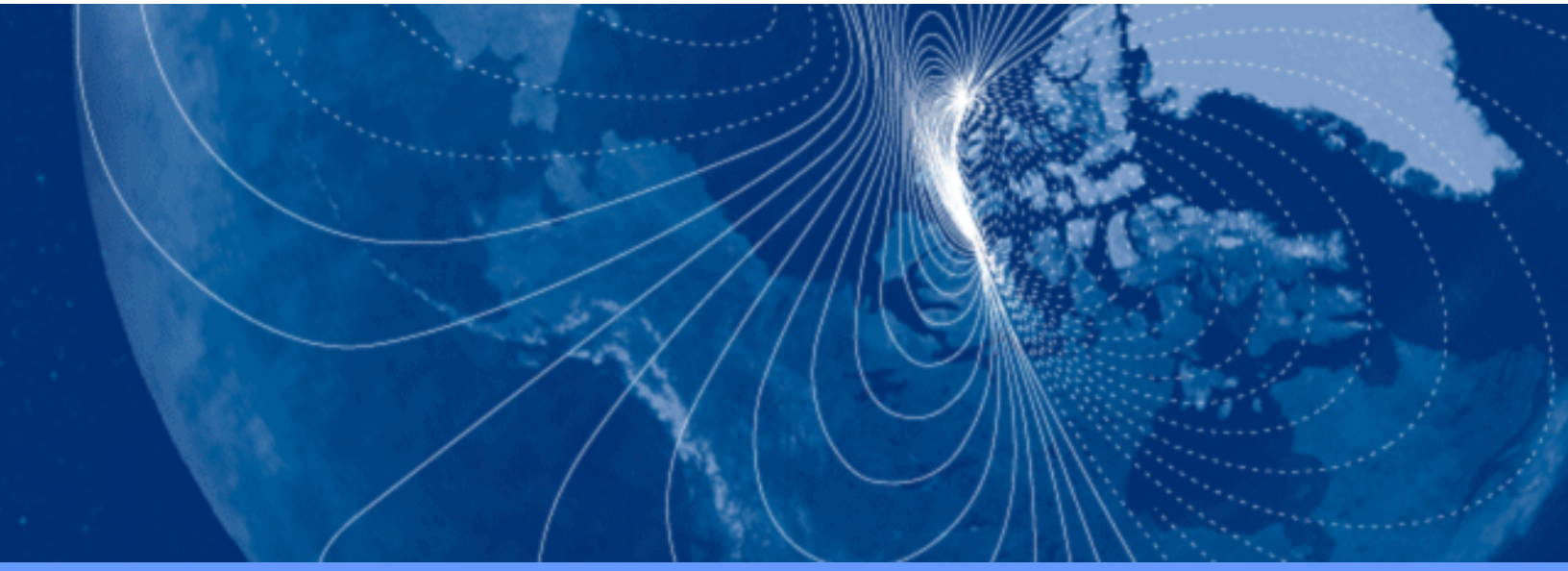


User Manual

SpacePoint Fusion

Demonstration Motion-Tracking Module

PNI SENSOR CORPORATION

Table of Contents

List of Tables

1 Copyright & Warranty Information

Revised June 2010: for the most recent version visit our website at www.pnicorp.com

PNI Sensor Corporation
133 Aviation Blvd, Suite 101
Santa Rosa, CA 95403, USA
Tel: (707) 566-2260
Fax: (707) 566-2261

Warranty and Limitation of Liability. PNI Sensor Corporation ("PNI") manufactures its Products from parts and components that are new or equivalent to new in performance. PNI warrants that each Product to be delivered hereunder, if properly used, will, for ninety (90) days following the date of shipment unless a different warranty time period for such Product is specified: (i) in PNI's Price List in effect at time of order acceptance; or (ii) on PNI's web site (www.pnicorp.com) at time of order acceptance, be free from defects in material and workmanship and will operate in accordance with PNI's published specifications and documentation for the Product in effect at time of order. PNI will make no changes to the specifications or manufacturing processes that affect form, fit, or function of the Product without written notice to the Customer, however, PNI may at any time, without such notice, make minor changes to specifications or manufacturing processes that do not affect the form, fit, or function of the Product. This warranty will be void if the Products' serial number, or other identification marks have been defaced, damaged, or removed. This warranty does not cover wear and tear due to normal use, or damage to the Product as the result of improper usage, neglect of care, alteration, accident, or unauthorized repair. THE ABOVE WARRANTY IS IN LIEU OF ANY OTHER WARRANTY, WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. PNI NEITHER ASSUMES NOR AUTHORIZES ANY PERSON TO ASSUME FOR IT ANY OTHER LIABILITY.

