Adafruit 128x64 OLED Bonnet for Raspberry Pi
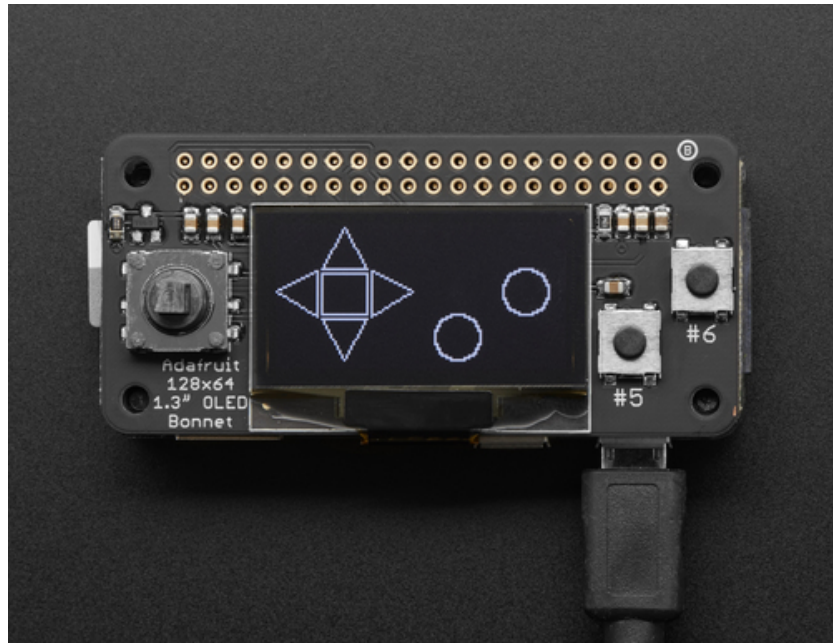
Created by lady ada
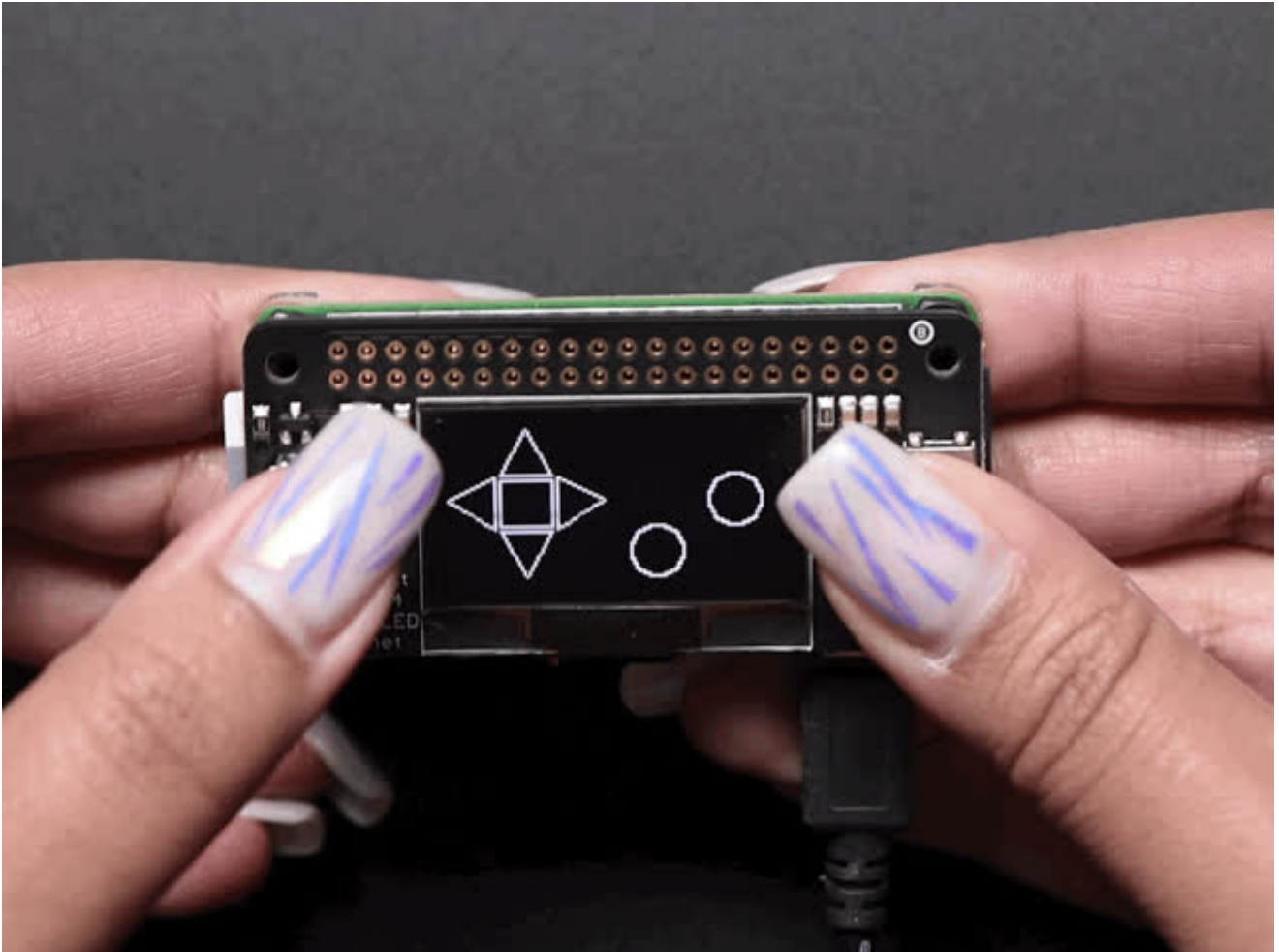

