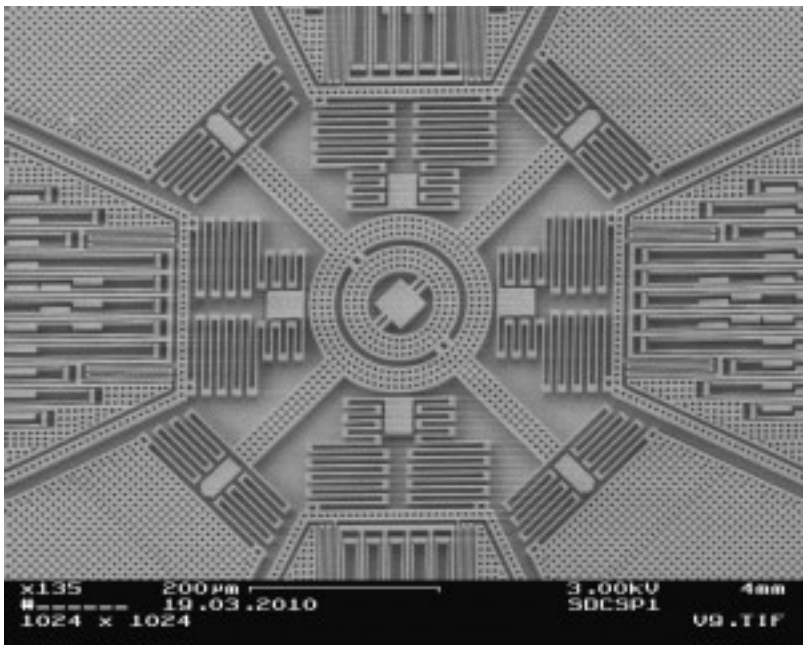
Gyroscopes and Accelerometers on a Chip

31. March 2013 · 190 comments · Categories: Angular Momentum, Arduino, Gyroscopes, Processing



The interior of a 3-D MEMS Gyroscope Sensor is intricate and tiny (this structure is only about 0.08 cm wide).

My last two blog entries discussed demonstrations of gyroscopes and angular momentum conservation at our school's science fair. One of the demonstrations I put together takes a look at how really small gyroscopic sensors, such as those in many smart phones, video game remotes or quad-copters provide information about their changing orientations. This information can be used as feedback for self-balancing (e.g. a two-wheeled scooter), navigation or as input to other applications like video games.

I didn't want to sacrifice my smart phone for this experiment. Fortunately, chips containing gyroscopic sensors are relatively cheap. In reading up on gyroscopic chips, I found that orientation data from gyroscope sensors is prone to drift significantly over time, so gyroscopic sensors are frequently combined with additional sensors, such as accelerometers or magnetometers to correct for this effect. This combination of sensors is frequently referred to as an IMU, or "Inertial Measurement Unit", and it is used in airplanes, spacecraft, GPS navigators (for use when GPS signals are unavailable) and other devices. The number of of sensor inputs in an IMU are referred to as "DOF"

(Degrees of Freedom), so a chip with a 3-axis gyroscope and a 3-axis accelerometer would be a 6-DOF IMU.

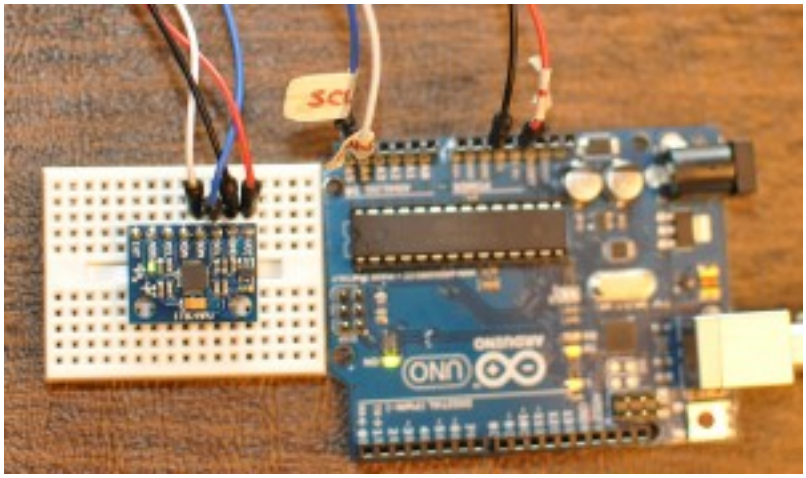
The GY-521 breakout board contains a 3-axis accelerometer and a 3-axis gyroscope in the tiny MPU-6050 chip in the center.

