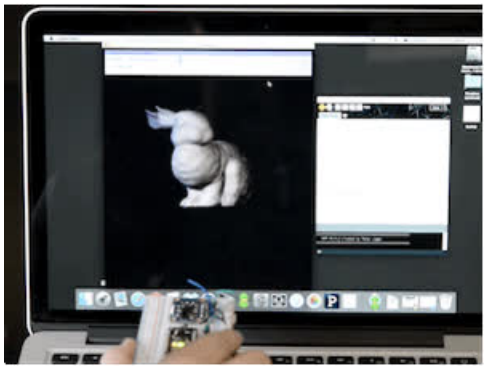
adafruit learning system

Adafruit BNO055 Absolute Orientation Sensor

Created by Kevin Townsend



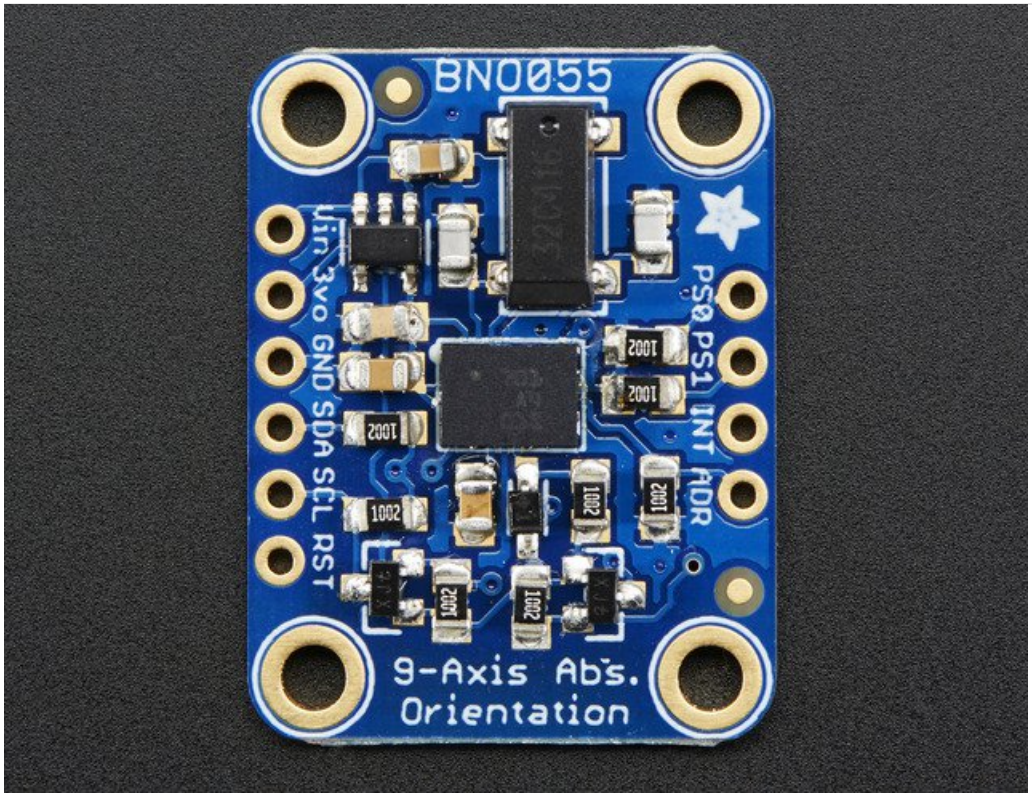
Last updated on 2018-04-29 11:32:36 PM UTC

Guide Contents

Guide Contents 2
Overview 4
Data Output 4
Related Resources 5
Pinouts 6
Power Pins 6
I2C Pins 6
Other Pins 6
Assembly 7
Prepare the header strip: 7
Add the breakout board: 8
And Solder! 8
Arduino Code 11
Wiring for Arduino 11
Software 11
Download the Driver from Github 11
Download Adafruit_Sensor 12
Adafruit Unified Sensor System 12
'sensorapi' Example 13
Raw Sensor Data 14
.getVector (adafruit_vector_type_t vector_type) 14
.getQuat(void) 15
.getTemp(void) 15
'rawdata' Example 15
Processing Test 17
Requirements 17
Opening the Processing Sketch 17
Run the Bunny Sketch on the Uno 18
Rabbit Disco! 18
Device Calibration 21
Interpretting Data 21
Generating Calibration Data 22
Persisting Calibration Data 22
Bosch Video 22
CircuitPython Code 23
Usage 24
FAQs 26
Can I manually set the calibration constants? 26
Does the device make any assumptions about its initial orientation? 26
Why doesn't Euler output seem to match the Quaternion output? 26
I'm sometimes losing data over I2C, what can I do about this? 26
I have some high frequency (> 2MHz) wires running near the BNO055 and I'm getting unusual results/hanging behavior 27

Downloads 28
Files 28
Pre-Compiled Bunny Rotate Binaries 28
Schematic 28
Board Dimensions 28

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.

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:

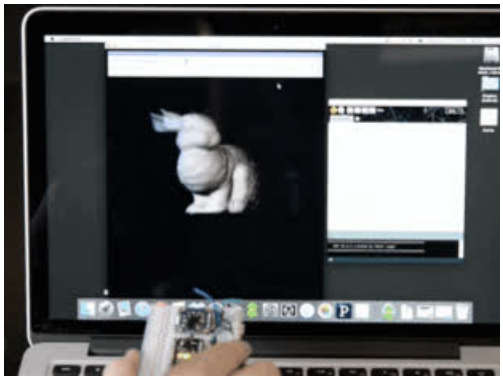
- **Absolute Orientation** (Euler Vector, 100Hz)
  Three axis orientation data based on a 360° sphere
- **Absolute Orientation** (Quaterion, 100Hz)
  Four point quaternion output for more accurate data manipulation
- **Angular Velocity Vector** (100Hz)
  Three axis of 'rotation speed' in rad/s
- **Acceleration Vector** (100Hz)

