Lab 3 – Reading data from a Gyroscope/Accelerometer with I2C.

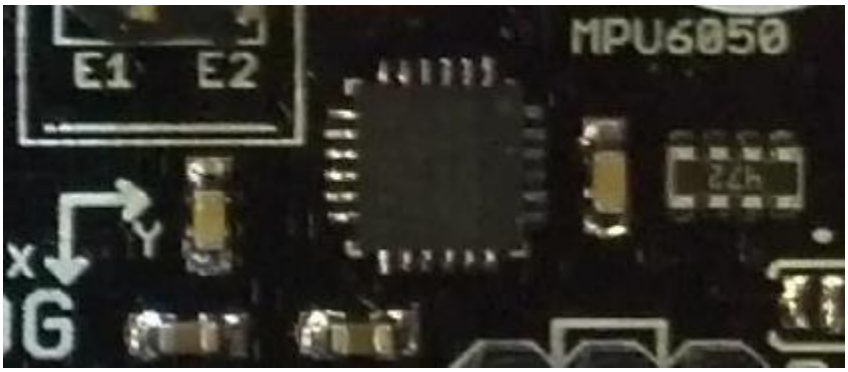
Part 1: Reading the Gyro

Objectives:

- Obtain data from a gyroscope with an I2C interface using a SFunction Driver
- Experimentally determine the conversion constant used to determine angle

Background Information:

This lab will utilize a single axis of the MPU6050 – an integrated circuit which contains a 3 axis gyroscope and a 3 axis accelerometer.



It has several noteworthy features:

- Configurable measurement ranges
  - Gyro angular rates: 250, ±500, ±1000, and ±2000°/sec
  - Accelerometer ranges: $\pm 2g$, $\pm 4g$, $\pm 8g$, and $\pm 16g$
- Configurable Low-pass and high pass filters

An Arduino library and sample code exists to make it easy to read from this device and set up the various options. This library is first tested in Arduino to ensure functionality, then ported to Simulink by "wrapping" the library into an SFunction block. This SFunction block is usually referred to as a "driver". In this case the "gyro driver" which is used to obtain the sensor data.

More information can be found by consulting the MPU6050 or by examining the actual library files included in this lab: `MPU6050.h` and `MPU6050.cpp`.

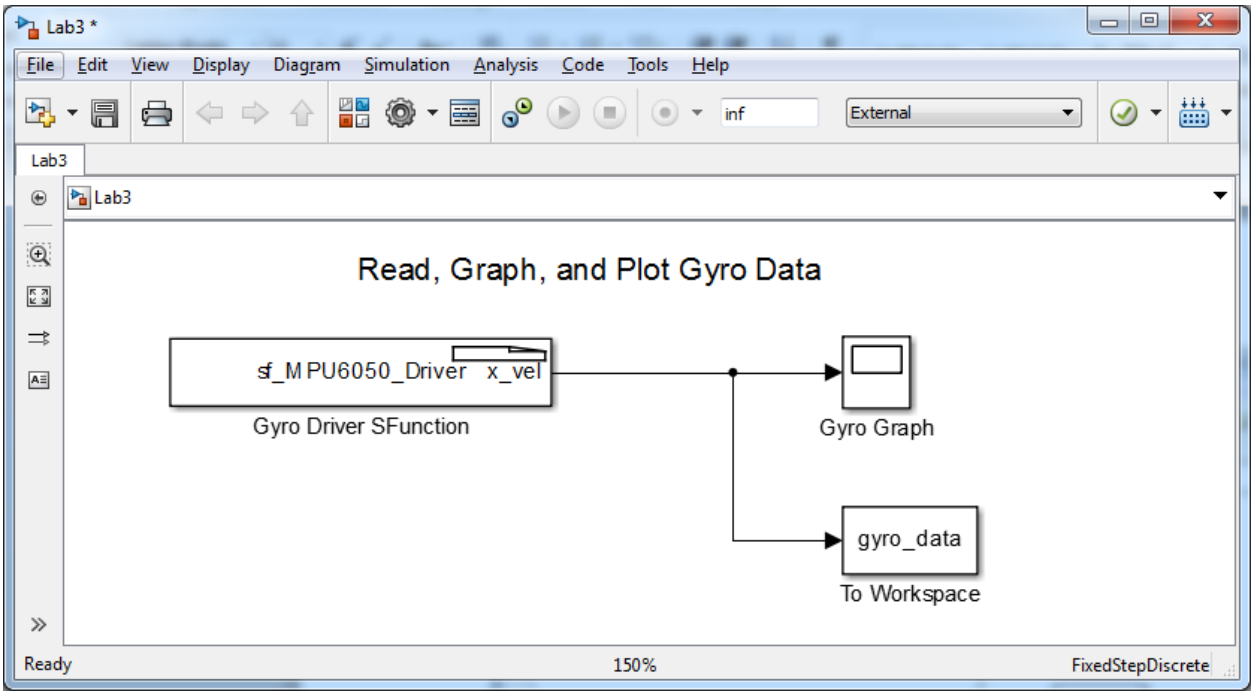
Question: What is the command needed to set the gyro low-pass filter to 42Hz? What is the delay associated with this? What is the command to set the gyro range to 250 degrees/second? (see `MPU6050.cpp`)