The MPU-6050 is a commonly used chip that combines a MEMS gyroscope and a MEMS accelerometer and uses a standard I2C bus for data transmission. More importantly to me, someone else has already done the hard work of reverse-engineering the MPU-6050 and provided freely available source code to use an Arduino microcontroller to retrieve the MPU-6050 data. There are a number different breakout boards available containing the MPU-6050 chip. You can find an overview of many of them on this page, under the heading “Breakout Boards”. I ordered the GY-521 module from Amazon.

There is a sophisticated library named **I2Cdevlib** for accessing the MPU-6050 and other I2C devices written by Jeff Rowberg. The specific code is provided in the sub-folder “MPU6050” . It utilizes a hardware buffer on the chip and Digital Motion Processing capabilities of the MPU-6050 to perform data conversions between different coordinate systems and combines data from multiple sensors to obtain greater accuracy and precision. Another useful library called **FreeIMU**, written by Fabio Varesano, is specifically geared towards calculating orientation from multiple-sensor IMUs, and can perform sophisticated data processing as well. FreeIMU makes use of the I2Cdevlib, but I found that the latest I2Cdevlib included with FreeIMU conflicted with the Rowberg’s version of I2Cdevlib, so be sure to use the correct version of I2Cdevlib if you use one of these libraries.

I was looking for a quick and dirty demo for our science fair, so I adapted the short code, by an author “Krodal”, at the bottom of this page to read the raw data from the

MPU-6050 and do simple (hah!) calculations of rotation angles myself. After going back to re-try my experiment with the libraries mentioned previously, I realized that it would have been much easier and more accurate to use one of the pre-written libraries instead. I did, however, learn a great deal in the process of converting the raw sensor data to rotation angles, so it was worthwhile to do it the hard way.



The GY-521 connected to an Arduino UNO. Additional connections are needed if using the I2Clib and FreeIMU libraries.

To set up the hardware, I soldered header pins to the GY-521, and connected it to an *Arduino UNO* via a tiny bread board. The wiring was trivial – only four jumper cables were needed. For power connect GND↔*GND* and VCC↔*3.3V* (the GY-521 board has a voltage regulator, and can take either 5V or 3.3V). Only two pins were needed to transmit data on the I2C bus: SDA (data)↔*A4* and SCL (clock)↔*A5*.

Getting the raw data was easy. Krodal's code worked immediately, and, generated output as shown below. A harder trick is to convert these raw numbers into meaningful data.

InvenSense MPU-6050

June 2012

WHO_AM_I : 68, error = 0

PWR_MGMT_2 : 0, error = 0

MPU-6050

Read accel, temp **and** gyro, error = 0

accel x,y,z: -276, -200, 14556

temperature: 27.141 degrees Celsius
gyro x,y,z : -5, -21, 53,

MPU-6050
Read accel, temp **and** gyro, error = 0
accel x,y,z: -280, -108, 14468
temperature: 27.282 degrees Celsius
gyro x,y,z : -16, 15, 73,

MPU-6050
Read accel, temp **and** gyro, error = 0
accel x,y,z: -296, -244, 14456
temperature: 27.282 degrees Celsius
gyro x,y,z : -5, -8, 46,

MPU-6050
Read accel, temp **and** gyro, error = 0
accel x,y,z: -288, -232, 14516
temperature: 27.235 degrees Celsius
gyro x,y,z : -3, -6, 49,
...

I modified Krodal's code to start with a calibration routine that averages the first 10 readings to compute the default sensor offsets. The offsets get subtracted from the raw sensor values before the values are converted to angles.

Now the task was figuring out how to compute the rotation angles. This can get quite complicated. There's a pretty good writeup of the mathematics involved here. For the GY-521 oriented flat as shown, the X and Y axes lie in the horizontal plane and the Z axis is vertical.

To compute orientation from the accelerometer, we rely on the fact that there is a constant gravitational pull of 1g (9.8 m/s^2) downwards. If no additional forces are acting on the accelerometer (a risky assumption, as we'll see later), the magnitude of the acceleration detected will always measure 1g, and the sensor's rotation can be computed

from the apparent position of the acceleration vector, as seen below. If the Z-axis is aligned along the gravitational acceleration vector, then it is impossible to compute rotation around the Z-axis from the accelerometer.