Last updated on 2017-06-09 09:15:51 PM UTC

Guide Contents

Overview

If you'd like a compact display, with buttons and a joystick - we've got what you're looking for. The Adafruit 128x64 OLED Bonnet for Raspberry Pi is the big sister to our mini PiOLED add-on (http://adafru.it/wVd). This version has 128x64 pixels (instead of 128x32) and a much larger screen besides. With the OLED display in the center, we had some space on either side so we added a 5-way joystick and two pushbuttons. Great for when you want to have a control interface for your project.
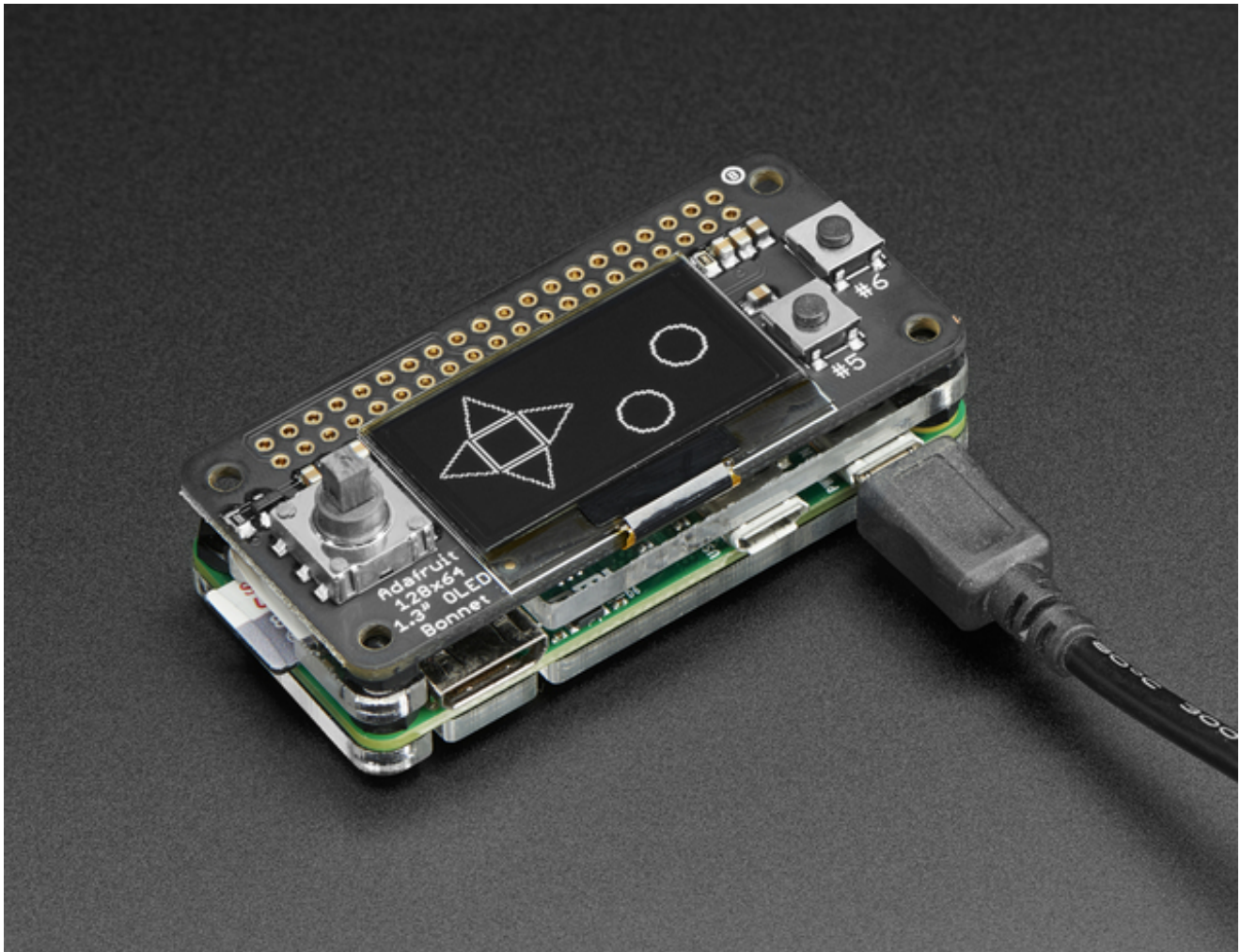
