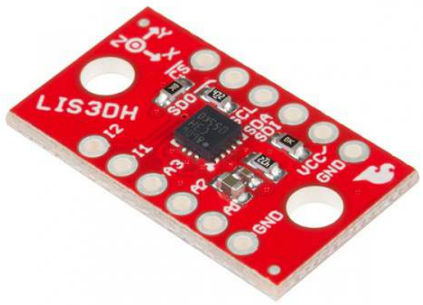
LIS3DH Hookup Guide

Introduction

The LIS3DH is a triple axis accelerometer you can use to add translation detection to your project. It would be classified as a 3DoF, or 3 Degrees of Freedom. Inertial Measurement Units (or IMUs), such as the LSM9DS1; the LSM6DS3; or the LSM303C, can provide additional space location data such as gyroscopic or magnetometric. This IC operates under the same principals but gives a few analog inputs to play with, and it has some built in movement detection abilities.



SparkFun Triple Axis Accelerometer Breakout - LIS3DH

SEN-13963

This guide presents the basics of plugging it into a processor, using the Arduino library to get acceleration data live or by FIFO collection, and describes the library usage.

Required Materials

To follow along, you'll need the following materials:

- LIS3DH Breakout Board
- Arduino UNO, RedBoard, or another Arduino-compatible board
- Straight Male Headers – Or wire. Something to connect between the breakout and a breadboard.
- Breadboard – Any size (even mini) should do.

- M/M Jumper Wires – To connect between Arduino and breadboard.

The LIS3DH is a 3.3V device! Supplying voltages greater than ~3.6V can permanently damage the IC. As long as your Arduino has a 3.3V supply output, and you're OK with using I²C; you shouldn't need any extra level shifting. But, if you want to use SPI, you may need a level shifter.

A logic level shifter is required for any 5V-operating Arduino (UNO, RedBoard, Leonardo, etc). If you use a 3.3V-based 'duino – like the Arduino Pro 3.3V or 3.3V Pro Mini – there is no need for level shifting.

Suggested Reading

If you're not familiar with some of the concepts below, we recommend checking out that tutorial before continuing on.

- Accelerometer Basics
- Gyroscopes
- Serial Peripheral Interface (SPI)
- Inter-IC Communication (I²C)
- Logic Levels
- Bi-Directional Level Shifter Hookup Guide

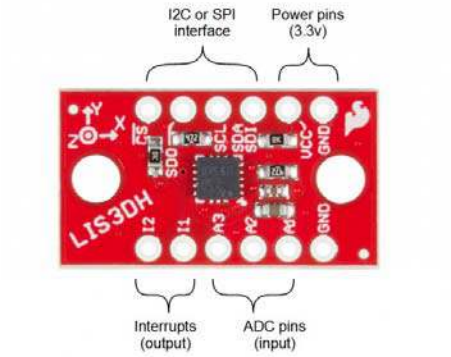
Also, the following ST documents are helpful for advanced users:

- LIS3DH_Datasheet – Hardware information and register map.
- LIS3DH_AppNote – Descriptive material showing basic usage.

Hardware Overview and Assembly

There are a few different methods with which you can use the LIS3DH.

The top side of the board has the LIS3DH sensor, some bypass caps and pull-up resistors.



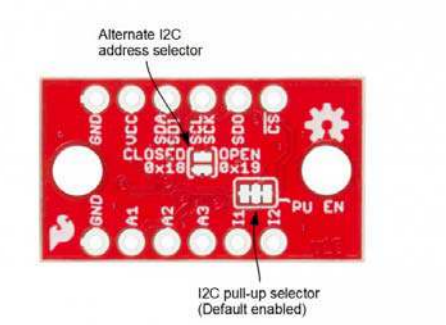
The pin connections

This table gives more information as to each pins functionality. The serial port can be connected as either SPI or I2C, and it uses the same physical pins for both. To get going, just wire up your choice of interface, supply 3.3v, and ground. Note that you will not need to use all the pins no matter which communication method you choose.

