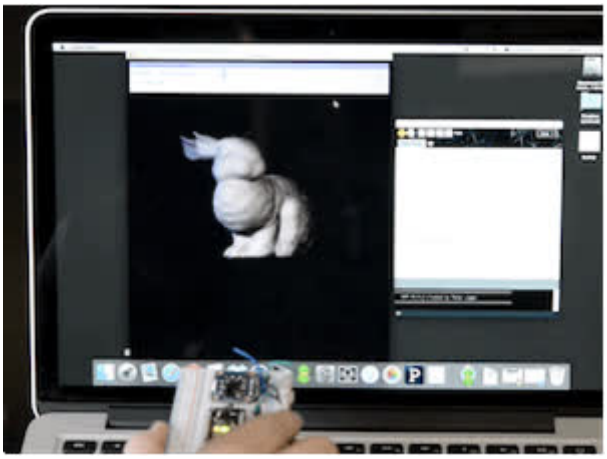
Adafruit BNO055 Absolute Orientation Sensor

Created by Kevin Townsend



https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor

Last updated on 2026-08-19 04:18:17 PM UTC

Table of Contents

Overview 5

- Data Output
- Related Resources

Pinouts 8

- Power Pins
- I2C Pins
- STEMMA QT version
- Other Pins

Assembly 11

- Prepare the header strip:
- Add the breakout board:
- And Solder!

Arduino Code 13

- Wiring for Arduino
- Software
- Adafruit Unified Sensor System
- 'sensorapi' Example
- Raw Sensor Data
- .getVector (adafruit_vector_type_t vector_type)
- .getQuat(void)
- .getTemp(void)
- 'rawdata' Example

WebSerial Visualizer 21

- Step 1 - Wire up the BNO055 to your Microcontroller using I2C
- Step 2 - Load the Sketch onto your device
- Step 3 - Install Chrome or Firefox
- Step 4 - Enable Web Serial API if necessary on older Chrome
- Step 5 - Visit the Adafruit 3D Model viewer
- Step 6 - Calibration
- Step 7 - Euler Angles or Quaternions

Processing Test 27

- Requirements
- Opening the Processing Sketch
- Run the Bunny Sketch on the Uno
- Rabbit Disco!

Device Calibration 31

- Interpretting Data
- Generating Calibration Data
- Persisting Calibration Data
- Bosch Video

Python & CircuitPython 33

- CircuitPython Microcontroller Wiring - I2C
- CircuitPython Microcontroller Wiring - UART
- Python Computer Wiring - I2C
- Python Computer Wiring - UART

- CircuitPython Installation of BNO055 Library
- Python Installation of BNO055 Library
- CircuitPython & Python Usage
- Usage
- Full Example Code

Python Docs 41

WebGL Example 42

- Dependencies
- Download the WebGL Example
- Start Server
- Sensor Calibration
- Usage
- More Info

BNO055 Sensor Calibration, Target Angle Offset, and Tap Detection in CircuitPython 46

- Overview
- BNo055 Sensor Calibration
- User Orientation Offset (Target Angle Offset)
- Tap Detection
- Additional Information

FAQs 53

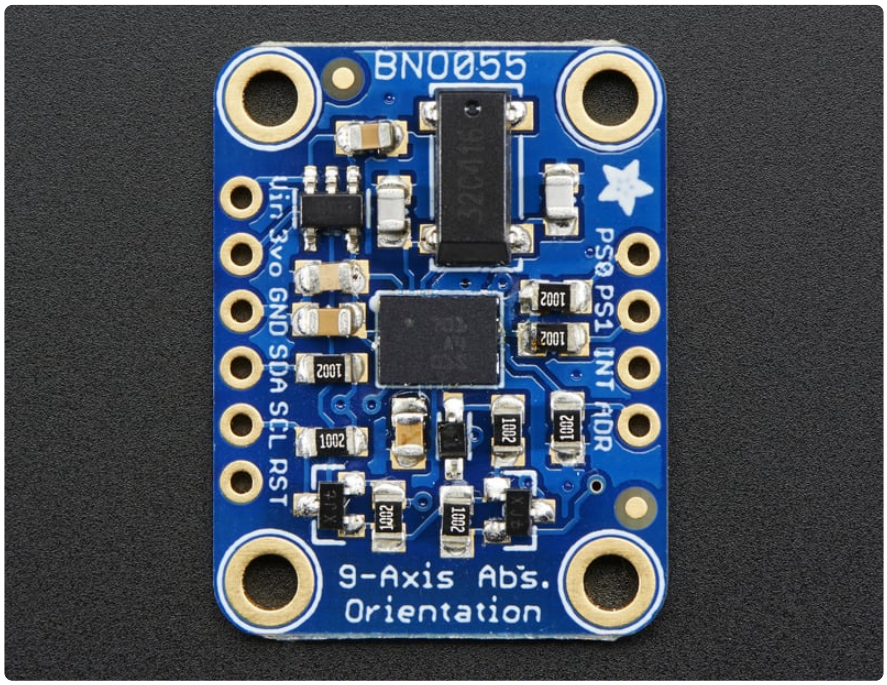
Downloads 58

- Files
- Pre-Compiled Bunny Rotate Binaries
- Schematic and Fab Print for Rev D (STEMMA QT)
- Schematic and Fab Print for Rev C (no STEMMA QT)

Overview

Text emphasized with a blue info:

There are two versions of this breakout: one with STEMMA QT ports and one without

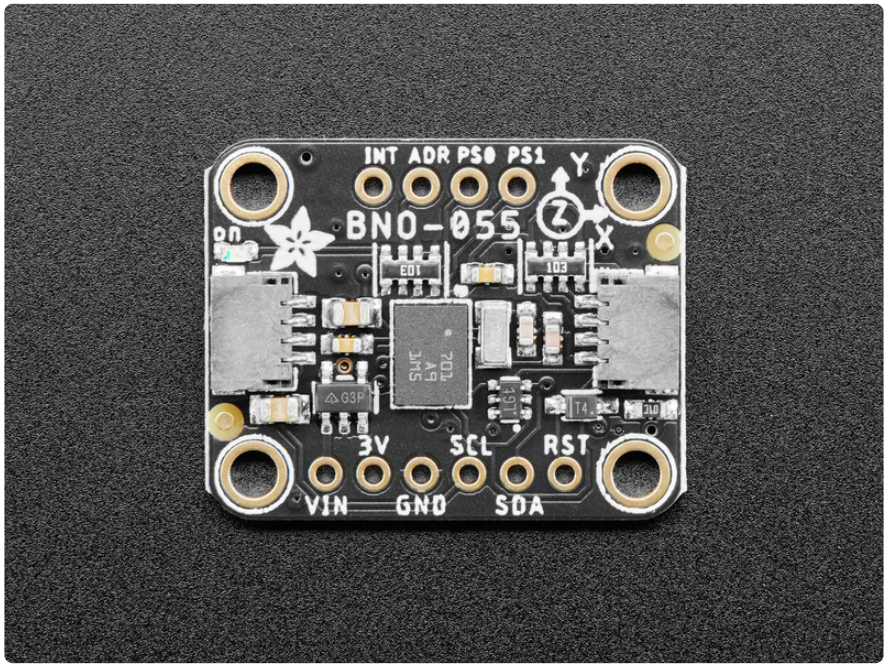


If you've ever ordered and wire up a 9-DOF sensor, chances are you've also realized the challenge of turning the sensor data from an accelerometer, gyroscope and magnetometer into actual "3D space orientation"! Orientation is a hard problem to solve. The sensor fusion algorithms (the secret sauce that blends accelerometer, magnetometer and gyroscope data into stable three-axis orientation output) can be mind-numbingly difficult to get right and implement on low cost real time systems.

Bosch is the first company to get this right by taking a MEMS accelerometer, magnetometer and gyroscope and putting them on a single die with a high speed ARM Cortex-M0 based processor to digest all the sensor data, abstract the sensor fusion and real time requirements away, and spit out data you can use in quaternions, Euler angles or vectors.

Text emphasized with a yellow exclamation:

The BNO055 I2C implementation violates the I2C protocol in some circumstances. This causes it not to work well with certain chip families. It does not work well NXP i.MX RT1011, and it does not work well with I2C multiplexers. With ESP32 and ESP32-S3, CircuitPython 9.2.2 and later (which use ESP-IDF 5.3.2 and later) work better. Earlier versions of CircuitPython and ESP-IDF do not work well. Operation with SAMD51, RP2040, STM32F4, and nRF52840 is more reliable.



The new version of the board includes SparkFun qwiic (https://adafru.it/Fpw) compatible STEMMA QT **(https://adafru.it/Ft4)** connectors for the I2C bus so **you don't even need to solder!** Use a a plug-and-play STEMMA QT cable to get 9 DoF data ASAP.

Rather than spending weeks or months fiddling with algorithms of varying accuracy and complexity, you can have meaningful sensor data in minutes thanks to the BNO055 - a smart 9-DOF sensor that does the sensor fusion all on its own!

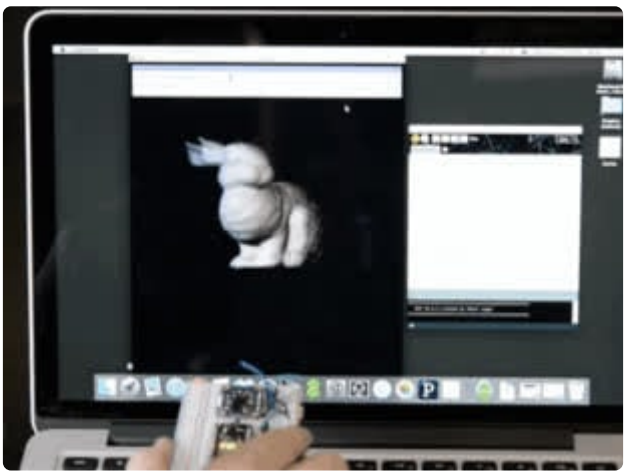
Data Output

The BNO055 can output the following sensor data:

- **Absolute Orientation** (Euler Vector, 100Hz)
  Three axis orientation data based on a 360° sphere
- **Absolute Orientation** (Quaterion, 100Hz)
  Four point quaternion output for more accurate data manipulation
- **Angular Velocity Vector** (100Hz)
  Three axis of 'rotation speed' in rad/s
- **Acceleration Vector** (100Hz)
  Three axis of acceleration (gravity + linear motion) in m/s^2
- **Magnetic Field Strength Vector** (20Hz)
  Three axis of magnetic field sensing in micro Tesla (uT)
- **Linear Acceleration Vector** (100Hz)
  Three axis of linear acceleration data (acceleration minus gravity) in m/s^2
- **Gravity Vector** (100Hz)
  Three axis of gravitational acceleration (minus any movement) in m/s^2
- **Temperature** (1Hz)
  Ambient temperature in degrees celsius

Related Resources

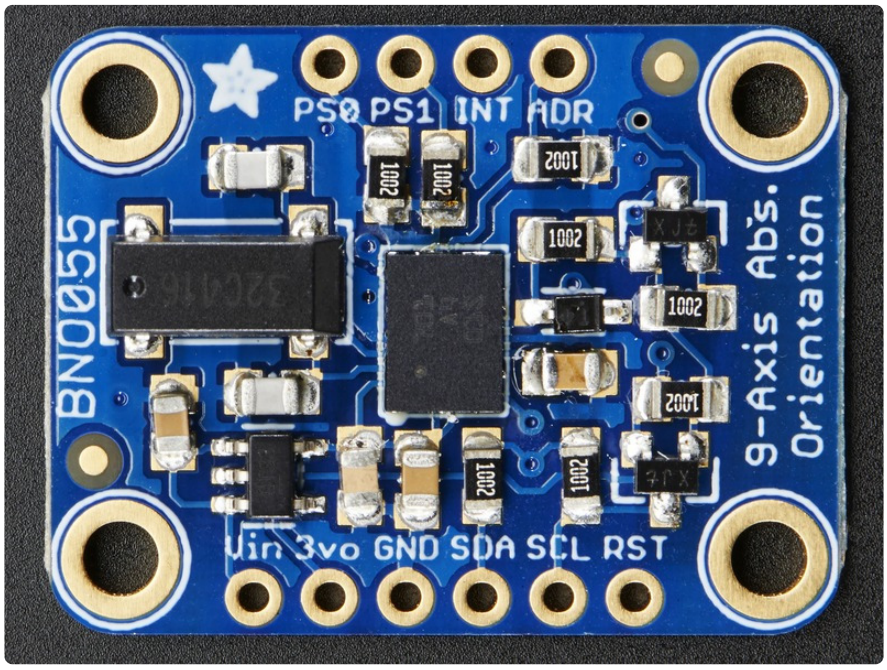
- Datasheet (https://adafru.it/f0H)
- Adafruit BNO055 Library (https://adafru.it/f0I) (GitHub)
- Comparing the BNO085 vs BNO055 (https://adafru.it/19Cb) (Adafruit Forums)

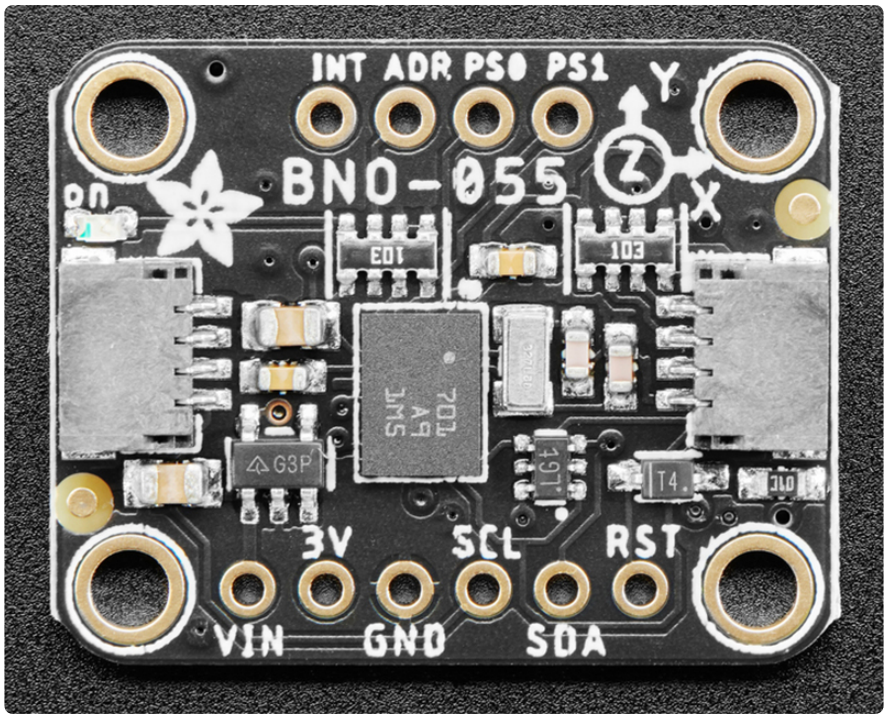


Pinouts

Text emphasized with a blue info:

There are two versions of this breakout: one with STEMMA QT ports and one without. The pin order on the STEMMA QT version of the board is not the same as the original version. The pins are the same otherwise.





Note: The **pin order** on the STEMMA QT version of the board is not the same as the original version. The pins are the same otherwise.

Text emphasized with a yellow exclamation:

The BNO055 I2C implementation violates the I2C protocol in some circumstances. This causes it not to work well with certain chip families. It does not work well with Espressif ESP32, ESP32-S3, and NXP i.MX RT1011, and it does not work well with I2C multiplexers. Operation with SAMD51, RP2040, STM32F4, and nRF52840 is more reliable.

