi edit the `/etc/rc.local` file like so:

```
sudo pico /etc/rc.local
```

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
```



```

# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

python /home/pi/adaiot/adaiot-temp.py &
exit 0

```

Now we'll go back to the dashboard and add a gauge block. Choose the deck-temp feed. (Or the feed name you changed it to). Set the min and max values appropriately. The code above sends the temperature in celcius. I live in Canada near Toronto so I need a pretty wide range between min and max.

CREATE A NEW BLOCK ✕

STEP 1: CHOOSE BLOCK TYPE EDIT

STEP 2: CHOOSE FEEDS EDIT

STEP 3: BLOCK SETTINGS

BLOCK TITLE


GAUGE MIN VALUE

GAUGE MAX VALUE

GAUGE WIDTH
THIN

GAUGE LABEL

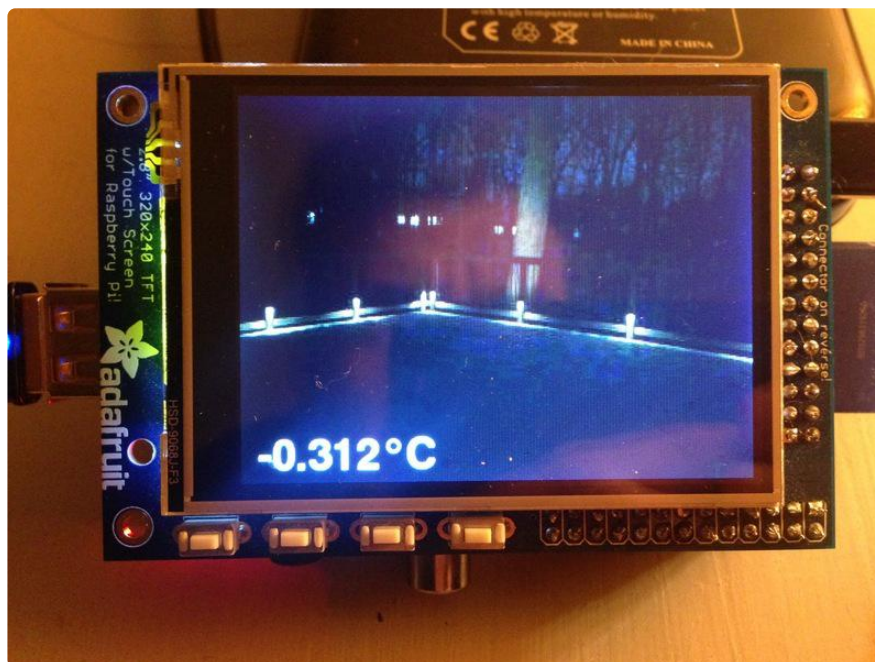
BLOCK PREVIEW



Hit create and you should see the new block on the dashboard next to your image block.



Receiver



As you can see from the picture, the receiver will be taking the images and temperature readings from the feed and displaying them on the PiTFT via Pygame. The overview from my [Raspberry Pi Pygame UI Basics \(\)](#) tutorial will get your environment all set up and running sweetly.

Once you've done that follow these steps to add some prerequisites for this project:

```
sudo pip install adafruit-io
sudo apt-get install python-matplotlib
```

Now go ahead and grab the project code again as we did on the Sender:

```
cd ~
git clone https://github.com/jerbly/adaiot.git
```

There's quite a lot going on in this code so let's walk through it in a few sections.

Setup

In this section we're importing all the required libraries, setting up a few constants and initialising pygame. You may want to change some of the constants:

- `ADAFRUIT_IO_KEY` and `ADAFRUIT_IO_USERNAME` - set these to your secret AIO key and your Adafruit account user name
- `PIC_FEED` and `TEMP_FEED` hold the feed names that must match the Sender
- `LCD_WIDTH` and `LCD_HEIGHT` are used throughout the code, set these to match your screen size
- `LUM_SAMPLE_POS` (explained later) is the pixel we will sample for luminance

```
import time
import base64
import pygame
from pygame.locals import *
import os
from Adafruit_IO import MQTTClient
import feed
import sys
import signal

# Set to your Adafruit IO key & username below.
ADAFRUIT_IO_KEY = 'SECRET'
ADAFRUIT_IO_USERNAME = 'SECRET' # See https://accounts.adafruit.com to find your
username.

