

# 3DOF – Orientation Implementation

Using only an accelerometer for positional tracking is inaccurate in practice as the double integral of the acceleration magnifies (quadratically) and accumulates error in the actual system's position (drift). This can be solved by combining the system with a GPS and a Kalman filter to correct for drift by estimating the true position with all three components. Here we implement orientation tracking via a gyroscope, part of the goal in implementation of the virtualized 6DOF system.

Runtimes of the code developed for processing and visualizing gyroscope data showed that visualization takes more than 10X as long as the calculations themselves. As a result, it was decided to split the system into two parts, a microcontroller which monitors an 9DOF inertial-measurement-unit (IMU) and performs all computations relating to the system, and an external microcontroller/computer that visualizes the data. The block diagram (Image 1) below illustrates this.

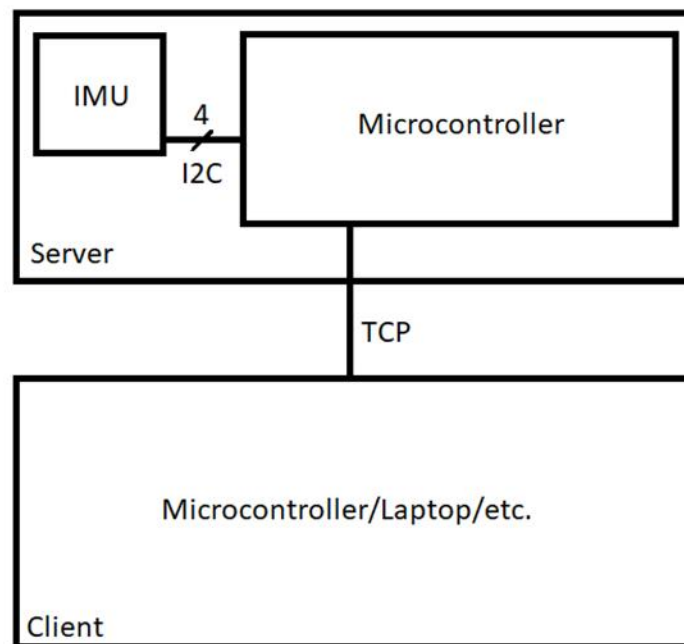


Figure 1

The following pages dive into the hardware and software design, decision making, and implementation in developing the 3DOF orientation-tracking system.

# RPi3B+ and an IMU System

## **IMUs and the LSM9DS1**

The IMU sensor chosen is the 9DOF LSM9DS1. [A link to its data sheet can be found here.](#) High-level information is summarized in this document. Before diving into the chip and system design itself, it is important to note what an accelerometer, magnetometer, and gyroscope do and why in conjunction they can give information on the location, directionality, and orientation of an object in 3D space.

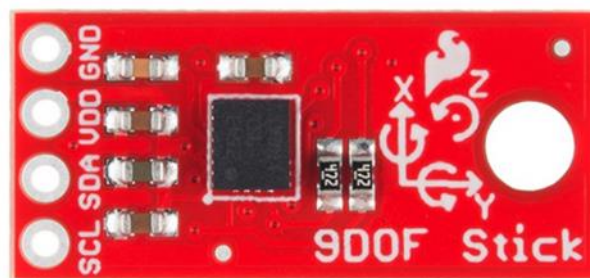
An accelerometer provides information on the linear acceleration of an object. In other words, 3D movement (x, y, z plane). This is typically accomplished through measurement of capacitance in a flexible MEMS capacitor. This changing capacitance is due to displacement of capacitor plates (due to motion in that direction) in the MEMS device. Thus, the capacitance of these plates is correlated to the motion or acceleration of the MEMS device in the relevant direction. Similar techniques with resistors and inductors can be used to measure this linear acceleration.