				Connection	
Group	**Name**	**Direction**	**Description**	**I2C**	**SPI**
Serial	!CS	I	Chip select (for SPI)	NC	!CS

	SDO	O	Data output (MISO for SPI)	NC	MISO
	SCL	I	Data clock	SCL	SCK
	SDA/SDI	I/O	Data in (SDA for I2C, MOSI for SPI)	SDA	MOSI
Interrupts	I1	O	Primary int has FIFO + motion	Optional MCU	
	I2	O	Secondary int has motion	Optional MCU	
ADC	A1	I	Analog in	Optional	
	A2	I	Analog in	Optional	
	A3	I	Analog in (unused for temp readings)	Optional	
Power	VCC	I	3.3V input	Supply	
	GND	I	Ground connection (either PTH)	Supply	

On the bottom, there are two jumpers that correspond to the I2C address and pull-up enable.



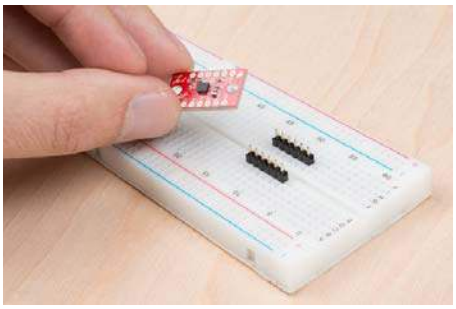
The following options are available:

- **The I2C Address Jumper** – Bridge to use alternate address 0x18, otherwise leave open for 0x19. Leave open for SPI use.
- **The I2C Pull-up Enable** – Closed by default, this connects a pull-up resistor between the I2C lines and VCC. This generally doesn't interfere with SPI operation, but, if less power consumption is required, carefully cut the copper traces.

Working with a Breadboard

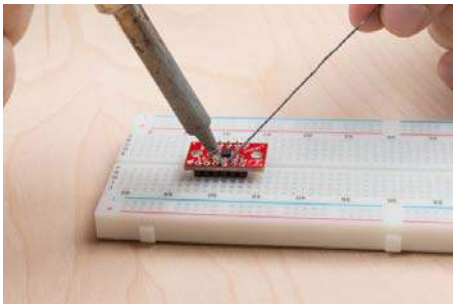
This sensor works nicely with a breadboard for easy connection, and, because it gives some mass to the accelerometer, it more closely matches what might be expected from a project or cellphone.

To add headers, break off two 6-pin lengths of 0.1 inch male headers, and set them into a breadboard to use as a soldering jig.



Two rows of headers placed and ready to solder.

Drop the breakout board onto the pins, and solder down the rows.



Soldering on the rows of pins.

Congratulations! You're now ready to connect the sensor to a microcontroller of your choosing.

Getting the Arduino Library

The examples in the guide use the Arduino IDE and a RedBoard to communicate with the LIS3DH.

To get the Arduino library, download from Github, or use the Arduino Library Manager.

Download the Github Repository

Visit the GitHub repository to download the most recent version of the library, or click the link below:

DOWNLOAD THE ARDUINO LIBRARY

Library Manager

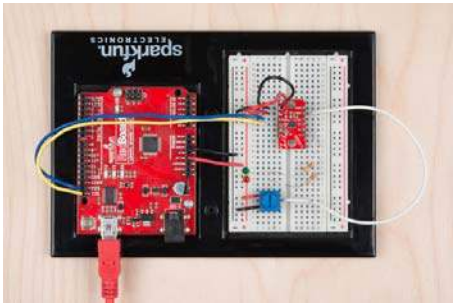
For help installing the library, check out our How To Install An Arduino Library tutorial.

If you don't end up using the manger, you'll need to move the *SparkFun_LIS3DH_Arduino_Library* folder into a *libraries* folder within your Arduino sketchbook.

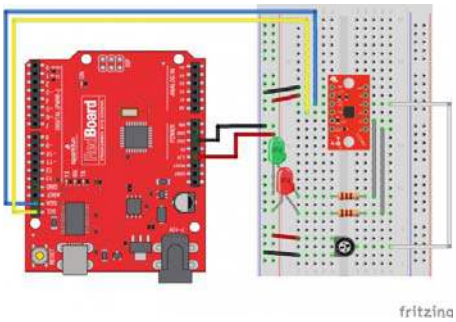
Example: I2C, analog, and interrupts

The first circuit allows a RedBoard to talk to the LIS3DH over I2C and provides connections on the interrupt and ADC pins. If you don't need them, just connect power, ground, and communication pins, and ignore the interrupt and ADC examples.

Use these two pictures as a guide for building the circuit.



