Openmag User's Guide

Sean D'Epagnier

1

Table of Contents

1 Software

1.1 Installation

The DUSIv2 uses the FTDI chip for usb communications. Drivers are available for all operating systems from ftdi.com once configured, it should function as a virtual comm port, the baud rate is 38400. Onces configured it should function as a virtual comm port just as the other devices. For DUSIv1 and magnetometer mouse, see below.

1. Windows OS – There is a binary distribution for windows. This is a zip (open-mag_win32.zip) which contains all programs and drivers. The magnetometer mouse can only function as one of: mouse, joystick, or cdc device. The best I can offer is to choose one device at a time due to bugs in the windows operating system. The cdc device is used for configuration, to force this mode, hold the second (right) button down when plugging in the device. This is only for the magnetometer mouse, DUSIv1 is always in cdc mode. Windows will prompt for a driver the first time. Manually specify the downloaded folder containing the file OpenMag.inf. Once the driver is installed, you will be able to communicate via serial to the device.

   Hyperterminal Setup – Normally you will use the dataclient, but if you want to use hyperterminal for some reason: go to File->Properties, click on Settings, then ASCII Setup. Check "Send line ends with line feeds" and "Echo typed characters locally"

2. Mac OSX – Mac OS has builtin drivers already. For cdc mode, a new file named /dev/tty.usbmodemXXXX where XXXX is anything is created when cdc mode is enabled (hold key 2 down on the pointer model when plugging in usb)

   To communicate without installing any software, from a console type: "screen /dev/tty.usbmodemXXXX" with the right name for XXXX. This gives you a dumb console, run the dataclient for a nicer interface.

3. Other OS – Most common OS's support usb mouse, joystick, and cdc devices. This means you do not need to do anything special and no special driver is needed. For the serial device you should get a virtual comm port device, on linux it is /dev/ttyACM0 (the kernel module is cdc-acm) (or /dev/ttyUSB0 for DUSIv2), on freebsd it is /dev/ttyU0 (the kernel module is umodem). You will need Qt 4.3.0 or better to run DataViewer.

1.2 DataClient

Once configured the device will show up as a comm port. This means you can communicate to the device with any terminal program (eg hyperterminal)

The dataclient is the console version of the interface to device. The gui version is dataviewer in the next section. The dataclient program provides a more interactive user interface than a basic serial terminal. The main features are lineediting and completion via readline, data and reply separation (stdout and stderr), and the ability to change directories using cd, as well as see the directory via pwd.

The datainterface that the dataclient provides direct access to has two important concepts:

- operators – commands that operate on accessors, eg: get, set, clear, ls, ops ...
- accessors – data on the device which can be accessed eg: softwareversion

1.2.1 Example Session

To connect to a device, fifo or file:

```
./dataclient <file>
```

To connect to a server

```
./dataclient <host:port>
```

Once successfully connected you should see the prompt

```
$->
```

To autodetect the device, run the dataclient with no arguments.

It is now possible to execute commands and query data. Many of the commands are similar to unix commands. To list all possible commands issue "allops"

```
$-> allops
type ops ls allops mem get set values clear
```

Note: operators may be added or removed depending on software version

To list the accessors, issue "ls"

```
$-> ls
mouse/ joystick/ softwareversion settings/ stats/ calc/ calibration/ sensors/█
```

Items with a trailing / are directories. To list their contents, you may either cd into them, or ls them

```
$-> ls stats
freeram runtime mainloopfreq watchdog_resets
```

It should be apparent that "stats" is an accessor as is "stats/runtime". For example you may "get stats/runtime".

```
$-> get stats/runtime
51.81s
$-> cd stats/
stats/ $-> get runtime
54.54s
$-> get sensors/mag/output
-14349 -14980 -23731
```

Not all accessors support all operators. To see which operators are supported, use the ops operator which is always supported

```
$-> ops softwareversion
type mem ops get
```

This means it is valid to replace "ops" with any of the valid operators in the above command. Notice that you cannot "set" softwareversion.

Whenever the sensors are set up to automatically output data, this streaming data appears on stderr from dataclient.

For example, you may log data

```
sean@sun ~ $ ./dataclient 2> log
$-> set sensors/accel/outputrate 10
$-> [ctrl-D to exit]
sean@sun ~ $ cat log
```

```
accel: 201 58 1273
accel: 204 65 1273
accel: 202 65 1271
...
```

This interface includes additional host side operators which work with the dataclient program, they will not work with a console program.