A gyroscope measures the angular velocity, or rotation/pitch/orientation, of an object. This is detected by taking advantage of gyroscopic properties (ie. the Coriolis Effect) and the force it may cause in a MEMS device. Again, like in the accelerometer, the force applied due to the angular velocity is correlated to the motion of MEMS capacitor plates and so changes in capacitance.

The last measurement device to take note of is the magnetometer, a device for measuring magnetic fields and thus directionality. Usually the Earth's magnetic field is measured. A magnetometer takes advantage of the Hall Effect (the production of a potential difference across a conductor when a magnetic field is applied in a direction perpendicular to that of the flow of current) to measure the direction of a magnetic field. Here, a MEMS conductor's resistance will change due to the direction of the field and its effect on the flow of current. Measurement of the resulting potential difference is how a magnetic field's directionality is deduced.

Together, these three ICs/measurement-devices create an inertial measurement unit (IMU with 9 degrees of freedom (9DOF)). Now, onto the LSM9DS1 breakout board.

## **Breakout Board**



*Figure 2*

This breakout board [\(link here\)](#) combines the LSM9DS1 (9DOF sensor) with the relevant materials to minimize system noise, avoid power-source ripple, incorporate pullup resistors, and more. Figure 1 contains the top of the PCB and Figure 2 below contains the bottom.

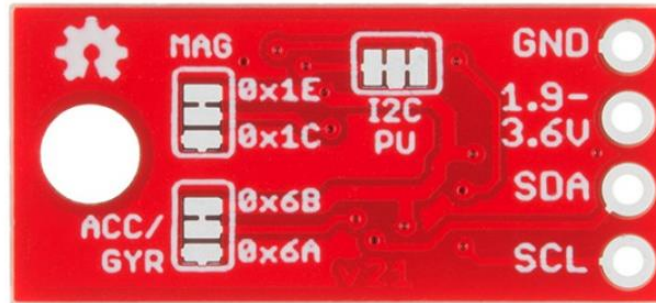


Figure 3

The board's schematic is below in Figure 3. Through this image it is easily understandable how the additional components from the LSM9DS1 ([datasheet here](#)) combine to create the breakout board.