Gyroscopes:

A gyroscope measures angular rate or 'angular'. With a known initial condition the angular rate can be integrated to obtain angular position.

Simulink Model

- Copy the following files to your Matlab folder where you will build the Simulink diagram [Note: Leave these files in the default Matlab folder or your project folder. Do not put them in a sub-folder unless you add the path to Matlab with 'addpath' command.]
    - MPU6050_Blcck.slx
    - sf_MPU6050_Driver.mexw32
    - sf_MPU6050_Driver.mexw64
    - sf_MPU6050_Driver.tlc
    - sf_MPU6050_Driver_wrapper.cpp
    - I2Cdev.cpp (I2C Library files for MPU6050 communication)
    - I2Cdev.h
    - MPU6050.cpp (gyro library for configuration/communication with mpu6050)
    - MPU6050.h
    - Wire.cpp (Arduino wire library - support for I2C communication)
    - Wire.h
    - twi.c (Arduino wire library - support for I2C communication)
    - twi.h
- Build the following Simulink diagram.



Notice the descriptive text. Although it is graphical code, it is still code and should be properly commented. Get into the habit of properly commenting graphical code.

- **Parameters**: The step-size should be 30 milliseconds to run in external mode

Checkpoint 1:

Run the Simulink diagram and observe the results as you move the sensor (Arduino board) in your hand. Observe how the graph changes as you rotate about different axes.

Question: If the system is at rest – what is the value of the gyro reading? How much is it fluctuating? Use the data written to the workspace to compute the average gyro reading when the sensor is at rest for several seconds. [Hint: Make sure outputs to workspace such as 'gyro_data' are set to 'array'. Do this from now on in labs, unless mentioned otherwise]

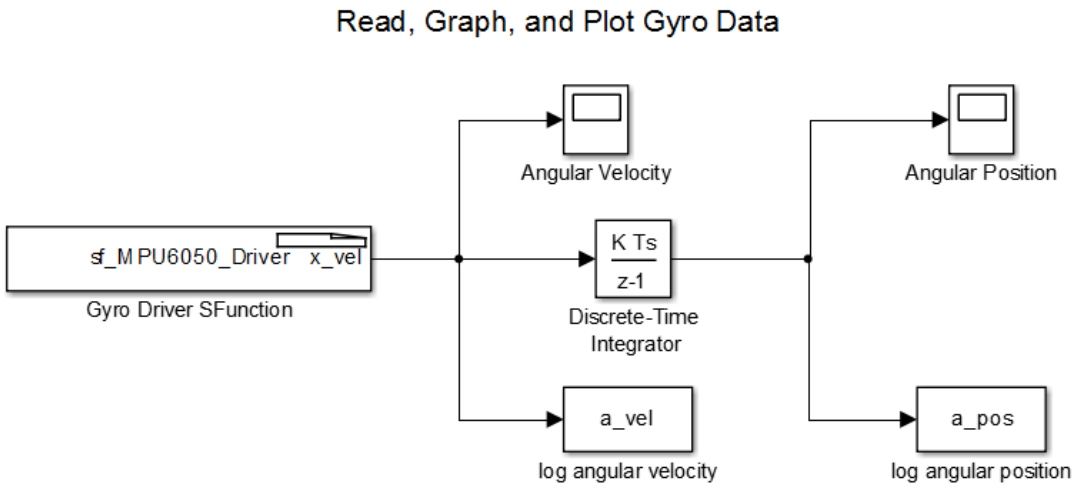
Note: The units are still unknown.

