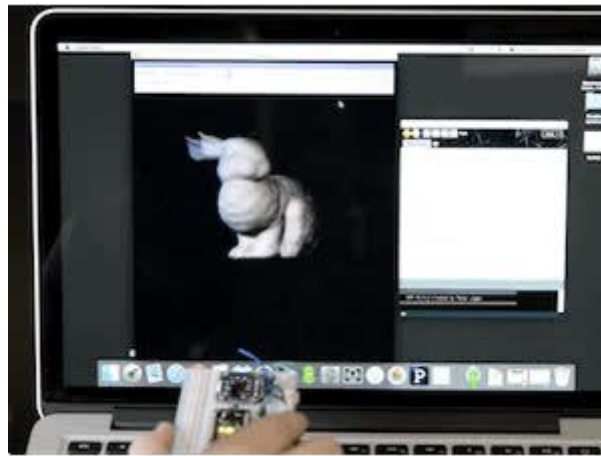




Adafruit BNO055 Absolute Orientation Sensor

Created by Kevin Townsend



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

Last updated on 2023-08-29 02:49:04 PM EDT

Table of Contents

Overview	5
<ul style="list-style-type: none">• Data Output• Related Resources	
Pinouts	7
<ul style="list-style-type: none">• Power Pins• I2C Pins• STEMMA QT version• Other Pins	
Assembly	9
<ul style="list-style-type: none">• Prepare the header strip:• Add the breakout board:• And Solder!	
Arduino Code	12
<ul style="list-style-type: none">• 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	19
<ul style="list-style-type: none">• Step 1 - Wire up the BNO055 to your Microcontroller using I2C• Step 2 - Load the Sketch onto your device• Step 3 - Install Chrome• Step 4 - Enable Web Serial API if necessary• Step 5 - Visit the Adafruit 3D Model viewer• Step 6 - Calibration• Step 7 - Euler Angles or Quaternions	
Processing Test	24
<ul style="list-style-type: none">• Requirements• Opening the Processing Sketch• Run the Bunny Sketch on the Uno• Rabbit Disco!	
Device Calibration	28
<ul style="list-style-type: none">• Interpreting Data• Generating Calibration Data• Persisting Calibration Data• Bosch Video	
Python & CircuitPython	30
<ul style="list-style-type: none">• 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](#) 38

[WebGL Example](#) 38

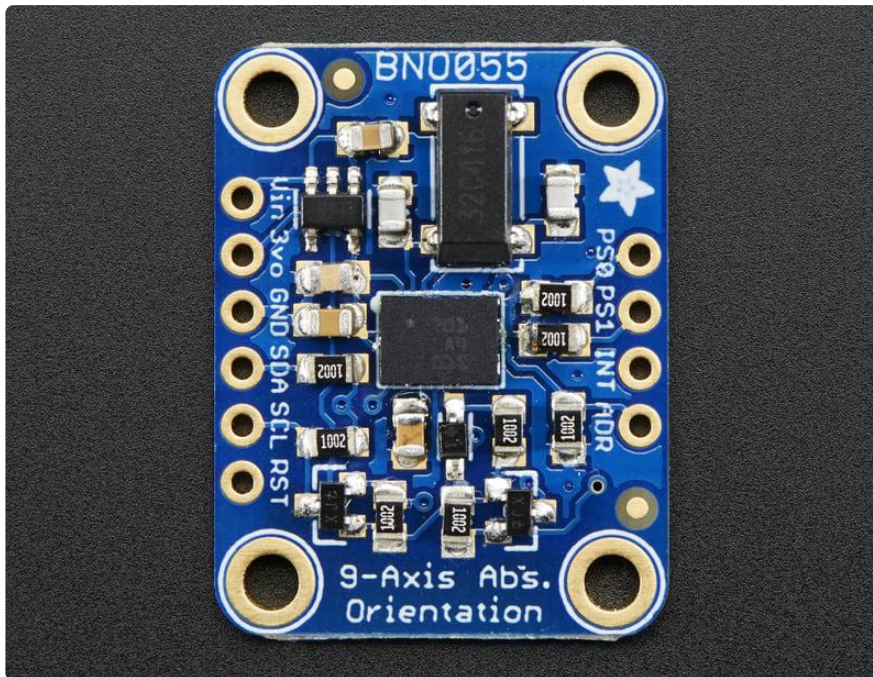
- [Dependencies](#)
- [Download the WebGL Example](#)
- [Start Server](#)
- [Sensor Calibration](#)
- [Usage](#)
- [More Info](#)

[FAQs](#) 43

[Downloads](#) 45

- [Files](#)
- [Pre-Compiled Bunny Rotate Binaries](#)
- [Schematic](#)
- [Board Dimensions](#)
- [Schematic for STEMMA QT](#)
- [Fab Print for STEMMA QT](#)

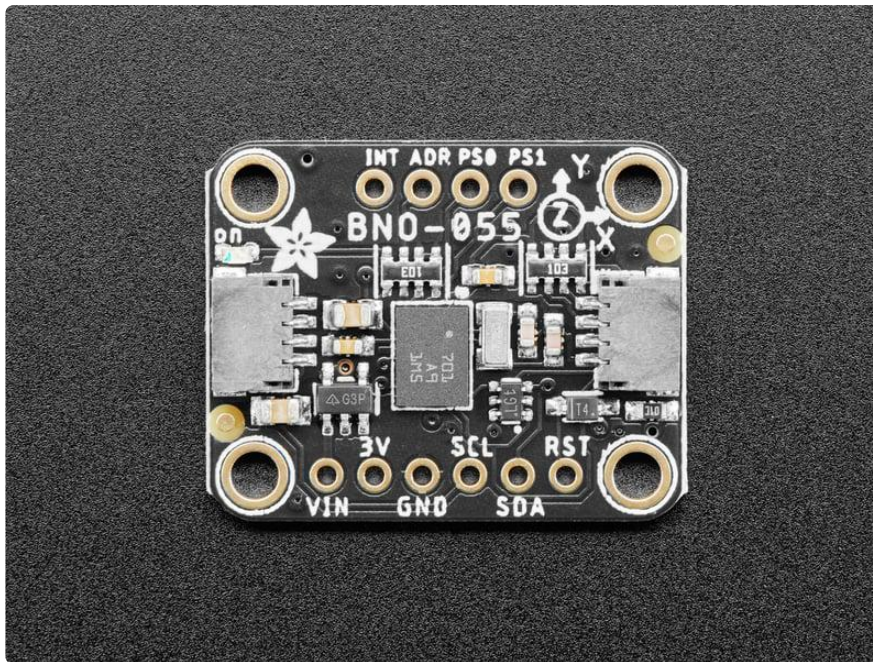
Overview



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.

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.



The new version of the board includes [SparkFun qwiic \(\)](#) compatible [STEMMA QT \(\)](#) 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 (Quaternion, 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/s²
- 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/s²
- Gravity Vector (100Hz)
Three axis of gravitational acceleration (minus any movement) in m/s²

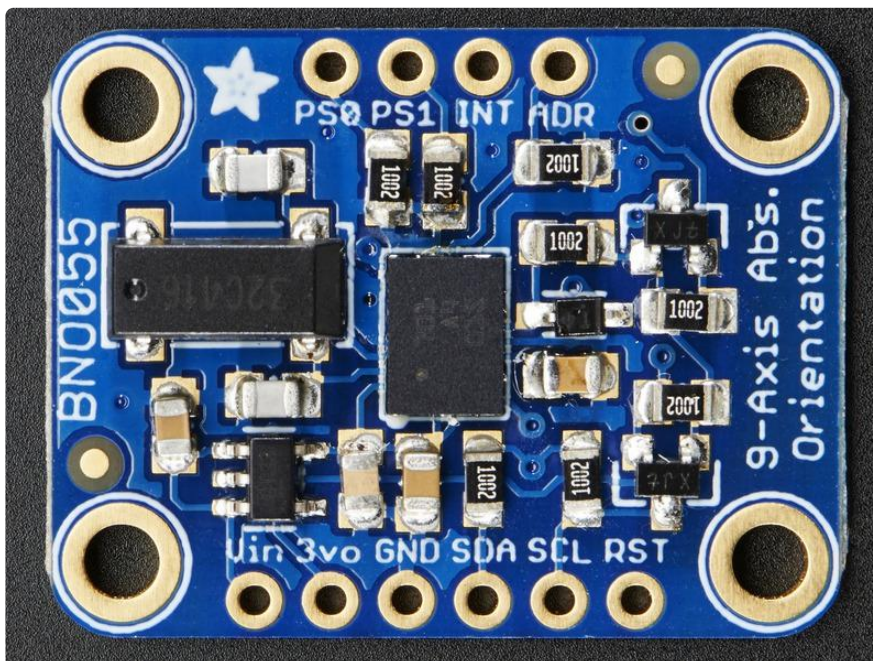
- Temperature (1Hz)
Ambient temperature in degrees celsius

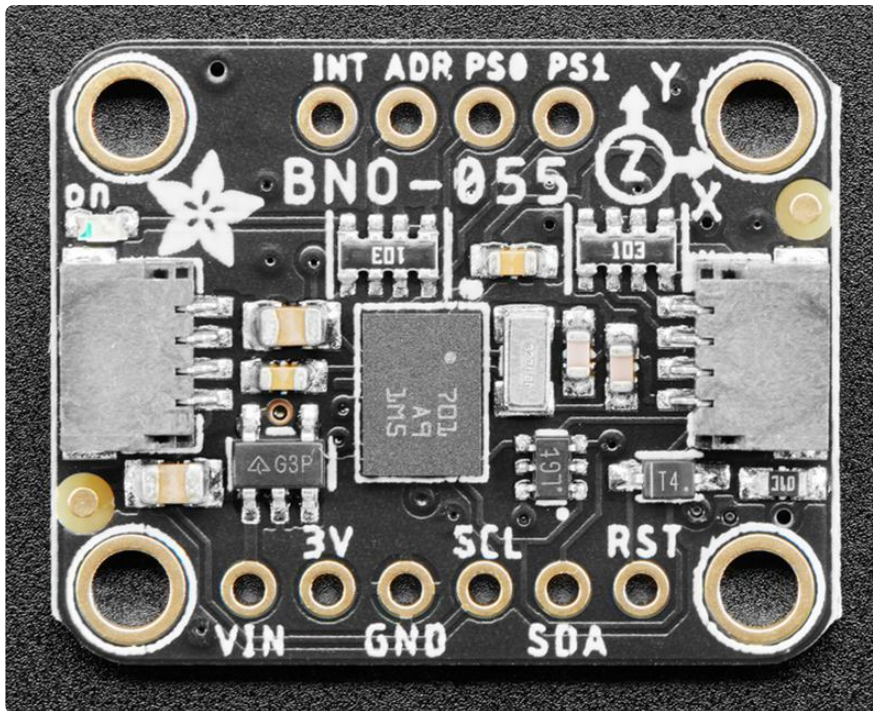
Related Resources

- [Datasheet \(\)](#)
- [Adafruit BNO055 Library \(\)](#) (Github)



Pinouts





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.