Three axis of acceleration (gravity + linear motion) in m/s^2

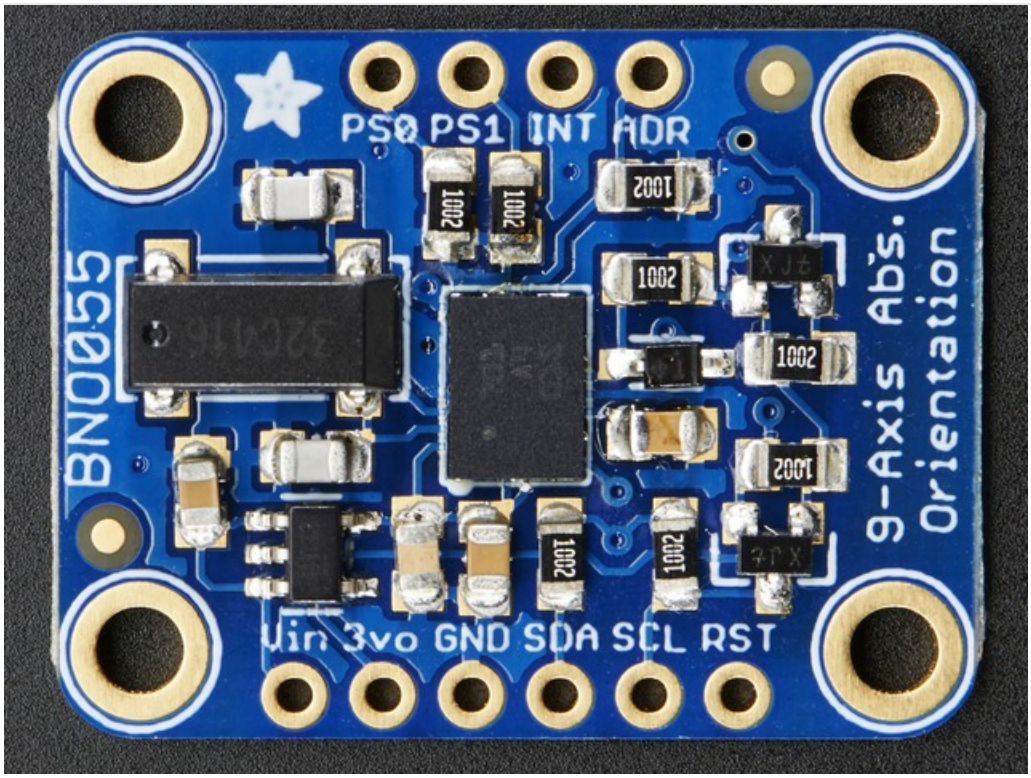
- **Magnetic Field Strength Vector** (20Hz)
  Three axis of magnetic field sensing in micro Tesla (uT)
- **Linear Acceleration Vector** (100Hz)
  Three axis of linear acceleration data (acceleration minus gravity) in m/s^2
- **Gravity Vector** (100Hz)
  Three axis of gravitational acceleration (minus any movement) in m/s^2
- **Temperature** (1Hz)
  Ambient temperature in degrees celsius

Related Resources

- Datasheet
- Adafruit BNO055 Library (Github)



Pinouts



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

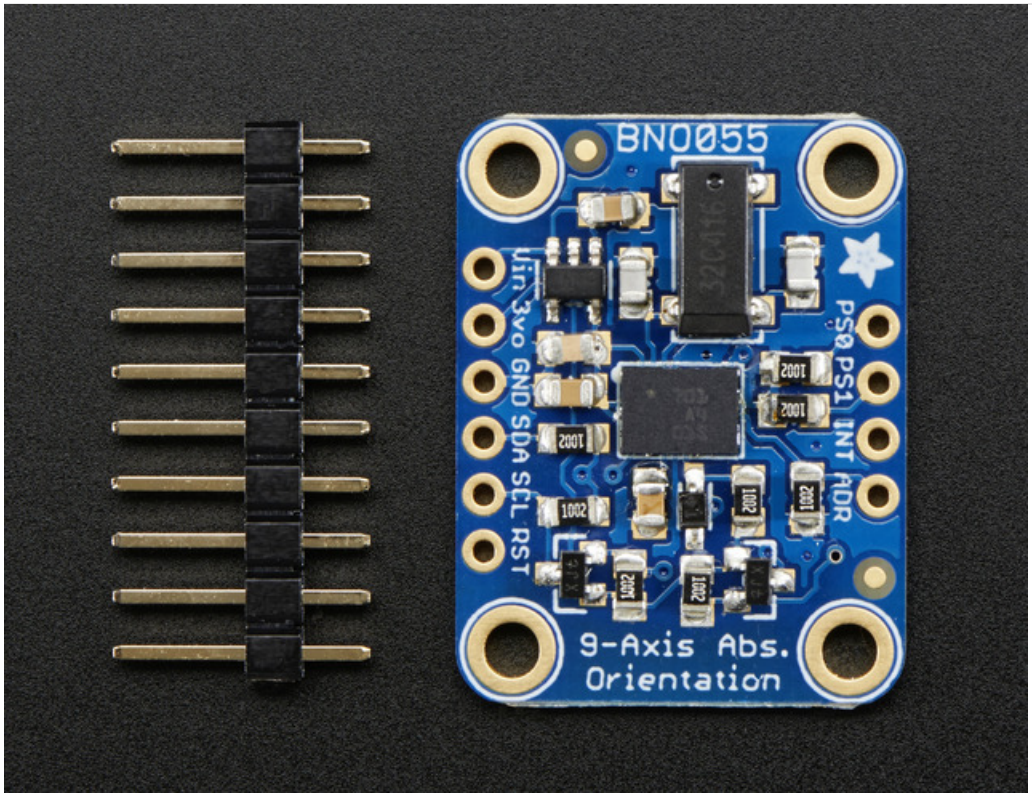
I2C Pins

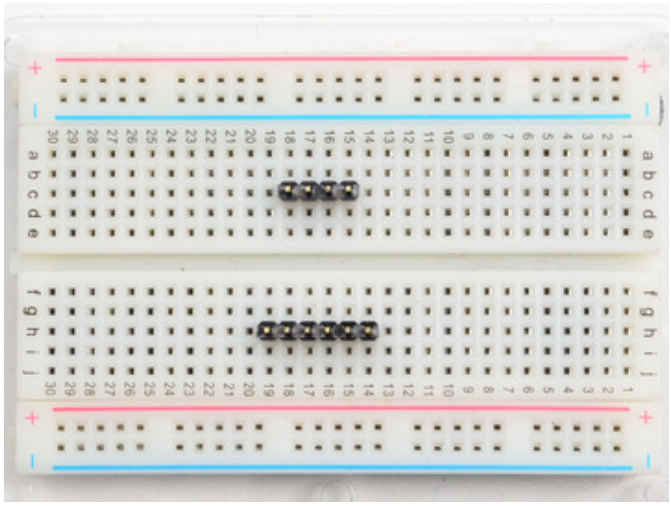
- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this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

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.