If any Product furnished hereunder fails to conform to the above warranty, Customer's sole and exclusive remedy and PNI's sole and exclusive liability will be, at PNI's option, to repair, replace, or credit Customer's account with an amount equal to the price paid for any such Product which fails during the applicable warranty period provided that (i) Customer promptly notifies PNI in writing that such Product is defective and furnishes an explanation of the deficiency; (ii) such Product is returned to PNI's service facility at Customer's risk and expense; and (iii) PNI is satisfied that claimed deficiencies exist and were not caused by accident, misuse, neglect, alteration, repair, improper installation, or improper testing. If a Product is defective, transportation charges for the return of the Product to Customer within the United States and Canada will be paid by PNI. For all other locations, the warranty excludes all costs of shipping, customs clearance, and other related charges. PNI will have a reasonable time to make repairs or to replace the Product or to credit Customer's account. PNI warrants any such repaired or replacement Product to be free from defects in material and workmanship on the same terms as the Product originally purchased.

Except for the breach of warranty remedies set forth herein, or for personal injury, PNI shall have no liability for any indirect or speculative damages (including, but not limited to, consequential, incidental, punitive and special damages) relating to the use of or inability to use this Product, whether arising out of contract, negligence, tort, or under any warranty theory, or for infringement of any other party's intellectual property rights, irrespective of whether PNI had advance notice of the possibility of any such damages, including, but not limited to, loss of use, revenue or profit. In no event shall PNI's total liability for all claims regarding a Product exceed the price paid for the Product. PNI neither assumes nor authorizes any person to assume for it any other liabilities.

Some states and provinces do not allow limitations on how long an implied warranty lasts or the exclusion or limitation of incidental or consequential damages, so the above limitations or exclusions may not apply to you. This warranty gives you specific legal rights and you may have other rights that vary by state or province.

2 Introduction

Thank you for purchasing PNI Sensor Corporation's SpacePoint Fusion demonstration motion-tracking module. We're certain you'll be impressed with its incredibly accurate and repeatable tracking performance and its low latency.

The SpacePoint Fusion module is intended for demonstrating the capabilities of PNI's SpacePoint motion-tracking technology. The module incorporates a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer. These 9 outputs are internally processed using PNI's SpacePoint algorithm to provide calculated orientation data in the form of quaternions. The USB/HID interface seamlessly integrates with a user's system to provide calculated quaternion orientation data. Because the SpacePoint Fusion demonstration module incorporates a USB/HID interface, no drivers are necessary, and no batteries are required since the host computer powers the module via the USB port.

3 Getting Started

To use the SpacePoint Fusion demonstration module, simply plug it into your computer's USB port. It is important to ensure the module is fully at rest for 5 seconds when plugging it into the computer's USB port, as the gyros initialize during this time. Once the module is connected to the USB port, its red and green LEDs will light up.

If using Windows, and this is the first time plugging in a SpacePoint Fusion demonstration module, Windows will automatically launch the "Found New Hardware" wizard, then indicate it has found the module and install the device.

If the LEDs do not come on or Windows otherwise does not seem to recognize the device, check the Device Manager, under Universal Serial Bus Controller – USB Composite Device. The SpacePoint Fusion demonstration module should be identified with "Location 0 (SpacePoint Fusion)", and under the Details tab it should indicate "VID_20FF&PID_0100". Alternatively you can check under Game Controllers on the Control Panel, where it should be recognized as a "7 axis 2 button device". Here, under Properties, you should observe changes in the 7 axis states as the device is moved, although these 7 states are not correlated to their description.

That's about it. The device now will be streaming orientation data in the form of quaternions to your computer via the USB/HID interface. Things you can do next include:

- Run a PNI demo program to demonstrate the performance of the SpacePoint Fusion demonstration module (and have a little fun). Two web-based Unity demo programs are available at PNI's website, and instructions for how to use these are found in Section 4 of this manual.

- Observe or log sensor data or calculated orientation data. PNI's Fusion Game Pad program allows you to do this and can be downloaded at PNI's website. Instructions for logging data using the Fusion Game Pad program are found in Section 6.