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
- 3V: 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.

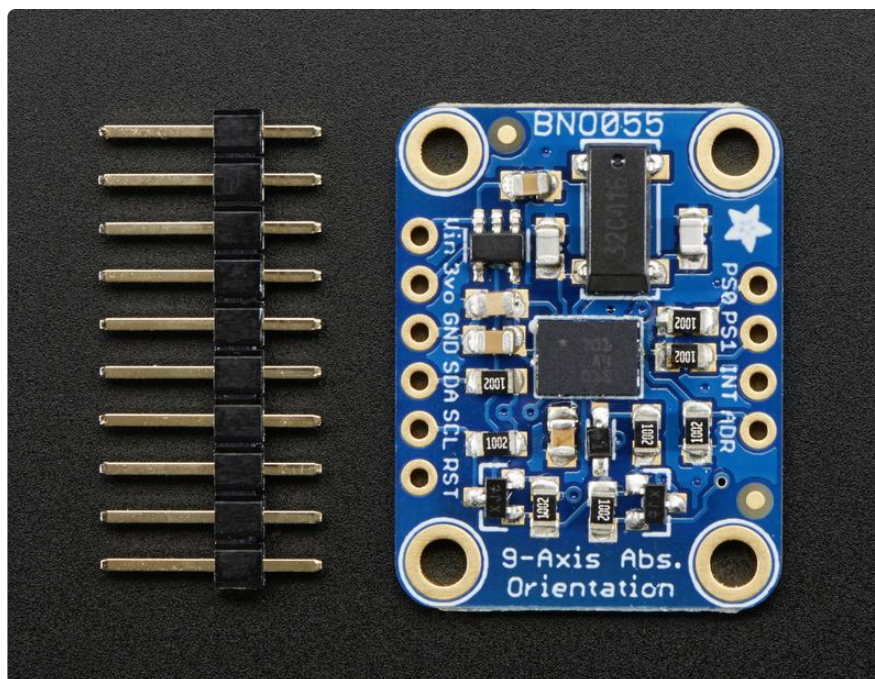
STEMMA QT version

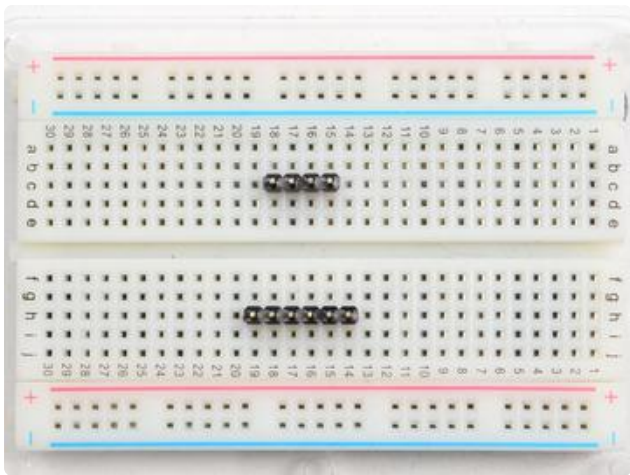
- [STEMMA QT \(\)](#) - These connectors allow you to connect to dev boards with STEMMA QT connectors or to other things with [various associated accessories \(\)](#)

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 address is 0x28. If this pin is connected to 3V, the address will be 0x29
- 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.

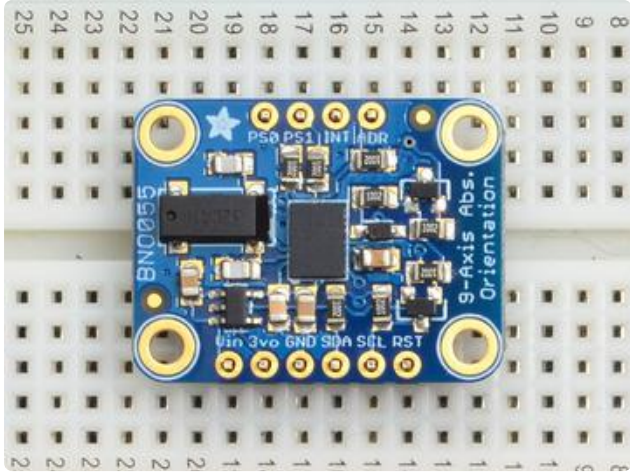
Assembly





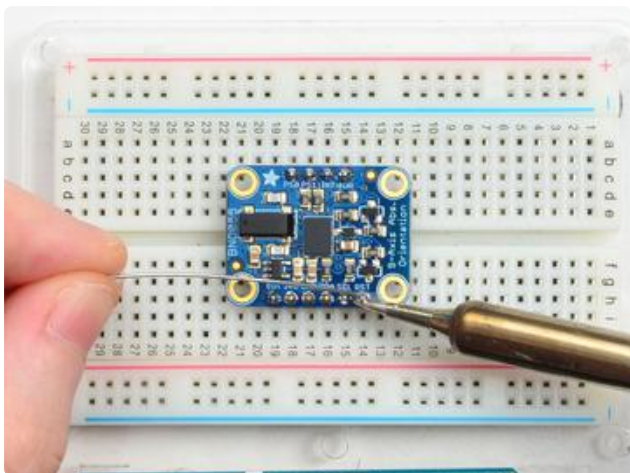
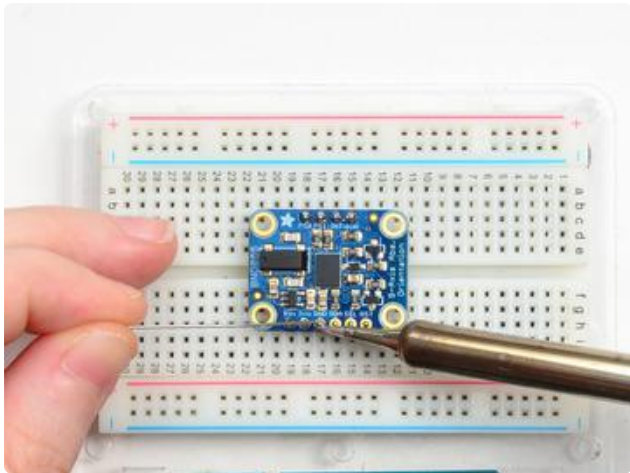
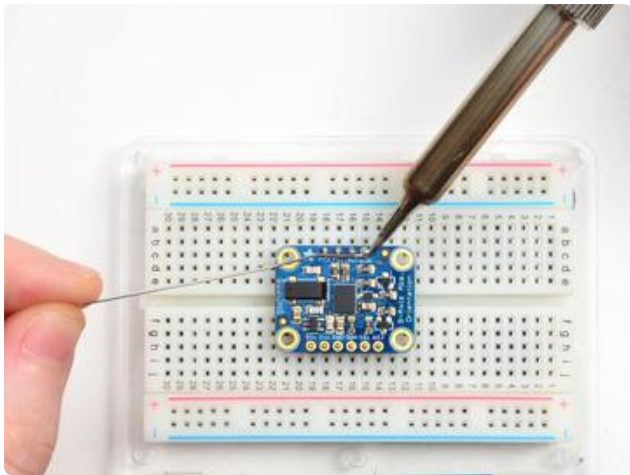
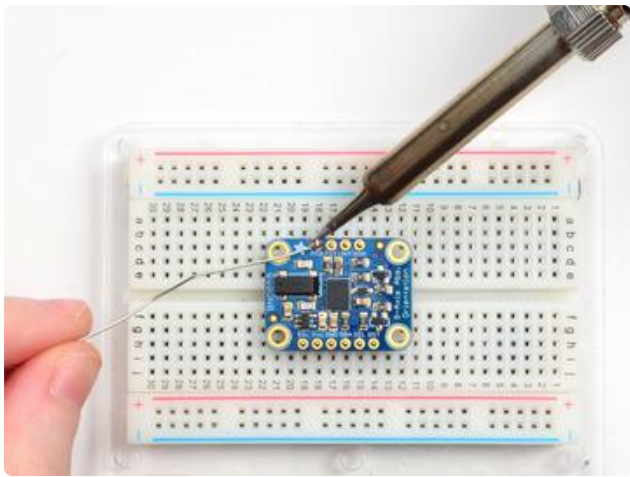
Prepare the header strip:

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

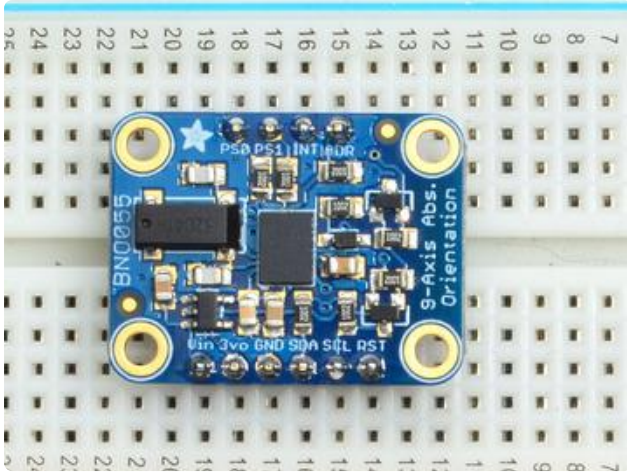


And Solder!