1.2.2 Host Side Commands

1. cd – change to a given directory if it exists, relative paths are supported, "cd ../../info" Lastdir is supported as well eg: "cd -"
2. pwd – display the current directory

It is recommended to run "dataclient --help" to show all the capabilites of dataclient.

1.3 DataViewer

The DataViewer is a graphical application used to query and interact with the device while it is running. When it is first run, you should see a tree view, and below it a console and an output window.

Hit populate to automatically completely query the device. This may take a few seconds. Now it is possible to view all of the data stored on the device. If you press "Get Values", it will re-request just the values. This is useful because many of the values update continuously. You may also check certain values and only request those values. It is possible to modify certain values. The ones marked "write only" or "read/write" can be modified. For values with only certain possible settings, a dropdown is provided.

The Console window displays the actual data being sent and received to the device. The dataclient is actually running and doing all of the communication with the device, the dataviewer communicates with the dataclient. There may be operations that can only be performed from the console, for this you will have to run the dataclient.

The Output window displays streaming data comming from the device.

2 Configuration

There are many settings which can easily be modified with serial communication. Some key settings are configuring the magnetometer. The bandwidth and range can be set.

For use in most areas away from magnetic distortions, the a range of 1.6 can be used (it is the default):

```
set settings/mag/range 1.6
```

When working near metal or other distortions, a range of 3.2 or 6.4 might be more suitable to avoid saturation. In some parts of the world (near the equator mostly) a range of 0.8 can be used without saturation. The smaller range gives lower noise sensor data.

Another important setting is bandwidth. There are 3 settings: slow – 4.5hz medium – 13hz (default for dusi) fast – 63hz (default for pointer)

The fast bandwidth while able to read from the sensors faster has _much_ higher noise. The slow or medium rates should be used for applications concerned with accuracy. These settings should be applied before calibration takes place. The fast bandwidth is needed for pointer movement and high-bandwidth applications.

3 Calibration

One of the key features is autocalibration. The device may not come pre-calibrated, so for precise measurements, the user should understand how to perform calibration. If you are interested in how the calibration works, see the Calibration document. The device computes the unknown calibration coefficients needed to deliver useful data.

3.1 Fast Calibration

The fast calibration is performed continuously and automatically. The accuracy is not very good, but there are a number of uses which still work fine, and this calibration is much easier to make use of. To enable output whenever the fast calibration is updated:

```
$-> set calibration/fastdebugging true
```

Now you will see:

```
accelfast bias: (-1e+02 -1e+01 1e+02)
accelfast magnitude: 1e+03
...
```

The device will only get new updates when it is moved to new positions.

You can of also read the calibration directly

```
$-> get calibration/accelfast
accelfast bias: (40.921 -244.21 -135.27)
accelfast magnitude: 945.64
$-> get calibration/magfast
magfast bias: (21699 13887 -13836)
magfast magnitude: 76213
```

The fast calibration is used to compute pitch, roll, and yaw when the deviation for the accel or magnetometer calibration is zero. If you clear this calibration and don't recompute it (see next section) then the fast calibration is used:

```
clear calibration/accel
clear calibration/mag
clear calibration/magalign
```

3.2 StillPoints

For accurate calibration the device uses stillpoints. What this means, is it can detect when it is not moving by looking at sensor noise levels. Once the unit is "still" the raw sensor data can be stored into a tabl. It is essential to place the device in various orientations and hold it perfectly still for 1-3 seconds. It is important to cover all possible orientations. To make it easier to make sure you do it correctly, I recommend each of the 6 sides in 2-3 rotations on each side. 10 positions is minimum, 12-15 positions will give better results.

Note: With the DUSI model it is possible to perform and validate all calibration using the menu system.

3.3 Verifying Calibration

To enable debugging output for the normal calibration and to calibrate the accelerometer:

```
$-> set calibration/debugging true
$-> clear calibration/accel
```

You will be notified whenever a stillpoint is added with:

```
stillpoint added: 23076.941000 14773.721000 28749.383000 nan nan nan
```

It is also possible to query the stillpoints with:

```
$-> get calibration/stillpoints
```

The still points are the raw data for the magnetometer followed by the accelerometer. If joint still points is disabled, it is possible to have a stillpoint for only one sensor, in that case the other sensor can have values of "nan"