The circuit built on a RedBoard



The connections shown in Fritzing

Basic Accelerometer Data Collection:

Start with just the basic accelerometer sketch, also called “MinimalistExample” from the library. This will periodically samples the sensor and displays data as number of Gs detected. Remember, the vertical axis will read 1G while sitting at rest.

```
#include "SparkFunLIS3DH.h"
#include "Wire.h"
#include "SPI.h"

LIS3DH myIMU; //Default constructor is I2C, addr 0x19.

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(1000); //relax...
  Serial.println("Processor came out of reset.\n");

  //Call .begin() to configure the IMU
  myIMU.begin();

}


void loop()
{
  //Get all parameters
  Serial.print("\nAccelerometer:\n");
  Serial.print(" X = ");
  Serial.println(myIMU.readFloatAccelX(), 4);
  Serial.print(" Y = ");
  Serial.println(myIMU.readFloatAccelY(), 4);
  Serial.print(" Z = ");
  Serial.println(myIMU.readFloatAccelZ(), 4);

  delay(1000);
}
```

Example output:

```
Processor came out of reset.


Accelerometer:
 X = -0.1481
 Y = -0.1361
 Z = 0.9768

Accelerometer:
 X = -0.1481
 Y = -0.1361
 Z = 0.9768

Accelerometer:
 X = -0.1481
 Y = -0.1361
 Z = 0.9768

Accelerometer:
 X = -0.1481
 Y = -0.1361
 Z = 0.9768
```

When run, the sketch will display data in Gs to the serial terminal. Every second, the data is collected and printed.

Using the ADC

To try out the analog inputs, load the example called “ADCUsage”, or copy paste from the following section. This example also shows some of the additional settings that can be applied within the `begin()` function.

```
#include "SparkFunLIS3DH.h"
#include "Wire.h"
#include "SPI.h"

LIS3DH myIMU; //Default constructor is I2C, addr 0x19.

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(1000); //relax...
  Serial.println("Processor came out of reset.\n");

  myIMU.settings.adcEnabled = 1;
  //Note:  By also setting tempEnabled = 1, temperature data i
s available
  //on ADC3.  Temperature *differences* can be read at a rate
of
  //1 degree C per unit of ADC3
  myIMU.settings.tempEnabled = 0;
  myIMU.settings.accelSampleRate = 50;  //Hz.  Can be: 0,1,10,
25,50,100,200,400,1600,5000 Hz
  myIMU.settings.accelRange = 2;      //Max G force readabl
e.  Can be: 2, 4, 8, 16
  myIMU.settings.xAccelEnabled = 0;
  myIMU.settings.yAccelEnabled = 0;
  myIMU.settings.zAccelEnabled = 0;

  //Call .begin() to configure the IMU
  myIMU.begin();

}

void loop()
{
  //Get all parameters
  Serial.print("\nADC:\n");
  Serial.print(" 1 = ");
  Serial.println(myIMU.read10bitADC1());
  Serial.print(" 2 = ");
  Serial.println(myIMU.read10bitADC2());
  Serial.print(" 3 = ");
  Serial.println(myIMU.read10bitADC3());

  delay(300);
}
```

Example output:

```
Processor came out of reset.


ADC:
 1 = 1020
 2 = 522
 3 = 506

ADC:
 1 = 1020
 2 = 544
 3 = 516

ADC:
 1 = 1020
 2 = 540
 3 = 517
```

The sketch prints the three ADC values every 300ms. Move the knob to see how the values change and how the effective voltage range is somewhat in the middle of the full range. Move the wire from on ADC pin to another to see that the controlled value changes.

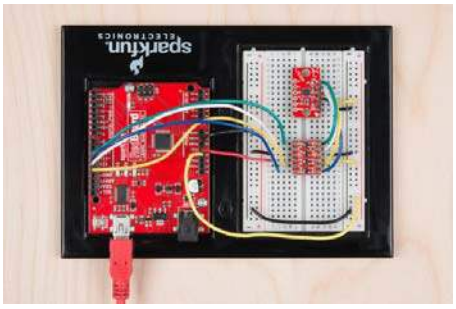
Using the Interrupt Pins

Interrupt behavior is highly configurable and is thus omitted as basic library functions. Instead, LIS3DH registers are directly written in accordance with the datasheet.