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 \(\)](#)).

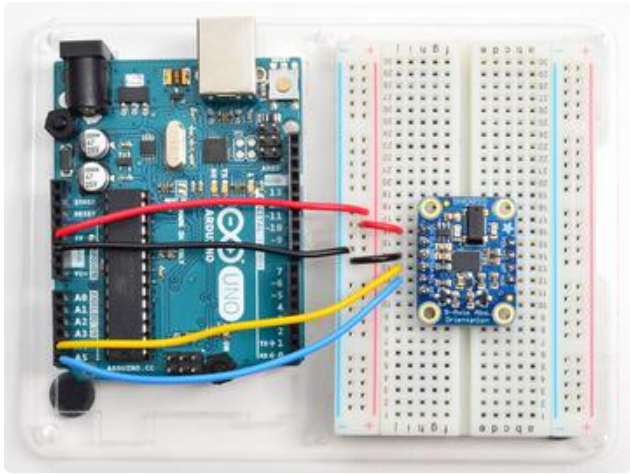


You're done! Check your solder joints visually and continue onto the next steps

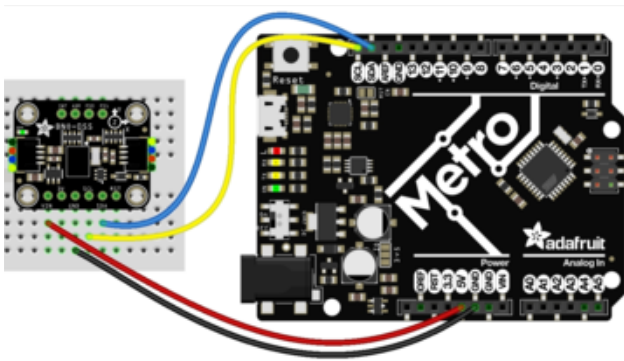
Arduino Code

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.

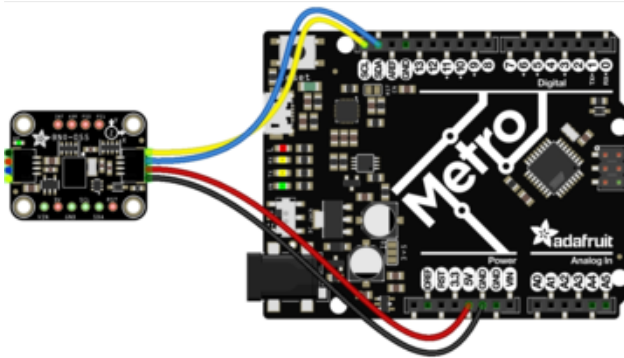


fritzing

Connect Vin (red wire) 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) to common power/data ground

Connect the SCL (yellow 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



fritzing

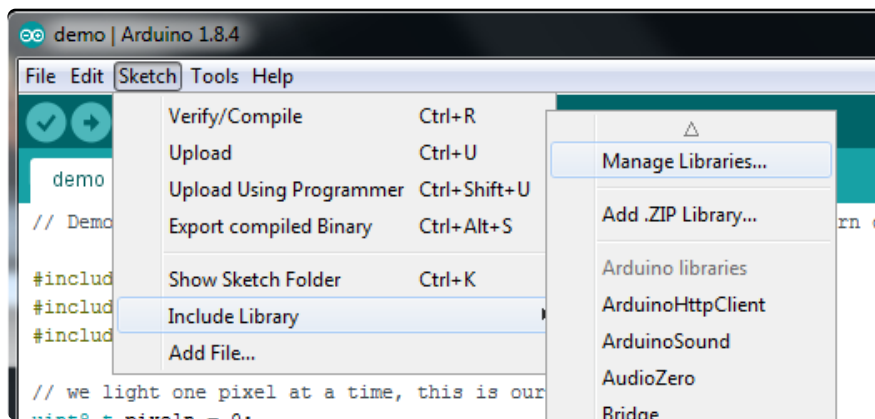
Connect the SDA (blue wire) pin to the I2C data SDA pin on your Arduino. On an UNO & '328 based Arduino, this is also known as A4, on a Mega it is also known as digital 20 and on a Leonardo/Micro, digital 2

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

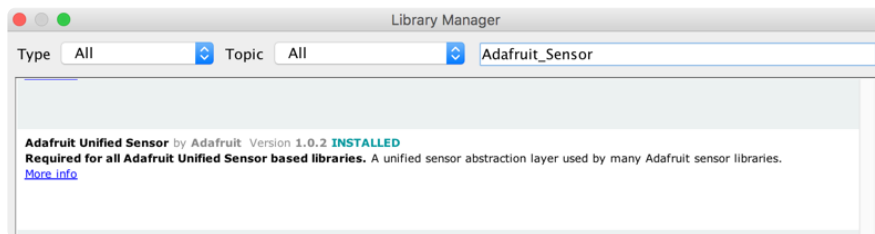
Software

The [Adafruit_BNO055 driver](#) () supports reading raw sensor data, or you can use the [Adafruit Unified Sensor](#) () system to retrieve orientation data in a standard data format.

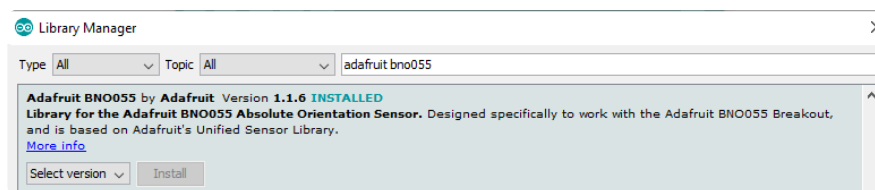
Open up the Arduino library manager:



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 \(\)](http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use)

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 \(\)](#) ([.getEvent \(\)](#), [.getSensor \(\)](#), 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);

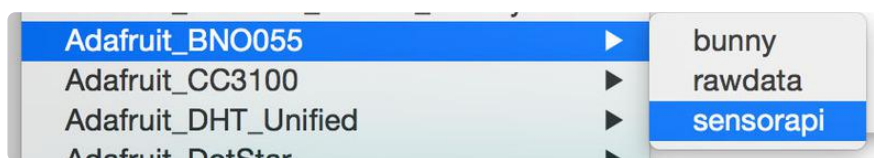
  /* Display the floating point data */
  Serial.print("X: ");
  Serial.print(event.orientation.x, 4);
  Serial.print("\tY: ");
  Serial.print(event.orientation.y, 4);
  Serial.print("\tZ: ");
  Serial.print(event.orientation.z, 4);
  Serial.println("");

  delay(100);
}

```

'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:

```
/dev/tty.usbmodem141121
X: 359.8125 Y: 11.8125 Z: 56.0000
X: 359.4375 Y: 13.8750 Z: 45.1250
X: 359.1875 Y: 14.3750 Z: 35.6250
X: 357.7500 Y: 14.6875 Z: 29.6250
X: 355.6250 Y: 13.3750 Z: 31.7500
X: 352.8625 Y: 13.8625 Z: 44.0000
X: 349.7500 Y: 13.6250 Z: 49.5000
X: 348.8125 Y: 14.1250 Z: 56.2500
X: 347.1250 Y: 13.8125 Z: 72.7500
X: 337.9375 Y: 16.5625 Z: 99.4375
X: 325.0000 Y: 20.6250 Z: 119.7500
X: 320.6250 Y: 22.4375 Z: 124.1875
X: 319.5000 Y: 23.6250 Z: 124.0000
X: 325.6875 Y: 23.3125 Z: 116.1250
X: 333.5625 Y: 21.0000 Z: 107.7500
X: 344.8625 Y: 15.5625 Z: 97.8125
X: 358.9375 Y: 5.0625 Z: 87.1875
X: 15.6250 Y: -10.7500 Z: 85.5625
X: 35.6250 Y: -28.8750 Z: 71.8125
X: 43.0000 Y: -35.5625 Z: 56.7500
X: 43.8750 Y: -36.5625 Z: 46.4375
X: 321.2500 Y: -33.8125 Z: 43.6250
X: 317.3125 Y: -29.2500 Z: 38.0625
X: 312.8125 Y: -23.5625 Z: 37.6250
X: 304.6875 Y: -14.5000 Z: 43.9375
X: 296.1875 Y: -4.0000 Z: 54.0000
X: 290.3125 Y: 1.1250 Z: 67.4375
X: 285.5625 Y: 4.3750 Z: 75.0000
X: 283.7500 Y: 5.3125 Z: 80.0625
X: 283.0000 Y: 5.0000 Z: 84.9375
X: 284.2500 Y: 3.0625 Z: 86.6875
X: 287.6875 Y: 0.1875 Z: 80.5625
X: 291.5000 Y: -3.2500 Z: 73.4375
X: 294.5000 Y: -6.2500 Z: 65.8125
```

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.y(), 4);
Serial.print(" qY: ");
Serial.print(quat.x(), 4);
Serial.print(" qZ: ");
Serial.print(quat.z(), 4);
Serial.println("");
```

.getTemp(void)

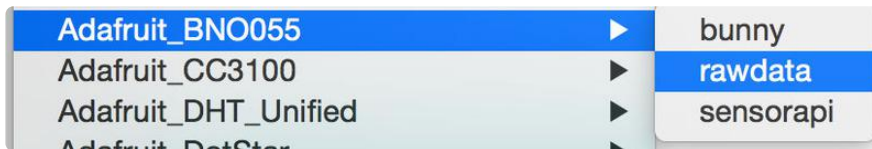
The .getTemp helper returns the current ambient temperature in degrees celsius, and can be read via the following function call:

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

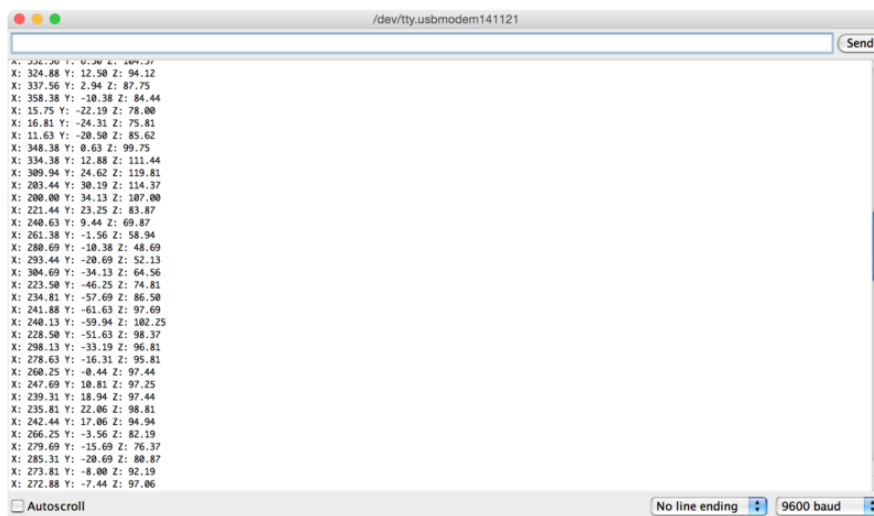
Serial.print("Current Temperature: ");
Serial.print(temp);
Serial.println(" C");
Serial.println("");
```

'rawdata' Example

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



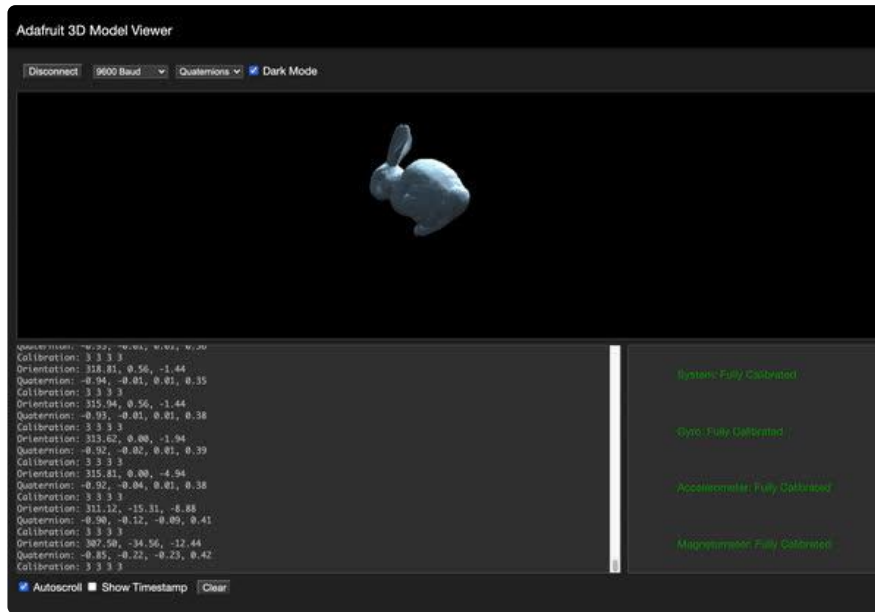
This should produce the following output on the Serial Monitor:



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  
// - VECTOR_MAGNETOMETER - uT  
// - VECTOR_GYROSCOPE - rad/s  
// - VECTOR_EULER - degrees  
// - VECTOR_LINEARACCEL - m/s^2  
// - VECTOR_GRAVITY - m/s^2  
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("");
```

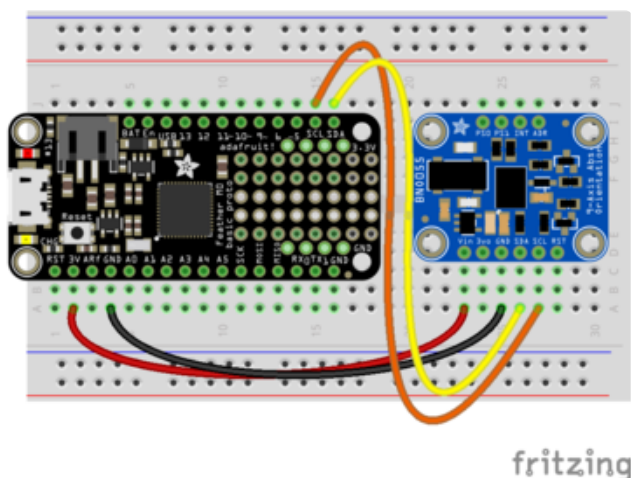
WebSerial Visualizer



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! \(\)](#)

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

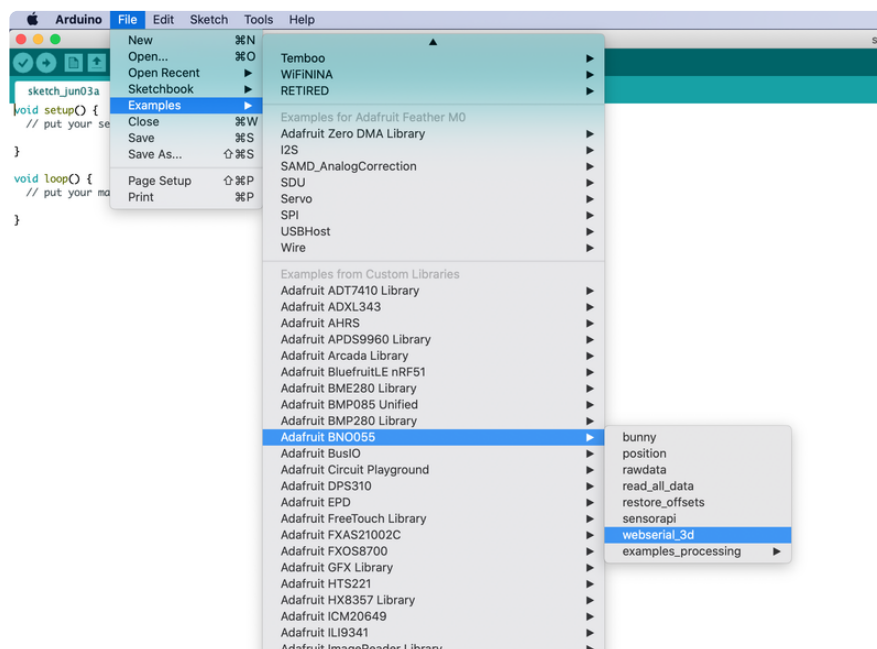
First wire up a BNO055 to your board exactly as shown on the previous pages using the I2C interface. Here's an example of wiring a Feather M0 to the sensor with I2C:



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

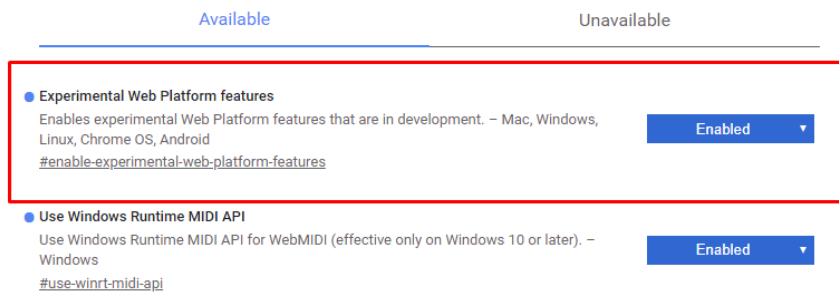
[Start by installing the Chrome browser if you haven't yet. \(\)](#)

Step 4 - Enable Web Serial API if necessary

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



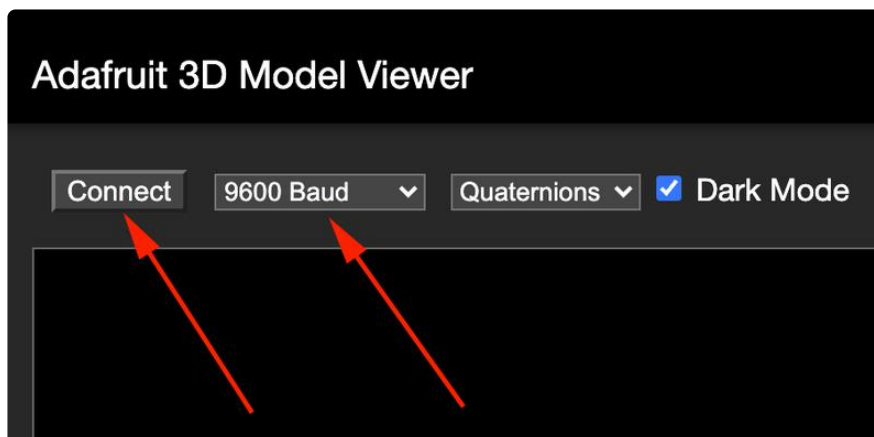
Restart Chrome

Step 5 - Visit the Adafruit 3D Model viewer

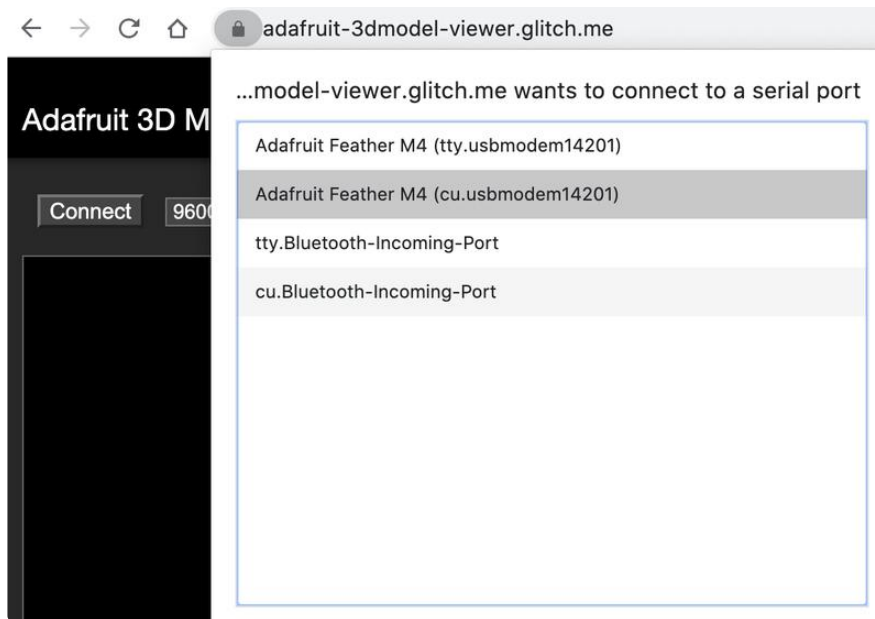
In Chrome, visit https://adafruit.github.io/Adafruit_WebSerial_3DModelViewer/ ()

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.

