Interface to and Calibrate the WiiMote

Aaron Feldman, Antonio Rohit

University of California, Berkeley

September 7, 2014

Abstract

Calibrating and interpreting data from a digital sensor, specifically the accelerometers from the Wii Remote. Interfacing a computer running Windows OS with the Wii Remote via Bluetooth HID wireless link, obtaining raw data from its accelerometer, calibration of the raw data, and display of the results in NI LabVIEW.

I. Pair a desktop computer with the WiiMote

Using the provided resourse LabVIEW VI WiiMote Pair.vi, we were able to connect the Wii Remote via Bluetooth to a Windows 7 Computer. This took unblocking multiple downloaded files and a system reboot. The MAC address of the Wii Remote used was 8C:56:C5:5A:0D:0A.

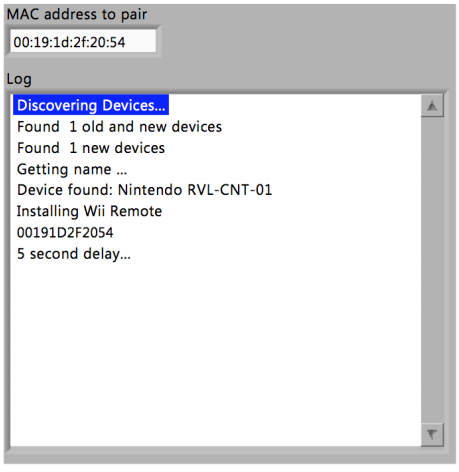

Figure 1: WiiMote Pair.vi

II. Plot the uncalibrated accelerometer signal from the WiiMote

After pairing we opened and ran the provided LabView VI Interface.vi. With this we were able to gain access to the raw accelerometer data. With the Wii Remote at rest on level ground with the face (z direction) up we took readings for the x, y, and z accelerometers.

$$x = 128 \tag{1}$$

$$y = 128 \tag{2}$$

$$z = 156 \tag{3}$$

As we can see the z accelerometer's value is noticeably higher. This is because this value is measuring both bias and gravitational force.

III. Control the WiiMote actuators

Using the front panel of WiiMote Interface.vi, we are able to send instructions to the Wii Remote using the Transmission panel.

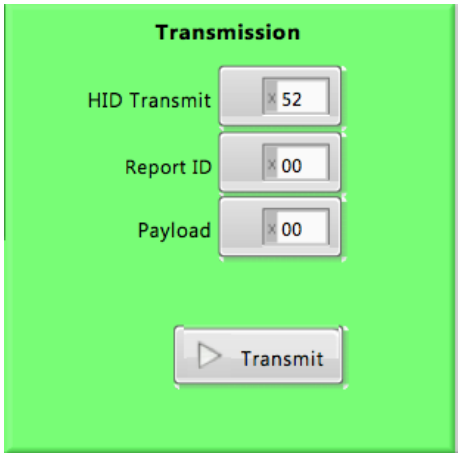

Figure 2: Transmission panel

Using $HIDTransmit$: $0x52$, $ReportID$: $0x11$, $Payload$: $0xa0$ we are able to turn the Wii Remote LEDs 2 and 4 on, and LEDs 1 and 3 off. This is because the microcontroller controls the LEDs with the first nibble of the $Payload$ byte, where a 0 represents off and a 1 represents on, with bits 4 controlling the first LED and bit 7 controlling the last and other bits respectively. Therefore 0b10100000 (0xa0) turn the LEDs 2 and 4 on, and LEDs 1 and 3 off. If we modify the instructions to $Payload$: $0xa1$ we are able to also enable to rumble motor, as the microcontroller controls the rumble motor with the same $ReportID$ with the last bit of the $Payload$ set to $0b1$.

IV. Measure sensitivity and bias of the accelerometer

We made the following modifications to the Event Structure in the LabVIEW Block Diagram to enable both the bias and sensitivity buttons on the control panel.

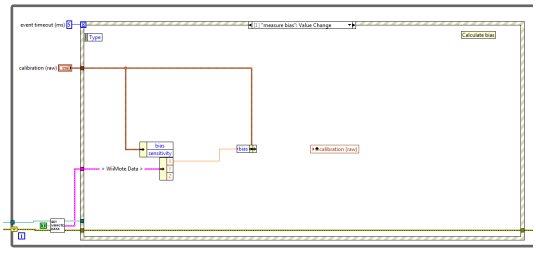

Figure 3: Event Structure Bias Modifications

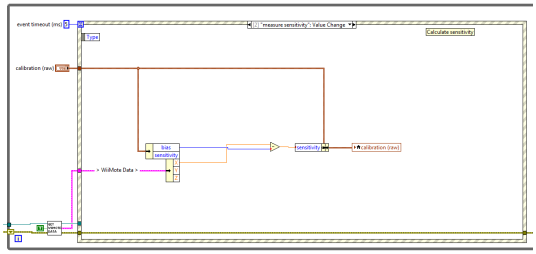

Figure 4: Event Structure Sensitivity Modifications

The bias and sensitivity are both stored in $I16$ type variables in the LabVIEW environment. We measured a bias of 128 ADC Units and a sensitivity of 27 ADC Units.

V. Calibrate the accelerometer

We modified the contents of the MathScript Node in WiiMote Interface.vi so that the Accelerometer (calibrated) chart plots acceleration of the x, y, and z axes, and the magnitude indicator displays the magnitude of acceleration, all in units of g^s. In order to calibrate the accelerometer we used the equation $\hat{A}_{calibrated} = (\hat{A}_{raw} - bias)/sensitivity$. In order to calculate the magnitude of the acceleration we used the equation $|A| = \sqrt{x^2 + y^2 + z^2}$.