Power Pins

- **VIN**: 3.3-5.0V power supply input
- **3VO**: 3.3V output from the on-board linear voltage regulator, you can grab up to about 50mA as necessary
- **GND**: The common/GND pin for power and logic

I2C Pins

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.

- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.

STEMMA QT version

- STEMMA QT **(https://adafru.it/Ft4)** - These connectors allow you to connectors to dev boards with **STEMMA QT** connectors or to other things with various associated accessories (https://adafru.it/Ft6)

Other Pins

- **RST**: Hardware reset pin. Set this pin low then high to cause a reset on the sensor. This pin is 5V safe.
- **INT**: The HW interrupt output pin, which can be configured to generate an interrupt signal when certain events occur like movement detected by the accelerometer, etc. (not currently supported in the Adafruit library, but the chip and HW is capable of generating this signal). The voltage level out is 3V
- **ADR**: Set this pin high to change the default I2C address for the BNO055 if you need to connect two ICs on the same I2C bus.

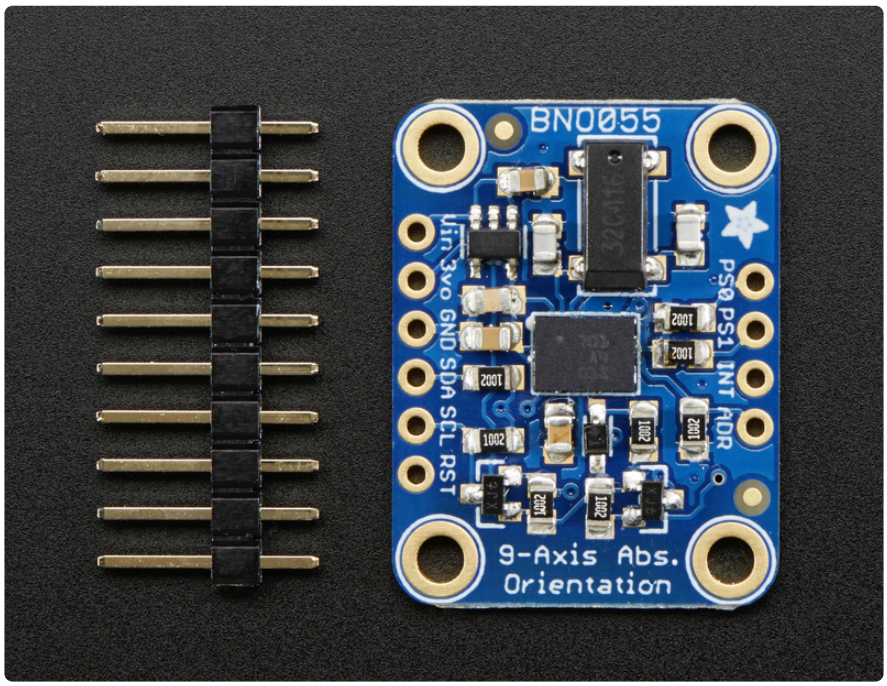
The default I²C address of the BNO055 device is 0101001b (0x29). The alternative address 0101000b (0x28), in I2C mode the input pin COM3 can be used to select between the primary and alternative I2C address as shown in Table 4-7.

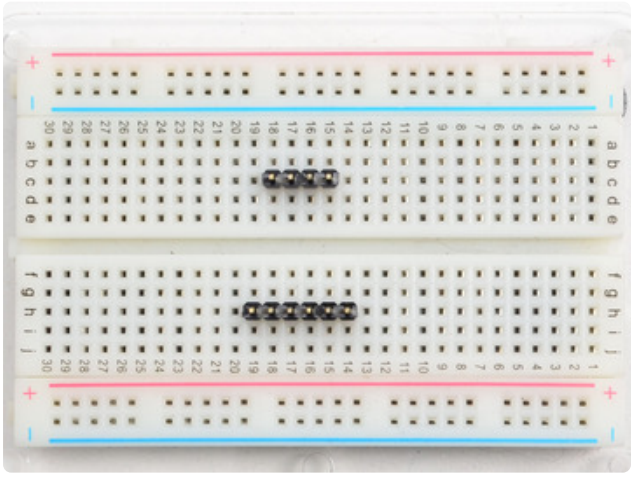
Table 4-7: I2C address selection

I2C configuration	COM3_state	I2C address
Slave	HIGH	0x29
Slave	LOW	0x28

- **PS0** and **PS1**: These pins can be used to change the mode of the device (it can also do HID-I2C and UART) and also are provided in case Bosch provides a firmware update at some point for the ARM Cortex M0 MCU inside the sensor. They should normally be left unconnected.

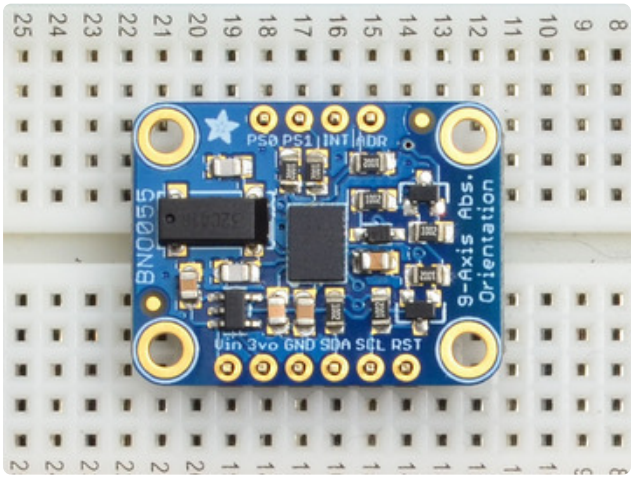
Assembly





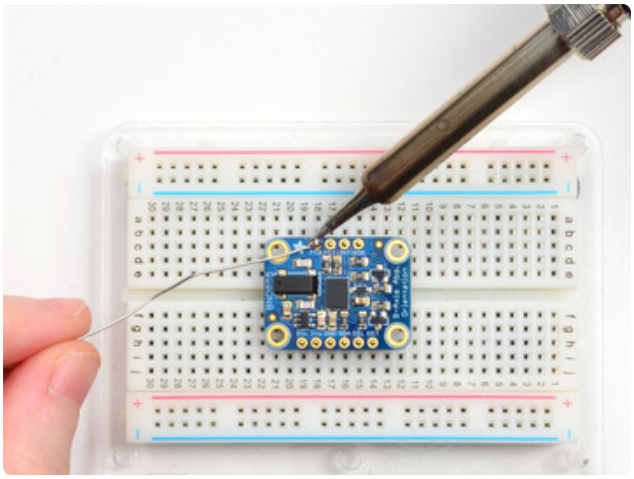
Prepare the header strip:

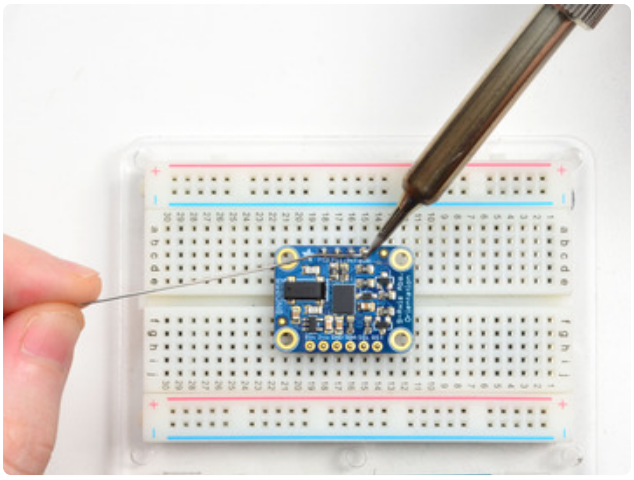
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

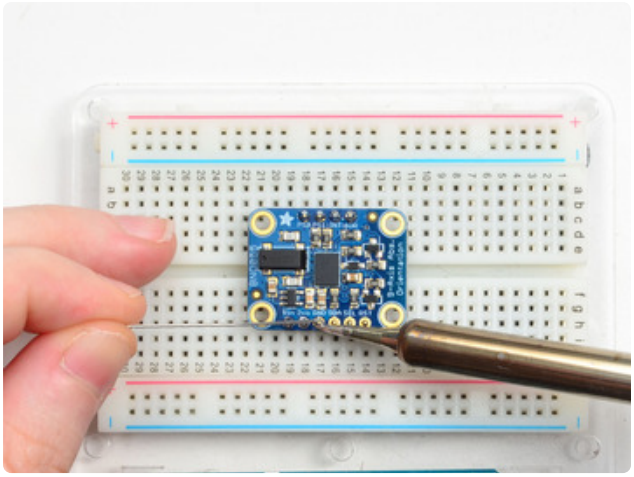


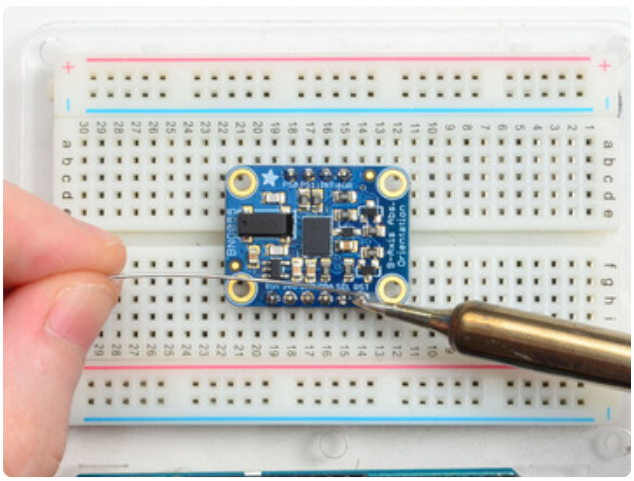
Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads







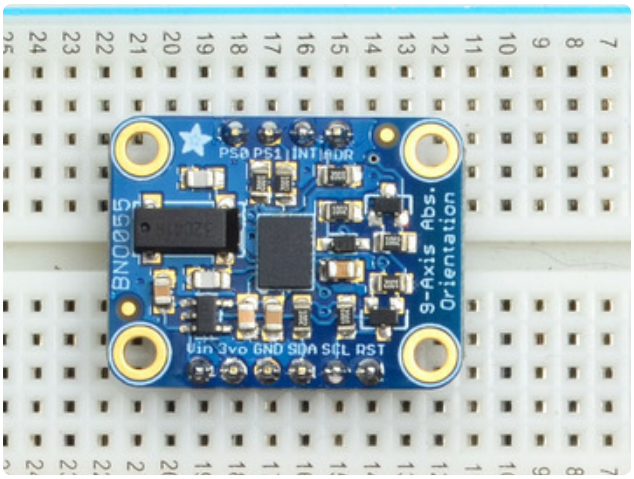


And Solder!

Be sure to solder all pins for reliable electrical contact.

Solder the longer power/data strip first

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk)).



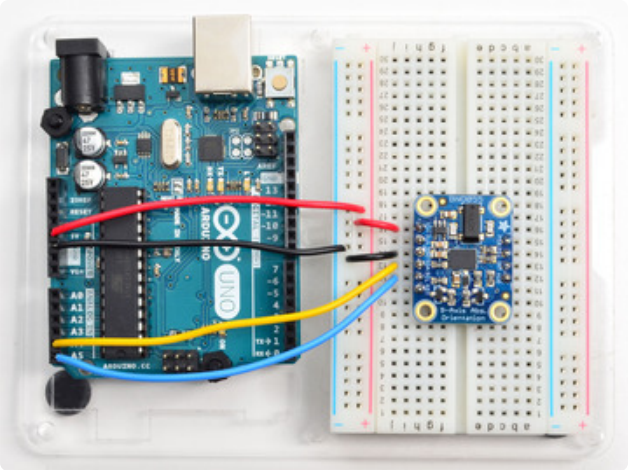
You're done! Check your solder joints visually and continue onto the next steps

---

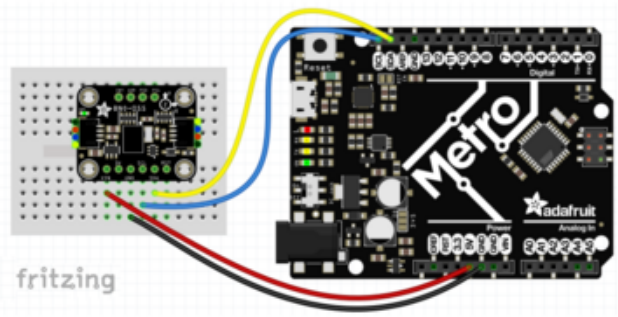
Arduino Code

Wiring for Arduino

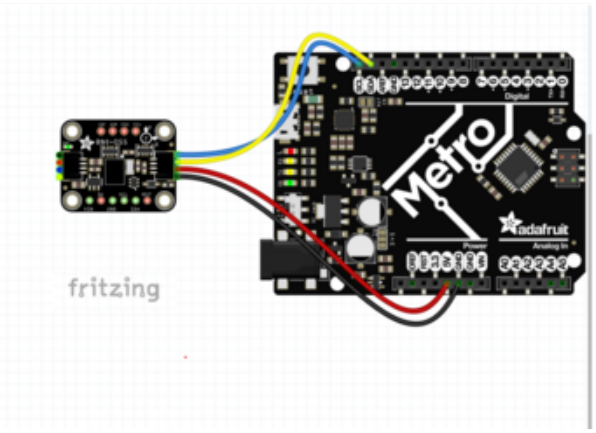
You can easily wire this breakout to any microcontroller, we'll be using an Arduino. For another kind of microcontroller, just make sure it has I2C capability, then port the code - its pretty simple stuff!



To connect the assembled BNO055 breakout to an Arduino Uno, follow the wiring diagram.



Connect **Vin (red wire, positive)** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
Connect **GND (black wire, negative)** to common power/data ground
Connect the **SCL (blue wire)** pin to the I2C clock **SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**



Connect the **SDA (yellow wire)** pin to the I2C data **SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**

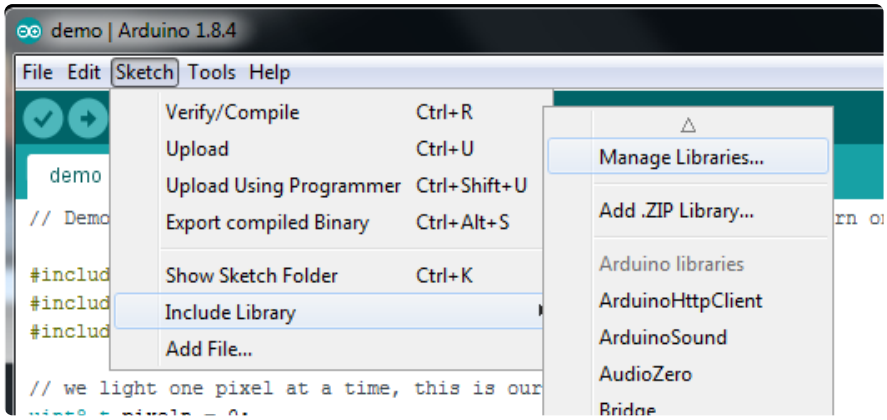
Text emphasized with a yellow exclamation:

If you're using a Genuino Zero or Arduino Zero with the built in EDBG interface you may need to use I2C address 0x29 since 0x28 is 'taken' by the DBG chip

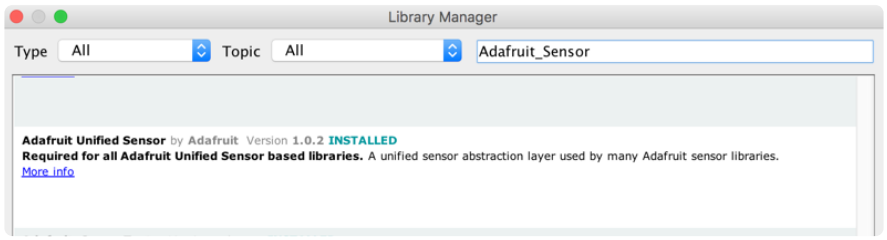
Software

The Adafruit_BNO055 driver (https://adafru.it/f0l) supports reading raw sensor data, or you can use the Adafruit Unified Sensor (https://adafru.it/dGB) system to retrieve orientation data in a standard data format.