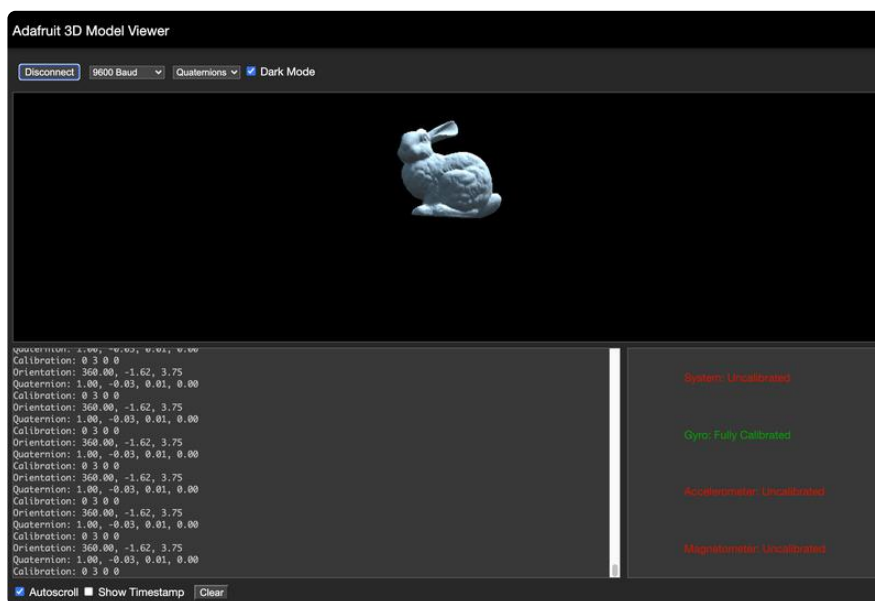
Click Connect



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.

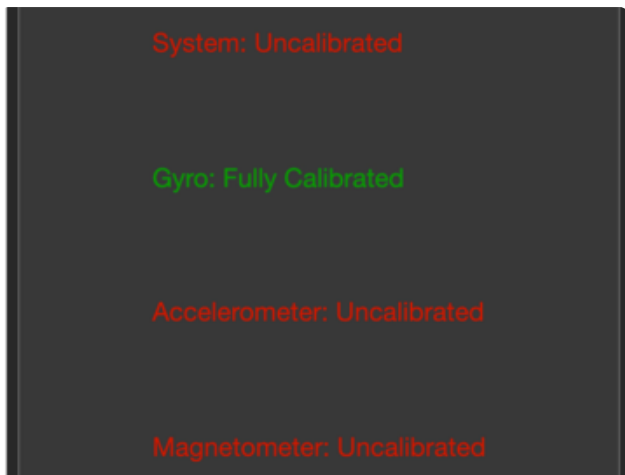


You'll see the serial port monitor on the bottom and a 3D bunny on the top. Try rotating and twisting the sensor to see it move!



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.



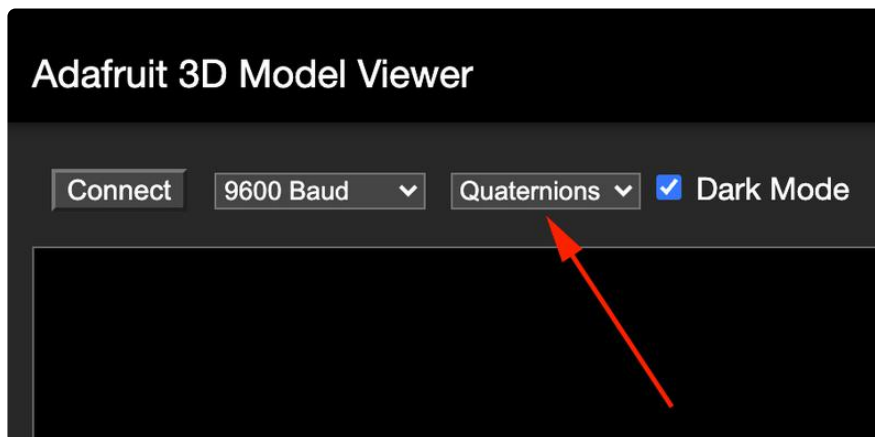
When you first connect, you'll see that most of the calibration registers show as Uncalibrated.



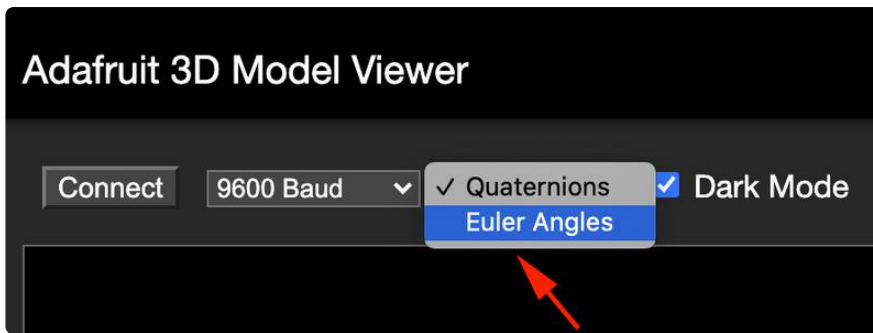
Once you have gone through the calibration steps, you will see that they are all fully calibrated.

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.

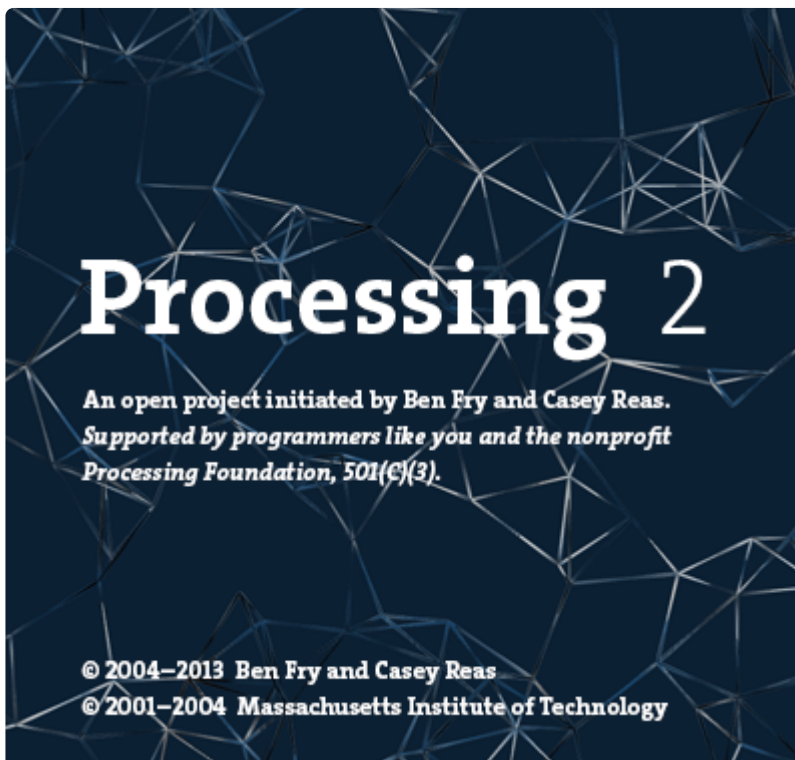


Try playing around with both by moving the bunny around and see if you can see the differences!

Processing Test

We don't recommend using processing for visualization, as its not easy. Check the previous page for how to use a Chrome browser