- Develop code to integrate the SpacePoint Fusion demonstration module's output into your application. Section 5 includes an explanation of the streaming data at the USB/HID interface, while Section 7 provides sample hard-copy source code. Additionally, PNI's Fusion Game Pad project, found on PNI's website, provides a complete Visual C++ program to leverage.

4 Using the Demo Programs

PNI provides two demo programs that work with the SpacePoint Fusion demonstration motion-tracking module. These are the FusionPointer and Seagull Island programs. They are web-based Unity programs which have been shown to run on Windows XP, Windows Vista, Windows 7, and Macintosh OX operating systems, and should work on Linux systems. As these programs are free and principally intended to demonstrate the possibilities enabled with PNI's SpacePoint technology, no guarantees are made regarding their ability to operate on any specific computer configuration. Also note that performance can vary depending on the computer's graphics card.

The FusionPointer demo is well suited for demonstrating the capabilities of SpacePoint technology, while the Seagull Island demo is more for entertainment. This is because the FusionPointer demo provides an orientation rendering mode (mimic mode) and a pointing mode, both of which provide absolute orientation referencing. In contrast, the Seagull Island demo couples both heading and roll to control the seagull's orientation, which provides an intuitive feel for gameplay, but sacrifices the inherent absolute orientation aspect of SpacePoint technology

4.1 Getting Started with the Demo Programs

For Windows computers, ensure Microsoft's .NET Framework is installed. Please use dotnetx35setup.exe to install this package if it is not already installed. This can be downloaded at http://msdn.microsoft.com/en-us/netframework/cc378097.aspx. If it does not install correctly, uninstall all versions of Microsoft .NET Framework, then try again.

Demo programs are available at www.pnicorp.com/support/downloads/spacepoint-fusion. Download the desired demo program's zip file and extract it.

Plug the SpacePoint Fusion demonstration module into the computer's USB port prior to launching the demo program. It is important that the module is fully at rest for 5 seconds after plugging it into the computer's USB port, as the gyros initialize during this time. Once the module is connected to the USB port, its red and green LEDs will light up.

Launch a demo program by double-clicking the program's .html icon. With the program running, point the module at your computer display and press "P" on your keyboard to set the viewpoint. You can reset the rendered viewpoint at anytime by pressing "P" again.

In the upper left corner of the screen you will see a box containing a series of numbers. The row labeled "q" displays the instantaneous calculated quaternion values; the row labeled "acc" displays the calculated acceleration values, in g, for each axis; while "acc.magnitude" is the magnitude of the acceleration vector (~1.0 g when the module is at rest)..

4.2 FusionPointer Demo Program

The FusionPointer program provides a 3-D rendering of the SpacePoint Fusion demonstration module. By simultaneously pressing the module's right button and twisting the module about its X-axis, you can zoom in or out on the rendered image. (i.e. the image can be enlarged or shrunk.)

The FusionPointer program operates in either Mimic or Pointer mode, as described below.

- **Mimic Mode:** In Mimic mode the image can be rotated about its center such that the rendered module will closely mimic the orientation of the module. (This assumes the viewpoint is initially aligned by pressing the "P" button when pointing straight at the screen).
- **Pointer Mode:** In Pointer mode the image on the screen acts as if it were attached by a stick to the module, such that changing the orientation of the module results in the image translating on the screen. When operating in Pointer mode, the image will have a bright red "+" superimposed over it. Note that the image only responds to changes in orientation, and not changes in position.

Since the rendered image does not know the actual distance of the SpacePoint Fusion demonstration module from the screen, the length of the virtual stick is set by how far the image is zoomed out. PNI generally recommends fully zooming out on the object (making it as small as possible) when operating in Pointer mode.

The program will start in Mimic mode. To temporarily switch to Pointer mode, hold down the left button: when the left button is released the program will be back in Mimic mode. To semi-permanently put the module in Pointer mode, double-click the left button. To exit Pointer mode, press the left button once.

4.3 Seagull Island Demo Program

The Seagull Island program is a 3rd person game in which the player controls the flight of a seagull around an island. A few points about the program:

To turn the seagull, you can either turn the module in the horizontal plane (adjust yaw) or you can roll the module around its X-axis (similar to banking an airplane).

To go faster, press the module's left button. (Pressing the right button has no effect.)