These displays are small, only about 1.3" diagonal, but very readable due to the high contrast of an OLED display. This screen is made of 128x64 individual white OLED pixels and because the display makes its own light, no backlight is required. This reduces the power required to run the OLED and is why the display has such high contrast; we really like this miniature display for its crispness!
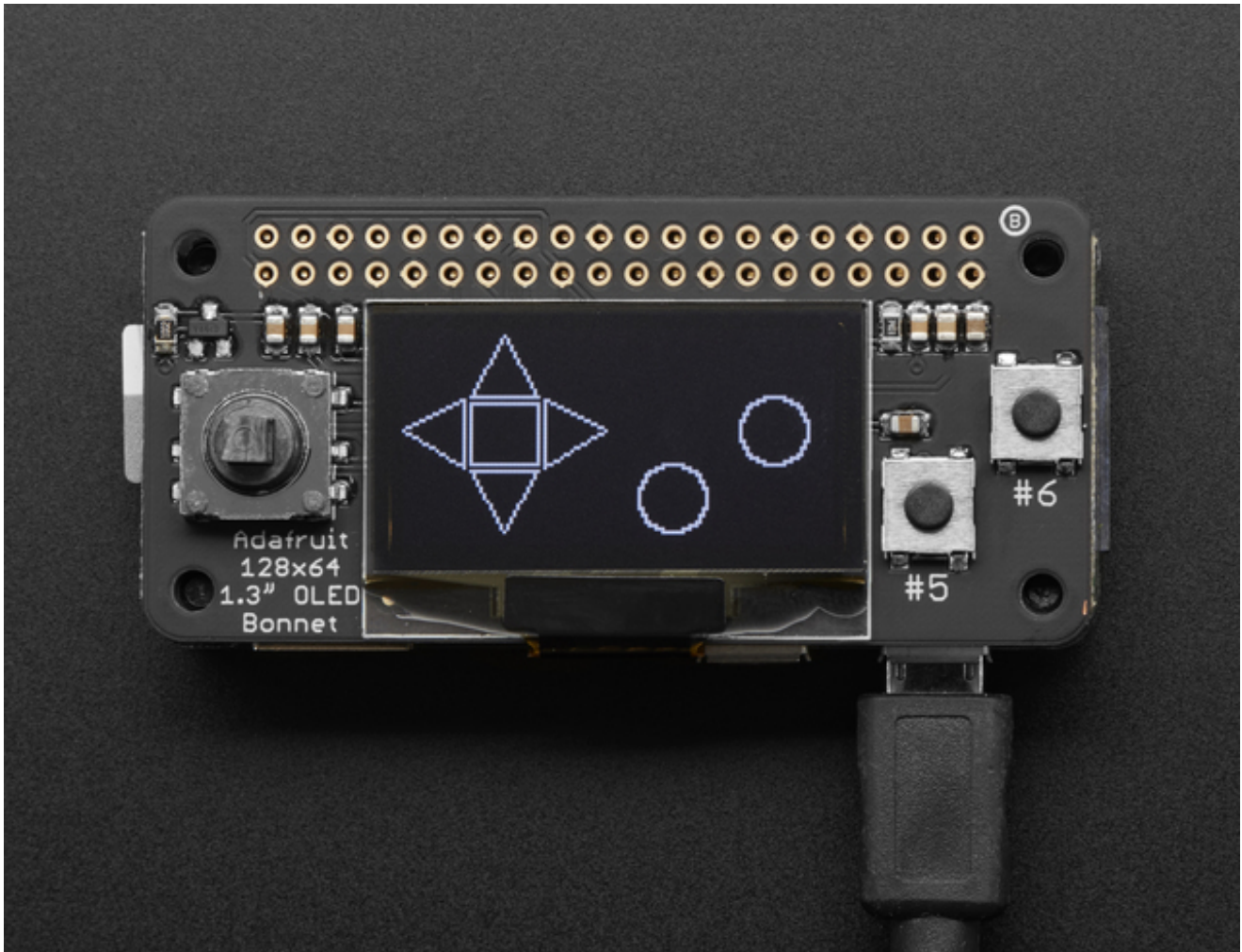
Please note that this display is too small to act as a primary display for the Pi (e.g. it can't act like or display what would normally be on the HDMI screen). Instead, we recommend using **pygame** for drawing or writing text.