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

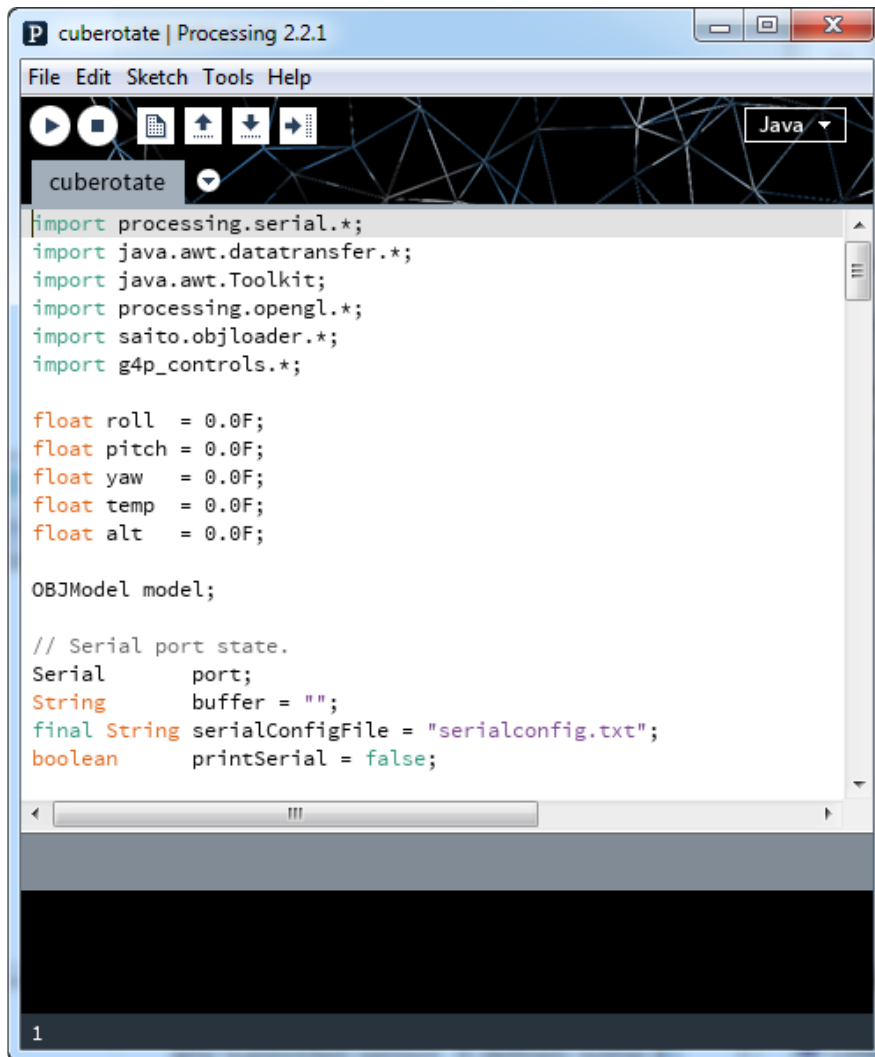
- [Processing 2.x \(\)](#)
 - Note that you can try later Processing versions like 3.0+ too. On some platforms Processing 2.2.1 has issues with supporting 3D acceleration (you might see 'NoClassDefFoundError: processing/awt/PGraphicsJava2D' errors). In those cases grab the later Processing 3.0+ release and use it instead of 2.x.
- [Saito's OBJ Loader \(\)](#) library for Processing (included as part of the Adafruit repo since Google Code is now 'End of Life').
- [G4P GUI library \(\)](#) for Processing ([download the latest version here \(\)](#) 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:



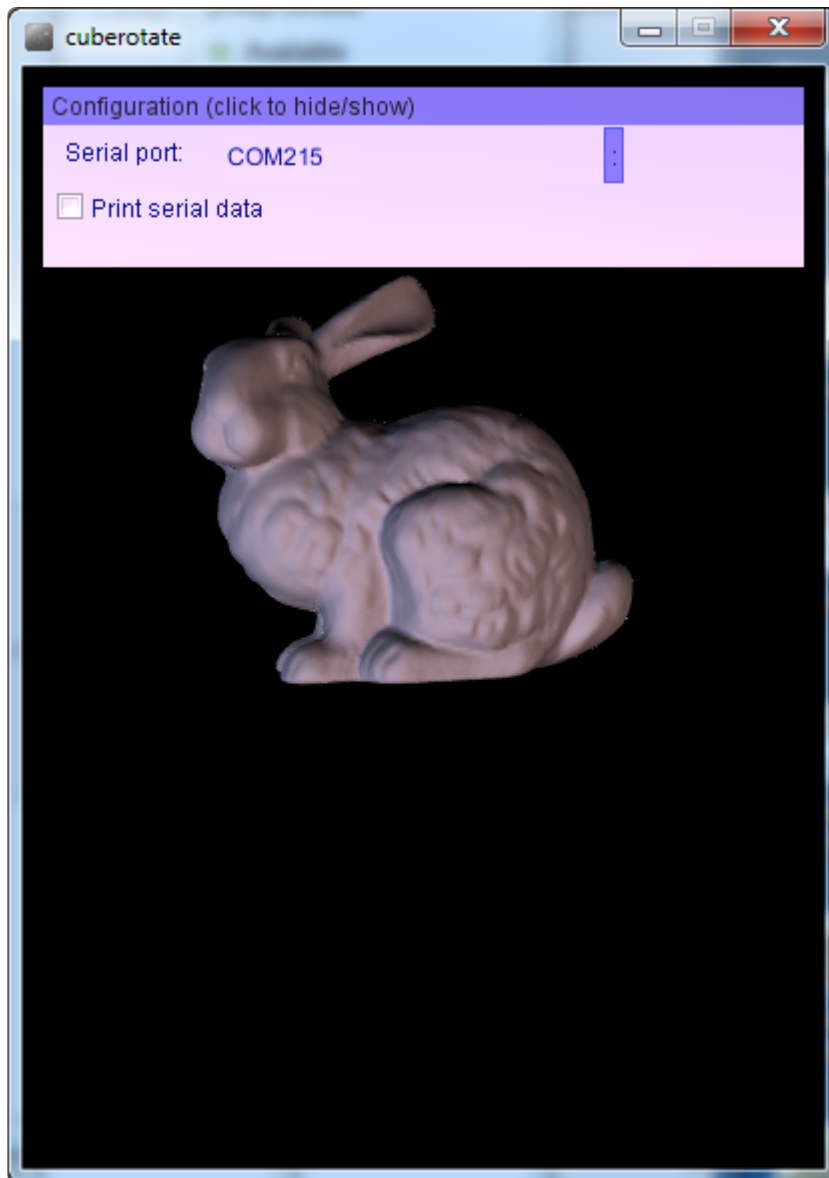
Run the Bunny Sketch on the Uno

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.

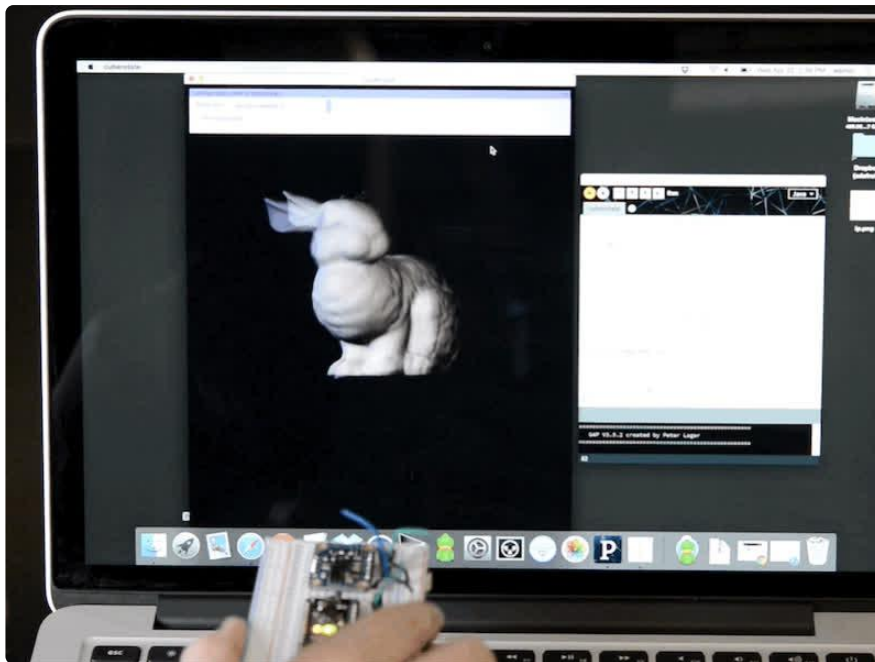
Rabbit Disco!

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 \(\)](#).