The "3rd person eye" acts as though it is attached to an elastic cord which is attached to the seagull at the other end. Consequently, when you start you'll see the seagull from behind. When the seagull turns, you'll see the seagull from a different viewpoint, but if the seagull

then continues on a straight course you'll soon be viewing it from behind again. Likewise, pressing the left button to go faster causes the seagull to move well ahead of you, and releasing the left button allows you to catch up.

As previously mentioned, the Seagull Island program primarily is intended for entertainment purposes by providing an intuitive feel for gameplay. But, since it couples both heading and roll to control the seagull's orientation, it does not demonstrate the absolute orientation capabilities of SpacePoint technology as effectively as the FusionPointer demo program..

5 Interpreting SpacePoint Fusion's Output

The SpacePoint Fusion demonstration module is a USB composite device with HID Game Pad Report output on endpoint 0x82. The Game Pad Report includes 15 bytes of data. For additional information regarding USB, refer to the USB 2.0 specification and the USB HID specification.

The output includes scaled acceleration values converted into g's and orientation values given as quaternions. These are calculated from sensor data using PNI's proprietary Spacepoint algorithm, which is embedded in the module's firmware. Button status and PNI reserved data fields also are reported. Data is updated and output at 125 Hz and all data is in hex format.

The 15 bytes of data on endpoint 0x82 is presented in little Endian format as follows:

Table 5-1: Endpoint Outputs

Byte	Description
0	Scaled Accel X Lower
1	Scaled Accel X Upper
2	Scaled Accel Y Lower
3	Scaled Accel Y Upper
4	Scaled Accel Z Lower
5	Scaled Accel Z Upper
6	Quaternion [0] Lower
7	Quaternion [0] Upper
8	Quaternion [1] Lower
9	Quaternion [1] Upper
10	Quaternion [2] Lower
11	Quaternion [2] Upper
12	Quaternion [3] Lower (scalar)
13	Quaternion [3] Upper (scalar)
14	Reserved/Button

The status of the module's left and right buttons are packed into the 14th byte. The least significant bit (LSB) is the left button and the second LSB is the right button.

Table 5-2: Sample Endpoint Output

Data stream: 2f85238a5b90ac9f496a126a1ef8d0

Raw Data	2f	85	23	8a	5b	90	ac	9f	49	6a	12	6a	1e	f8	d0
Description	**Scaled AccelX**		**Scaled AccelY**		**Scaled AccelZ**		**q[0]**		**q[1]**		**q[2]**		**q[3]**		**Res/ B0/B1**
Hex	852f		8a23		905b		9fac		6a49		6a12		f81e		d0
Decimal	34095		35363		36955		40876		27209		27154		63518		-
Offset	32768		32768		32768		32768		32768		32768		32768		-
Counts	1327		2595		4187		8108		-5559		-5614		30750		-
Scale Factor	6		6		6		1		1		1		1		-
Scaled Counts	7962		15570		25122		8108		-5559		-5614		30750		-
Normal. Factor	32768		32768		32768		32768		32768		32768		32768		-
Value	0.2430		0.4752		0.7667		0.2474		-0.1697		-0.1713		0.9384		-

Where:

- "Offset" = the middle of the sensor's total range, or 32768.
- "Counts" = "Dec" - "Offset"
- "Scale Factor" = the 'g' rating of the accelerometer
- "Scaled Counts" = "Counts" * "Scale Factor"
- "Normal. Factor" = the normalization factor for the sensor, or 32768
- "Value" = "Scaled Counts" / "Normalization Factor"

6 Fusion Game Pad Sample Project & Data Logger

The FusionGamePad.zip file includes a complete Visual C++ project, with a functioning build in the "Release" folder. (The "Release" folder is created when the zip file is extracted.) The executable file in the "Release" folder can be used to display or log sensor or calculated data from the module. The overall project folder can act as a guide for generating new programs that utilize the SpacePoint Fusion demonstration module. Note that Fusion Game Pad was developed on a Windows XP platform.

To view the source code in native format, Visual C++ Express can be downloaded at www.microsoft.com/express/vc/. Conversely, much of the source code is provided in hardcopy format in Section 7.

The key resources and source code in this project are:

- FusionGamePad.cpp
- UsbHidApi.h
- UsbHidApi.dll
- UsbHidApi.lib

Visual C++ Tree

FusionGamePad.cpp is the main source code to read data from the module's USB interface. This data can be output to the display or logged to a text file. Hardcopy source code is provided in Section 7.1.