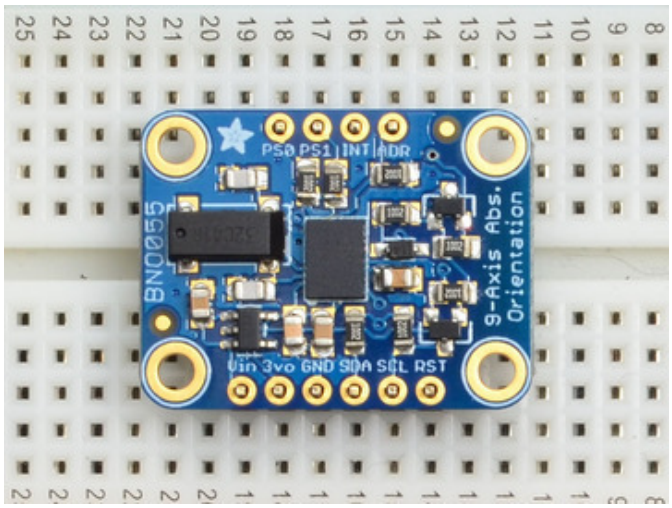
Assembly





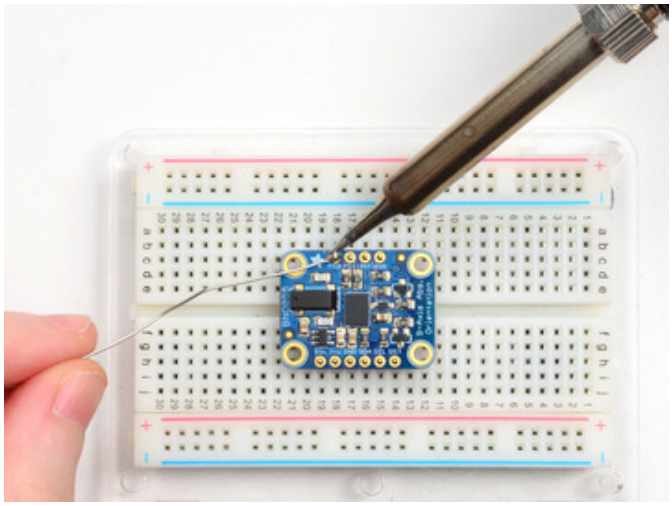
Prepare the header strip:

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

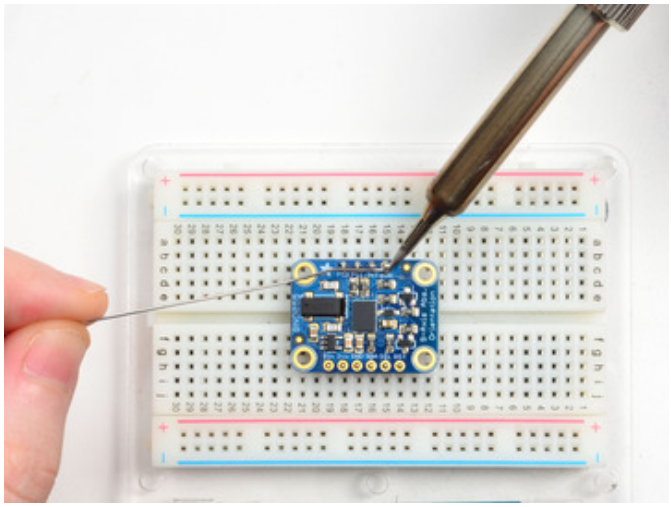


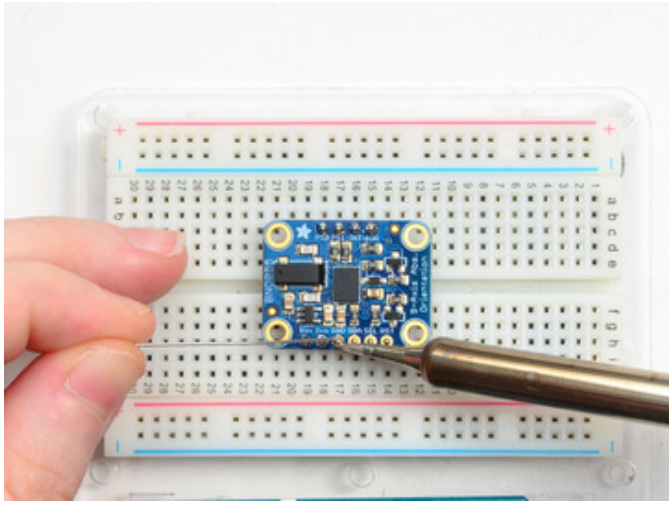
And Solder!

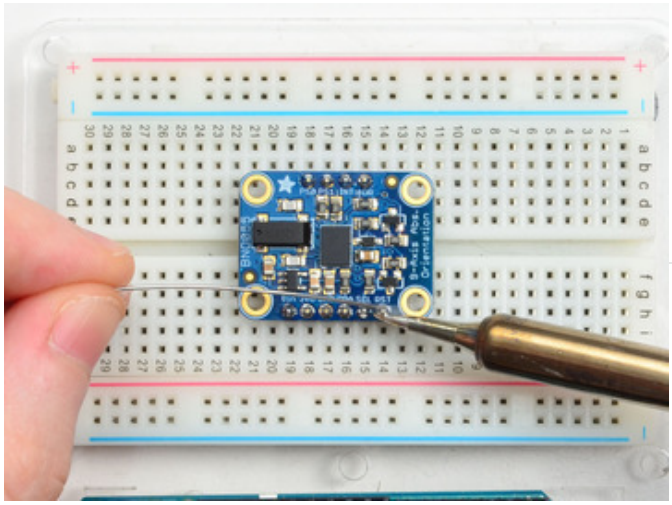
Be sure to solder all pins for reliable electrical contact.