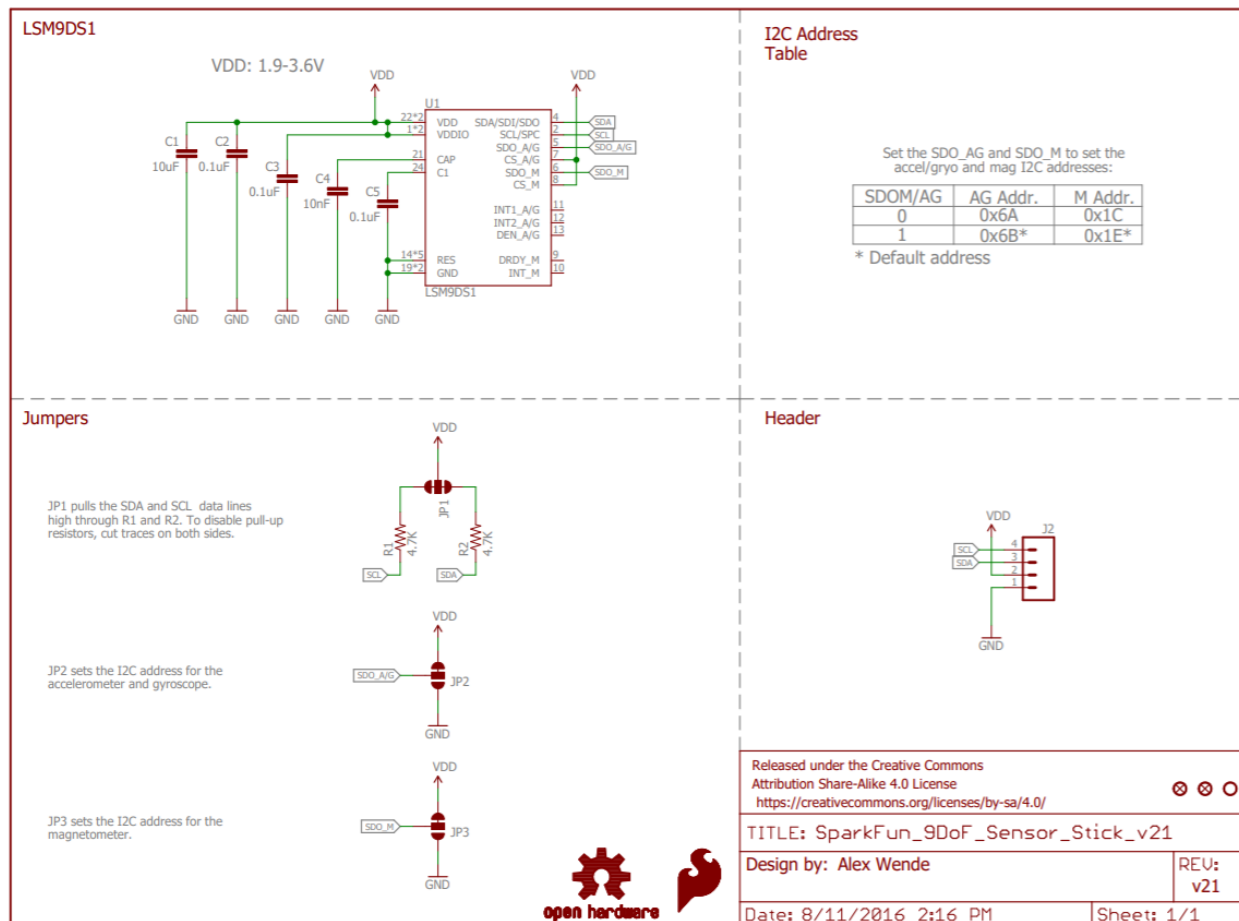


Figure 4

C1 is for removing the power-source ripple (hence the large capacitance). C2-C5 serve as low-pass filters to remove high-frequency noise which may disturb the internal electronics. R1 and R2 are pull-up resistors for connecting power to the system. In the I2C address section it is clear what the default addresses are for the accelerometer/gyroscope and magnetometer information.

Table 1 below describes the necessary high-level information for correctly utilizing the breakout board shown described above.

Characteristic	Value	Comments
$t_{PWR}$	210ms (round-up to 300ms)	Time for System to Power-Up
$V_{DD}$	3.3V	Power Supply to System
$T_{Op}$	-40°C to 85°C	Temperature Operating Range
I2C Clock Speed	400kHz	Clock Speed for I2C
ADC	16 bits	Analog Data Sample Granularity
Default I2C Gyro/Acc Address	0x6B	
Default I2C Magn Address	0x1E	To Change, Physically Alter Board

Table 1

### **RPi3B+ Microcontroller**

An RPi3B+ is an excellent choice for this project for the following reasons. Firstly, it comes with an onboard 3.3V/5V output power as well as I2C connections. This means that it provides all of the necessary inputs and outputs to interact with and power the breakout board. But more importantly, the RPi3B+ also contains a 64bit 1.4GHz SoC, 1GB RAM, multiple other communication abilities (I2C, UART, USB, SPI, etc.), GPIO pins, Bluetooth, is powerable via USB or 5V/2.5A input, is under \$50, can run C++/Python, and more. This means that in addition to meeting our immediate needs, the RPi3B+ can also be useful in the future as our system develops and incorporates tools such as cameras, motors, etc.

