**Recommended development environment:** Visual Studio 2010 Express

The included Visual Studio 2010 solution can be imported into Visual Studio 2012 as well.

This zip file contains a single Visual Studio 2010 solution (.SLN) file. This solution contains both the GUI project and the firmware project. The firmware project will not be compiled by Visual Studio; it is included in the solution because some people, including me, prefer to write code in the Visual Studio IDE over the Arduino IDE. You’ll still need to open the .ino project in the Arduino IDE to compile it.

Other Notes:

* (If you look at the code) The sensor board's magnetometer has 2 of its axes swapped. This is a hardware problem. Not anything to worry about, but the magnetometer z/y axis is swapped.
* Any Arduino library functionality that requires timers (tone, PWM, etc) will not work, since timer A is being used directly in this project.

Short description of the more technical stuff  
----------------------------------------------------------  
**Headtracker:**  
The sensors are sampled with 128 samples/sec. As we have 9 sensors, we get 128 \* 9 = 1152 measurements/samples every second (more than plenty for a headtracker). The sampling-rate is controlled by timer0, that will set a flag high - indicating that the sensors should be read and new angles etc. calculated (timer0 running in CTC-mode, using compare-register A). While it should be possible to make all the calculations etc. within the interrupt, it's not recommended. A lot of heavy/slow calculations are done, and in case of timing-problems, we just want it to skip a sample and continue (should never happen, but just in case).   
If you turn on DEBUG you will get a lot of extra information, be able to see timing-problems etc. (please note, using serial.print is time comsuming and will in mast cases give timing-problems. Rarely a real problem when debugging, but be aware that it will change the samplerate a bit.)  
  
**Accelerometer:**  
The accelerometer data is converted to G-force. Not really necessary, but more convenient to work with. The total G-vector is calculated, and the angle calculated by using the total vector-length and the G-force in each direction.   
  
**Gyro:**  
The gyro indicates the angular change in degrees/sec. All axes accounts for tilt etc and data is continously integrated, to give an estimation of the angle.   
  
**Magnetometer:**  
The magnetometer is compensated for tilt (x- y-direction), and the final x and y values used to calculate the magnetic heading.   
  
**Filter:**  
The filter is a basic complementary filter, and an optional lowpass filter. The weight for gyro/accelerometer/magnetometer can be sat in the GUI, and lowpass filter changed.   
  
  
**PPM**  
Please note that only PPM-out is activated as default. PPM-mixing etc. need some work.   
The PPM in/out is a bit tricky, as we can only do one thing at a time. Oh - and we only have one 16 bit timer available. Timer1 (16 bit timer) is used to generate the PPM output. It's set to change pin at interrupt to get an accurate hardware-timed pin-change. The compare-interrupt will set the next compare-match.   
  
PPM-in is detected by pin-change interrupt (higher priority than timer1 compare interrupt). It will use the time from timer1, and detect if the timer have been reset.   
----------------------------------------  
**A few notes:**  
  
**Use of Serial output**  
The headtracker uses a lot of slow calculations with floating points, cos, sin atan etc. Furthermore the sampling frequency is pretty high. Using Serial.print is great for getting info, data etc. - but it requires time. As it's running "on the edge", you can only print a few characters without loosing a sample. It's programmed to just skip a sample in case of timing problems, but just be sure to note this when testing.   
  
**Updating variables**  
All variables stored in EEPROM have to be changed in the GUI or by changing this:  
// If the device have just been programmed, write initial config/values to EEPROM:  
if (EEPROM.read(0) != 42) {   
  
But as all settings saved in EEPROM can be changed from GUI I'll suggest that, as it's a lot easier. 