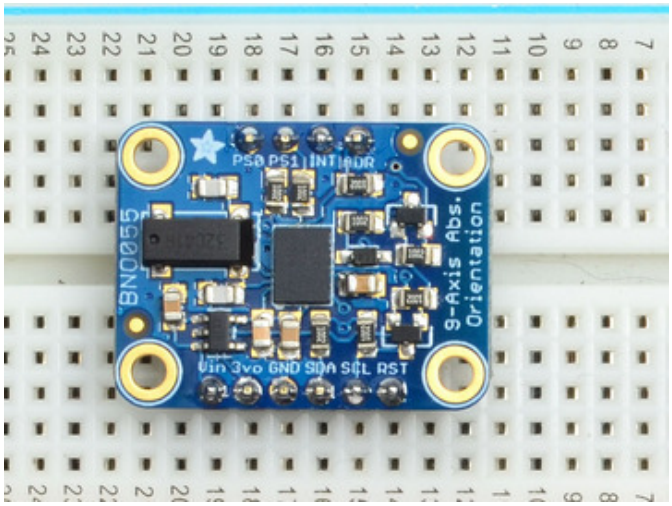
Solder the longer power/data strip first

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









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

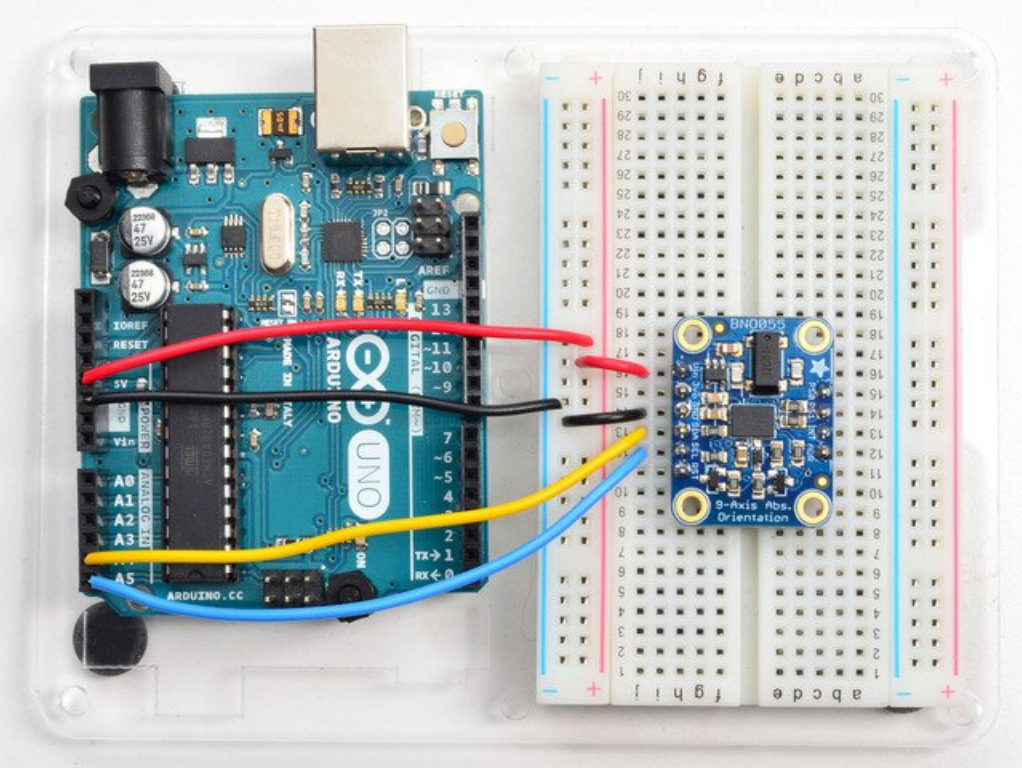
Arduino Code

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 below.

- Connect **Vin** 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** to common power/data ground
- Connect the **SCL** 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** 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.

Download the Driver from Github

To begin controling the motor chip, you will need to download the Adafruit_BNO055 Library from our github

repository. You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download Adafruit_BNO055 from Github

https://adafru.it/f0K

Rename the uncompressed folder **Adafruit_BNO055** and check that the **Adafruit_BNO055** folder contains **Adafruit_BNO055.cpp** and **Adafruit_BNO055.h**

Place the **Adafruit_BNO055** library folder your ***arduinosketchfolder*/libraries/** folder.
You may need to create the **libraries** subfolder if its your first library. Restart the IDE.

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

Download Adafruit_Sensor

We also have a core sensor library that helps manage sensor readings. So, just like the BNO055 library, download Adafruit_Sensor

Install just like you did with **Adafruit_BNO055**

Download Adafruit_Sensor

https://adafru.it/cMO

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.

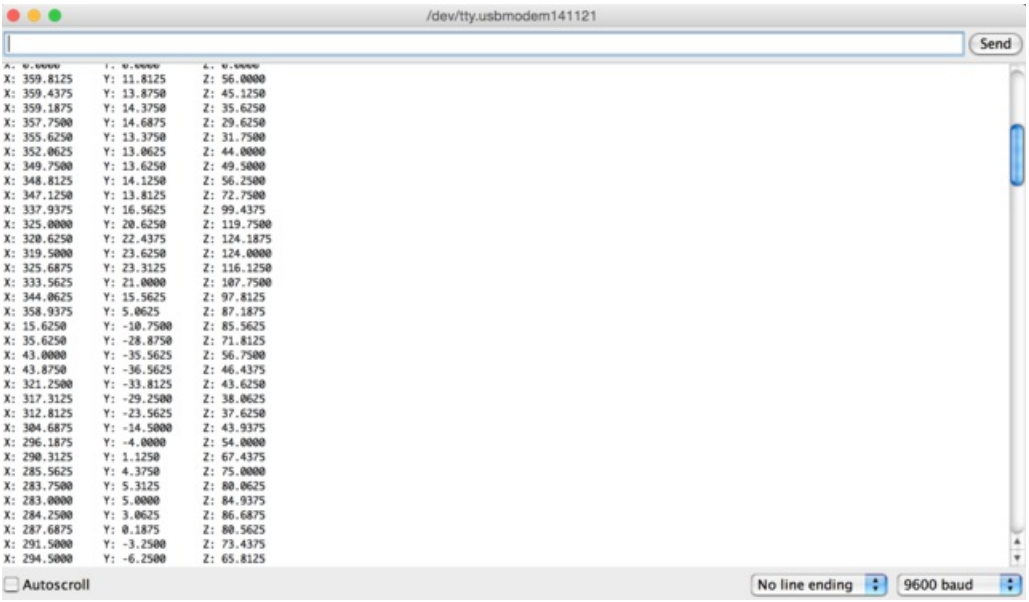
For example, the following code snippet shows the core of what is needed to start reading data using the Unified Sensor System:

```
#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:



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("");
```