PIC_FEED = 'pic'
TEMP_FEED = 'deck-temp'
OUT_DIR = '/home/pi/adafruit/'
LCD_WIDTH = 320
LCD_HEIGHT = 240
LCD_SIZE = (LCD_WIDTH, LCD_HEIGHT)
LUM_SAMPLE_POS = 30,220

BLACK = 0,0,0
GREEN = 0,255,0
RED = 255,0,0
WHITE = 255,255,255

os.putenv('SDL_FBDEV', '/dev/fb1')
os.putenv('SDL_MOUSEDRV', 'TSLIB')
os.putenv('SDL_MOUSEDEV', '/dev/input/touchscreen')
pygame.init()
pygame.mouse.set_visible(False)
lcd = pygame.display.set_mode(LCD_SIZE)
lcd.fill(BLACK)
font_big = pygame.font.Font(None, 40)

image_surface = None
text_surface = None
lum = 100
page = 0
```

MQTT events

Here we set up some event handlers for the MQTT session.

- **connected** is called when the connection to the Adafruit servers is established. At this point we subscribe to the two feeds. The Adafruit servers immediately send back the last entry in the feed and then every new entry as it comes in.
- **disconnected** is called when ever our connection drops. The underlying MQTT library has a reconnect loop which will kick in and attempt to get you connected again. So we just print a message for debug.
- **message** is called whenever a new feed entry comes in.
 - If it's a new image we decode the base64 to a jpeg and then load it back into a pygame surface. Next we grab the [colour of a pixel to get its luminance \(\)](#) this is so we can determine whether we need a light or dark text colour over the background.
 - If it's a temperature value we choose black or white text colour according to the background luminance and render this to a surface.

```
def connected(client):
    print 'MQTT Connected'
    client.subscribe(PIC_FEED)
    client.subscribe(TEMP_FEED)

def disconnected(client):
    print 'MQTT Disconnected from Adafruit IO!'

def message(client, feed_id, payload):
    global image_surface, text_surface, lum
    if feed_id == PIC_FEED:
        print 'MQTT received pic'
        fh = open(OUT_DIR+"testjr.jpg", "wb")
        fh.write(payload.decode('base64'))
        fh.close()
        image_surface = pygame.image.load(OUT_DIR+"testjr.jpg")
        col = image_surface.get_at(LUM_SAMPLE_POS)
        lum = (0.299*col.r + 0.587*col.g + 0.114*col.b)
        # IF RESIZE REQUIRED
        #surf = pygame.transform.scale(surf, LCD_SIZE)
    elif feed_id == TEMP_FEED:
        print 'MQTT received temp: {}'.format(payload)
        if lum < 75:
            col = WHITE
        else:
            col = BLACK
        text_surface = font_big.render(payload+u'\u00B0C', True, col)
    show_dash()
```

Drawing and Switching pages

`show_dash` is the main function for rendering the surfaces to the LCD.

- If we're on page 0 it first paints the `image_surface` if we have one, otherwise it's just a black fill. Next it positions the `text_surface` in the bottom left of the screen.
- If we're on page 1 it paints the temperature graph. (More on this later)

You'll see later how we're using the touchscreen to flip between pages.

`switch_page` simply flips between page 0 and 1.

```
def show_dash():
    if page == 0:
        if image_surface:
            lcd.blit(image_surface, (0,0))
        else:
            lcd.fill(BLACK)
        if text_surface:
            rect = text_surface.get_rect()
            rect.x = 10
            rect.y = LCD_HEIGHT-rect.height-2
            lcd.blit(text_surface, rect)
    elif page == 1:
        feed_surface = pygame.image.load(OUT_DIR+"temps.png")
        lcd.blit(feed_surface, (0,0))

    pygame.display.update()

def switch_page():
    global page
    if page == 1:
        page = 0
    else:
        page += 1

    show_dash()
```

Connecting and looping

In this last section there are three significant things happening:

- First we set up the signal handling so we can close the program cleanly. There are multiple threads in this program so we need to handle ctrl+c and kill signals to gracefully stop.
- Second we set up and connect to Adafruit.io over MQTT

- Finally the main loop. Here we're doing to key things:
 - We check for `MOUSEBUTTONUP` events so we can call `switch_page` if the screen is touched.
 - Next, we use a counter to trigger a secondary thread every 5 minutes. This other thread creates the temperature graph explained in the next section.

```
def signal_handler(signal, frame):
    print 'Received signal - exiting'
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)
signal.signal(signal.SIGTERM, signal_handler)