Part 2: Computing Angle from a Gyro

The gyro provides angular velocity – the change in angular position over time. To obtain the angle the angular velocity is integrated.

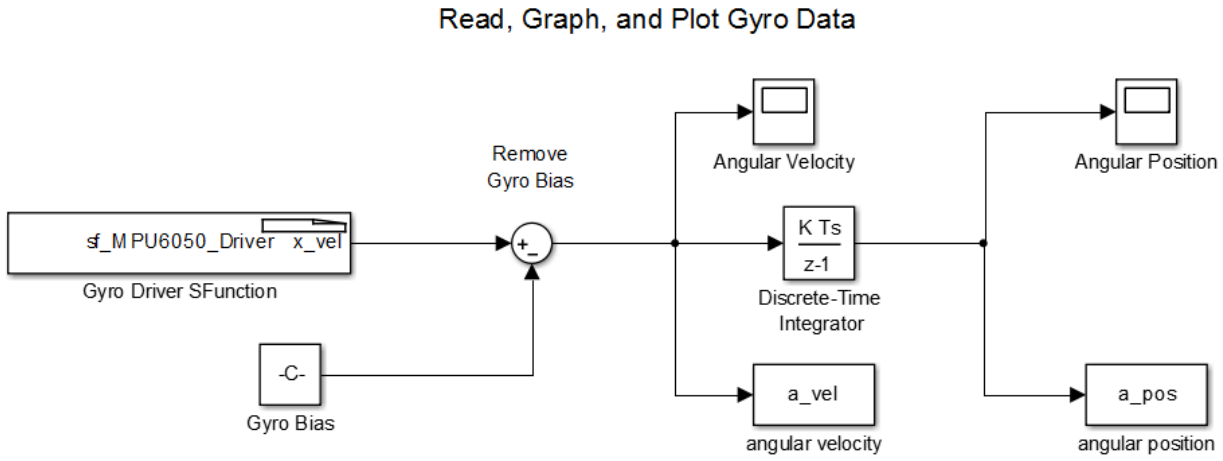
Simulink Model

Build and run the following Simulink diagram.



Question: What do you notice about the angular position and velocity when the sensor is sitting still? Does this make sense?

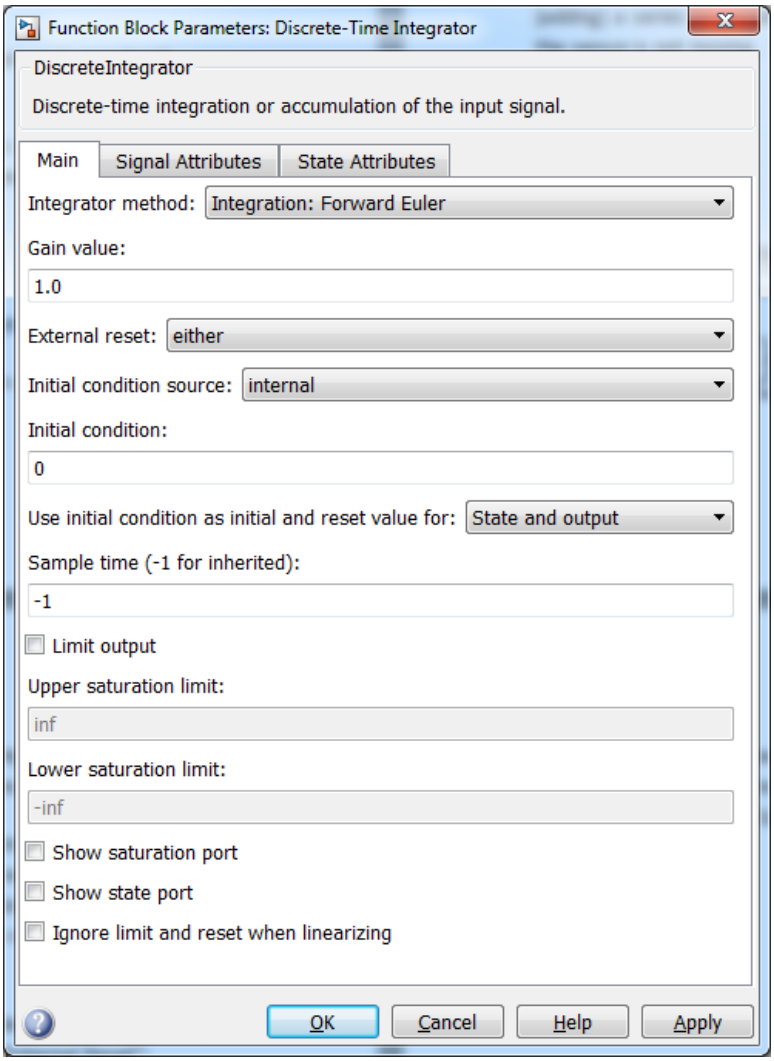
Since the sensor is not moving the values should be zero (no velocity). Any bias in velocity is integrating (adding) a series of nonzero numbers to the computed position even though the sensor is not moving. To eliminate this remove the bias before integrating.