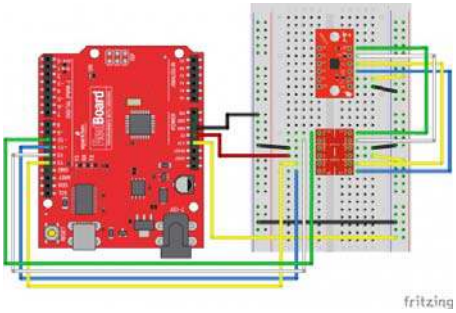
An example is provided that has the relevant registers configured with comments in a template function that can be copied into a project and modified. Run the example named IntUsage, which will throw an interrupt on one pin when an exceeded acceleration is detected and a pulse on the other when a tap is detected.

Example: SPI and FIFO usage

The second method in which to communicate with the LIS3DH is with the SPI interface. The SPI interface operates at 3.3v, so use a logic level converter or a MCU that operates at 3.3V. Use the following pictures to help build the circuit.



The circuit built on a RedBoard



The connections shown in Fritzing

Basic Accelerometer Data Collection:

SPI is not the default configuration, so you'll have to pass extra information to the library by constructing with parameters. Modify "MinimalistExample" by changing `LIS3DH myIMU;` to `LIS3DH myIMU(SPI_MODE, 10);` for SPI mode with the !CS pin connected to pin 10.

The modified "MinimalistExample" is listed here:

```
#include "SparkFunLIS3DH.h"
#include "Wire.h"
#include "SPI.h"

LIS3DH myIMU(SPI_MODE, 10); // constructed with parameters fo
r SPI and cs pin number

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(1000); //relax...
  Serial.println("Processor came out of reset.\n");

  //Call .begin() to configure the IMU
  myIMU.begin();

}


void loop()
{
  //Get all parameters
  Serial.print("\nAccelerometer:\n");
  Serial.print(" X = ");
  Serial.println(myIMU.readFloatAccelX(), 4);
  Serial.print(" Y = ");
  Serial.println(myIMU.readFloatAccelY(), 4);
  Serial.print(" Z = ");
  Serial.println(myIMU.readFloatAccelZ(), 4);

  delay(1000);
}
```

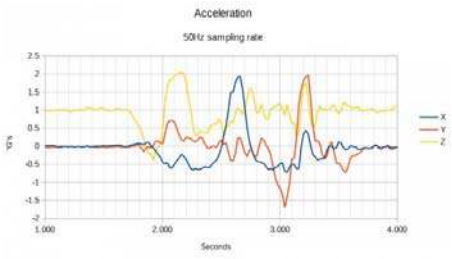
Example output:

```
Processor came out of reset.


Accelerometer:
 X = -0.1481
 Y = -0.1361
 Z = 0.9768

Accelerometer:
 X = -0.1481
 Y = -0.1361
 Z = 0.9768

Accelerometer:
 X = -0.1481
 Y = -0.1361
 Z = 0.9768

Accelerometer:
 X = -0.1481
 Y = -0.1361
 Z = 0.9768
```

When run, the sketch will display data in Gs to the serial terminal. Every second, the data is collected and printed.

FIFO usage:

The SPI bus can operate faster than I2C, so for high speed data collections where periodic sampling is required, SPI is advisable.