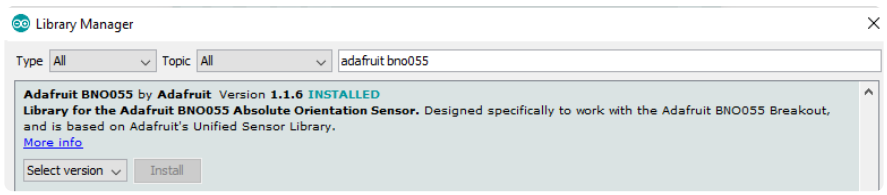
Open up the Arduino library manager:



Search for the **Adafruit Sensor** library and install it



Search for the **Adafruit BNO055** library and install it



We also have a great tutorial on Arduino library installation at: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

Adafruit Unified Sensor System

Since the Adafruit_BNO055 driver is based on the Adafruit Unified Sensor system, you can retrieve your three axis orientation data (in Euler angles) using the standard types and functions described in the Adafruit Sensor learning guide (https://adafru.it/kA0) (.getEvent (https://adafru.it/kA0), .getSensor (https://adafru.it/kA0), etc.).

This is probably the easiest option if all you care about is absolute orientation data across three axis.

For example, the following code snippet shows the core of what is needed to start reading data using the Unified Sensor System:

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

Adafruit_BNO055 bno = Adafruit_BNO055(55);

void setup(void)
{
  Serial.begin(9600);
  Serial.println("Orientation Sensor Test"); Serial.println("");

  /* Initialise the sensor */
  if(!bno.begin())
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
    while(1);
  }

  delay(1000);

  bno.setExtCrystalUse(true);
}

void loop(void)
{
  /* Get a new sensor event */
  sensors_event_t event;
  bno.getEvent(&event);

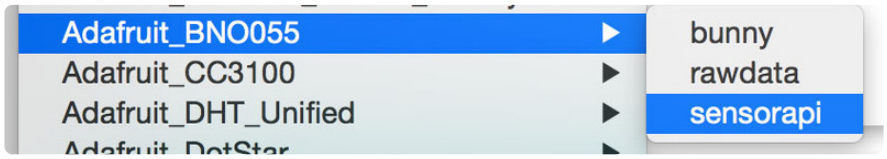
  /* Display the floating point data */
  Serial.print("X: ");
  Serial.print(event.orientation.x, 4);
  Serial.print("\tY: ");
  Serial.print(event.orientation.y, 4);
  Serial.print("\tZ: ");
  Serial.print(event.orientation.z, 4);
  Serial.println("");

```

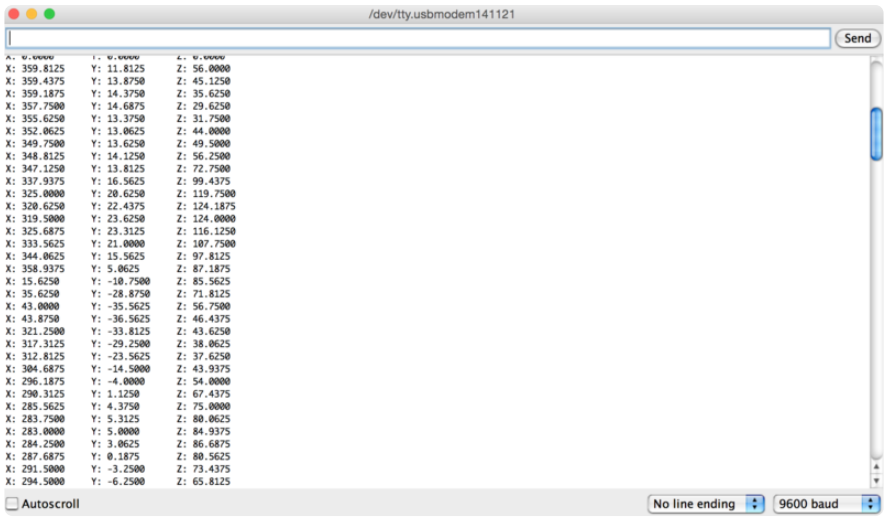
```
41    delay(100);
42  }
```

'sensorapi' Example

To test the Unified Sensor System output, open the **sensorapi** demo in the Adafruit_BNO055 examples folder:



This should produce the following output on the Serial Monitor:



Raw Sensor Data

If you don't want to use the Adafruit Unified Sensor system (for example if you want to access the raw accelerometer, magnetometer or gyroscope data directly before the sensor fusion algorithms process it), you can use the raw helper functions in the driver.

The key raw data functions are:

- **getVector** (adafruit_vector_type_t vector_type)
- **getQuat** (void)
- **getTemp** (void)

.getVector (adafruit_vector_type_t vector_type)

The .getVector function accepts a single parameter (vector_type), which indicates what type of 3-axis vector data to return.

The vector_type field can be one of the following values:

- **VECTOR_MAGNETOMETER** (values in uT, micro Teslas)
- **VECTOR_GYROSCOPE** (values in rps, radians per second)
- **VECTOR_EULER** (values in Euler angles or 'degrees', from 0..359)
- **VECTOR_ACCELEROMETER** (values in m/s^2)
- **VECTOR_LINEARACCEL** (values in m/s^2)
- **VECTOR_GRAVITY** (values in m/s^2)

For example, to get the Euler angles vector, we could run the following code:

```
imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);

/* Display the floating point data */
Serial.print("X: ");
Serial.print(euler.x());
Serial.print(" Y: ");
Serial.print(euler.y());
Serial.print(" Z: ");
Serial.print(euler.z());
Serial.println("");
```

.getQuat(void)

The .getQuat function returns a Quaternion, which is often easier and more accurate to work with than Euler angles when doing sensor fusion or data manipulation with raw sensor data.

You can get a quaternion data sample via the following code:

```
imu::Quaternion quat = bno.getQuat();

/* Display the quat data */
Serial.print("qW: ");
Serial.print(quat.w(), 4);
Serial.print(" qX: ");
Serial.print(quat.x(), 4);
Serial.print(" qY: ");
Serial.print(quat.y(), 4);
Serial.print(" qZ: ");
Serial.print(quat.z(), 4);
Serial.println("");
```

.getTemp(void)

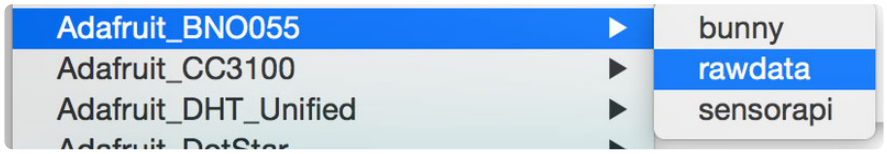
The .getTemp helper returns the current ambient temperature in degrees celsius, and can be read via the following function call:

```
/* Display the current temperature */
int8_t temp = bno.getTemp();

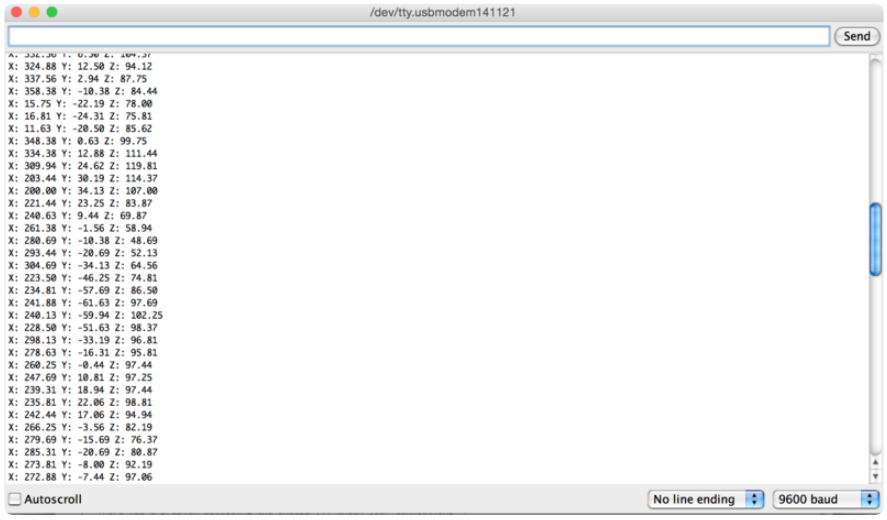
Serial.print("Current Temperature: ");
Serial.print(temp);
Serial.println(" C");
Serial.println("");
```

'rawdata' Example

To test the raw data ouput, open the **rawdata** demo in the Adafruit_BNO055 examples folder:



This should produce the following output on the Serial Monitor:

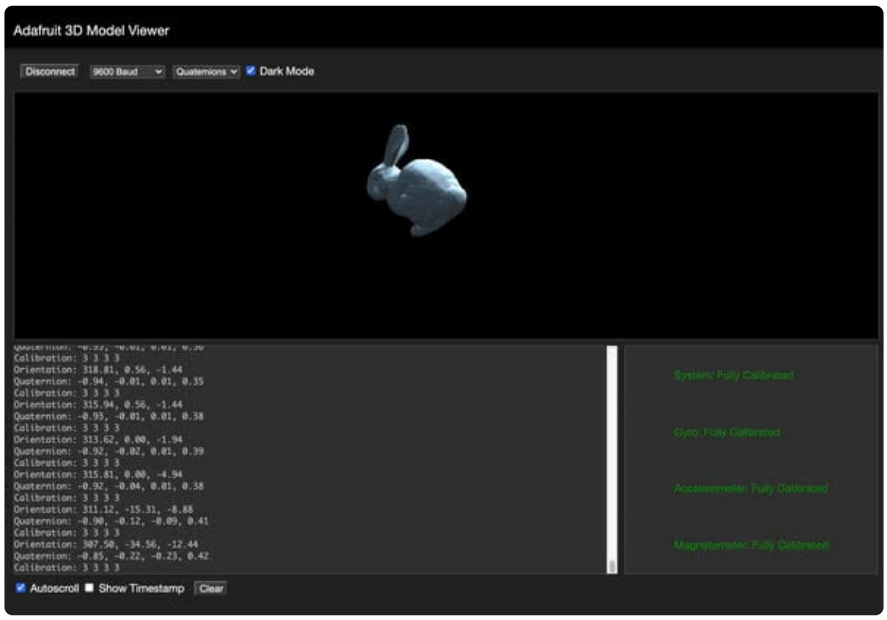


By default, the sketch generates **Euler angle** absolute orientation data, but you can easily modify the data displayed by changing the value provided to **.getVector** below:

```
// Possible vector values can be:
// - VECTOR_ACCELEROMETER - m/s^2
```

```
3   // - VECTOR_MAGNETOMETER  - uT
4   // - VECTOR_GYROSCOPE     - rad/s
5   // - VECTOR_EULER         - degrees
6   // - VECTOR_LINEARACCEL   - m/s^2
7   // - VECTOR_GRAVITY       - m/s^2
8   imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
9
10  /* Display the floating point data */
11  Serial.print("X: ");
12  Serial.print(euler.x());
13  Serial.print(" Y: ");
14  Serial.print(euler.y());
15  Serial.print(" Z: ");
16  Serial.print(euler.z());
17  Serial.println("");
```

WebSerial Visualizer

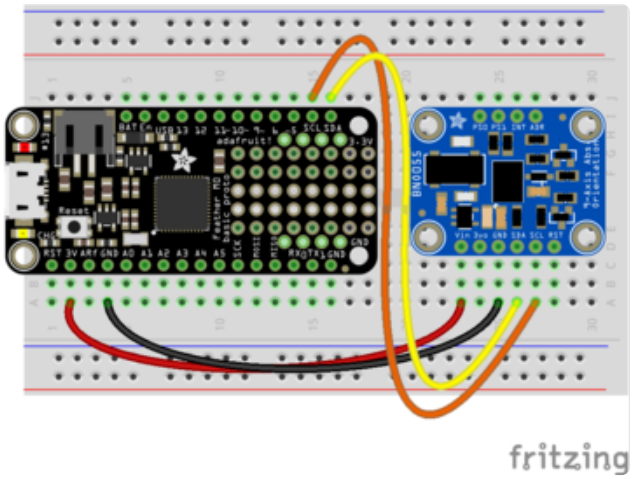


That raw data is all fine and good, but we want to see what they mean in 3D space, right? Traditionally, a Processing sketch would be used to read the serial data and convert it to a 3D rotation - but [thanks to Web Serial API we can use any Chrome browser - a lot easier than installing Processing!](https://adafru.it/JaV) (https://adafru.it/JaV)

Text emphasized with a blue exclamation:	Web Serial support is also available in Firefox as of release 151.

Step 1 - Wire up the BNO055 to your Microcontroller using I2C

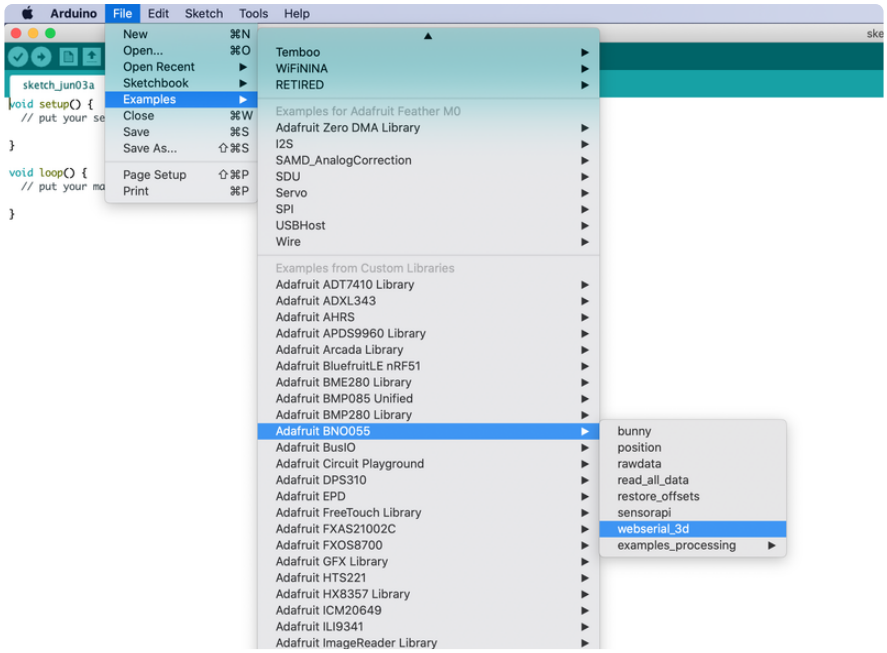
First wire up a BNO055 to your board exactly as shown on the previous pages using the I2C interface. Here's an example of wiring a Feather M0 to the sensor with I2C:



Board 3V to **sensor VIN**
Board GND to **sensor GND**
Board SCL to **sensor SCL**
Board SDA to **sensor SDA**

Step 2 - Load the Sketch onto your device

Continue by making sure you still have the Arduino IDE open and have the latest version of the Adafruit BNO055 library installed. Open the sketch at **Examples → Adafruit BNO055 → webserial_3d**



Upload the sketch to your Microcontroller Board.

Step 3 - Install Chrome or Firefox

Web Serial support requires at least Firefox 151 or a Chrome-89-based browser.

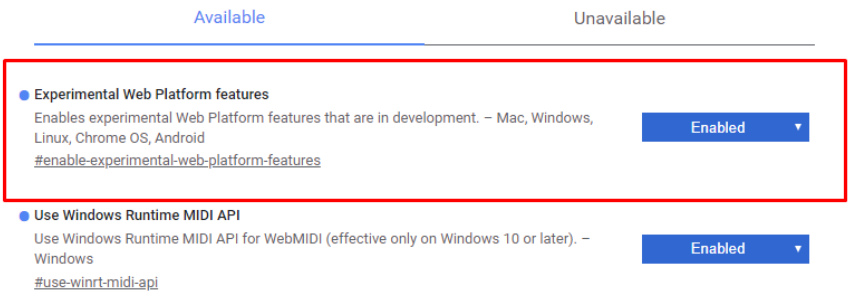
Step 4 - Enable Web Serial API if necessary on older Chrome

Text emphasized with a blue exclamation:

As of Chrome 89, Web Serial is enabled by default.

At the time of this tutorial, you'll need to enable the Serial API, which is really easy.

Visit **chrome://flags** from within Chrome. Find and enable the **Experimental Web Platform features**



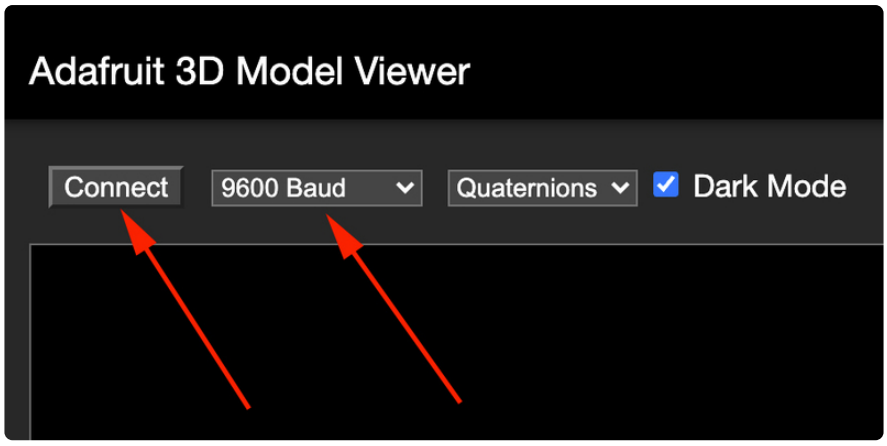
Restart Chrome

Step 5 - Visit the Adafruit 3D Model viewer

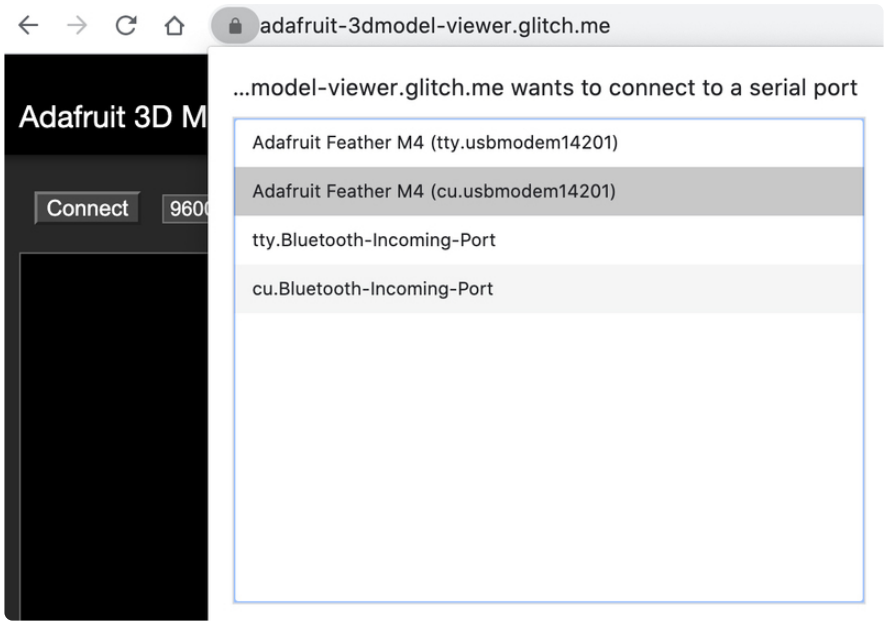
In Chrome, visit https://adafruit.github.io/Adafruit_WebSerial_3DModelViewer/ (https://adafru.it/PPE)

Verify you have **9600 Baud** selected (it only really matters for non-native-serial devices but might as well make sure its right). If you changed it in the sketch, be sure it matches.

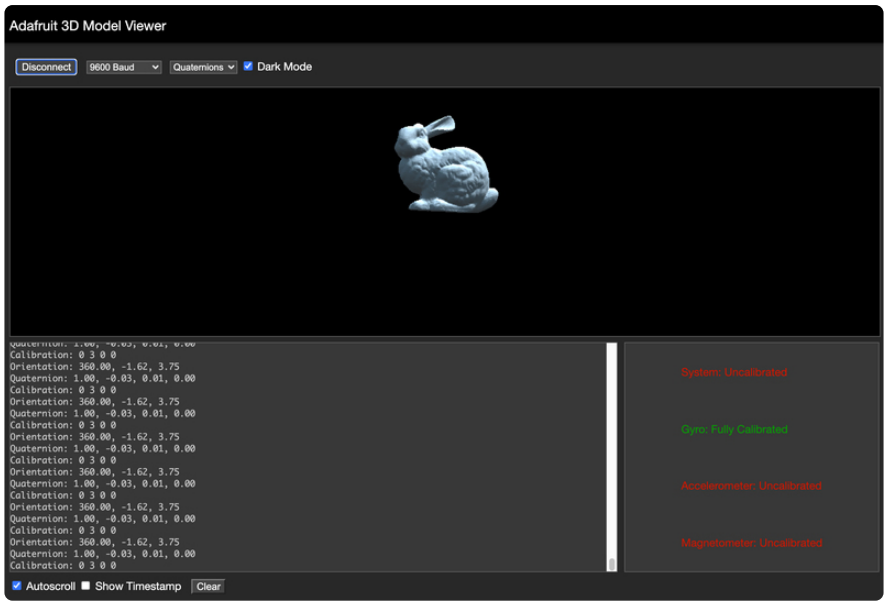
Click **Connect**



When the security window pops up, pick the matching Serial/COM port for your board running the AHRS sketches. Make sure the serial port isn't open in Arduino or something else.

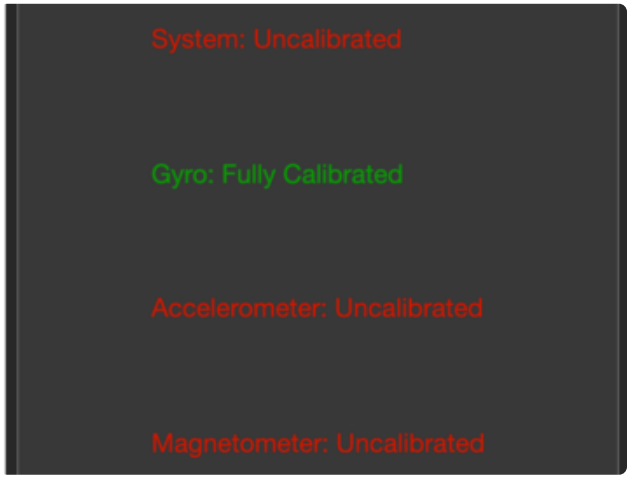


You'll see the serial port monitor on the bottom and a 3D bunny on the top. Try rotating and twisting the sensor to see it move!

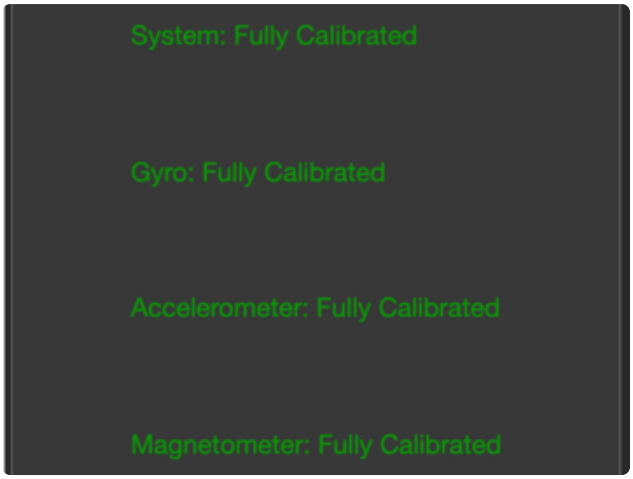


Step 6 - Calibration

The devices will need to be calibrated each time it is powered up. You can see the Device Calibration page for more details on performing the actual calibration, but the WebSerial interface provides a convenient way to check the current calibration status.



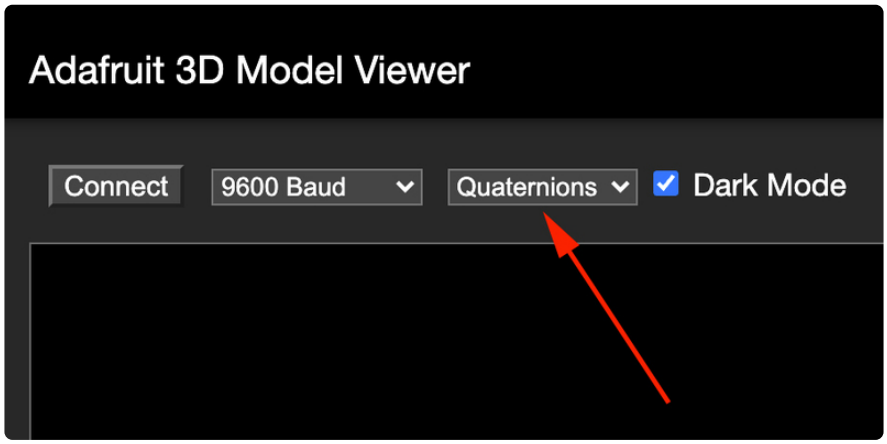
When you first connect, you'll see that most of the calibration registers show as Uncalibrated.



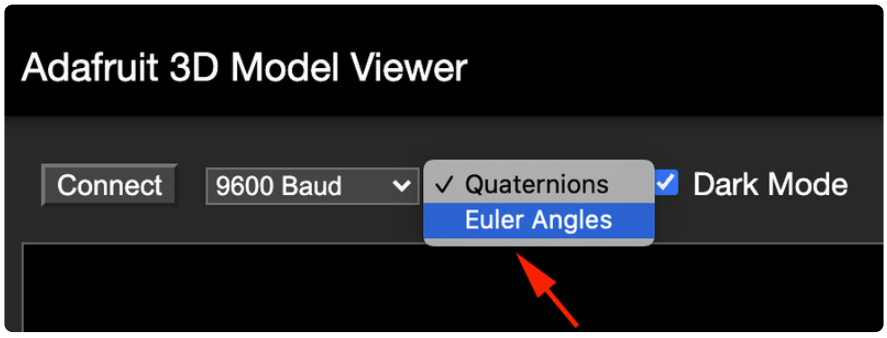
Once you have gone through the calibration steps, you will see that they are all fully calibrated.

Step 7 - Euler Angles or Quaternions

The WebSerial interface is also able to use both Euler Angles and Quaternions. Euler angles represent the X, Y, and Z axes and are easier to understand, but also have the disadvantage of "Gimbal Lock" at certain angles. To get around that, quaternions can be used. The angle type selection is at the top.



You can choose between using Euler Angles and Quaternions.



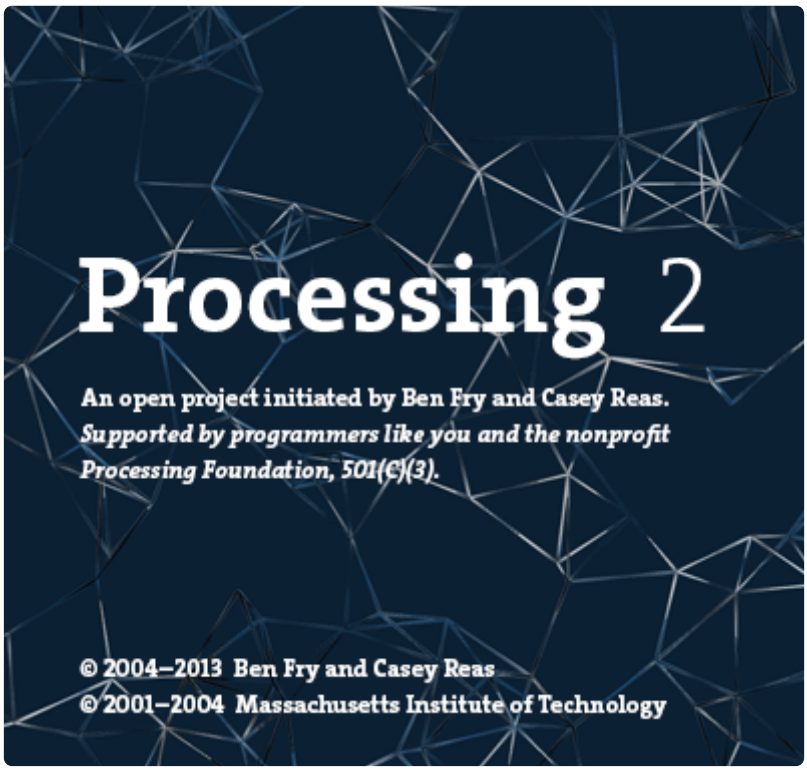
Try playing around with both by moving the bunny around and see if you can see the differences!

Processing Test