Using the display and controls in python is very easy, we have a library ready-to-go for the SSD1306 OLED chipset and the joystick/buttons are connected to GPIO pins on the Pi. Our example code allows you to draw images, text, whatever you like, using the Python imaging library. We also have example code for using the joystick/buttons/OLED together. Our tests showed 15 FPS update rates once you bump the I2C speed to 1MHz, so you can do animations or simple video.
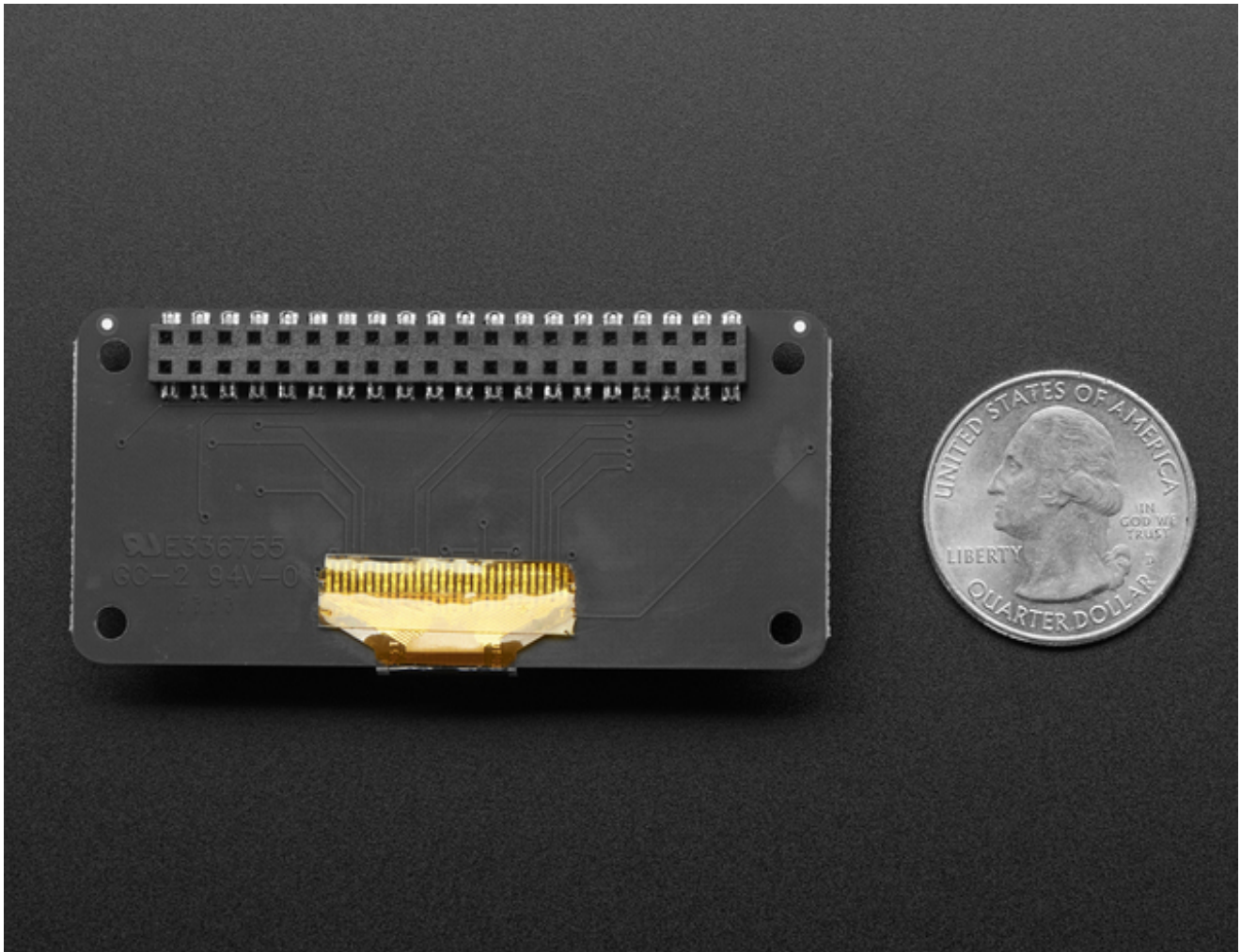
Comes completely pre-assembled and tested so you don't need to do anything but plug it in and install our Python code! Works with any Raspberry Pi computer, including the original Pi 1, B+, Pi 2, Pi 3 and Pi Zero.

Usage

We'll be using Python to control the display. In theory you can use any language you like that gives you access to the computer's I2C ports, but our library is for Python only!

This guide assumes you have your Raspberry Pi all set up with an operating system, network connectivity and SSH!

Step 1. Dependencies

Before using the library you will need to make sure you have a few dependencies installed. Connect to your Pi using SSH (http://adafru.it/vbC) and follow the steps below.