When enough stillpoints are aquired, you can "apply" the stillpoints to the accel calibration:

```
$-> apply calibration/stillpoints accel
$-> accel bias: (-136.67 -8.5935 135.41)
accel magnitude: 1311.7
accel magnitude ratios: [0.992 0.995]
accel cross coupling: {0.00217545 0.00549039 0.00179621}
accel second order terms: <5.8745e-07 1.7386e-06 4.0053e-07>
accel third order terms: <6.316e-11 2.3483e-09 1.775e-09>
accel dev: 0.022264
accel bias: (-133.19 -8.4023 133.13)
accel magnitude: 1325.4
accel magnitude ratios: [0.993 0.996]
accel cross coupling: {0.00212276 0.00534561 0.00174033}
accel second order terms: <5.1716e-07 1.6266e-06 1.3047e-06>
accel third order terms: <6.7473e-11 2.3116e-09 1.7148e-09>
accel dev: 0.002142
accel bias: (-133.11 -8.3985 133.09)
accel magnitude: 1325.5
accel magnitude ratios: [0.993 0.996]
accel cross coupling: {0.00212184 0.00534325 0.00173963}
accel second order terms: <5.1596e-07 1.6241e-06 1.3229e-06>
accel third order terms: <6.6307e-11 2.3128e-09 1.7109e-09>
accel dev: 0.002136
```

The calibration takes a few seconds, and will only output if debugging is set to true. You can see that the second and third order terms are quite small. The magnitude ratios are also small as well because the accelerometer is high quality.

The accelerometer calibration must be applied first because the magalign calibration (computed when mag is calibrate) depends on it.

The magnetometer uses the same process to calibrate, but calibration for both the mag, and magalign is performed. The mag calibration is just for the magnetometer. The magalign calibration is the alignment between the accelerometer and magnetometer. Make sure the data is read away from distortions when calibrating.

```
$-> apply calibration/stillpoints mag
```

The closer the deviation is to zero, the more correct the calibration is, a deviation of .01 or better is normally acceptable.

The stillpoints are lost on power reset, but they can also be cleared manually:

```
$-> clear calibration/stillpoints
```

3.4 Alignment

Once calibration for the sensors is performed, accelerometer and magnetometer vectors are available in the sensor coordinate system. This coordinate system typically has some arbitrary rotation away from the desired coordinate system.

There are two alignments performed. One rotates from sensor coordinates into box coordinates. Box coordinates are used to calculate pitch, roll, yaw and dip (the current measured inclination). The other rotation rotates into laser coordinates. Unlike box coordinates, laser coordinates do not havfe a concept of roll since the alignment is to an axis (laser or sight). Laser coordinates provide incline and azimuth which are like pitch and yaw except they use laser coordinates not box coordinates.

You may perform alignments external to the device with your own software and not use this feature, in this case you might as well read from the sensors directly.

To set the box alignment there are two options.

- if you know magnetic north, place the device level and facing magnetic north:

```
set boxalignment 1
```

Now the boxalignment is set

- If you do not know magnetic north, then place the device level:

```
set boxalignment 2
```

Next place the device pointed straight up (against a wall should work):

```
set boxalignment 3
```

If you want to reset boxalignment to no rotation.

```
clear boxalignment
```

The laser alignment has separate alignments for the accelerometer and magnetometer. Mathematically speaking, these alignments are identical, but in practice they are not. Similar results for the two alignments indicate good calibration.

To set the laser alignment, you have to take shots around an axis. Typically this axis has a laser pointer. To take a shot, align through the laser axis to a known point and:

```
set laseralignment 1
```

repeat this command for at least 5 shots, or until the error is low enough.

If you make a bad shot, you will need to reset and start over. To reset the laser alignment:

```
clear laseralignment
```

4 Emulation

Emulation allows you to run the calibration algorithms, and the menu system on a pc computer rather than the device. If you had an embedded system running a regular OS talking to the device, it would be possible to run the algorithms on that system therefore offloading the floating point computations.

Note: Currently Emulation does not compile on win32.

The program "calibrationhost" can be run which runs a tcp server. Connect to this server using the dataclient.

```
./calibrationhost -q /dev/ttyACM0 &
[1] 12351
Listening on port 7029 for connections from telnet or dataclient
./dataclient localhost:7029
$-> ls
calc/ sensors/
```

The emulation provides the exact same interface. The device (/dev/ttyACM0) should previously be configured to output accel, mag, and temperature data. There is a script "runcalibrationhost.sh" which runs the calibrationhost program and sets up the device to output raw data correctly.

It is also possible to relay the data as a server:

```
./dataclient -p 3000 -q
Listening on port 3000 for connections from telnet or dataclient
$->
```

You may then connect to this dataclient with other dataclients from remote hosts on port 3000. These dataclients may in turn run as servers as well.

It is also possible to run the menu interface if you have opengl. This program "menuhost" automatically runs the calibration algorithms, so it is equivilant to calibrationhost, with the addition of the menu system.