Text emphasized with a yellow exclamation:	We DO NOT RECOMMEND using Processing for visualization, as its not easy. Check the previous page for how to use a Chrome browser

Processing is a language similar to Arduino but aimed at graphics on computers. Programs, like Arduino, are also called sketches. More at processing.org (https://adafru.it/ddm).

To help you visualize the data, we've put together a basic Processing sketch that loads a 3D model (in the .obj file format) and renders it using the data generated by the BNO055 sketch on the Uno. The "bunny" sketch on the uno published data over UART, which the Processing sketch reads in, rotating the 3D model based on the incoming orientation data.



Requirements

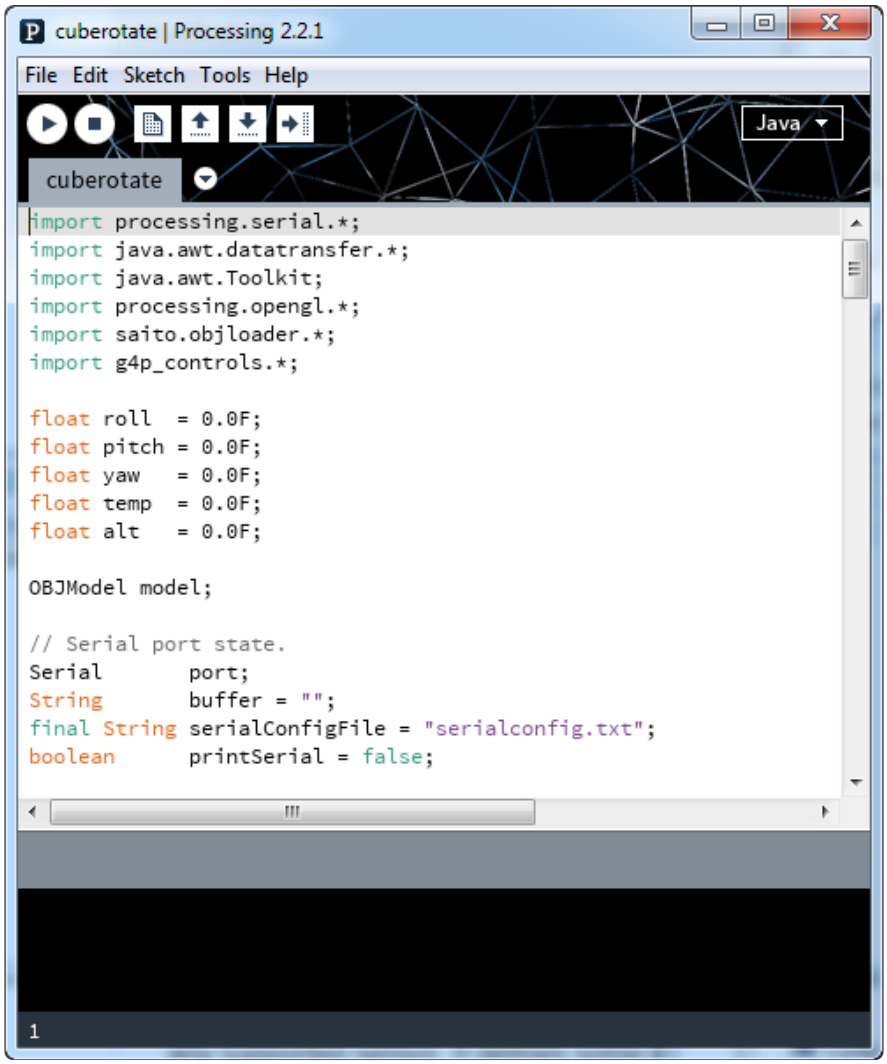
- Processing 2.x (https://adafru.it/19Cc)
    - Note that you can try later Processing versions like 3.0+ too. On some platforms Processing 2.2.1 has issues with supporting 3D acceleration (you might see 'NoClassDefFoundError: processing/awt/PGraphicsJava2D' errors). In those cases grab the later Processing 3.0+ release and use it instead of 2.x.
- Saito's OBJ Loader (https://adafru.it/mf0) library for Processing (included as part of the Adafruit repo since Google Code is now 'End of Life').
- G4P GUI library (https://adafru.it/dMO) for Processing (download the latest version here (https://adafru.it/dMP) and copy the zip into the processing libraries folder along with the OBJ loader library above). Version 3.5.2 was used in this guide.

The OBJ library is required to load 3D models. It isn't strictly necessary and you could also render a boring cube in Processing, but why play with cubes when you have rabbits?!

Opening the Processing Sketch

The processing sketch to render the 3D model is contained in the sample folder as the ahrs sketch for the Uno.

With Processing open, navigate to you Adafruit_BNO055 library folder (ex.: '**libraries/Adafruit_BNO055**'), and open '**examples/bunny/processing/cuberotate/cuberotate.pde'.** You should see something like this in Processing:



Run the Bunny Sketch on the Uno

Make sure that the "bunny" example sketch is running on the Uno, and that the Serial Monitor **is closed**.

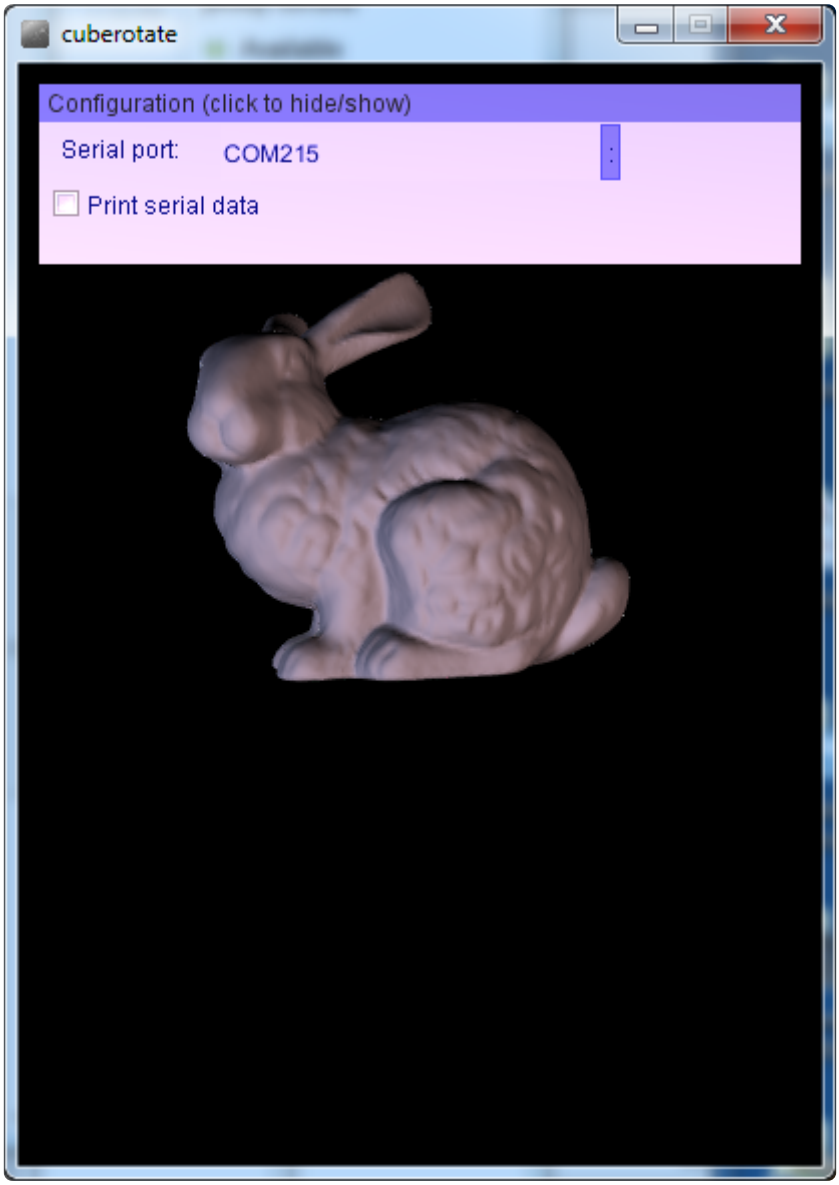
With the sample sketch running on the Uno, click the triangular 'play' icon in Processing to start the sketch.

Text emphasized with a blue exclamation:

Note: Verify your serial port number function: setSerialPort is correct for your computer, if you get an error, you likely have the wrong port selected.

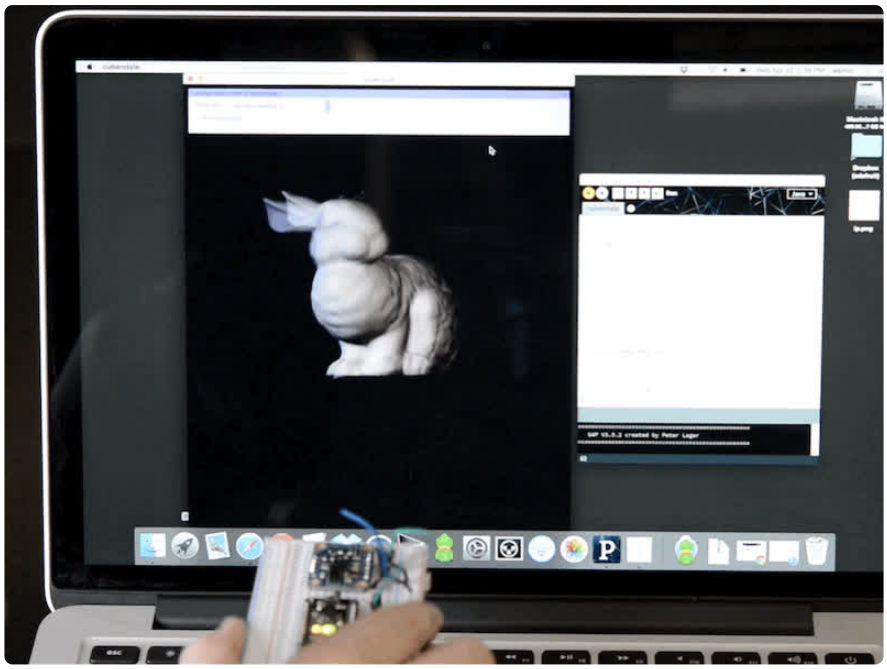
Rabbit Disco!

You should see a rabbit similar to the following image:



Before the rabbit will rotate you will need to click the **:** to the right of the serial port name. This will open a list of available serial ports, and you will need to click the appropriate serial port that your Arduino uses (check the Arduino IDE to see the port name if you're unsure). The chosen serial port should be remembered if you later run the sketch again.

As you rotate your breakout board, the rabbit should rotate to reflect the movement of the breakout in 3D-space, as seen in the video below



Also notice in the upper right corner of the dialog box at the top that the calibration of each sensor is displayed. It's important to calibrate the BNO055 sensor so that the most accurate readings are retrieved. Each sensor on the board has a separate calibration status from 0 (uncalibrated) up to 3 (fully calibrated). Check out the video and information from this guide for how to best calibrate the BNO055 sensor (https://adafru.it/kA1).

Device Calibration

The BNO055 includes internal algorithms to constantly calibrate the gyroscope, accelerometer and magnetometer inside the device.

The exact nature of the calibration process is a black box and not fully documented, but you can read the calibration status of each sensor using the **.getCalibration** function in the Adafruit_BNO055 (https://adafru.it/f0l) library. An example showing how to use this function can be found in the sensorapi demo, though the code is also shown below for convenience sake.

The four calibration registers -- an overall system calibration status, as well individual gyroscope, magnetometer and accelerometer values -- will return a value between '0' (uncalibrated data) and '3' (fully calibrated). The higher the number the better the data will be.