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 \(\)](#) 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);
}

```

Interpreting 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).

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!

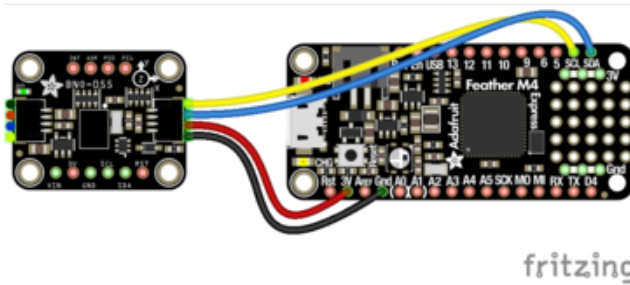
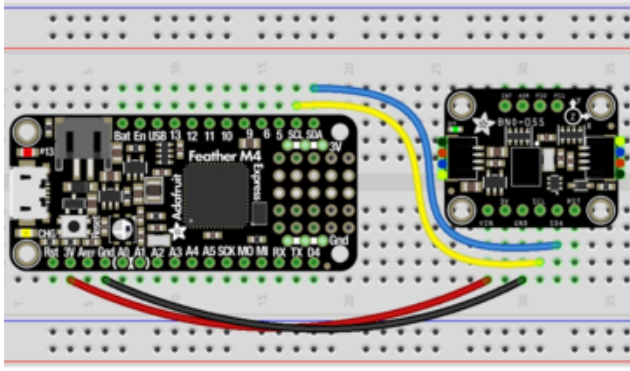
Python & CircuitPython

It's easy to use the BNO055 sensor with Python and CircuitPython, and the [Adafruit CircuitPython BNO055 \(\)](#) library. This library allows you to easily write Python code that reads the acceleration and orientation of the sensor.

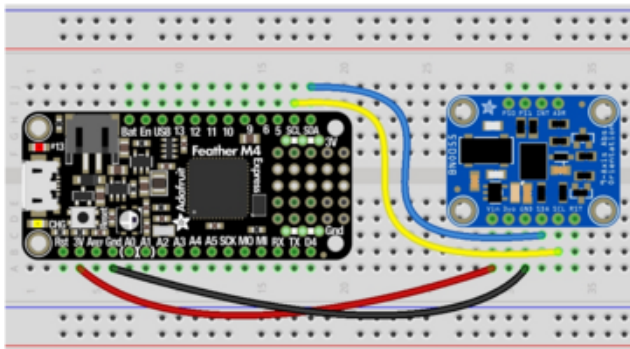
You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python [thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library \(\)](#).

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:

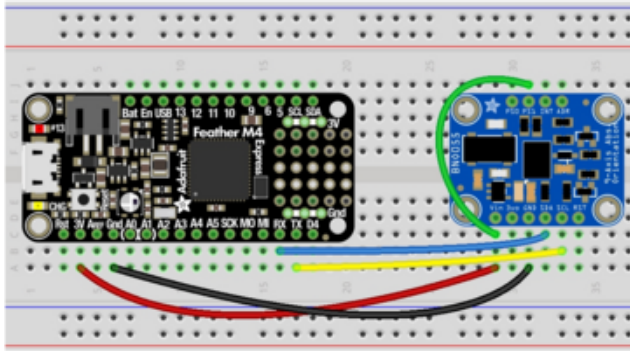


- Board 3V to sensor VIN (red wire for STEMMA QT)
- Board GND to sensor GND (black wire for STEMMA QT)
- Board SCL to sensor SCL (yellow wire for STEMMA QT)
- Board SDA to sensor SDA (blue wire for STEMMA QT)



CircuitPython Microcontroller Wiring - UART

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

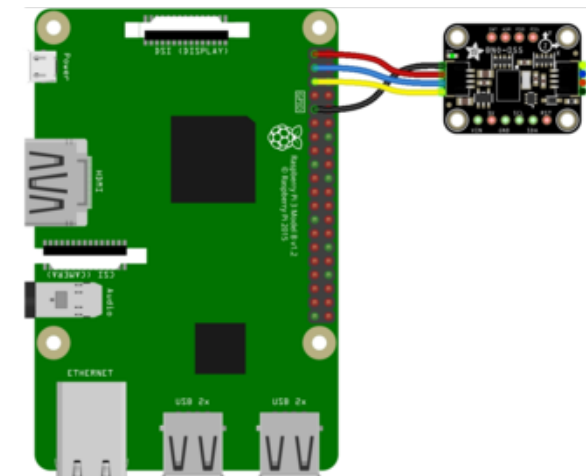
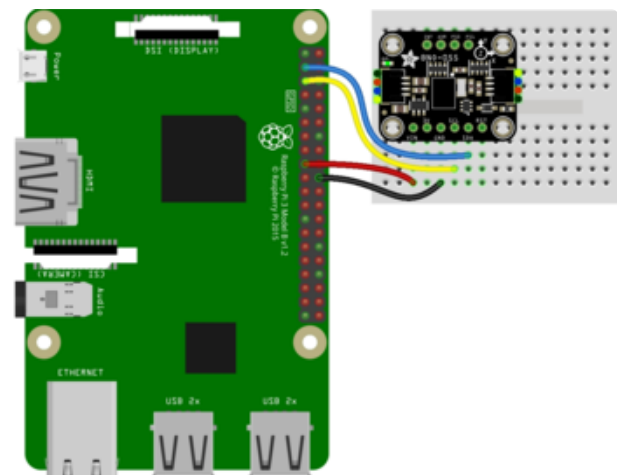
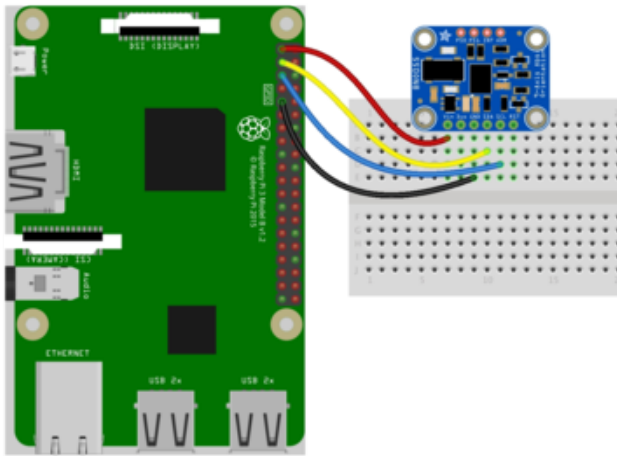


Board 3V to sensor VIN
Board GND to sensor GND
Board TX to sensor SCL
Board RX to sensor SDA
sensor PS1 to sensor VIN

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 \(\)](#).

Here's the Raspberry Pi wired with I2C:

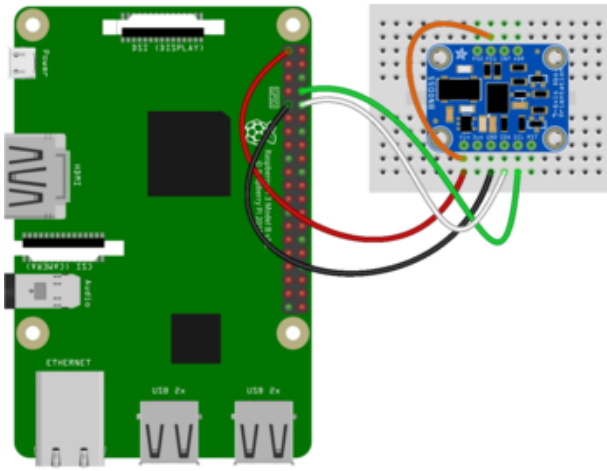


- Pi 3V3 to sensor VIN (red wire for STEMMA QT)
- Pi GND to sensor GND (black wire for STEMMA QT)
- Pi SCL to sensor SCL (yellow wire for STEMMA QT)
- Pi SDA to sensor SDA (blue wire for STEMMA QT)

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

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 \(\)](#) library on your CircuitPython board.

First make sure you are running the [latest version of Adafruit CircuitPython \(\)](#) 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 \(\)](#). For example the Circuit Playground Express guide has [a great page on how to install the library bundle \(\)](#) 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 \(\)](#) so you are at the CircuitPython `>>>` prompt.

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 \(\)!](#)

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

- `sudo 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!

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! \(\)](#)

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 your 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 your 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
import adafruit_bno055

i2c = board.I2C() # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C() # For using the built-in STEMMMA QT connector on a
# microcontroller
sensor = adafruit_bno055.BNO055_I2C(i2c)

# If you are going to use UART uncomment these lines
# uart = board.UART()
# sensor = adafruit_bno055.BNO055_UART(uart)

last_val = 0xFFFF

def temperature():
    global last_val # pylint: disable=global-statement
    result = sensor.temperature
    if abs(result - last_val) == 128:
        result = sensor.temperature
        if abs(result - last_val) == 128:
            return 0b00111111 & result
    last_val = result
    return result
```

```
while True:
    print("Temperature: {} degrees C".format(sensor.temperature))
    """
    print(
        "Temperature: {} degrees C".format(temperature())
    ) # Uncomment if using a Raspberry Pi
    """
    print("Accelerometer (m/s^2): {}".format(sensor.acceleration))
    print("Magnetometer (microteslas): {}".format(sensor.magnetic))
    print("Gyroscope (rad/sec): {}".format(sensor.gyro))
    print("Euler angle: {}".format(sensor.euler))
    print("Quaternion: {}".format(sensor.quaternion))
    print("Linear acceleration (m/s^2): {}".format(sensor.linear_acceleration))
    print("Gravity (m/s^2): {}".format(sensor.gravity))
    print()

    time.sleep(1)
```

Python Docs

[Python Docs \(\)](#)

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 library, you'll need to install the [flask Python web framework \(\)](#).

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 \(\)](#) on your computer or laptop. I recommend and have tested the code for this project with the latest version of [Chrome \(\)](#).

Download the WebGL Example

The example can be found in the library repo [here](#) (). 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_BN0055.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
```

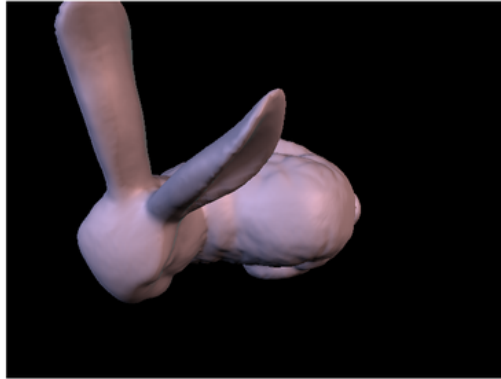
You should see text like the following after the server starts running:

```
* 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/> () might work, or on a BeagleBone Black <http://beaglebone:5000/> () is the URL to try. If neither URL works you'll need to [look up the IP address of your device](#) () 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/> ().

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

Adafruit BNO055 Absolute Orientation Sensor Demo



Orientation (degrees):

Heading = 359.9375

Roll = -0.6875

Pitch = -89.25

Calibration:

(0=uncalibrated, 3=fully calibrated)

System = 0

Gyro = 3

Accelerometer = 0

Magnetometer = 0

Actions:

Model:

Bunny

Straighten

Save Calibration

Load Calibration

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.

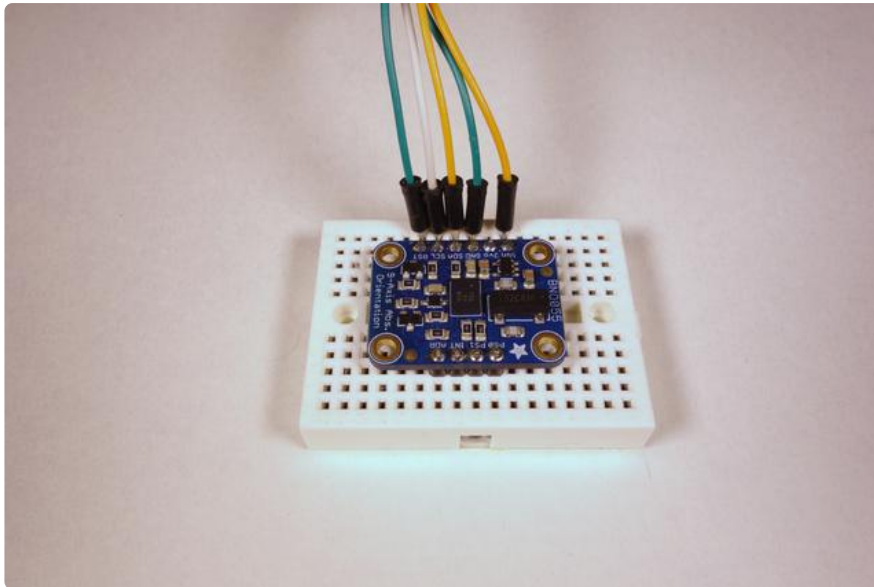
Sensor Calibration

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.