Install the RPi.GPIO library by running the following at the command line:

**sudo apt-get update
sudo apt-get install build-essential python-dev python-pip
sudo pip install RPi.GPIO**

Finally, install the Python Imaging Library (http://adafru.it/dvB) and smbus library by executing:

sudo apt-get install python-imaging python-smbus

Now to download and install the latest Adafruit SSD1306 python library code and examples, execute the following commands:

**sudo apt-get install git
git clone
https://github.com/adafruit/Adafruit_Python_SSD1306.git (http://adafru.it/dEH)
cd Adafruit_Python_SSD1306
sudo python setup.py install**

```
pi@raspberrypi: ~/Adafruit_Python_SSD1306
Processing dependencies for Adafruit-SSD1306==1.6.1
Searching for Adafruit-GPIO==1.0.2
Best match: Adafruit-GPIO 1.0.2
Processing Adafruit_GPIO-1.0.2-py2.7.egg
Adafruit-GPIO 1.0.2 is already the active version in easy-install.pth

Using /usr/local/lib/python2.7/dist-packages/Adafruit_GPIO-1.0.2-py2.7.egg
Searching for spidev==3.2
Best match: spidev 3.2
Processing spidev-3.2-py2.7-linux-armv7l.egg
spidev 3.2 is already the active version in easy-install.pth

Using /usr/local/lib/python2.7/dist-packages/spidev-3.2-py2.7-linux-armv7l.egg
Searching for Adafruit-PureIO==0.2.1
Best match: Adafruit-PureIO 0.2.1
Processing Adafruit_PureIO-0.2.1-py2.7.egg
Adafruit-PureIO 0.2.1 is already the active version in easy-install.pth

Using /usr/local/lib/python2.7/dist-packages/Adafruit_PureIO-0.2.1-py2.7.egg
Finished processing dependencies for Adafruit-SSD1306==1.6.1
pi@raspberrypi:~/Adafruit_Python_SSD1306 $
```

Step 2. Enable i2c
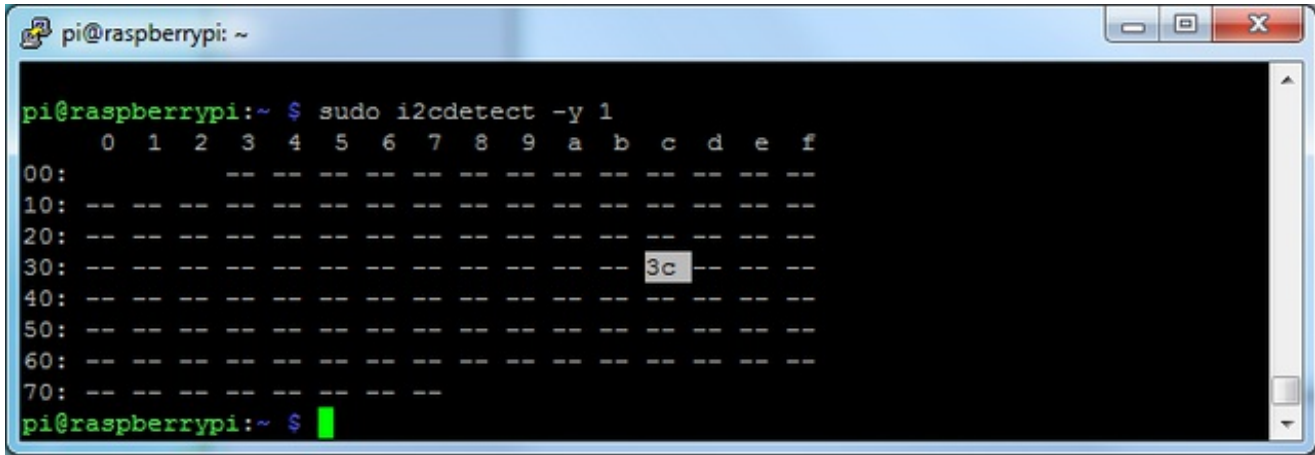
To enable i2c, you can follow our detailed guide on configuring the Pi with I2C support here. (http://adafru.it/dEO)

After you've enabled I2C you will need to shutdown with**sudo shutdown -h now**

Once the Pi has halted, plug in the PiOLED. Now you can power the Pi back up, and log back in. Run the following command from a terminal prompt to scan/detect the I2C devices