Determine the bias which results in an angular velocity fluctuating about zero. Integrating this velocity will provide the angular position. The correct units are still unknown.

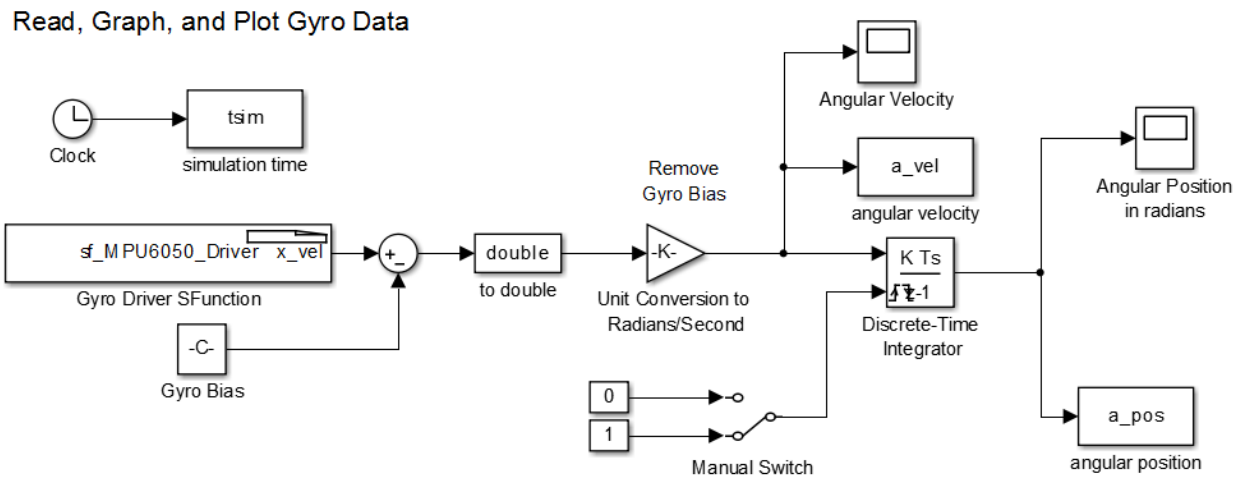
If there are any small errors in the bias, integrating these errors will eventually result in the position deviating from what is expected. In this case the only way to correct it is to start over again at a known point. This is done by resetting the integrator. When the code is first downloaded the integrator initial condition is zero. The only way to reset the integrator is to download the code again – this can be time consuming for initial testing.

To remedy this add an external reset to the integrator so it can be reset while running the simulation. Double click the integrator block and select "Either" for "External Reset":



Add a switch in the simulation diagram so the integrator can be reset manually – this will reset the angular position to zero when the switch is changed.

A unit conversion must be performed to get the data into meaningful units. A gain block is added to accommodate this. Start with this value at initially at 1.



Determining the Unit Conversion:

- Build and run the Simulink diagram.
- Modify the bias until the velocity is fluctuating around zero.
- Toggle the switch to reset the angular position.
- Rotate the sensor about the x-axis 90 degrees
- Use the resulting angular position indicated to determine the unit conversion factor
- Test your results

Question: What is the proper unit conversion factor to obtain the results in radians/sec and radians? Create a plot showing both the angular position and angular velocity versus time as the sensor moves from 0 to 90 degrees and back to 0. The units for this plot should be radians, radians/second and seconds. Clearly label this graph.

Checkpoint 2:

Create a demonstration where the LED 13 will change from off at zero degrees to fully bright at 180 degrees. Be prepared to demonstrate this. Include a copy of your Simulink diagram with the rest of the lab questions.