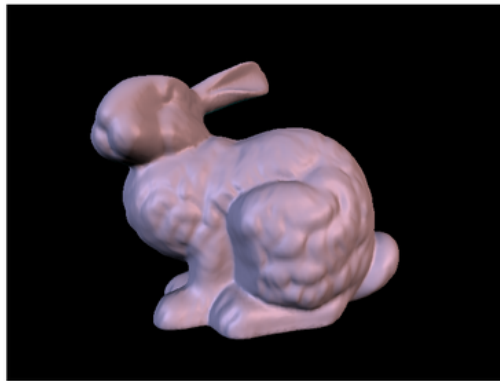
Usage

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:

Adafruit BNO055 Absolute Orientation Sensor Demo



Orientation (degrees):

Heading = 99.5625
Roll = -0.4375
Pitch = -89.375

Calibration:

(0=uncalibrated, 3=fully calibrated)
System = 3
Gyro = 3
Accelerometer = 3
Magnetometer = 3

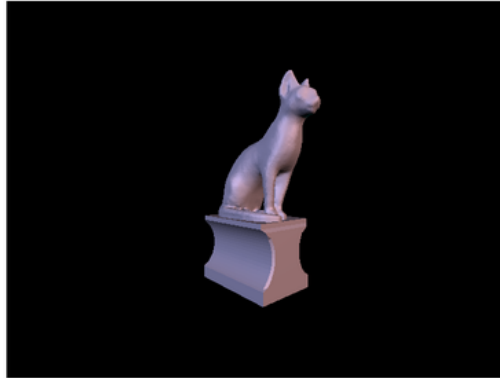
Actions:

Model:
Bunny
[Straighten](#)
[Save Calibration](#)
[Load Calibration](#)

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:

Adafruit BNO055 Absolute Orientation Sensor Demo



Orientation (degrees):

Heading = 39.8125

Roll = -0.375

Pitch = -89.625

Calibration:

(0=uncalibrated, 3=fully calibrated)

System = 3

Gyro = 3

Accelerometer = 0

Magnetometer = 3

Actions:

Model:

Cat Statue

Straighten

Save Calibration

Load Calibration

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 \(\)](#): 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 \(\)](#): 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 \(\)](#) 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 \(\)](#) that isn't necessary for simple apps like this demo).
- [Three.js \(\)](#): This is the JavaScript library that handles all the 3D model rendering.
- [Bootstrap \(\)](#) & [jQuery \(\)](#): 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!

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://github.com/adafruit/Adafruit_BN...ffsets.ino) ()

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://www.bosch-sensortec.com/en/home...1/bno055_4) ()

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://www.adafruit.com/datasheets/BST...000_12.pdf) ()

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.

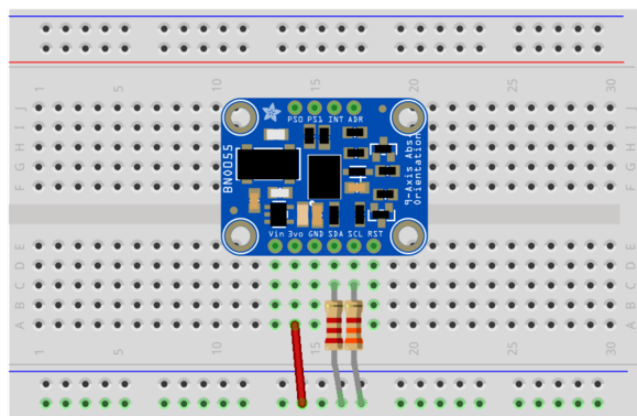
For absolute orientation, quaternions should always be used, and they can be converted to Euler angles at the last moment via the `.toEuler()` helper function in [quaternion.h](#) ().

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. ()

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

Downloads

Files

- [Arduino Library](#) ()
- [EagleCAD PCB files on GitHub](#) ()
- [BNO055 Stemma 3D models on GitHub](#) ()
- [BNO055 Datasheet](#) ()
- [Fritzing object in the Adafruit Fritzing Library](#) ()

Pre-Compiled Bunny Rotate Binaries

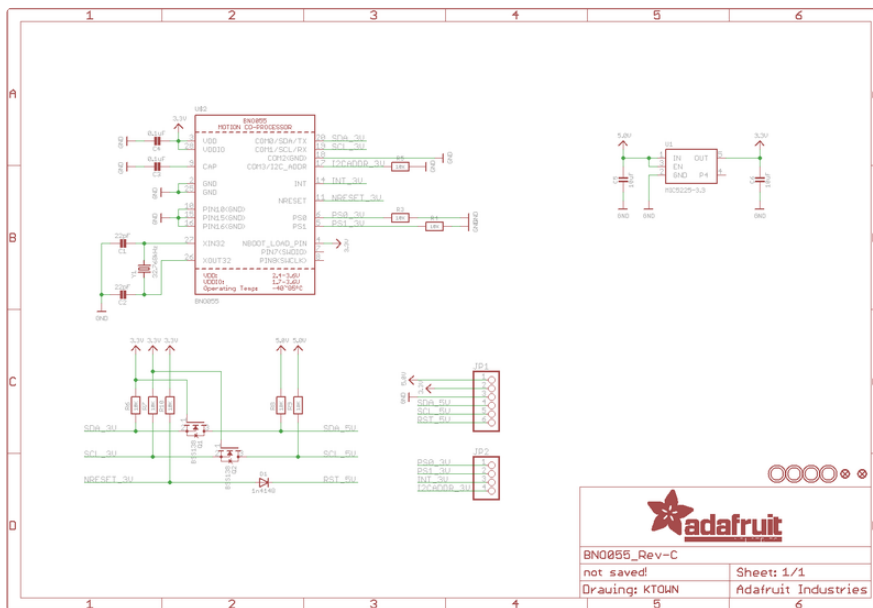
The following binary images can be used in place of running the Processing Sketch, and may help avoid the frequent API and plugin changes.

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

[cuberotate.app.zip](#)

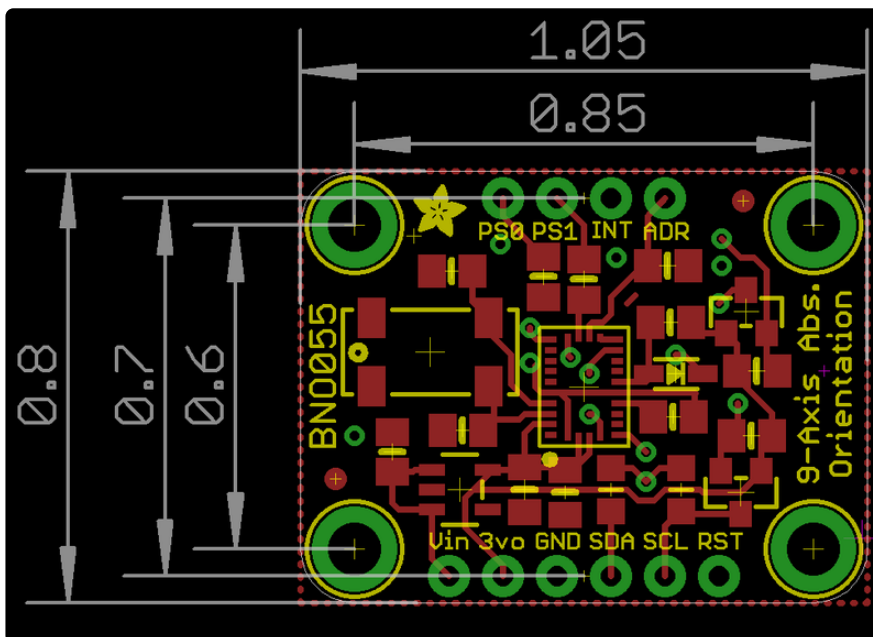
Schematic

The latest version of the Adafruit BNO055 breakout can be seen below (click the image to view the schematic in full resolution):

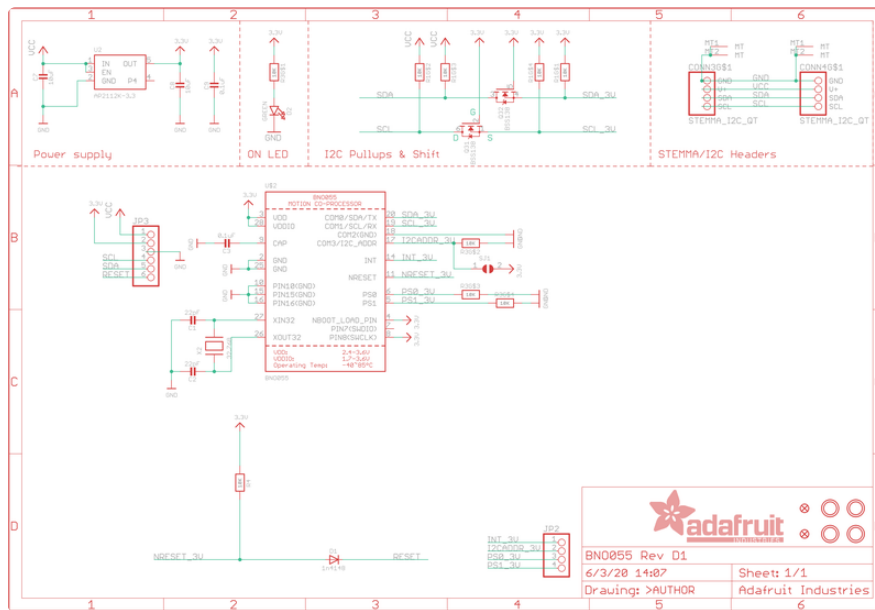


Board Dimensions

The BNO055 breakout has the following dimensions (in inches):



Schematic for STEMMA QT



Fab Print for STEMMA QT