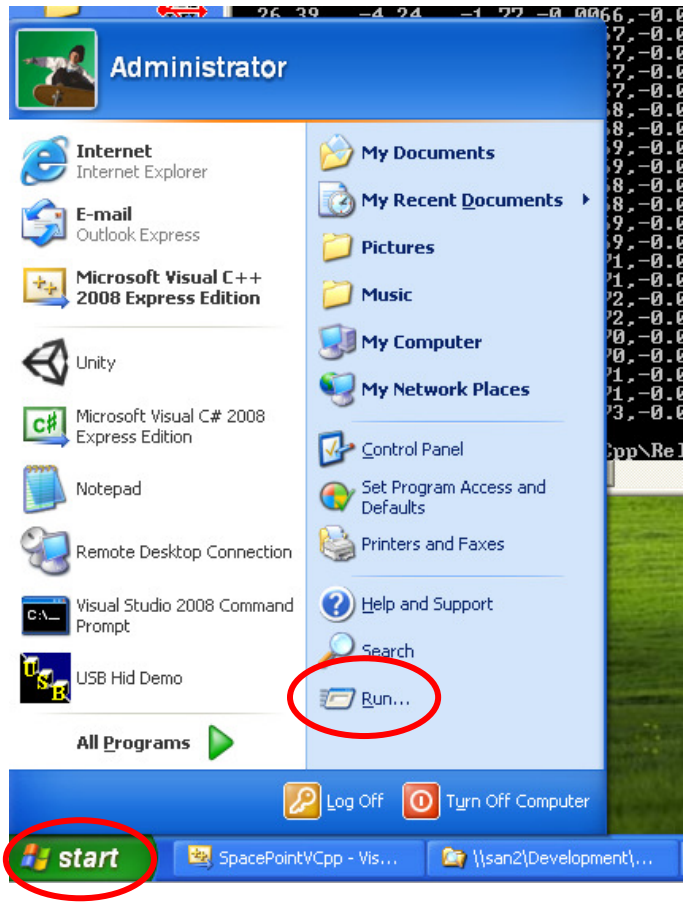
UsbHidApi.h includes the UsbHidApi library functions. Hardcopy source code is provided in Section 7.2.

6.1 Running Fusion Game Pad Data Logger

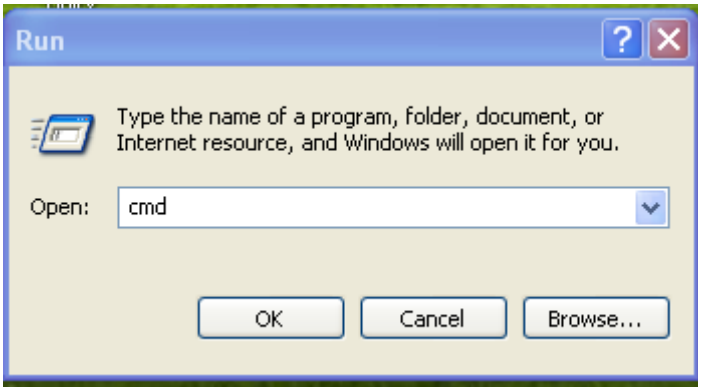
Instructions for running Fusion Game Pad follow. Note it is important to ensure the SpacePoint Fusion demonstration module is fully at rest for 5 seconds when plugging it into the computer's USB port, as the gyros initialize during this time.

- Open a Command window:

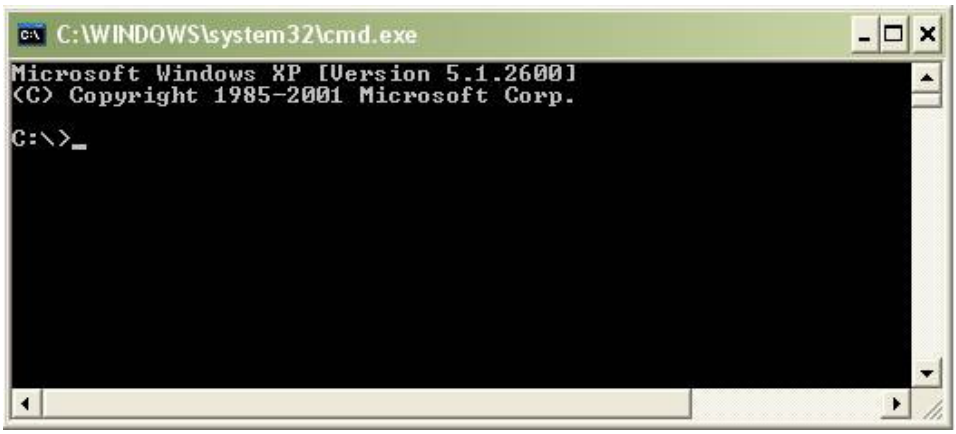
Click on "Start", then "Run".