sudo i2cdetect -y 1

You should see the following, indicating that address**0x3c** (the OLED display) was found

```
pi@raspberrypi: ~

pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```
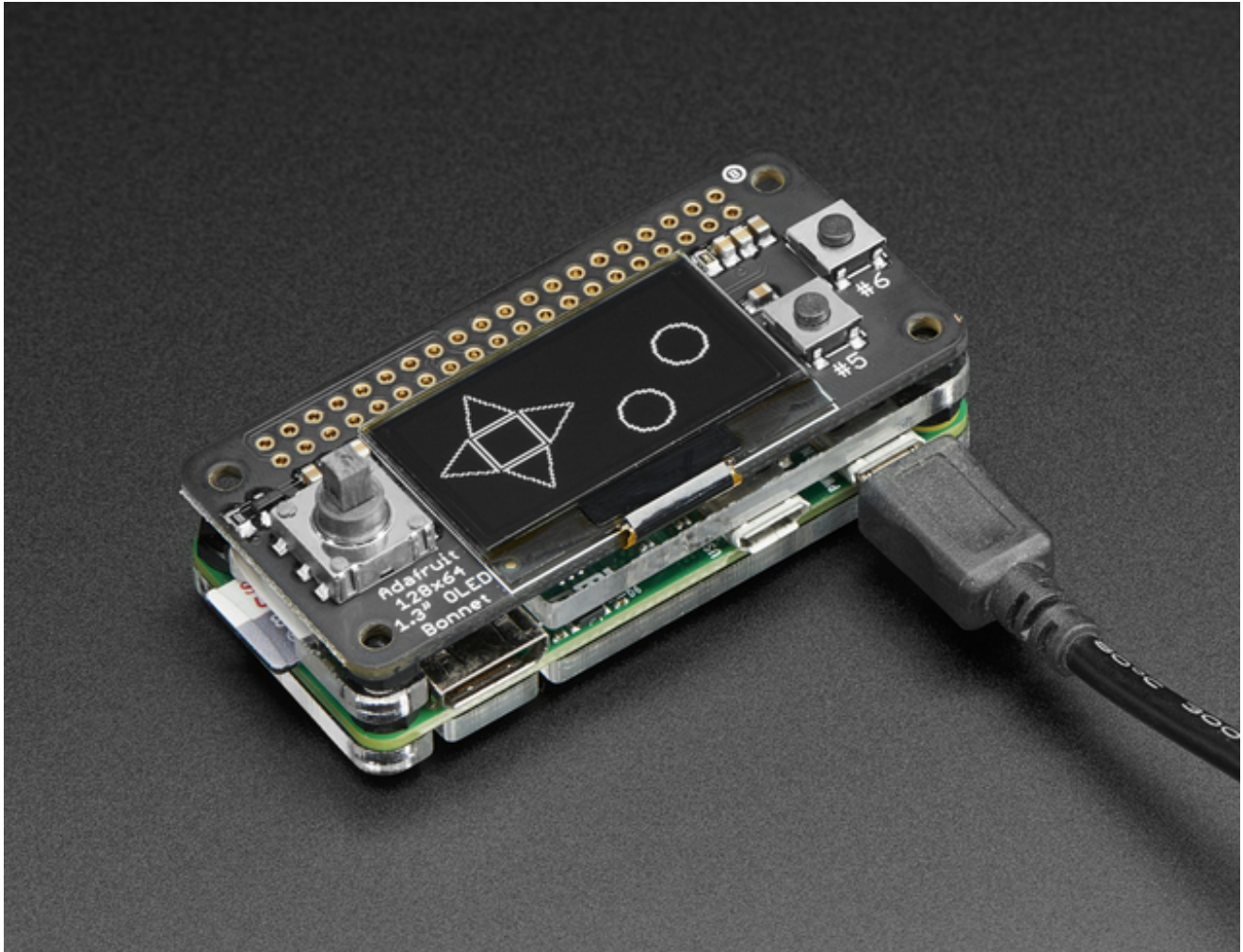
Step 3. Verify I2C Device

While in the **Adafruit_Python_SSD1306** folder, you can run our buttons example, which will let you press various buttons and see them mimicked on the OLED

Run sudo python examples/buttons.py to run the demo, you should see something like the below:

Press buttons to interact with the demo. Press the joystick + buttons at once for an Easter egg!

Running Scripts on Boot

You can pretty easily make it so this program (or whatever program you end up writing) run every time you boot your Pi.

The fastest/easiest way is to put it in**/etc/rc.local**

Run **sudo nano /etc/rc.local** and add the line

sudo python /home/pi/Adafruit_Python_SSD1306/examples/buttons.py &

on its own line right before**exit 0**

Then save and exit. Reboot to verify that the screen comes up on boot!

Library Usage

Inside the examples subdirectory you'll find python scripts which demonstrate the usage of the library. These are covered in more detail in our OLED guide here, so do check them out. (http://adafru.it/wF9)

To help you get started, I'll walk through the**buttons.py** code below, that way you can use this file as the basis of a future project.

Python library setup

import RPi.GPIO as GPIO

import time

import Adafruit_GPIO.SPI as SPI
import Adafruit_SSD1306

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

First a few modules are imported, including the**Adafruit_SSD1306** module which contains the OLED display driver classes. You can also see some of the **P**ython **I**maging **L**ibrary modules like **Image**, **ImageDraw**, and **ImageFont** being imported. Those are, as you can imagine, are for drawing images, shapes and text/fonts!

Pin Setup