```
/**************************************************************************/
/*
    Display sensor calibration status
```

```
 */
/**************************************************************************/
void displayCalStatus(void)
{
  /* Get the four calibration values (0..3) */
  /* Any sensor data reporting 0 should be ignored, */
  /* 3 means 'fully calibrated" */
  uint8_t system, gyro, accel, mag;
  system = gyro = accel = mag = 0;
  bno.getCalibration(&system, &gyro, &accel, &mag);

  /* The data should be ignored until the system calibration is > 0 */
  Serial.print("\t");
  if (!system)
  {
    Serial.print("! ");
  }

  /* Display the individual values */
  Serial.print("Sys:");
  Serial.print(system, DEC);
  Serial.print(" G:");
  Serial.print(gyro, DEC);
  Serial.print(" A:");
  Serial.print(accel, DEC);
  Serial.print(" M:");
  Serial.println(mag, DEC);
}
```

Interpretting Data

The BNO055 will start supplying sensor data as soon as it is powered on. The sensors are factory trimmed to reasonably tight offsets, meaning you can get valid data even before the calibration process is complete, but particularly in NDOF mode **you should discard data as long as the system calibration status is 0 if you have the choice**.

The reason is that system cal '0' in NDOF mode means that the device has not yet found the 'north pole', and orientation values will be off The heading will jump to an absolute value once the BNO finds magnetic north (the system calibration status jumps to 1 or higher).

Text emphasized with a blue exclamation:	When running in NDOF mode, any data where the system calibration value is '0' should generally be ignored

Generating Calibration Data

To generate valid calibration data, the following criteria should be met:

- **Gyroscope**: The device must be standing still in any position
- **Magnetometer**: In the past 'figure 8' motions were required in 3 dimensions, but with recent devices fast magnetic compensation takes place with sufficient normal movement of the device
- **Accelerometer**: The BNO055 must be placed in 6 standing positions for +X, -X, +Y, -Y, +Z and -Z. This is the most onerous sensor to calibrate, but the best solution to generate the calibration data is to find a block of wood or similar object, and place the sensor on each of the 6 'faces' of the block, which will help to maintain sensor alignment during the calibration process. You should still be able to get reasonable quality data from the BNO055, however, even if the accelerometer isn't entirely or perfectly calibrated.

Persisting Calibration Data

Once the device is calibrated, the calibration data will be kept until the BNO is powered off.

The BNO doesn't contain any internal EEPROM, though, so you will need to perform a new calibration every time the device starts up, or manually restore previous calibration values yourself.

Bosch Video

Here's a video from the BNO055 makers on calibration!

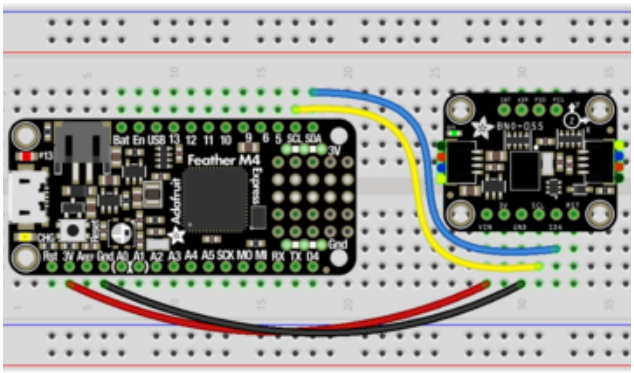
Python & CircuitPython

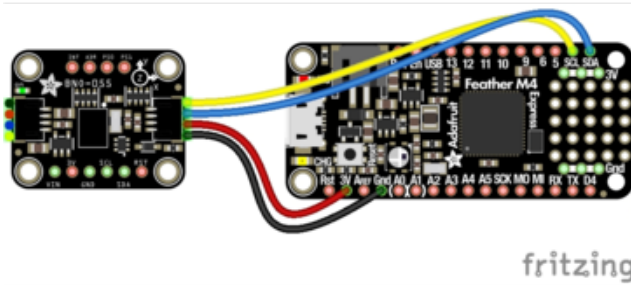
It's easy to use the BNO055 sensor with Python and CircuitPython, and the Adafruit CircuitPython BNO055 (https://adafru.it/C43) library. This library allows you to easily write Python code that reads the acceleration and orientation of the sensor.

You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

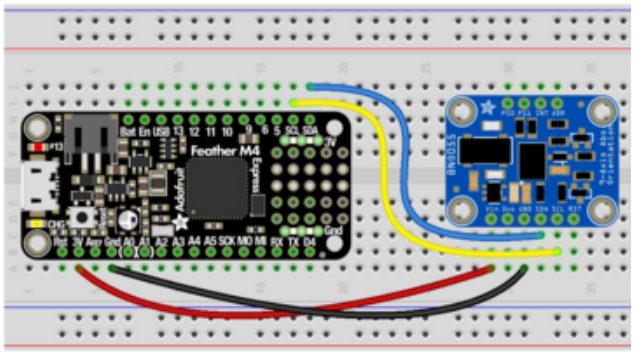
CircuitPython Microcontroller Wiring - I2C

First wire up a BNO055 to your board exactly as shown on the previous pages for Arduino using the I2C interface. Here's an example of wiring a Feather M4 to the sensor with I2C:



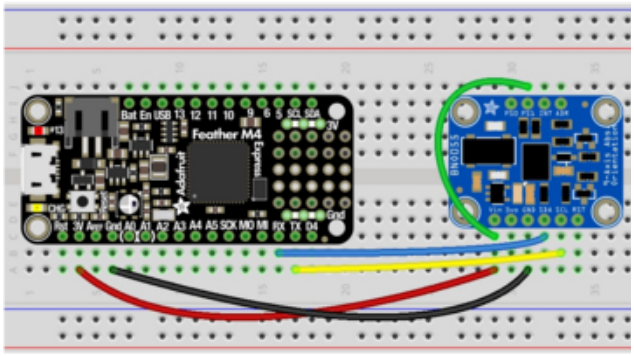


Board 3V to **sensor VIN (red wire for STEMMA QT)**
Board GND to **sensor GND (black wire for STEMMA QT)**
Board SCL to **sensor SCL (yellow wire for STEMMA QT)**
Board SDA to **sensor SDA (blue wire for STEMMA QT)**



CircuitPython Microcontroller Wiring - UART

Here's an example of wiring a Feather M4 to the sensor with UART:

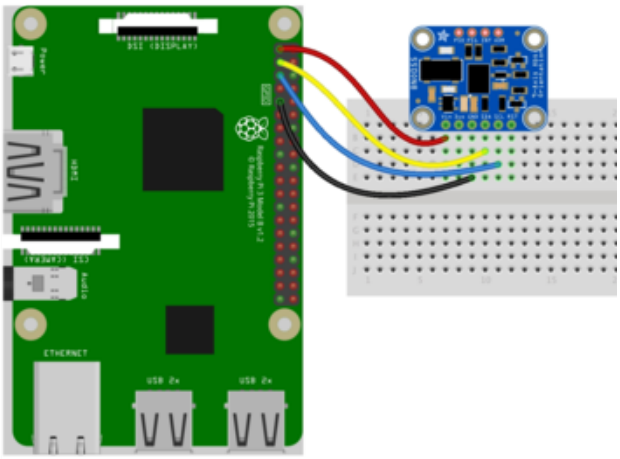


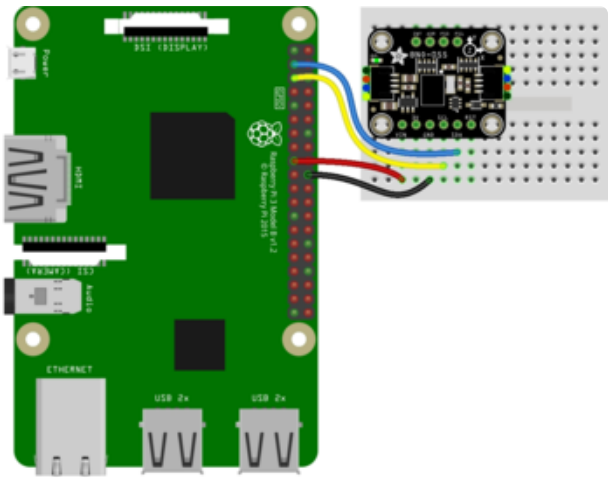
Board 3V to **sensor VIN**
Board GND to **sensor GND**
Board TX to **sensor SCL**
Board RX to **sensor SDA**
sensor PS1 to **sensor VIN**

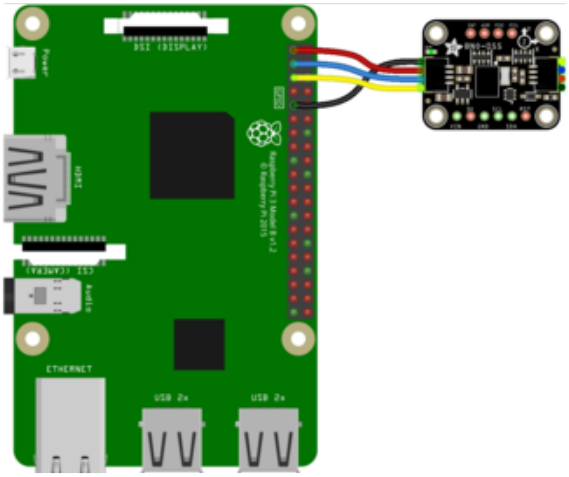
Python Computer Wiring - I2C

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, [please visit the guide for CircuitPython on Linux to see whether your platform is supported](https://adafru.it/BSN) (https://adafru.it/BSN).

Here's the Raspberry Pi wired with I2C:







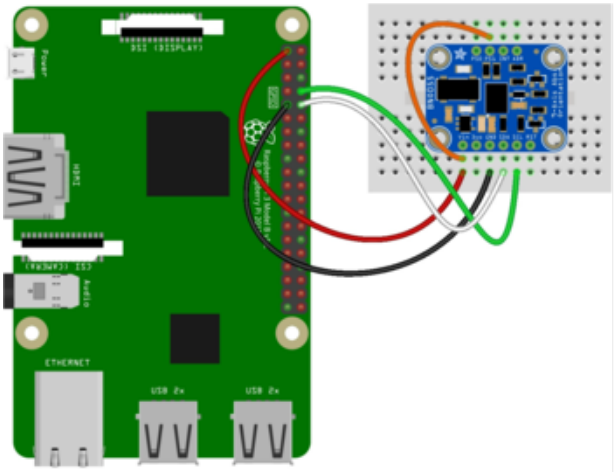
Pi 3V3 to **sensor VIN (red wire for STEMMA QT)**
Pi GND to **sensor GND (black wire for STEMMA QT)**
Pi SCL to **sensor SCL (yellow wire for STEMMA QT)**
Pi SDA to **sensor SDA (blue wire for STEMMA QT)**

Text emphasized with a yellow exclamation:

Older versions of the Raspberry Pi firmware do not have I2C clock stretching support so they don't work well with the BNO. Please ensure your firmware is updated to the latest version before continuing and slow down the I2C as explained here https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/i2c-clock-stretching

Python Computer Wiring - UART

Here's the Raspberry Pi wired with UART:



Pi 3V3 to **sensor VIN**
Pi GND to **sensor GND**
Pi TXD to **sensor SCL**
Pi RXD to **sensor SDA**
sensor PS1 to **sensor VIN**

Text emphasized with a yellow exclamation: You will also need to configure the Pi to enable the UART and disable the login console from using it.

CircuitPython Installation of BNO055 Library

Next you'll need to install the Adafruit CircuitPython BNO055 (https://adafru.it/C43) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). For example the Circuit Playground Express guide has a great page on how to install the library bundle (https://adafru.it/Bf2) for express boards.

The **lib** folder on your **CIRCUITPY** drive should contain at least the following libraries:

- **adafruit_bno055.mpy**

- **adafruit_bus_device**
- **adafruit_register**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_bno055.mpy, adafruit_bus_device,** and **adafruit_register** files and folders copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.



Text emphasized with a yellow exclamation: The BNO055 CircuitPython library is large, and cannot be imported on a SAMD21 due to lack of RAM space.

Python Installation of BNO055 Library

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Once that's done, from your command line run the following command:

- `pip3 install adafruit-circuitpython-bno055`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!



Text emphasized with a yellow exclamation: Older versions of the Raspberry Pi firmware do not have I2C clock stretching support so they don't work well with the BNO. Please ensure your firmware is updated to the latest version before continuing and slow down the I2C as explained here https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/i2c-clock-stretching

To use this sensor, you must enable i2c slowdown on the Raspberry Pi device tree overlay. Check out this guide for instructions! (https://adafru.it/C7x)

CircuitPython & Python Usage

To demonstrate the usage of the sensor we'll initialize it and read the acceleration, orientation (in Euler angles), and more from the board's Python REPL. The difference between I2C and UART is only with the initialization. After that, you can use the sensor the same with either connection.

I2C Initialization

If you are using the I2C connection, create you sensor object as follows:

```
import board
import busio
import adafruit_bno055
i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_bno055.BNO055_I2C(i2c)
```

UART Initialization - CircuitPython

If you are using the UART connection with a board (like a Feather) running CircuitPython, create your sensor object as follows:

```
import board
import busio
import adafruit_bno055
uart = busio.UART(board.TX, board.RX)
sensor = adafruit_bno055.BNO055_UART(uart)
```

UART Initialization - Python

Check how you specific board supports UART and where the port entry is created and named. For the Raspberry Pi, this is done using the **pyserial** module and the UART used is `/dev/serial0`. Then you create your sensor object as follows:

```
import serial
import adafruit_bno055
uart = serial.Serial("/dev/serial0")
sensor = adafruit_bno055.BNO055_UART(uart)
```

Usage

Now you're ready to read values from the sensor using any of these properties:

- **temperature** - The sensor temperature in degrees Celsius.
- **acceleration** - This is a 3-tuple of X, Y, Z axis accelerometer values in meters per second squared.
- **magnetic** - This is a 3-tuple of X, Y, Z axis magnetometer values in microteslas.
- **gyro** - This is a 3-tuple of X, Y, Z axis gyroscope values in degrees per second.
- **euler** - This is a 3-tuple of orientation Euler angle values.
- **quaternion** - This is a 4-tuple of orientation quaternion values.
- **linear_acceleration** - This is a 3-tuple of X, Y, Z linear acceleration values (i.e. without effect of gravity) in meters per second squared.
- **gravity** - This is a 3-tuple of X, Y, Z gravity acceleration values (i.e. without the effect of linear acceleration) in meters per second squared.

```
>>> print('Temperature: {} degrees C'.format(sensor.temperature))
Temperature: 23 degrees C
>>> print('Accelerometer (m/s^2): {}'.format(sensor.acceleration))
Accelerometer (m/s^2): (0.0, -0.6, 9.47)
>>> print('Magnetometer (microteslas): {}'.format(sensor.magnetic))
Magnetometer (microteslas): (-16.75, -7.1875, -42.0625)
>>> print('Gyroscope (deg/sec): {}'.format(sensor.gyro))
Gyroscope (deg/sec): (-0.00109083, 0.00218166, 0.0)
>>> print('Euler angle: {}'.format(sensor.euler))
Euler angle: (117.937, -0.25, 3.1875)
>>> print('Quaternion: {}'.format(sensor.quaternion))
Quaternion: (0.514832, -0.0124512, 0.0251465, -0.856812)
>>> print('Linear acceleration (m/s^2): {}'.format(sensor.linear_acceleration))
Linear acceleration (m/s^2): (0.01, 0.0, -0.31)
>>> print('Gravity (m/s^2): {}'.format(sensor.gravity))
Gravity (m/s^2): (-0.04, -0.54, 9.79)
>>>
```

That's all there is to using the BNO055 sensor with CircuitPython!

Here's a complete example that prints each of the properties every second. Save this as **code.py** on your board and look for the output in the serial REPL.

Full Example Code

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time

import board
```

```
7
8   import adafruit_bno055
9
10  i2c = board.I2C()  # uses board.SCL and board.SDA
11  # i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
12  sensor = adafruit_bno055.BNO055_I2C(i2c)
13
14  # If you are going to use UART uncomment these lines
15  # uart = board.UART()
16  # sensor = adafruit_bno055.BNO055_UART(uart)
17
18  last_val = 0xFFFF
19
20
21  def temperature():
22      global last_val  # noqa: PLW0603
23      result = sensor.temperature
24      if abs(result - last_val) == 128:
25          result = sensor.temperature
26          if abs(result - last_val) == 128:
27              return 0b00111111 & result
28      last_val = result
29      return result
30
31
32  while True:
33      print(f"Temperature: {sensor.temperature} degrees C")
34      """
35      print(
36          "Temperature: {} degrees C".format(temperature())
37      )  # Uncomment if using a Raspberry Pi
38      """
39      print(f"Accelerometer (m/s^2): {sensor.acceleration}")
40      print(f"Magnetometer (microteslas): {sensor.magnetic}")
41      print(f"Gyroscope (rad/sec): {sensor.gyro}")
42      print(f"Euler angle: {sensor.euler}")
43      print(f"Quaternion: {sensor.quaternion}")
44      print(f"Linear acceleration (m/s^2): {sensor.linear_acceleration}")
45      print(f"Gravity (m/s^2): {sensor.gravity}")
46      print()
47
48      time.sleep(1)
```

https://github.com/adafruit/Adafruit_CircuitPython_BNO055/blob/main/examples/bno055_simpletest.py

Python Docs

Python Docs (https://adafru.it/C45)

WebGL Example

Included with the BNO055 library is an example of how to send orientation readings to a webpage and use it to rotate a 3D model. Follow the steps below to setup and run this example.

This example is for use with Raspberry Pi and other Linux computers - flask doesn't run on CircuitPython yet!

Dependencies

In addition to the BNO055 libary, you'll need to install the flask Python web framework (https://adafru.it/cFR).

Connect to your board in a command terminal and run the following commands:

```
sudo apt-get update
sudo apt-get install python3-flask
```

You will also need to be using a web browser that supports WebGL (https://adafru.it/fEf) on your computer or laptop. I recommend and have tested the code for this project with the latest version of Chrome (https://adafru.it/fEg).

Download the WebGL Example

The example can be found in the library repo here (https://adafru.it/ENo). There are various ways you can get all the code. The easiest is probably to just clone the repo to your Pi:

```
git clone https://github.com/adafruit/Adafruit_CircuitPython_BNO055.git
```

And then you can navigate to the examples folder in the repo. Or copy the example to another location, etc.

Start Server

Navigate to the **webgl_demo** example folder that you downloaded above. Then you can start the server by running:

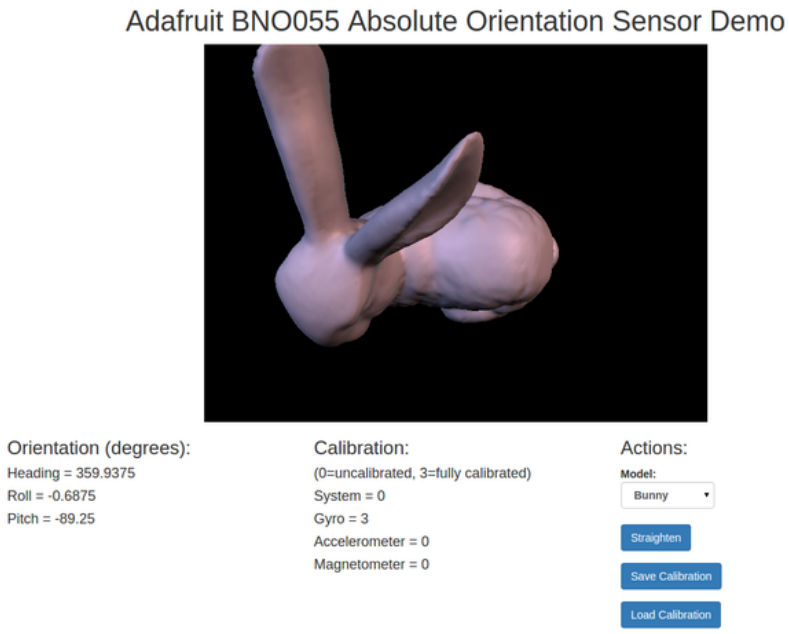
```
sudo python3 server.py
```

You should see text like the following after the server starts running:

```
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
```

Now open a web browser on your computer and navigate to your board's IP address or hostname on port 5000. For example on a Raspberry Pi http://raspberrypi.local:5000/ (https://adafru.it/fEh) might work, or on a BeagleBone Black http://beaglebone:5000/ (https://adafru.it/fEi) is the URL to try. If neither URL works you'll need to look up the IP address of your device (https://adafru.it/fEj) and then access it on port 5000. For example if your board has the IP address 192.168.1.5 you would access http://192.168.1.5:5000/ (https://adafru.it/fEk).