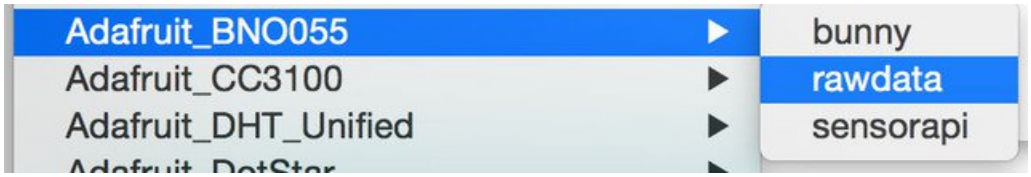
.getTemp(void)

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

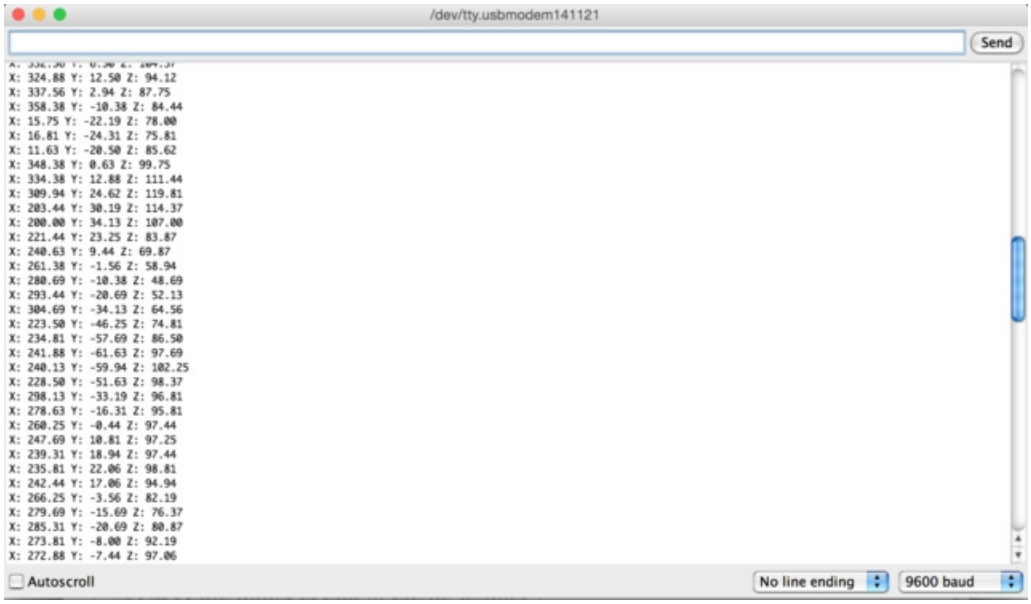
```
/* Display the current temperature */
int8_t temp = bno.getTemp();

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

'rawdata' Example

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



This should produce the following output on the Serial Monitor:

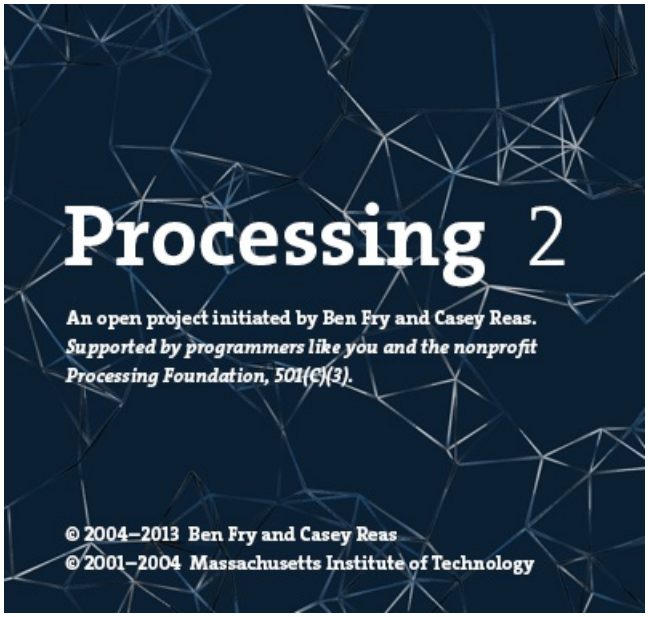


By default, the sketch generates **Euler angle** absolute orientation data, but you can easily modify the data displayed by changing the value provided to **.getVector** below:

```
// Possible vector values can be:
// - VECTOR_ACCELEROMETER - m/s^2
// - 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("");
```

Processing Test

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

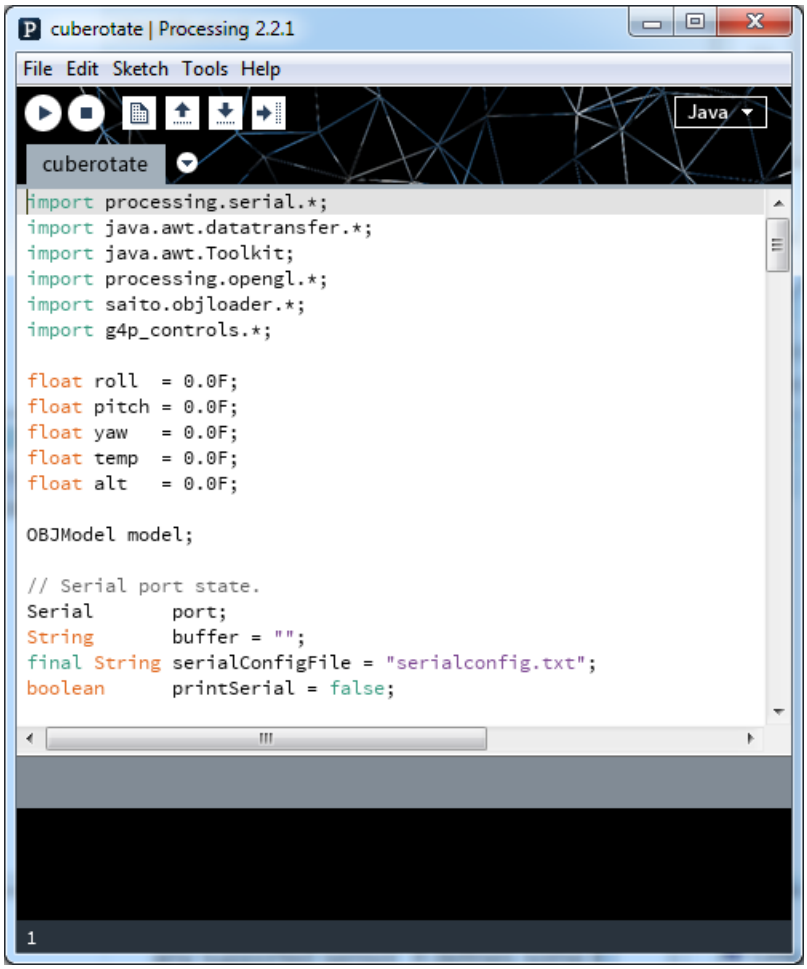
- Processing 2.x
    - Note that you can try later Processing versions like 3.0+ too. On some platforms Processing 2.2.1 has issues with supporting 3D acceleration (you might see 'NoClassDefFoundError: processing/awt/PGraphicsJava2D' errors). In those cases grab the later Processing 3.0+ release and 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:



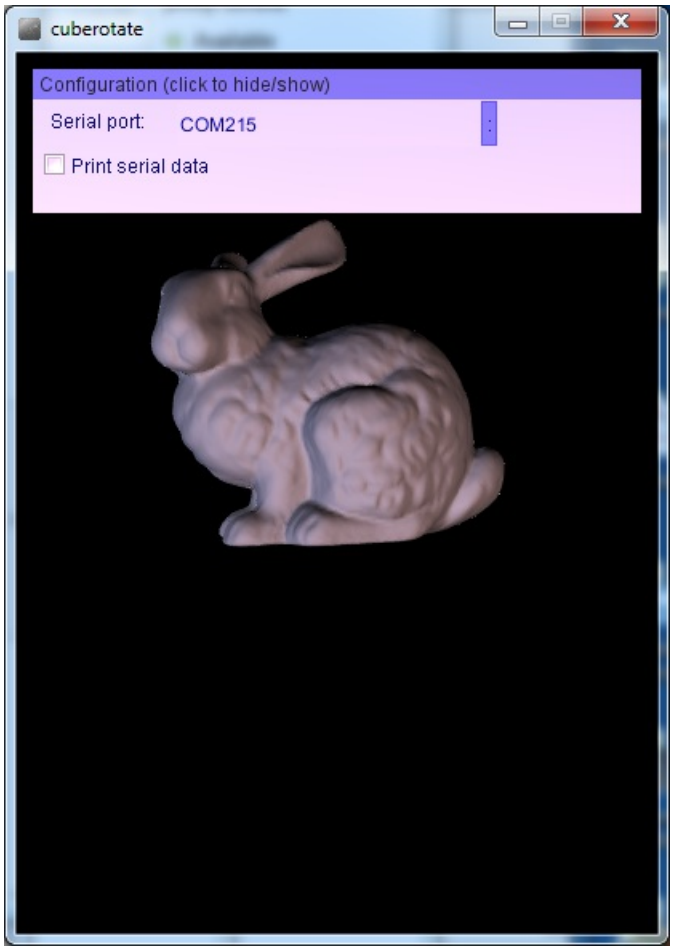
Run the Bunny Sketch on the Uno

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.

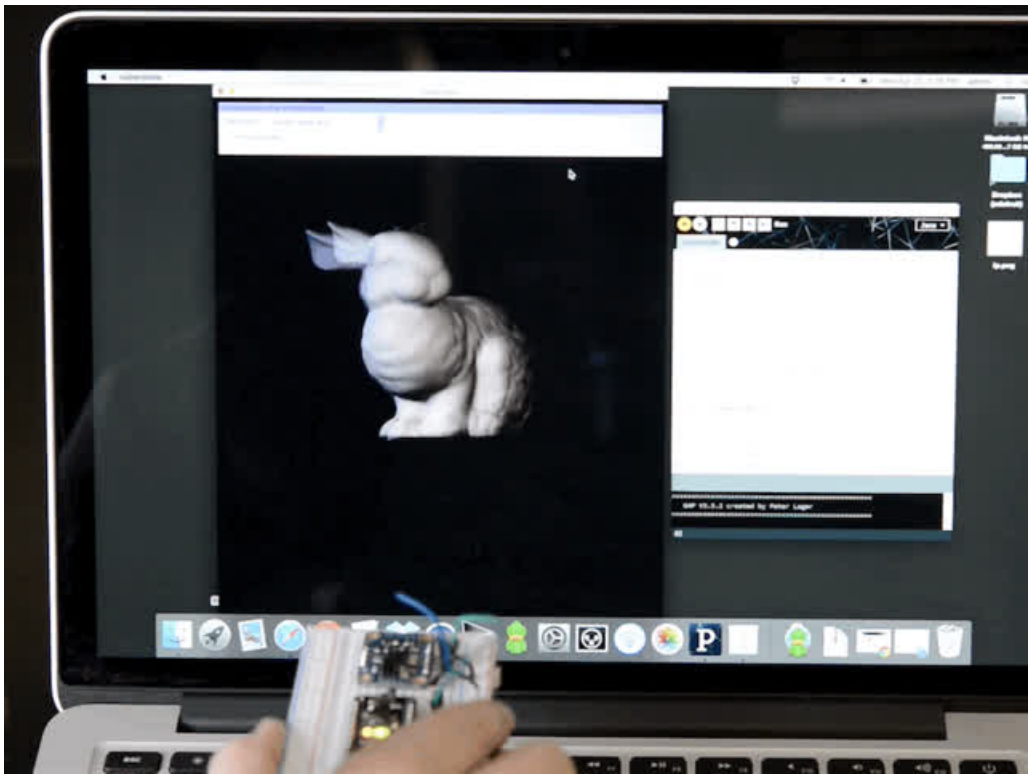
Rabbit Disco!

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);
}
```

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).

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.