```
# Input pins:
L_pin = 27
R_pin = 23
C_pin = 4
U_pin = 17
D_pin = 22

A_pin = 5
B_pin = 6

GPIO.setmode(GPIO.BCM)
GPIO.setup(A_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Input with pull-up
GPIO.setup(B_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Input with pull-up
GPIO.setup(L_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Input with pull-up
GPIO.setup(R_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Input with pull-up
```

```
GPIO.setup(U_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Input with pull-up
GPIO.setup(D_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Input with pull-up
GPIO.setup(C_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Input with pull-up
```

Next up we define the pins that are used for the joystick and buttons. The Joystick has Left, Right, Center (press in), Up and Down. There's also the A and B buttons on the right. Each one should be set as an input with pull-up resistor.

Display Setup

```
# Raspberry Pi pin configuration:
RST = None

# 128x64 display with hardware I2C:
disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST)
```

Below the configuration values is the display class setup. There are 4 variants of OLED displays, with 128x32 pixels or 128x64 pixels, and with I2C or with SPI.

However since the OLED Bonnet is a 128x64 I2C display *only* you should only use the

```
# 128x64 display with hardware I2C:
disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST)
```

variant for creating the display object! The rest can remain commented out.

Note that above, we initialize RST = None because the OLED Bonnet does not require a reset pin.

Display Initialization

```
# Initialize library.
disp.begin()

# Clear display.
disp.clear()
disp.display()

# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
width = disp.width
height = disp.height
image = Image.new('1', (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
```

```
draw.rectangle((0,0,width,height), outline=0, fill=0)
```

The next bit of code will initialize the display library with begin() and clear the display with clear() and display().

Then it will configure a PIL drawing class to prepare for drawing graphics. Notice that the image buffer is created in 1-bit mode with the '1' parameter, this is important because the display only supports black and white colors.

We then re-draw a large black rectangle to clear the screen. In theory we don't have to clear the screen again, but its a good example of how to draw a shape!

Button Input & Drawing

Once the display is initialized and a drawing object is prepared, you can draw shapes, text and graphics using [PIL's drawing commands](http://adafru.it/dfH) (http://adafru.it/dfH).

```
try:
  while 1:
    if GPIO.input(U_pin): # button is released
      draw.polygon([(20, 20), (30, 2), (40, 20)], outline=255, fill=0)  #Up
    else: # button is pressed:
      draw.polygon([(20, 20), (30, 2), (40, 20)], outline=255, fill=1)  #Up filled

    if GPIO.input(L_pin): # button is released
      draw.polygon([(0, 30), (18, 21), (18, 41)], outline=255, fill=0)  #left
    else: # button is pressed:
      draw.polygon([(0, 30), (18, 21), (18, 41)], outline=255, fill=1)  #left filled

    if GPIO.input(R_pin): # button is released
      draw.polygon([(60, 30), (42, 21), (42, 41)], outline=255, fill=0) #right
    else: # button is pressed:
      draw.polygon([(60, 30), (42, 21), (42, 41)], outline=255, fill=1) #right filled

    if GPIO.input(D_pin): # button is released
      draw.polygon([(30, 60), (40, 42), (20, 42)], outline=255, fill=0) #down
    else: # button is pressed:
      draw.polygon([(30, 60), (40, 42), (20, 42)], outline=255, fill=1) #down filled

    if GPIO.input(C_pin): # button is released
      draw.rectangle((20, 22,40,40), outline=255, fill=0) #center
    else: # button is pressed:
      draw.rectangle((20, 22,40,40), outline=255, fill=1) #center filled

    if GPIO.input(A_pin): # button is released
      draw.ellipse((70,40,90,60), outline=255, fill=0) #A button
    else: # button is pressed:
      draw.ellipse((70,40,90,60), outline=255, fill=1) #A button filled
```

```
        if GPIO.input(B_pin): # button is released
            draw.ellipse((100,20,120,40), outline=255, fill=0) #B button
        else: # button is pressed:
            draw.ellipse((100,20,120,40), outline=255, fill=1) #B button filled

        # Display image.
        disp.image(image)
        disp.display()
    time.sleep(.01)
```

This is a basic polling example - we'll check each GPIO.input in order, and draw a different shape - a directional arrow or a round circle) depending on whether the button is pressed. If the button is pressed we have the shape filled in. If the button is not pressed, we draw an outline only

Then we run **disp.image(image)** and **disp.display()** to actually push the updated image to the OLED. **This is required to actually make the changes appear!**

A small time.sleep() delay just keeps the OLED from getting flickery and 'de-bounces' the button inputs.

More Demos & Examples

You can check out our other examples in the example, just make sure to edit each one with **nano animate.py** for example, and find the line that says:

```
# Raspberry Pi pin configuration:
RST = 24
```

and change it to:

```
# Raspberry Pi pin configuration:
RST = None # PiOLED does not require reset pin
```

and make sure that the configuration section where you choose which type of display, looks like this

```
# 128x32 display with hardware I2C:
#disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST)

# 128x64 display with hardware I2C:
disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST)

# 128x32 display with hardware SPI:
# disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST, dc=DC, spi=SPI.SpiDev(SPI_POR$

# 128x64 display with hardware SPI:
```

disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST, dc=DC, spi=SPI.SpiDev(SPI_POR$
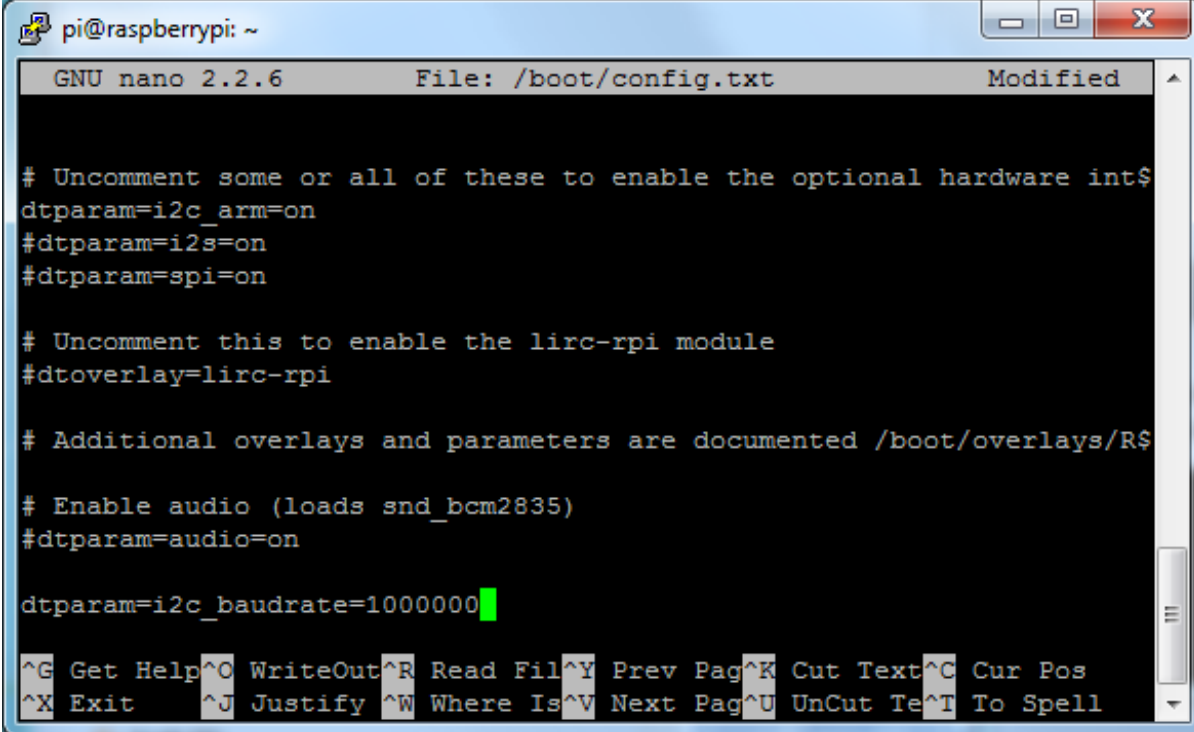
That is, we'll be using I2C 128x64 display!

Speeding Up the Display

For the best performance, especially if you are doing fast animations, you'll want to tweak the I2C core to run at 1MHz. By default it may be 100KHz or 400KHz

To do this edit the config with **sudo nano /boot/config.txt**

and add to the end of the file

dtparam=i2c_baudrate=1000000

```
  GNU nano 2.2.6          File: /boot/config.txt              Modified

# Uncomment some or all of these to enable the optional hardware int$
dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on

# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/R$

# Enable audio (loads snd_bcm2835)
#dtparam=audio=on

dtparam=i2c_baudrate=1000000

^G Get Help ^O WriteOut ^R Read Fil ^Y Prev Pag ^K Cut Text ^C Cur Pos
^X Exit      ^J Justify  ^W Where Is ^V Next Pag ^U UnCut Te ^T To Spell
```

reboot to 'set' the change.

Downloads

Files

- EagleCAD PCB files on GitHub (http://adafru.it/wWC)
- UG-2864HSWEG01 (http://adafru.it/aJI) Datasheet
- UG-2864HSWEG01 (http://adafru.it/wWD) User Guide
- SSD1306 (http://adafru.it/aJK) Datasheet
- Fritzing objects available in the Adafruit Fritzing Library(http://adafru.it/aP3)

Schematic & Fabrication Print

Dimensions in mm

Adafruit
128x64
1.3" OLED
Bonnet