Once the page loads you should see something like the following:



If you move the BNO055 sensor you should see the 3D model move too. However when the demo first runs the sensor will be uncalibrated and likely not providing good orientation data. Follow the next section to learn how to calibrate the sensor.

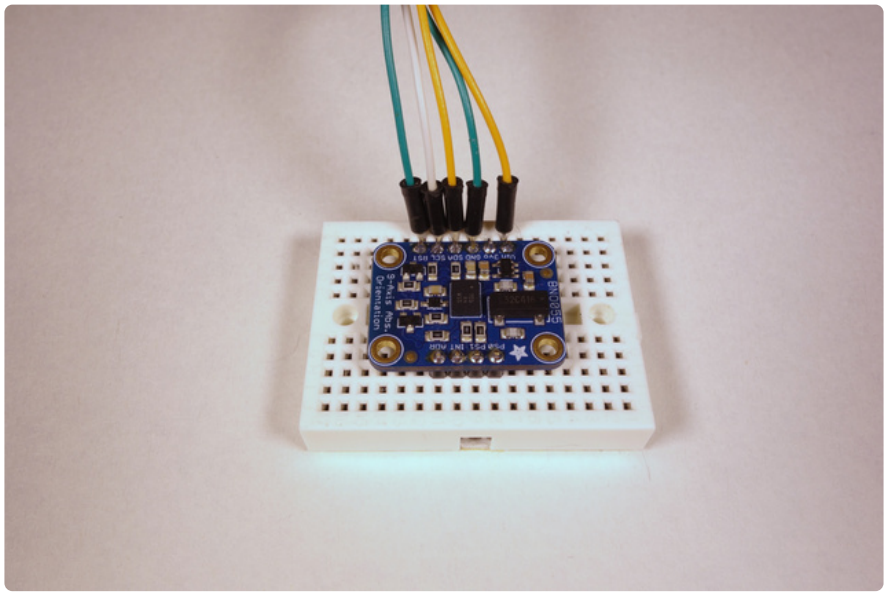
Sensor Calibration

This feature is currently not active. Once the library has been updated to allow calibration data to be saved and loaded, this section will get updated.

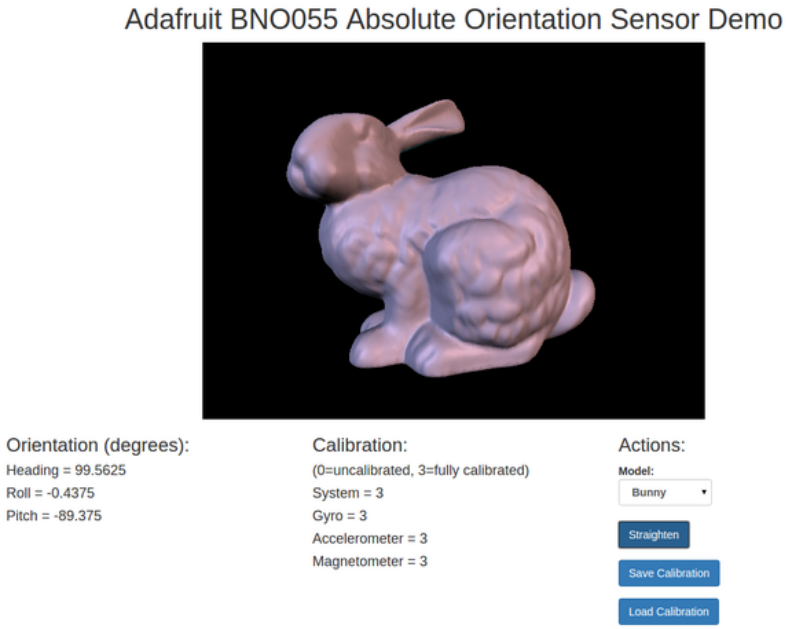
For now, just have fun spinning the little 3D rabbit around.

Usage

You can line up the axes of the sensor and 3D model by using the **Straighten** button. First you'll need to place the BNO sensor in a very specific orientation. Place the sensor flat in front of you and with the row of SDA, SCL, etc. pins facing away from you like shown below:

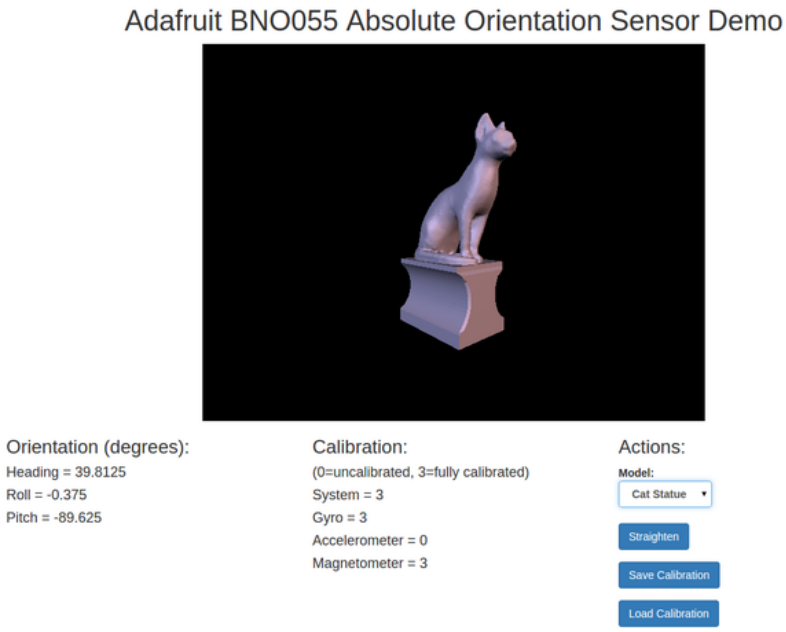


Then click the **Straighten** button and you should see the 3D model snap into its normal position:



Now move the BNO055 sensor around and you should see your movements exactly matched by the 3D model!

You can also change the 3D model by clicking the **Model** drop down on the right and changing to a different model, like a cat statue:



That's all there is to using the BNO055 WebGL demo!

To stop the server go back to the terminal where it was started and press **Ctrl-C**.

More Info

Describing how all of the WebGL code works is a little too complex for this guide, however the high level components of the example are:

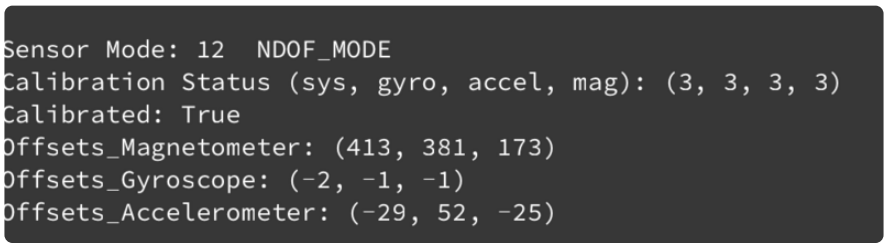
- flask web service framework (https://adafru.it/cFR): This is a great, simple web framework that is used by server.py to serve the main index.html page and expose a few web service endpoints to read BNO sensor data and save/load calibration data.
- HTML5 server sent events (https://adafru.it/dh7): This is how data is sent from the server to the webpage. With SSE a connection is kept open and data is pushed to the client web page. BNO sensor readings are taken and sent over SSE where they're use to update the orientation of the model. This page (https://adafru.it/fEl) has a little more info on how to use HTML5 SSE with the flask framework (although it uses a more complex multiprocessing

framework called gevent (https://adafru.it/fEm) that isn't necessary for simple apps like this demo).

- Three.js (https://adafru.it/fEn): This is the JavaScript library that handles all the 3D model rendering.
- Bootstrap (https://adafru.it/fEo) & jQuery (https://adafru.it/fEp): These are a couple other JavaScript libraries that are used for the layout and some core functionality of the page.

That's all there is to using the BNO055 WebGL demo. Enjoy using the BNO055 absolute orientation sensor in your own projects!

BNO055 Sensor Calibration, Target Angle Offset, and Tap Detection in CircuitPython



Overview

This page of notes by CGrover (https://adafru.it/197E) was used to develop BNO055 9-DoF sensor algorithms for the PowerWash Simulator Controller project and discusses three essential characteristics of the sensor.

First, the relative and absolute calibration of the sensor can be performed to improve initial sensor stability and positioning. Stand-alone sensor calibrator code is shown and was submitted to the driver library's examples folder.

Next we'll talk about how to measure and adjust for user position orientation without changing the sensor's absolute position calibration.

Finally, since tap detection is not native to the BNO055 chip, an example of how to detect single and double-taps with the accelerometer component of the sensor is described.

BNo055 Sensor Calibration

The sensor's offset registers each contain `(0, 0, 0)` after power-on, indicating that one or more of the sensor components isn't fully calibrated. If left alone, the sensor would eventually calibrate as its internal background calibration routine watches the sensor's movement. It can take a long time to calibrate the sensor in this manner unless the user executes a calibration dance just after power-up.

The preferred approach is to conduct a calibration dance using a separate module that provides offset values that can be inserted into the project code to preset the sensor's offset registers just after power-up. Presetting the magnetometer is important since reliable absolute positioning is dependent on the magnetometer knowing its geographic location relative to magnetic north. Presetting the gyroscope and accelerometer registers isn't as critical, but is a good practice.

Here's the stand-alone calibrator method that's included in the driver library's examples folder:

```
# SPDX-FileCopyrightText: 2023 JG for Cedar Grove Maker Studios
# SPDX-License-Identifier: MIT

"""
`bno055_calibrator.py`
================================================================================
A CircuitPython module for calibrating the BNo055 9-DoF sensor. After manually
calibrating the sensor, the module produces calibration offset tuples for use
in project code.

* Author(s): JG for Cedar Grove Maker Studios

Implementation Notes
--------------------
**Hardware:**
* Adafruit BNo055 9-DoF sensor
**Software and Dependencies:**
* Driver library for the sensor in the Adafruit CircuitPython Library Bundle
* Adafruit CircuitPython firmware for the supported boards:
  https://circuitpython.org/downloads
"""

import time

import board

import adafruit_bno055


class Mode:
    CONFIG_MODE = 0x00
    ACCONLY_MODE = 0x01
    MAGONLY_MODE = 0x02
    GYRONLY_MODE = 0x03
```

```
    ACCMAG_MODE = 0x04
    ACCGYRO_MODE = 0x05
    MAGGYRO_MODE = 0x06
    AMG_MODE = 0x07
    IMUPLUS_MODE = 0x08
    COMPASS_MODE = 0x09
    M4G_MODE = 0x0A
    NDOF_FMC_OFF_MODE = 0x0B
    NDOF_MODE = 0x0C


# Uncomment these lines for UART interface connection
# uart = board.UART()
# sensor = adafruit_bno055.BNO055_UART(uart)

# Instantiate I2C interface connection
# i2c = board.I2C()  # For board.SCL and board.SDA
i2c = board.STEMMA_I2C()  # For the built-in STEMMA QT connection
sensor = adafruit_bno055.BNO055_I2C(i2c)
sensor.mode = Mode.NDOF_MODE  # Set the sensor to NDOF_MODE

print("Magnetometer: Perform the figure-eight calibration dance.")
while not sensor.calibration_status[3] == 3:
    # Calibration Dance Step One: Magnetometer
    #   Move sensor away from magnetic interference or shields
    #   Perform the figure-eight until calibrated
    print(f"Mag Calib Status: {100 / 3 * sensor.calibration_status[3]:3.0f}%")
    time.sleep(1)
print("... CALIBRATED")
time.sleep(1)

print("Accelerometer: Perform the six-step calibration dance.")
while not sensor.calibration_status[2] == 3:
    # Calibration Dance Step Two: Accelerometer
    #   Place sensor board into six stable positions for a few seconds each:
    #    1) x-axis right, y-axis up,    z-axis away
    #    2) x-axis up,    y-axis left,  z-axis away
    #    3) x-axis left,  y-axis down,  z-axis away
    #    4) x-axis down,  y-axis right, z-axis away
    #    5) x-axis left,  y-axis right, z-axis up
    #    6) x-axis right, y-axis left,  z-axis down
    #   Repeat the steps until calibrated
    print(f"Accel Calib Status: {100 / 3 * sensor.calibration_status[2]:3.0f}%")
    time.sleep(1)
print("... CALIBRATED")
time.sleep(1)

print("Gyroscope: Perform the hold-in-place calibration dance.")
while not sensor.calibration_status[1] == 3:
    # Calibration Dance Step Three: Gyroscope
    #  Place sensor in any stable position for a few seconds
    #  (Accelerometer calibration may also calibrate the gyro)
    print(f"Gyro Calib Status: {100 / 3 * sensor.calibration_status[1]:3.0f}%")
    time.sleep(1)
print("... CALIBRATED")
time.sleep(1)
```

```
91
92  print("\nCALIBRATION COMPLETED")
93  print("Insert these preset offset values into project code:")
94  print(f"  Offsets_Magnetometer:  {sensor.offsets_magnetometer}")
95  print(f"  Offsets_Gyroscope:     {sensor.offsets_gyroscope}")
96  print(f"  Offsets_Accelerometer: {sensor.offsets_accelerometer}")
```

https://github.com/adafruit/Adafruit_CircuitPython_BNO055/blob/main/examples/bno055_calibrator.py

Dance Step One: The Figure-Eight

- To calibrate the magnetometer, wave the sensor slowly in a figure-8 pattern until the REPL says "CALIBRATED."

Dance Step Two: The Six-Step Rotate

- The accelerometer is then calibrated by holding it on an edge facing you for a few seconds then rotating it clockwise 90 degrees, wait, and repeat for a total of 4 positions. Then place it face-up on a flat surface and hold it there for a few seconds. Finally, flip it face down and hold it to complete the accelerometer calibration.

Dance Step Three: The Look Up and Wait

- The last step is for the gyroscope. All it needs is to be held still for a few seconds in a face-up position. The accelerometer calibration usually takes care of the gyroscope calibration.

After all three calibration dances complete, the preset offset values will appear in the REPL.

Repeating the calibration process produces some variance in the offset values, but the scale and magnitude are usually close. Since the sensor is continuously calibrating, close is good enough for most projects. The primary benefit of calibrating the sensor once using the stand-alone code is that the project application begins with a useful orientation from the get-go and won't require a calibration dance recital for each power-on startup.



Text emphasized with a blue exclamation:

The sensor components (particularly the magnetometer) will gradually deviate from the preset offset by a few counts during operation. The automatic background calibration process is designed to accommodate the drift and maintain the accuracy of the sensors.

Optional: Preserving Calibration between Power On/Off Cycles

Rather than the copy/paste method described above, storing and reusing configuration offsets from one power-on/off session to the next is possible since the offset register properties can be read and changed. After conducting a single stand-alone calibration, store the calibration offset registers into the NVM memory or an SD card file for use during subsequent power-on startups. You may also want to consider updating the NVM or file periodically during regular use as the offset registers are continually adjusted by the internal background calibration task.