This graph was made by taking the output of the example and copy-pasting it into a spreadsheet program, then creating a chart. During the data collection, the sensor was moved about a foot back and forth on each axis.

```
#include "SparkFunLIS3DH.h"
#include "Wire.h"
#include "SPI.h"

LIS3DH myIMU(SPI_MODE, 10); //Constructing with SPI interface
information
//LIS3DH myIMU(I2C_MODE, 0x19); //Alternate constructor for I2
C

uint32_t sampleNumber = 0; //Used to make CSV output row numbe
rs

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(1000); //relax...
  Serial.println("Processor came out of reset.\n");

  myIMU.settings.adcEnabled = 0;
  //Note:  By also setting tempEnabled = 1, temperature data i
s available
  //instead of ADC3 in.  Temperature *differences* can be rea
d at a rate of
  //1 degree C per unit of ADC3 data.
  myIMU.settings.tempEnabled = 0;
  myIMU.settings.accelSampleRate = 10;  //Hz.  Can be: 0,1,10,
25,50,100,200,400,1600,5000 Hz
  myIMU.settings.accelRange = 2;      //Max G force readabl
e.  Can be: 2, 4, 8, 16
  myIMU.settings.xAccelEnabled = 1;
  myIMU.settings.yAccelEnabled = 1;
  myIMU.settings.zAccelEnabled = 1;

  //FIFO control settings
  myIMU.settings.fifoEnabled = 1;
  myIMU.settings.fifoThreshold = 20;  //Can be 0 to 31
  myIMU.settings.fifoMode = 1;  //FIFO mode.
  //fifoMode can be:
  //  0 (Bypass mode, FIFO off)
  //  1 (FIFO mode)
  //  3 (FIFO until full)
  //  4 (FIFO when trigger)

  //Call .begin() to configure the IMU (except for the fifo)
  myIMU.begin();

  Serial.print("Configuring FIFO with no error checking...");
  myIMU.fifoBegin(); //Configure fifo
  Serial.print("Done!\n");

  Serial.print("Clearing out the FIFO...");
  myIMU.fifoClear();
  Serial.print("Done!\n");
  myIMU.fifoStartRec(); //cause fifo to start taking data (re-
applies mode bits)

}

void loop()
{
  //float temp;  //This is to hold read data
  //uint16_t tempUnsigned;
  //
  while(( myIMU.fifoGetStatus() & 0x80 ) == 0) {};  //Wait fo
```

```
r watermark

  //Now loop until FIFO is empty.
  //If having problems with the fifo not restarting after read
ing data, use the watermark
  //bits (b5 to b0) instead.
  //while(( myIMU.fifoGetStatus() & 0x1F ) > 2) //This checks
that there is only a couple entries left
  while(( myIMU.fifoGetStatus() & 0x20 ) == 0) //This checks f
or the 'empty' flag
  {
      Serial.print(sampleNumber);
      Serial.print(",");
      Serial.print(myIMU.readFloatAccelX());
      Serial.print(",");
      Serial.print(myIMU.readFloatAccelY());
      Serial.print(",");
      Serial.print(myIMU.readFloatAccelZ());
      Serial.println();
      sampleNumber++;
  }

}
```

Example output:

```
Processor came out of reset.

Configuring FIFO with no error checking...Done!
Clearing out the FIFO...Done!
0,-0.15,-0.14,1.04
1,-0.17,-0.12,1.02
2,-0.21,-0.10,0.95
3,-0.21,-0.10,1.01
4,-0.22,-0.12,1.07
5,-0.17,-0.12,0.99
6,-0.12,-0.15,0.96
7,-0.18,-0.12,0.94
8,-0.19,-0.10,0.98
9,-0.20,-0.14,1.04
10,-0.19,-0.12,0.99
11,-0.20,-0.10,0.95
12,-0.21,-0.12,1.06
13,-0.14,-0.12,0.98
14,-0.10,-0.11,0.95
15,-0.12,-0.10,0.94
16,-0.14,-0.09,0.90
...
```

Notice that the output produces batches of data periodically. Even though the data waits to be collected, it is still sampled periodically. The data is collected when the FIFO is past the watermark configured in the line `myIMU.settings.fifoThreshold = 20;` .

Extra Examples and Arduino Library Reference

The following examples are included in the Arduino library:

- **ADCUsage** - Demonstrates analog in reads and has notes about temperature collection
- **FifoExample** - Demonstrates using the built-in buffer to burst-collect data - **Good demonstration of settings**

- **FullSettingExample** - Shows all settings, with non-used options commented out
- **IntUsage** - shows configuration of interrupt bits
- **LowLevelExample** - Demonstrates using only the core driver without math and settings overhead
- **MinimalistExample** - The **easiest** configuration
- **MultiI2C** - Using two LIS3DHs over I2C
- **MultiSPI** - Using two LIS3DHs over SPI

Library Usage

Take the following steps to use the library

- construct an object in the global space with one of these constructions
  - No parameters – I2C mode at address 0x19
  - I2C_MODE, address
  - SPI_MODE, pin number
- With in begin, set the .settings. values
- run `.begin()`

Example:

```
LIS3DH myIMU; //This creates an instance the library object.

void setup()
{
    myIMU.settings.adcEnabled = 1;
    myIMU.settings.tempEnabled = 0;
    myIMU.settings.accelSampleRate = 50;  //Hz.  Can be: 0,1,1
0,25,50,100,200,400,1600,5000 Hz
    myIMU.settings.accelRange = 2;      //Max G force readabl
e.  Can be: 2, 4, 8, 16
    myIMU.begin();
}
```

Settings

The main LIS3DH class has a public member, which is named settings. To configure settings, use the format `myIMU.settings.accelSampleRate = (...);` . Then, call `.begin()` to apply.