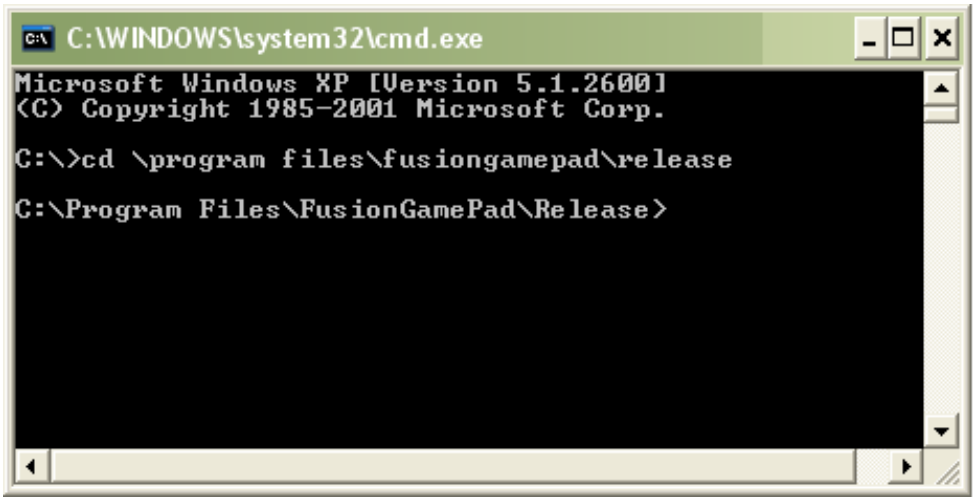
This will open the Run window. Type “cmd” then press <enter>.



This will open the Command window.



- In the Command window, type “cd \[file path]” to set the directory to the “Release” folder where FusionGamePad.exe resides.



- Run the executable file:

  In the Command window (and in the "Release" directory) type the following:

  FusionGamePad.exe [Nsample]

  This will display the results on the computer screen. To store the data to a log file use the pipe command ">>" as in the example below:

  FusionGamePad.exe 100 >> test.log

  This creates a file named test.log with calculated orientation data from 100 samples, and this file will be saved in the "Release" folder. The data will not be displayed at the same time it is obtained. This data can be imported into a spreadsheet if desired.

6.2 Interpreting Fusion Game Pad Output

The displayed and logged outputs have the same format, as shown below. The first line of output identifies the program, the second line is the header line for the data, and subsequent lines are the data.

Table 6-1: Sample Fusion Game Pad Log Output

AX	AY	AZ	Q0	Q1	Q2	Q3	LButton	RButton
0.9126	0.0837	0.1106	0.1061	-0.6336	0.0611	0.7639	1	0
0.9126	0.0837	0.1106	0.1061	-0.6336	0.0611	0.7639	1	0
0.9018	0.0782	0.1053	0.1059	-0.6336	0.0616	0.7639	1	0
0.9018	0.0782	0.1053	0.1059	-0.6336	0.0616	0.7639	1	0
0.9099	0.0837	0.1106	0.1055	,-0.6335	0.0621	0.7640	1	0

AX, AY, and AZ are the 3-axis normalized acceleration values in g. Q0, Q1, Q2, and Q3 are the normalized quaternion values. LButton and RButton represent the status of the module's buttons: "1" is pressed, "0" is not pressed.