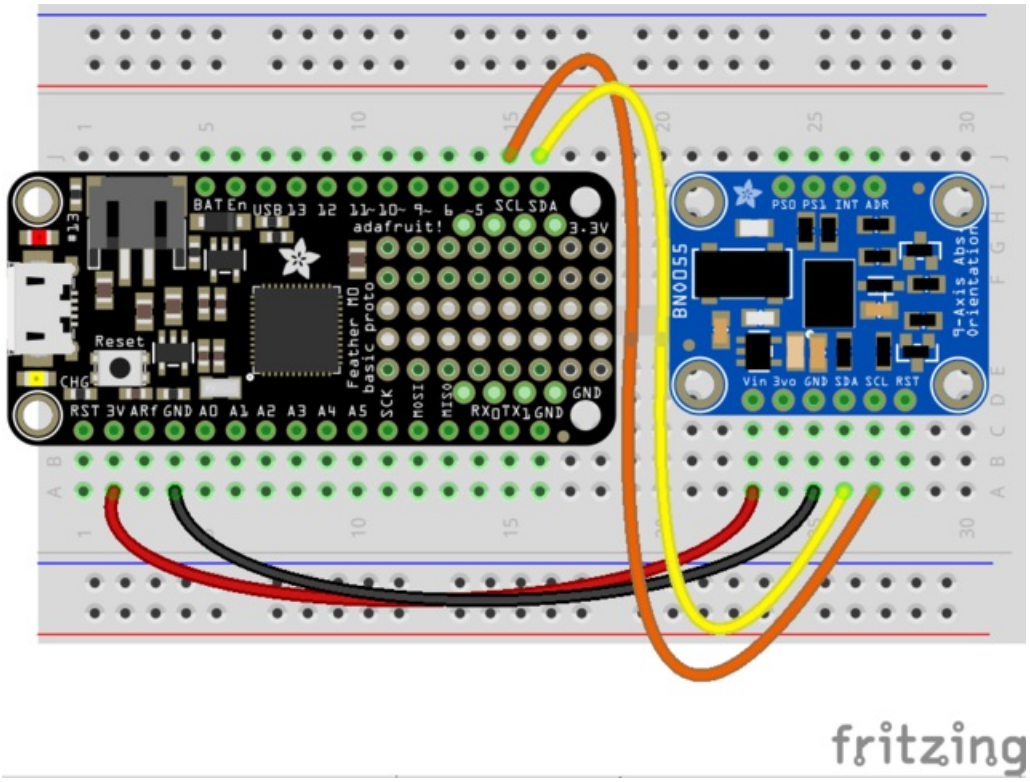
Bosch Video

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

CircuitPython Code

It's easy to use the BNO055 sensor with 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.

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 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**

Next you'll need to install the Adafruit CircuitPython BNO055 library on your CircuitPython board. **Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!**

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 both express and non-express boards.

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries from the bundle:

- **adafruit_bno055.mpy**
- **adafruit_bus_device**
- **adafruit_register**

You can also download the **adafruit_bno055.mpy** from its releases page on Github.

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.

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. First initialize the I2C bus and create an instance of the sensor by running:

```
import board
import busio
import adafruit_bno055
i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_bno055.BNO055(i2c)
```

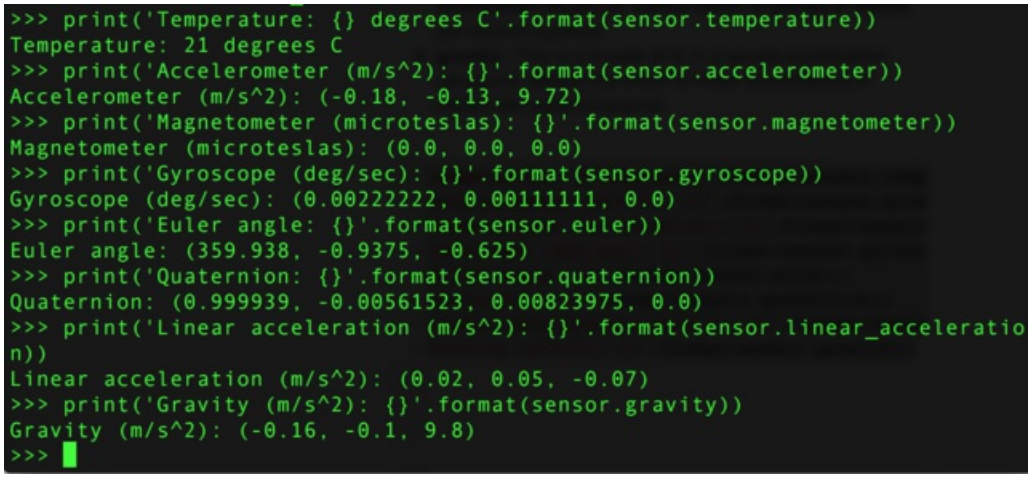
Remember if you're using a board that doesn't support hardware I2C (like the ESP8266) you need to use the **bitbangio** module instead:

```
import board
import bitbangio
import adafruit_bno055
i2c = bitbangio.I2C(board.SCL, board.SDA)
sensor = adafruit_bno055.BNO055(i2c)
```

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

- **temperature** - The sensor temperature in degrees Celsius.
- **accelerometer** - This is a 3-tuple of X, Y, Z axis accelerometer values in meters per second squared.
- **magnetometer** - This is a 3-tuple of X, Y, Z axis magnetometer values in microteslas.
- **gyroscope** - 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))
print('Accelerometer (m/s^2): {}'.format(sensor.accelerometer))
print('Magnetometer (microteslas): {}'.format(sensor.magnetometer))
print('Gyroscope (deg/sec): {}'.format(sensor.gyroscope))
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))
```



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

Here's a complete example that prints the acceleration and orientation (as Euler angles) every second. Save this as **main.py** on your board and look for the output in the serial REPL. Remember if using the ESP8266 and**bitbangio** module you need to change the initialization as mentioned above!