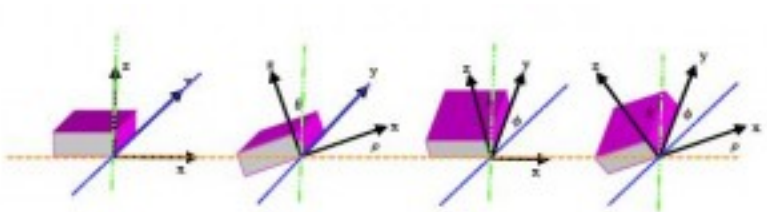


Figure 8. Three Axis for Measuring Tilt

$$\rho = \arctan\left(\frac{A_X}{\sqrt{A_Y^2 + A_Z^2}}\right)$$

$$\phi = \arctan\left(\frac{A_Y}{\sqrt{A_X^2 + A_Z^2}}\right)$$

$$\theta = \arctan\left(\frac{\sqrt{A_X^2 + A_Y^2}}{A_Z}\right)$$

This diagram shows calculations of tilt angles from the measured acceleration vectors. Diagram comes from this site. ρ and ϕ are defined as tilt with respect to the X- and Y-axes respectively, while θ is defined with as tilt of the Z-axis with respect to gravity. Please note that while ρ and ϕ are the same as pitch and roll, θ is NOT the same as Yaw. Yaw cannot be computed from the accelerometer.

According to the MPU-6050 datasheet, page 13, you can convert the raw accelerometer data from Krodal's code into multiples of g (9.8 m/s^2) by dividing by a factor of 16384. However, since the conversion to angles uses ratios of the acceleration vector components, this factor divides out. I computed rotation around the X-axis (ϕ) and Y-axis ($-\rho$) using the formulas in the image to the left. Interpreting the diagram at left is a little tricky, since the tilt of the X-axis actually shows rotation around the Y-axis, and the angling of the Y-axis shows (negative) rotation around the X-axis.

It is important to note that the accelerometer results provide accurate orientation angles as long as gravity is the only force acting on the sensor. However, when moving and rotating the sensor, we are applying forces to it, which causes the measurements to fluctuate. The net result is that accelerometer data tends to be very noisy, with brief but

significant perturbations. If these can be averaged out, the accelerometer provides accurate results over timescales longer than the perturbations.

Computing orientation from the gyroscope sensor is different, since the gyroscope measures *angular velocity (the rate of change in orientation angle),* not angular orientation itself. To compute the orientation, we must first initialize the sensor position with a known value (possibly from the accelerometer), then measure the angular velocity (ω) around the X, Y and Z axes at measured intervals (Δt). Then $\omega \times \Delta t$ = change in angle. The new orientation angle will be the original angle plus this change. The problem with this approach is that we are *integrating* - adding up many small computed intervals – to find orientation. Repeatedly adding up increments of $\omega \times \Delta t$ will result in small systematic errors becoming magnified over time. This is the cause of gyroscopic drift, and over long timescales the gyroscope data will become increasingly inaccurate.

The MPU-6050 datasheet shows that dividing the raw gyroscope values by 131 gives angular velocity in degrees per second, which multiplied by the time between sensor readings, gives the change in angular position. If we save the previous angular position, we simply add the computed change each time to find the new value.

As explained above, both the accelerometer and gyroscope data are prone to systematic errors. The accelerometer provides accurate data over the long term, but is noisy in the short term. The gyroscope provides accurate data about changing orientation in the short term, but the necessary integration causes the results to drift over longer time scales.

The solution to these problems is to fuse the accelerometer and gyroscope data together in such a way that the errors cancel out. The standard method of combining these two inputs is with a *Kalman Filter*,which is quite a complex methodology. Fortunately, there is a simpler approximation for combining these two data types, called a *Complementary Filter*. Far better explanations than I could ever write are givenhere (AVRFreaks) and here (Shane Colton), but the approximate formula to combine the accelerometer and gyroscope data is:

Filtered Angle = α **× (Gyroscope Angle) + (1 −** α**) × (Accelerometer Angle)** where $\alpha = \tau / (\tau + \Delta t)$ and **(Gyroscope Angle) = (Last Measured Filtered Angle) +** $\omega \times \Delta t$

Δt = sampling rate, τ = time constant greater than timescale of typical accelerometer noise

I had a sampling rate of about 0.04 seconds and chose a time constant of about 1 second, giving $\alpha \approx 0.96$.

So, now I had gyroscope data, accelerometer data and a filtered combination of the two. To visualize how these data all compared, I used Processing, an open source programming language and environment (very similar to the Arduino IDE) that is particularly well suited for data visualization. I modified the Arduino sketch to send the processed sensor and filtered data through the serial port, and wrote a Processing sketch to show the sensor and filter output as applied to three 3-D rectangles. Note that you must use the 32-bit version of Processing to get serial port access. For anyone who is interested,

here is the Arduino sketch:

GY_521_send_serial

Download gy_521_send_serial

and here is the Processing sketch (On Jan. 15, 2015, I updated the Processing sketch to fix a display bug that appeared in Processing 2.1.1):

ShowGY521Data

Download ShowGY521Data