7 Sample Code

7.1 FusionGamePad.cpp

The FusionGamePad.cpp defines the entry point for the console application. The sample code below demonstrates how to communicate with PNI's SpacePoint. It is provided "as is" without any express or implied warranty.

```
#include "stdafx.h"
#include <windows.h>
#include "UsbHidApi.h"
#include <math.h>
#include <iostream>

using namespace std;

const int DATALEN = 15;

void displayindata(unsigned char indata[DATALEN],int Nbytes){
   int i, byteindex = 1;
   int rawaxes[14];
   float acc_scaled[3];
   float q_scaled[4];
   int buttons[2];

   // Parse indata
   for (i = 0; i < 7; i++)
   {
         rawaxes[i] = (int)indata[byteindex] +
256*(int)indata[byteindex+1];
         byteindex += 2;
   }
   buttons[0] = indata[byteindex]&1;
   buttons[1] = (indata[byteindex]>>1)&1;
   rawaxes[7] = (indata[byteindex]>>4)&0xf;

   //16 bit raw values centered at 32768
   //acc_scaled = 6*(acc_received - 32768)/32768;
   for(i=0;i<3;i++)
         acc_scaled[i] = 6.0f* (rawaxes[i]-32768) / 32768.0f;
   printf("%7.4f,%7.4f,%7.4f,",acc_scaled[0],acc_scaled[1],acc_scaled[2]);

   //16 bit raw values centered at 32768
   //q_scaled = 3.0518e-005*(qraw - 32768)
   for(i=0;i<4;i++)
         q_scaled[i] = 3.0518e-005f*(rawaxes[3+i]-32768);

   printf("%7.4f,%7.4f,%7.4f,%7.4f,",q_scaled[0],q_scaled[1],q_scaled[2],q
_scaled[3]);
   printf("   %1i,\t%1i\n",buttons[0],buttons[1]);
}
```

```
int main(int argc, char* argv[])
{
   int connected = 0;
   int sample = 0;
   int mode = 0;
   int interfaceid = 0;
   int Nbytes = 0;
   int Nsamples = 10;
   int VID = 0x20ff;
   int PID = 0x0100;
   unsigned char indata[DATALEN];

   printf("\nPNI Corp, Fusion Game Pad data logger V3.00\n");
   if((argc != 2)&&(argc != 4)){
         cout << "\nusage: FusionGamePad Nsamples <VID:8447=0x20ff>
<PID:256=0x0100>\n" << endl;
   }
   else{
         Nsamples = atoi(argv[1]);
         if(argc>2){
               VID = atoi(argv[2]);
               PID = atoi(argv[3]);
         }
         //cout << VID << "\t" << PID << endl;
   }

   SetInterface(1);
   printf("\n   AX,     AY,     AZ,       Q0,    Q1,    Q2,    Q3,
LButton, RButton\n");

   if(connected = Open(VID,PID,NULL,NULL,NULL,1)){
         //printf("Connected\n");
         sample = 0;

         while(connected){
               if(sample < Nsamples){
                     if(Nbytes = Read(indata)>0){
                           displayindata(indata,Nbytes);
                           sample++;
                     }
               }
               else{
                     CloseRead();
                     connected = 0;
               }
         }
   }

   return 0;
}
```

7.2 UsbHidApi.h

This DLL provides the client application an easy method for accessing an HID device via USB link. The acts of reading from and writing to a USB device under Windows are significantly different and more involved than for other communication devices, such as serial comm ports. For this reason, PNI has encapsulated the complexity of the interface within a DLL. This DLL provides all the required functions for accessing the device, while hiding the details of the implementation.

The USB link is implemented as a Human Interface Device (HID) class function. As such, the DLL is dependent on the following Windows drivers:

- hidclass.sys
- hidparse.sys
- hidusb.sys

In addition to being USB-compliant, the host PC should have the latest versions of the drivers listed above.

The module was built using Microsoft Visual C++, 6.0 as a Win32, non-MFC DLL. Since there is no dependence on the Microsoft Foundation Class (MFC), there is no need to copy additional MFC DLLs to your Windows system folder. Also, no dependence on MFC means the size of the DLL itself is minimal. (Although the DLL does not internally use the MFC, the client application is not restricted. Exported functions in the DLL may be called by either MFC or non-MFC applications.)