Settings contains the following members:

- `uint8_t adcEnabled` – Set to 1 to enable ADCs
- `uint8_t tempEnabled` – Set to 1 to override ADC3 with delta temperature information
- `uint16_t accelSampleRate` – Can be: 0,1,10,25,50,100,200,400,1600,5000 Hz
- `uint8_t accelRange` – Max G force readable. Can be: 2, 4, 8, 16
- `uint8_t xAccelEnabled` – Set to 1 to enable x axis
- `uint8_t yAccelEnabled` – Set to 1 to enable y axis
- `uint8_t zAccelEnabled` – Set to 1 to enable z axis
- `uint8_t fifoEnabled` – Set to 1 to enable FIFO
- `uint8_t fifoMode` – Can be 0x0,0x1,0x2,0x3
- `uint8_t fifoThreshold` – Number of bytes read before watermark is detected (0 to 31)

Functions

Advanced programmers: The LIS3DH class inherits the LIS3DHCore, which can be used to communicate without all these

functions, so you can write your own. This class is not covered in this hookup guide.

uint8_t begin(void);

Call after providing settings to start the wire or SPI library as indicated by construction and runs `applySettings()` . Returns 0 for success.

void applySettings(void);

This configures the IMU's registers based on the contents of .settings.

int16_t readRawAccelX(void);

int16_t readRawAccelY(void);

int16_t readRawAccelZ(void);

These functions return axis acceleration information as a 16 bit, signed integer.

float readFloatAccelX(void);

float readFloatAccelY(void);

float readFloatAccelZ(void);

These functions call the Raw functions, then apply math to convert to a float expressing acceleration in number of Gs.

uint16_t read10bitADC1(void);

uint16_t read10bitADC2(void);

uint16_t read10bitADC3(void);

These functions return the ADC values read from the pins. Values will be 10 bit and the detectable range is about 0.9V to 1.8V.

Note: When `tempEnabled == 1` , ADC3 reads as an unreferenced temperature in degrees C. Read twice and calculate the change in temperature.

void fifoBegin(void);

This enables the FIFO by writing the proper values into the FIFO control reg, and control reg 5. This does not start the data collection to the FIFO, run `fifoStartRec()` when ready.

Sample rate depends on data rate selected in .settings.

void fifoClear(void);

This reads all data until the status says none is available, discarding the data. Use to start with new data if the FIFO fills with old data.

void fifoStartRec(void)

This enables FIFO data collection. Run this before starting to check if data is available.

After fifoStartRec is used, data from the X, Y, Z registers is not real time, but is the next available sample.

uint8_t fifoGetStatus(void)

This returns the FIFO status byte. The contents of the byte are as follows:

- bit 7: Watermark exceeded
- bit 6: FIFO has overflowed
- bit 5: FIFO is empty
- bit 4 through 0: Number of samples available (0 to 31)

void fifoEnd(void);

This stops the FIFO and returns the device to regular operation.

Resources and Going Further

You should now have a basic understanding of how to use the LIS3DH, but if you need some more information check out the following links:

- LIS3DH Breakout Github repo – Design files.
- SparkFun LIS3DH Arduino Library Github Repo – arduino library.
- LIS3DH Datasheet – Hardware information and register map.
- LIS3DH AppNote – Descriptive material showing basic usage.

Going Further

Need a little inspiration? Check out some of these other great SparkFun tutorials.

Vernier Photogate
Vernier Photogate Timer -- using the Serial Enabled LCD Kit.

Simon Tilts Assembly Guide
This tutorial will guide you through assembling your Simon Tilts PTH Kit.

LSM9DS0 Hookup Guide
How to assemble, connect to, and use the LSM9DS0 -- an accelerometer, gyroscope, and magnetometer all-in-one.

9DoF Razor IMU M0 Hookup Guide
How to use and re-program the 9DoF Razor IMU M0, a combination of ATSAMD21 ARM Cortex-M0 microprocessor and MPU-9250 9DoF-in-a-chip.