Based on our calibration parameters and the bit resolution and sensitivity of the Wii Remote we can calculate an estimate for the maximum acceleration that the Wii Remote can measure. To do this we take the Wii's bit resolution of 256 bits, subtract the bias and divide by the sensitivity. $(256 - 128)/27 \approx 4.7g$.

When the Wii Remote is at rest the magnitude indicator should be ≈ 1. This is because the Wii Remote should be experiencing no acceleration other than that due to gravitational forces.

It is better to perform calibration at runtime, rather than hard-coding values for the sensitivity and bias of the sensor for many reasons. One reason is that it makes it possible to easily replace sensors if they are malfunctional. Another is that it allows you to easily change the bias if the bias happens to change for a sensor and also compensate in a mass design setting for manufacture differences between different sensors without having to measure and manually calculate each devices sensors individually.

To implement all of this into LabVIEW we used the MathScript codes:

$$a = (aRaw - bias)/sens; \tag{4}$$

$$mag = sqrt(a(1)^2 + a(2)^2 + a(3)^2); \tag{5}$$

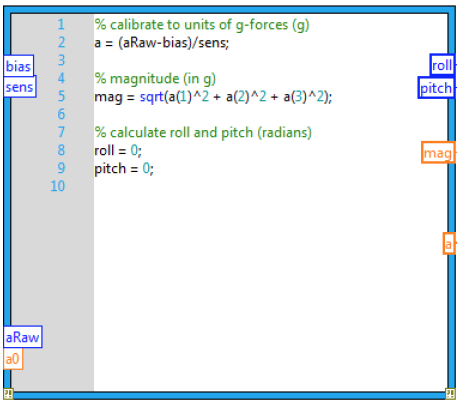

Figure 5: Code Block Modifications

VI. Measure Pitch and Roll

We were able to modify the contents of the MathScript Node in WiiMote Interface.vi so that the Orientation (calibrated) chart plots pitch and roll or the Wii Remote in units of degrees. To do this we calculated pitch and roll with:

$$pitch = \arctan\left(\frac{G_y}{\sqrt{G_x^2 + G_z^2}}\right) \tag{6}$$

$$roll = \arctan\left(-\frac{G_x}{G_z}\right) \tag{7}$$

Converting equation 6 to MathScript:

$$atan2(a(2), sqrt(a(1)^2 + a(3)^2)); \tag{8}$$

Converting equation 7 to MathScript:

$$atan2(-a(1), a(3)); \tag{9}$$

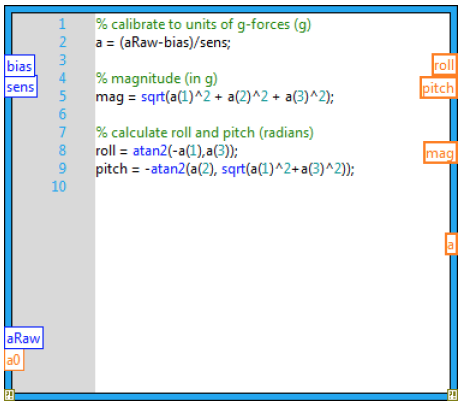

Figure 6: Code Block Modifications

VII. Filter the accelerometer signal

Accelerometers are often used to detect high frequency signals such as noise and vibration; for a game controller, lower-frequency movements of the user should be isolated. In particular, the rumble motor will introduce a high-frequency signal. We implemented a moving average lowpass filter to smooth the signal of the calibrated accelerometer when the rumble motor is on.

On a discrete signal like the digital signals used in electronics and more specifically the Wii Remote, we can weight both the current value of the remotes accelerometer output with the previous value. In averaging these weighted values together, we can adjust the weights to provide more or less sensitivity of our output to the current input value. If there are extreme fluctuations in our value, say due to a rumble motor, we are in this way able to average them out. For our moving average low pass filter we used the equation $y[n] = (1-\alpha) * y[n-1] + \alpha * x[n]$. This has a frequency response of $H(z) = \left(\frac{\alpha}{1-(1-\alpha)z^{-1}}\right)$. We used values from $\alpha = 0.1 \rightarrow \alpha = 0.001$ with good results.

$$A_{filtered} = (1-k) + a0 + k * a; \tag{10}$$

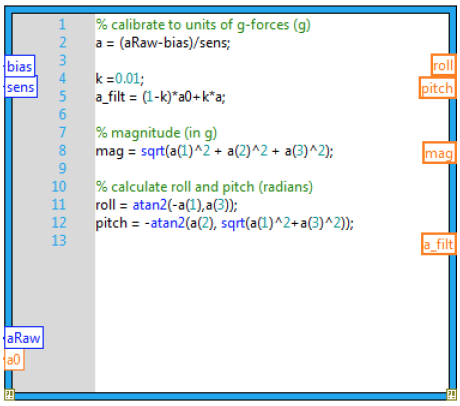

Figure 7: Moving Average Low Pass Filter

VIII. Lab Feedback

This lab was a great introduction to connecting via Bluetooth and calibrating sensors. In the future it would be nice to see this lab implemented in an OS other than Windows (Linux/OS X). (Yes that would require re-imaging/dual booting the lab computers). This would possibly bring this lab more in line with newer trends in the marketplace and remove many of the head aches that are Windows specific.