# 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.
print 'Attempting to connect MQTT...'
client.connect()

client.loop_background()
chart_counter = 0
while True:
    for event in pygame.event.get():
        if event.type is MOUSEBUTTONUP:
            switch_page()
    time.sleep(0.1)
    # Create a new chart approx every 5 mins
    chart_counter += 1
    if chart_counter == 3000:
        chart_counter = 0
        feed.ChartThread(ADAFRUIT_IO_KEY, TEMP_FEED, OUT_DIR).start()
```

Temperature Graph

`feed.py` defines a separate thread to be called periodically to construct the temperature graph. This is done in a separate thread to not spoil the responsiveness of the touchscreen to flip between pages. As you will see it's quite a heavy operation to gather the data and generate the graph.

After connecting to Adafruit.io using the REST client we make a call to retrieve the data from the temperature feed. This uses the [Data Retrieval \(\)](#) API to get all the data from a feed. At the moment there's no way to restrict what's returned, you get everything. So we have to filter the data to only grab entries from the last 24 hours.

Each data object as it comes in actually has a number of elements to it. Here's what one entry actually looks like:

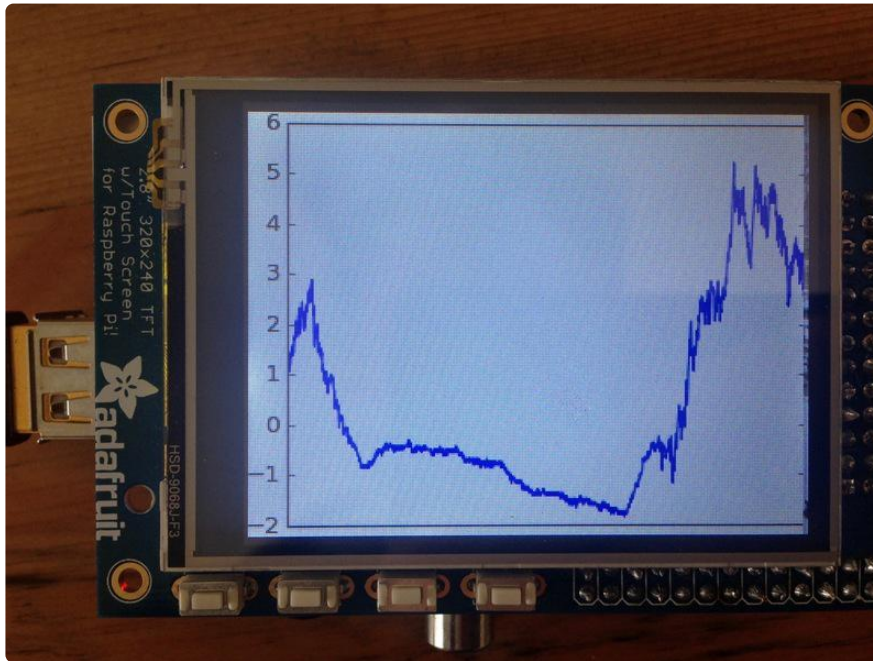
```
Data(created_epoch=1459912484.47247,  
created_at=u'2016-04-06T03:14:44.472Z',  
updated_at=u'2016-04-06T03:14:44.472Z', value=u'-0.875',  
completed_at=None, feed_id=514989, expiration=None, position=None,  
id=348921349)
```

We use the `created_epoch` to determine if the value was sent in the last 24 hours and as the x axis on the graph, `dates`. The temperature values for the y axis are stored in the `temps` list.

Finally these two lists are passed to [Matplotlib \(\)](#) to render a graph as an image. I'm certainly no expert with this graphing library but it does the trick. Getting it to create you an image the correct size is a black art!

```
from Adafruit_IO import Client  
import datetime  
import matplotlib  
matplotlib.use('Agg')  
import matplotlib.pyplot as plt  
from matplotlib.dates import date2num  
  
import threading  
  
class ChartThread(threading.Thread):  
    def __init__(self, client_key, feed_name, out_dir):  
        threading.Thread.__init__(self)  
        self._client_key = client_key  
        self._feed_name = feed_name  
        self._out_dir = out_dir  
  
    def run(self):  
        print "ChartThread connecting"  
        aio = Client(self._client_key)  
  
        print "ChartThread fetching data"  
        data = aio.data(self._feed_name)  
  
        today = datetime.datetime.now()  
        one_day = datetime.timedelta(days=1)  
        yesterday = today - one_day  
  
        dates = []  
        temps = []  
  
        print "ChartThread treating data"  
        for d in data:  
            ts = datetime.datetime.fromtimestamp(d.created_epoch)  
            if ts > yesterday:  
                dates.append(ts)  
                temps.append(d.value)  
  
        print "ChartThread plotting"  
        dates = date2num(dates)
```

```
fig = plt.figure()
fig.set_size_inches(4, 3)
plt.subplots_adjust(left=0.0, right=0.925, bottom=0.0, top=0.948)
ax = fig.add_subplot(111)
ax.plot_date(dates, temps, '-')
ax.axes.get_xaxis().set_visible(False)
plt.savefig(self._out_dir+'temps.png', dpi = 80, bbox_inches='tight',
pad_inches = 0)
plt.close(fig)
print "ChartThread done"
```



That's it! Remember to run the dashboard as root:

```
sudo python dash.py
```