# System Implementation

## Hardware Connections

The RPi3B+ is setup as a TCP-Server where it reads LSM9DS1 gyroscope values, calculates the system's orientation, and sends an update to a client over the internet (for information on setting up RPi3B+ internet communications, see the setup document) every n-calculations. In this way, measuring the gyroscope readings is far faster (1-10 milliseconds per cycle) than if the RPi3B+ also performed visualization (over 50ms/cycle). Calculations are not done client-side as they may cause runtime issues with how fast server measurements are sent. The LSM9DS1 breakout board is connected to the RPi3B+ via 4 wires. The grounds are coupled via pin 6 of the RPi3B+, VDD is supplied by the 3V pin, RPi3B+ pin 1, and SDA/SCL by RPi3B+ pins 3/5 respectively. Figure 5 shows the RPi3B+ pinout.

Raspberry Pi 3 GPIO Header





















Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I <sup>2</sup> C)		DC Power 5v	04
05	GPIO03 (SCL1 , I <sup>2</sup> C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I <sup>2</sup> C ID EEPROM)		(I <sup>2</sup> C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Figure 5

The pullup resistors, high-pass capacitors, etc. are already included in the IMU's breakout board as explained above (Figure 4). Hardware assembly is trivial (simply solder and hookup the pins).

## Server Software – Server Class

The RPi3B+ is setup as a TCP-Client-Server which can only host one client at a time. If host-name issues appear, adjust the host variable in line 18 of "IMU\_Server.py" accordingly. Likewise, with line 19 if port issues arise (ie. the port is already in use). The "dataTransfer" method is not used in this application, feel free to ignore or delete it (it is intended for future applications of this software). The "sendData"

method sends utf-8 encoded data, the client receives it as a string. The “killServer” method shuts down the communication between the Server and Client.

### **Server Software – IMU Class**

The IMU class initializes the I2C communications between the RPi3B+ and LSM9DF1 breakout board. It also contains methods which ease the communication between the two devices (ie. handling data-types, processing data, decoding communications, etc.). This software should work for any RPi3B+, however, if issues arise - make sure the addresses used in the provided code match that of your own RPi3B+. Furthermore, if you adjust initialization settings, problems may arise (but no software error may appear) in functions such as “sampleDPS” where the raw data is scaled to expected data range. If you want to adjust any conditions, first consult the LSM9DS1 data sheet before adjusting the software. An important aspect of this class is that it requires the hardware of the system to remain “still” during initialization so that any errors can be minimized (method gyrERR).

### **Server Software – RectPrismV Class**

The RectPrismV class deals with initializing, adjusting, and tracking the information related to the physical orientation of the system – specifically its vertices. It makes the assumption the system will be a rectangular prism. To adjust the prism’s dimensions, change the class’s inputs. This class deals in radians, not degrees.

### **Server Software – “Main”**

The IMU class (gyro) and prism need to be declared before main executes in order to avoid errors due to non-initialized variables. For faster position updates to the client, but poorer accuracy, the if statement (“if j%n==0”) in the while loop can be edited. This function first calculates the gyroscope’s error, then uses it to offset all further readings. Readings are passed through the class’s “degreesRot” (how many degrees the object rotated around each access in that dt) and “rotate” (rotates the vertices) methods which returns the new vertices.

### **Client Software – Communication Setup and Data Manipulation**

The variable “host” must be equal to your RPi3B+’s IP address (as a string). Port can be left as 5560 or adjusted if it is in use.

All communications are done via utf-8 encoded strings. The method “strTOMatrix” converts the server’s packet to a Python 3 - numpy matrix containing the vertices of the rectangular prism (system). “vertsTOsides” then takes this data and creates the sides of the rectangular prism for visualization. These two functions are in a try-except statement in order to ignore any data exchange problems, such as multiple packages being read at once, which can crash the program. Even if a package is skipped the tracking is still accurate as it is handled server-side, so the next successful update will give the actual orientation regardless.

### **Client Software – Visualization**

Every loop of the while statement a new plot is created with the updated information and same plot settings (axis, maxima/minima, window size, etc.). The plot pauses for only 1 ms to ensure it is drawn,

then it awaits the next update before replotting. If instead of vertex data, "KILL" is received from the server, the Client will shut-down.