```
import board
import busio
import time

import adafruit_bno055


# Initialize I2C and the sensor.
i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_bno055.BNO055(i2c)

# Main loop runs forever printing acceleration and Euler angles every second.
while True:
    print('Accelerometer (m/s^2): {}'.format(sensor.accelerometer))
    print('Euler angle: {}'.format(sensor.euler))
    time.sleep(1.0)
```

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

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

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

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?

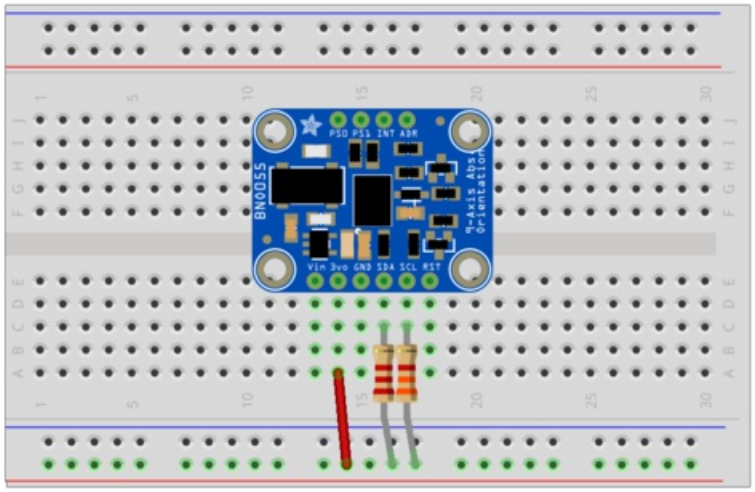
The Euler angles coming out of the chip are based on 'automatic orientation detection', which has the drawback of not being continuous for all angles and situations.

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.

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 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.

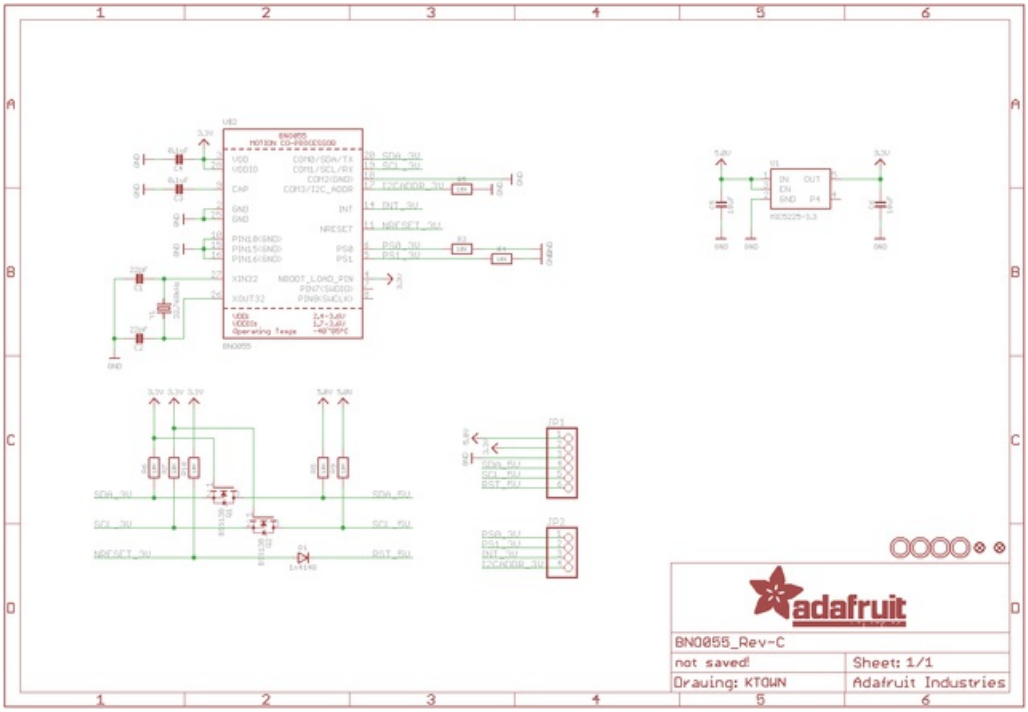
For OS X download **cuberotate.app.zip**, which was built on OS X 10.11.6 based on Processing 2.2.1:

cuberotate.app.zip

https://adafru.it/wB4

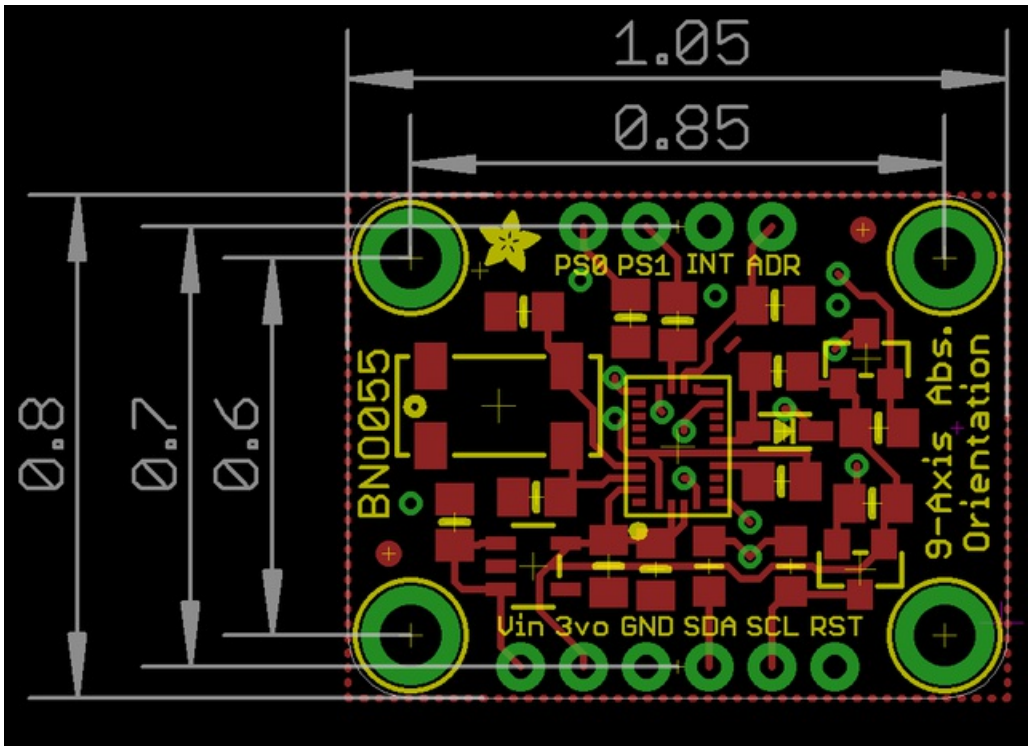
Schematic

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):


1.05
0.85
0.8
0.7
0.6
PS0 PS1 INT ADR
BNO055
9-Axis Abs. Orientation
Vin 3vo GND SDA SCL RST