**Notes for IMU Implementation on ATRIAS**

1. Overview: The main purpose for the IMU is to sense the attitude (and angular velocity) of the robot's torso. The model we are currently using is the MEMS-based Microstrain 3DM-GX3-25 (<http://www.microstrain.com/inertial/3DM-GX3-25>), which combines a triaxial gyro, triaxial accelerometer, and a triaxial magnetometer along with a filtering algorithm to determine the torso posture referenced to the world frame. It also provides angular velocities, which will be necessary for the robot control processing.
   1. Note: The accelerometers can be used to estimate the gravity vector, which provides an **absolute** measurement of the relationship to the “world frame” or “fixed frame”. However, the accelerometers have to be integrated twice to get position, so they are very sensitive to noise. The gyros measure angular velocity which can be integrated (once) to estimate angular position (i.e. attitude), but these measurements are measured **relative** to the initial orientation of the gyro and are subject to drift. The combination of accelerometers with gyros through a filtering algorithm provides a solution which combines the strengths of the accelerometers with the strengths of the gyros (or maybe it would be better to say that the strengths of the accelerometer compensate for the weaknesses of the gyros, and vice versa). The video at [Enabling Robotic Attitude Sensing and Autonomous Navigation Through Inertial Sensor Technology](http://www.microstrain.com/video/enabling-robotic-attitude-sensing) (on the documentation tab of the Microstrain 3DM-GX3-25 website) provides a good overview on all of this.
2. IMU software and drivers: All of the necessary software and drivers for the IMU can be downloaded from <http://updates.microstrain.com/3DM-GX3_Software_CD_1.7.zip> (on the documentation tab of the Microstrain 3DM-GX3-25 website). That software kit includes USB drivers (for interfacing the IMU with a PC via USB) as well as a Monitor application which can be loaded on a PC to interact with the IMU over USB. There is also an application which can be used for calibrating the magnetometer. (For the time being, it’s probably best just to leave the magnetometer off, as will be discussed in the next section). While you’re on the Microstrain website, also download the 3DM-GX3-25 Single Byte Data Communications Protocol (<http://files.microstrain.com/3DM-GX3-25%20Single%20Byte%20Data%20Communications%20Protocol.pdf>) which provides all of the necessary info regarding communication and configuration of the IMU. [Note: Whenever you’re looking at documentation or website into, select the “Single Byte” version rather than the “MIP” version. The MIP version is for firmware version 2.0 and above, and our IMU is firmware 1.1.32 and can’t be upgraded to 2.0.]
3. Configuring the basic IMU settings: Some of the basic IMU settings can be configured most easily by using the Monitor application that was downloaded in the previous step. Connect the IMU to the PC with the USB cable, and start the Monitor application. It should detect the IMU and display the info in the window at the top of the GUI; if it doesn’t, click on refresh list a couple of times and/or unplug the USB and try again. Once it detects the IMU, select it and click on Device Settings.

In the settings menu, the main things you need to consider are the data rate (and baud rate) and the setting for the magnetometer (i.e. either off or on). (All of the other settings should be left at the default values.) Our experience has been that the magnetometer is not very reliable, and it may be best just to turn it off, by deselecting “Enable Heading Correction”. This means that you won’t get reliable heading information, but the most important data for control is the torso pitch and roll, so we can live without good heading data (at least for the time being).

The data rate can be set from 1 Hz to 1 kHz and should be set as high as possible, with the following constraints.

* 1. The highest data rate which still permits calculation of orientation (i.e. angles, not just angular velocities) is 500 Hz. (See p.40 of the Communications Protocol.)
  2. The max data rate is constrained by the baud rate of the computer interface. This isn’t an issue for USB, but for RS232 serial communications (which we’ll be using for the actual implementation on the robot), the baud rate has to be set to at least 460800 to support a data rate of 500 Hz. (Again, see p.40 of the Communications Protocol.)

After closing out the Settings dialog, you can have a little bit of fun checking out the IMU performance. Click on “Attitude & 3D Object” in the upper right hand corner, and you’ll see a graphical representation of the IMU, moving as you move the actual IMU.