User Orientation Offset (Target Angle Offset)

A user orientation offset to correct for the alignment of the display in relationship with the sensor will usually be needed by a project, initiated with a button press or other event like an accelerometer double-tap. Changing the target angle offset doesn't recalibrate the sensor, it just uses the current Euler angle to provide the offset for future position readings.

```
# The target angle offset used to reorient the sensor
# (heading, roll, pitch)
target_angle_offset = (0, 0, 0)

# The project's main while loop
while True:
    # Get the Euler angle values from the sensor
    # The Euler angle limits are: +180 to -180 pitch, +360 to -360 heading, +90 to -90 roll
    sensor_euler = sensor.euler
    print(f"Euler angle: {sensor_euler}")
    # Adjust the Euler angle values with the target_angle_offset
    heading, roll, pitch = [position - target_angle_offset[idx] for idx, position in enumerate(sensor_euler)]

    # Scale the heading for horizontal movement range
    horizontal_mov = int(map_range(heading, -20, 20, -30, 30))
    print(f"mouse x: {horizontal_mov}")

    # Scale the roll for vertical movement range
    vertical_mov = int(map_range(roll, -25, 25, 30, 30))
    print(f"mouse y: {vertical_mov}")

    # Translate to stuff needed for HID
    mouse.move(x=horizontal_mov)
    mouse.move(y=vertical_mov)

    # Check the "reorient" button was pressed
    if reorientation_button:
        print(f"Reorient the sensor")
        # Use the current Euler angle values to reorient the target angle
        target_angle_offset = [angle for angle in sensor_euler]
```

Tap Detection

Here's a fairly simple non-blocking single and double tap detection scheme that takes advantage of the BNO055's 100Hz-ish measurement data rate. The accelerometer's data rate acts like a high pass filter when measuring the delta between two measurements.

The tap sensitivity threshold can be set to accommodate the sensor's mechanical mounting scheme; 1.0 is overly sensitive, 5.0 is typical, and 10 is somewhat numb. The tap debounce setting can also vary somewhat depending on the sensor mount; 0.1 seconds works for nicely for sensors that are securely attached, 0.3 is typical if the sensor is suspended in foam, and 0.5 may be needed if mounted loosely. Adjust these values for your project's particulars.

Single-Tap Detection

```
def euclidean_distance(reference, measured):
    """Calculate the Euclidean distance between reference and measured points
    in a universe. The point position tuples can be colors, compass,
    accelerometer, absolute position, or almost any other multiple value data
    set.
    reference: A tuple or list of reference point position values.
    measured: A tuple or list of measured point position values."""

    # Create list of deltas using list comprehension
    deltas = [(reference[idx] - count) for idx, count in enumerate(measured)]
    # Resolve squared deltas to a Euclidean difference and return the result
    return math.sqrt(sum([d ** 2 for d in deltas]))

# Set the tap detector parameters
TAP_THRESHOLD = 6  # Tap sensitivity threshold; depends on the physical sensor mount
TAP_DEBOUNCE = 0.3  # Time for accelerometer to settle after tap (seconds)

# The project's main while loop
while True:
    # Detect a single tap on any axis of the BNo055 accelerometer
    accel_sample_1 = sensor.acceleration  # Read one sample
    accel_sample_2 = sensor.acceleration  # Read the next sample
    if euclidean_distance(accel_sample_1, accel_sample_2) >= TAP_THRESHOLD:
        # The difference between two consecutive samples exceeded the threshold
        # (equivalent to a high-pass filter)
        print(f"SINGLE tap detected")
        #
        # Perform the single-tap task here
        #
        time.sleep(TAP_DEBOUNCE)  # Debounce delay
```

Double-Tap Detection

```
def euclidean_distance(reference, measured):
    """Calculate the Euclidean distance between reference and measured points
    in a universe. The point position tuples can be colors, compass,
    accelerometer, absolute position, or almost any other multiple value data
    set.
    reference: A tuple or list of reference point position values.
    measured: A tuple or list of measured point position values."""

    # Create list of deltas using list comprehension
    deltas = [(reference[idx] - count) for idx, count in enumerate(measured)]
    # Resolve squared deltas to a Euclidean difference and return the result
    return math.sqrt(sum([d ** 2 for d in deltas]))

# Set the BNo055 tap detector parameters and initialize tap event history list
TAP_THRESHOLD = 6  # Tap sensitivity threshold; depends on the physical sensor mount
TAP_DEBOUNCE = 0.1  # Time for accelerometer to settle after tap (seconds)
TAP_TIMEOUT = 1500  # Remove tap event from history timeout (milliseconds)
tap_events = []  # Initialize the tap event history list

# The project's main while looop
while True:
    # Detect a tap on any axis of the BNo055 accelerometer
    accel_sample_1 = sensor.acceleration  # Read one sample
    accel_sample_2 = sensor.acceleration  # Read the next sample
    if euclidean_distance(accel_sample_1, accel_sample_2) >= TAP_THRESHOLD:
        # The difference between two consecutive samples exceeded the threshold
        # (equivalent to a high-pass filter)
        print(f"SINGLE tap detected {ticks_ms()}")
        tap_events.append(ticks_ms() + TAP_TIMEOUT)  # save tap expiration time in event stack
        time.sleep(TAP_DEBOUNCE)  # Debounce delay

    # Clean up tap event history after timeout period expires
    if len(tap_events) > 0:
        # Check for expired events
        if tap_events[0] <= ticks_ms():
            # The oldest event has expired
            tap_events = tap_events[1:]  # Remove the oldest event

    # Check see if two taps are in the event history list
    if len(tap_events) == 2:
        # Double-tap: execute the task and clear event history
        print(f"DOUBLE tap detected {ticks_ms()}")
        #
        # Perform the double-tap task here
        #
        tap_events = []  # Clear event history
```

Additional Information

A very good calibration reference by MathWorks, but the axis orientation doesn't represent the default setting: https://www.mathworks.com/help/supportpkg/arduinoio/ug/calibrate-sensors.html (https://adafru.it/19Ce)

Bosch:

https://www.youtube.com/watch?v=Bw0WuAyGsnY (https://adafru.it/19Cf)

BNO055 Sensor CircuitPython Driver GitHub:

https://github.com/adafruit/Adafruit_CircuitPython_BNO055 (https://adafru.it/C43)

BNO055 Sensor ReadTheDocs:

https://docs.circuitpython.org/projects/bno055/en/latest/ (https://adafru.it/19Cg)

FAQs

Can I manually set the calibration constants?

Yes you can save and restore the calibration of the sensor, check out the restore_offsets example: https://github.com/adafruit/Adafruit_BN ... ffsets.ino (https://adafru.it/kBv)

One thing to keep in mind though is that the sensor isn't necessarily 'plug and play' with loading the calibration data, in particular the magnetometer needs to be recalibrated even if the offsets are loaded. The magnetometer calibration is very dynamic so saving the values once might not really help when they're reloaded and the EMF around the sensor has changed.

For further details check out the datasheet and Bosch's info on the sensor for calibration info: https://

www.bosch-sensortec.com/en/home ... 1/bno055_4 (https://adafru.it/kBw)

Does the device make any assumptions about its initial orientation?

You can customize how the axes are oriented (i.e. swap them around, etc.) but the Adafruit Arduino library doesn't expose it right now. Check out section 3.4 Axis Remap of the BNO055 datasheet for info on the registers to adjust its orientation: https://www.adafruit.com/datasheets/BST ... 000_12.pdf (https://adafru.it/kBx)

Another thing to be aware of is that until the sensor calibrates it has a relative orientation output (i.e. orientation will be relative to where the sensor was when it powered on).

A system status value of '0' in NDOF mode means that the device has not yet found the 'north pole', and orientation values will be relative not absolute. Once calibration and setup is complete (system status > '0') the heading will jump to an absolute value since the BNO has found magnetic north (the system calibration status jumps to 1 or higher). See the Device Calibration page in this learning guide for further details.

Why doesn't Euler output seem to match the Quaternion output?

The Euler angles coming out of the chip are based on 'automatic orientation detection', which has the drawback of not being continuous for all angles and situations.

According to Bosch BNO055 Euler angle output should only be used for eCompass, where pitch and roll stay below 45 degrees.

For absolute orientation, quaternions should always be used, and they can be converted to Euler angles at the last moment via the .toEuler() helper function in quaternion.h (https://adafru.it/lee).

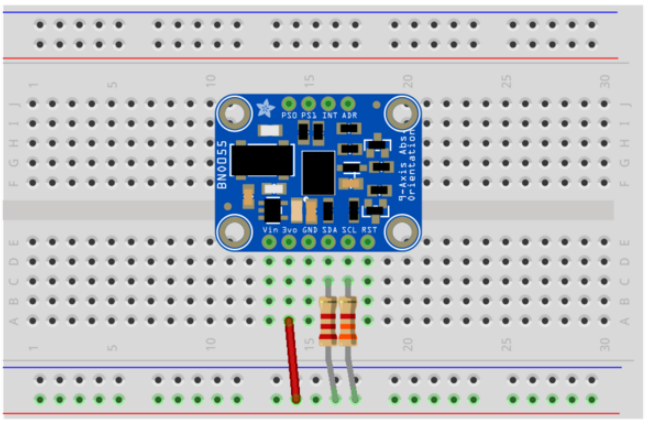
Why do I get Input/Output or I2C errors when trying to use the sensor in CircuitPython?

The BNO055 I2C implementation violates the I2C protocol in some circumstances. This causes it not to work well with certain chip families. It does not work well with Espressif ESP32, ESP32-S3, and NXP i.MX RT1011, and it does not work well with I2C

multiplexers. Operation with SAMD51, RP2040, STM32F4, and nRF52840 is more reliable.

I'm sometimes losing data over I2C, what can I do about this?

Depending on your system setup, you might need to adjust the pullups on the SCL and SDA lines to be a bit stronger. The BNO055 has very tight timing requirements on the I2C bus, requiring short setup and rise times on the signals. By default the breakout board has 10K pullups, which might be too weak on some setups. You can shorten the rise times and extend the setup time on the I2C lines with 'stronger' pullups. To do this simply add a 3.3K pullup on SCL and a 2.2K pullup on the SDA line with a breadboard or perma-proto board, which will override to weaker 10K pullups that are populated by default. See the image below for details:



I have some high frequency (> 2MHz) wires running near the BNO055 and I'm getting unusual results/hanging behavior

Turns out the BNO0055 breakout board is quite sensitive to RF interference from nearby wires with higher frequency square waves. (https://adafru.it/PPF)

Try to keep high frequency lines/wires away from the BNO055!

Downloads

Text emphasized with a blue info:	There are two versions of this breakout: one with STEMMA QT ports and one without

Files

- Arduino Library (https://adafru.it/f0I)
- EagleCAD PCB files on GitHub (https://adafru.it/sEn)
- BNO055 Stemma 3D models on GitHub (https://adafru.it/11Am)
- BNO055 Datasheet (https://adafru.it/se6) (2016)
- BNO055 Datasheet (https://adafru.it/19Cj) (10/2021)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/aP3)

https://adafru.it/19Cj

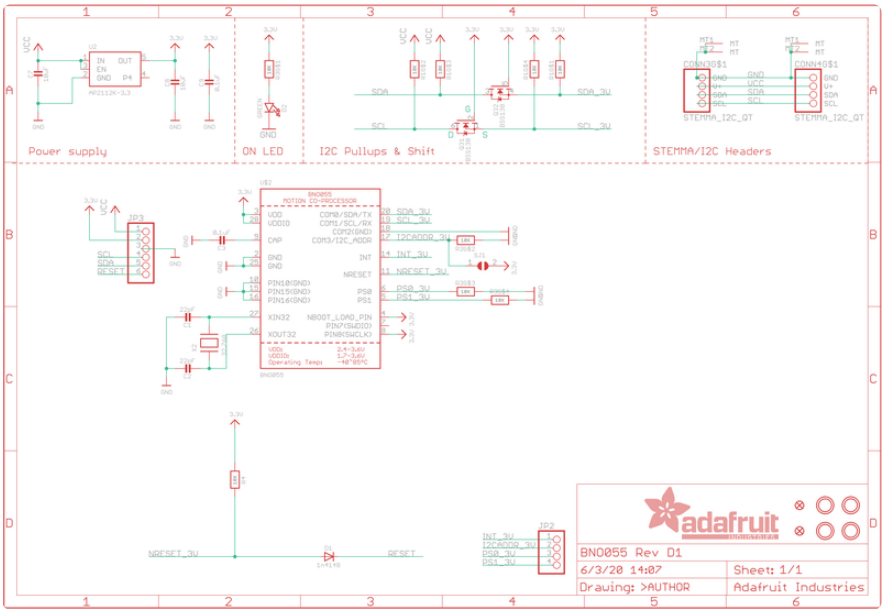
Pre-Compiled Bunny Rotate Binaries

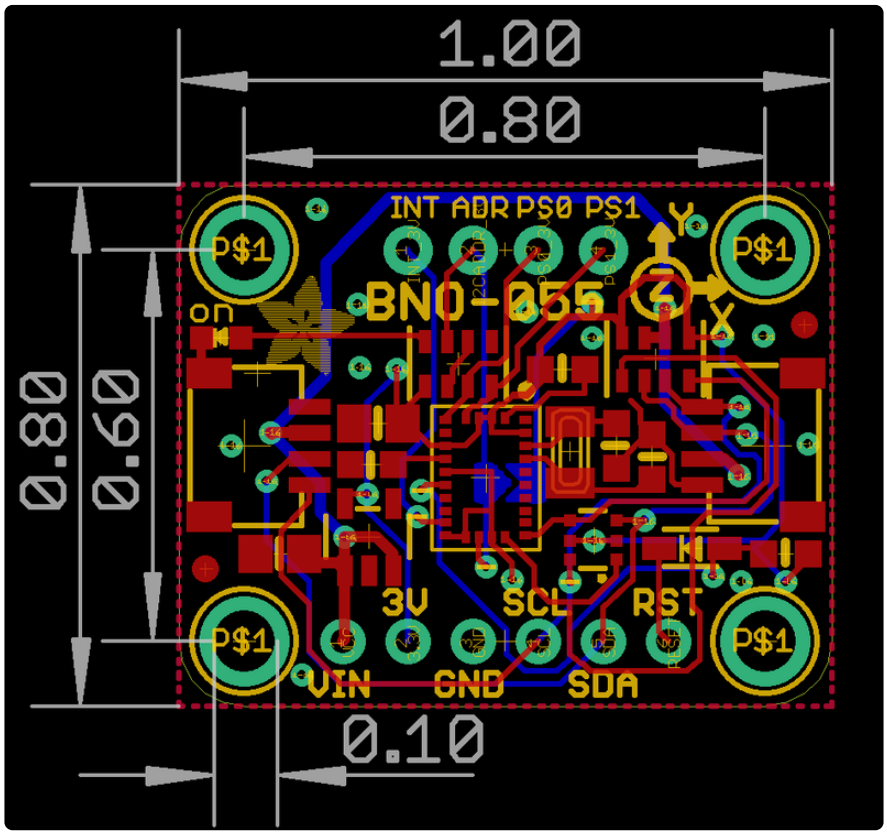
The following binary images can be used in place of running the Processing Sketch, and may help avoid the frequent API and plugin changes.

For OS X download **cuberotate.app.zip**, which was built on OS X 10.11.6 based on Processing 2.2.1:

https://adafru.it/wB4

Schematic and Fab Print for Rev D (STEMMA QT)





Schematic and Fab Print for Rev C (no STEMMA QT)

(Click the image or click here (https://adafru.it/19Cn) to view the schematic in full resolution):