1. Serial communication with the IMU: While USB communication is good for debugging and initial testing of the IMU, we’ll be using RS232 serial communication for actual implementation on the robot. (See p. 40 on the Communications Protocol for a good discussion of USB vs. serial.) The actual communication route will need to go from IMU to Medulla to the computer. I believe that Medulla already has a variety of RS232 “drivers” ready, and we’ll just need to figure out the best one to match the IMU settings. The basic IMU serial comms settings are on p.6 of the Communications Protocol; as mentioned previously, we’ll want a baud rate of at least 460800. While it would be possible to do some of the basic IMU packet processing and parsing on Medulla, it may be simpler at the outset to just use Medulla to translate RS232 to Ethercat and let the main computer (on the robot) do the necessary processing.
   1. Communication to the IMU: Communication to the IMU is needed for configuring things such as desired data type, IMU mode (continuous, active, etc.), and some things like that. All the commands are listed in the Communications Protocol document on p.7, with hyperlinks to the part of the document which describes things in a bit more detail. We had a bit of a challenge with this, but ended up doing most of it with the Simulink model serial\_comms\_to\_IMU.mdl included in this package. The basic idea is that you just need to send the hex byte code listed in the manual. For our Simulink implementation, we used hex2dec to figure out the decimal representation and then put that as the input for the Simulink model. For C++ implementation, you’ll definitely want to check out the sample code which is available in the Microstrain SDK <http://updates.microstrain.com/GX3_Orientation_SDK.zip> (the Single Byte version of the SDK, which is on the Documentation tab of the 3DM-GX3-25 website). We had trouble getting the “Set sampling settings” command to work properly, but that may have been an issue with packet length constraints imposed by our low baud rate.
      1. Note: The IMU is currently set to immediately (upon power-up) go into continuous transmit mode, sending out the data type Accelerations + Angular Rates + Orientation Matrix (0xC8 from the Communications Protocol). This was set up using the 0xD6 Continuous Preset command.
   2. Communication from the IMU: The IMU transmits a byte stream of data with packets corresponding to the requested data type. These packets also have a time stamp and a checksum. Your code will need to buffer, look for packet headers (e.g. 0xCE for Euler Angles), and then parse the packet into useable data. Note that all multi-byte data is transmitted in Big Endian order, and that all floating point values are 32 bit (4 bytes) in IEEE-754 format. We did online processing with the Simulink model IMU\_euler\_angles\_and\_rates\_with\_checksum.mdl and offline processing with a MATLAB script offline\_processing.m. Both of those are included in this package, and should give some insight on how to set up your code. (The offline\_processing.m script is commented pretty thoroughly.) Pay particular attention to the type-casting and byte-swapping necessary when converting the four-byte quantities to floating point numbers. You’ll also want to check for dropped packets, which you can determine by comparing successive time-stamps from the IMU. If you’re missing a lot of them, then there may be a problem with the data rate or baud rate.
2. Data type: The ideal data type for our implementation would be 0xC8 Accelerations + Angular Rates + Orientation Matrix. Having the orientation matrix in hand allows us to use different Euler Angle conventions and to apply transformations to compensate for mounting misalignment, etc. If there’s questions in regards to frames, Euler Angles, etc., I have some info on that as well.
3. Mounting: There are several sets of blueprints and diagrams on the Microstrain website, which may be helpful for the mounting. Check them out under “Mechanical Prints” on the Documentation page of the 3DM-GX3-25.

Included files:

* “serial\_comms\_to\_IMU.mdl”
* “IMU\_euler\_angles\_and\_rates\_with\_checksum.mdl”
* “offline\_processing.m”

References (all gathered in one place, just for good measure):

* <http://files.microstrain.com/3DM-GX3-25-Attitude-Heading-Reference-System-Data-Sheet.pdf>
* <http://files.microstrain.com/3DM-GX3-25%20Single%20Byte%20Data%20Communications%20Protocol.pdf>
* <http://www.microstrain.com/video/enabling-robotic-attitude-sensing>
* <http://updates.microstrain.com/3DM-GX3_Software_CD_1.7.zip>
* <http://updates.microstrain.com/GX3_Orientation_SDK.zip>

Contact:

Kevin Galloway

Postdoctoral Research Fellow  
4234 EECS Building  
University of Michigan  
1301 Beal Avenue  
Ann Arbor, MI 48109-2122 USA

[kevinsg@umich.edu](mailto:kevinsg@umich.edu)