The client application may link implicitly or explicitly with the DLL. In explicit linking, the client application makes function calls to explicitly load and unload the DLL's exported functions. In implicit linking, the application links to an import library (UsbHidApi.lib) and makes function calls just as if the functions were contained within the executable.

```
//
#ifndef __USBHID_API_H__
#define __USBHID_API_H__
#define USBHIDAPI_DLL_NAME "UsbHidApi.dll"

// The following ifdef block is the standard way of creating macros which
make exporting
// from a DLL simpler. All files within this DLL are compiled with the
USBHIDAPI_EXPORTS
// symbol defined on the command line. this symbol should not be defined
on any project
// that uses this DLL. This way any other project whose source files
include this file see
// USBHIDAPI_API functions as being imported from a DLL, whereas this DLL
sees symbols
```

```
// defined with this macro as being exported.
#ifdef USBHIDAPI_EXPORTS
#define USBHIDAPI_API __declspec(dllexport)
#else
#define USBHIDAPI_API __declspec(dllimport)
#endif

   // This structure defines an entry in a device list
   // The list describes the parameters associated with
   // a device.  It is mainly used with the GetList()
   // function.
   // modified 3/12/07
   typedef struct {
    char         DeviceName[50];   // Device name
    char         Manufacturer[50]; // Manufacturer
    char         SerialNumber[20]; // Serial number
    unsigned int VendorID;         // Vendor ID
    unsigned int ProductID;        // Product ID
    int          InputReportLen;   // Length of HID input report (bytes)
    int          OutputReportLen;  // Length of HID output report (bytes)
    int          Interface;        // Interface
    int          Collection;       // Collection
   } mdeviceList;

   // This structure was created to ease VB access.  It alleviates
   // some of the issues associated with inter-compiler data handling
   // such as alignment.
   // added 3/8/05
#pragma pack (push, 4)
   typedef struct {
    char         *DeviceName;   // Device name
    char         *Manufacturer; // Manufacturer
    char         *SerialNumber; // Serial number
    unsigned int VendorID;         // Vendor ID
    unsigned int ProductID;        // Product ID
    int          InputReportLen;   // Length of HID input report (bytes)
    int          OutputReportLen;  // Length of HID output report (bytes)
    int          Interface;        // Interface
    int          Collection;       // Collection
   } mdeviceList2;
#pragma pack (pop,4)


////////////////////////////////////////////////////////////////////////
/////////////////
   // These declarations define special non-member functions for VB
access.

////////////////////////////////////////////////////////////////////////
/////////////////

   extern "C" int _stdcall SetInstance(int instance);          // Set
object instance

   extern "C" int _stdcall GetLibVersion(char *buf);           // Get DLL
version string
```

```
   extern "C" void _stdcall ShowVersion(void);                  // Show a
message box containing version

   extern "C" int _stdcall Read(void *pBuf);                    // Read the
from the HID device.

   extern "C" void  _stdcall CloseRead(void);                   // Close
the read pipe

   extern "C" void  _stdcall CloseWrite(void);                  // Close
the write pipe

   extern "C" int _stdcall Write(void *pBuf);                   // Write to
the HID device

   extern "C" void _stdcall GetReportLengths (int *input_len,   // Pointer
for storing input length
                                              int *output_len); //
Pointer for storing output length

   extern "C" void _stdcall SetCollection(int);                 //
Specifies a collection (call prior to Open())  (0xffff = unspecified)

   extern "C" int _stdcall GetCollection();                     //
Retrieves collection setting (0xffff = unspecified)

   extern "C" void _stdcall SetInterface(int);                  //
Specifies an interface (call prior to Open())  (0xffff = unspecified)

   extern "C" int _stdcall GetInterface();                      // Rerieves
interface setting   (0xffff = unspecified)

   extern "C" int _stdcall Open (unsigned int VendorID,         // Vendor
ID to search     (0xffff if unused)
                                 unsigned int ProductID,        //
Product ID to search    (0xffff if unused)
                                 char         *Manufacturer, //
Manufacturer            (NULL if unused)
                                 char         *SerialNum,       //
Serial number to search (0xffff if unused)
                                 char         *DeviceName,   // Device
name to search   (NULL if unused)
                                 int          bAsync);          // Set
TRUE for non-blocking read requests.



   extern "C" int _stdcall GetList(unsigned int VendorID,       // Vendor
ID to search  (0xffff if unused)
                                   unsigned int ProductID,      //
Product ID to search (0xffff if unused)
                                   char         *Manufacturer, //
Manufacturer            (NULL if unused)
                                   char         *SerialNum,    // Serial
number to search (NULL if unused)
                                   char         *DeviceName,   // Device
name to search   (NULL if unused)
```

```
                                        mdeviceList2 *pList,        //
Caller's array for storing matching device(s)
                                        int          nMaxDevices);  // Size
of the caller's array list (no.entries)

// This class is exported from the UsbHidApi.dll
class USBHIDAPI_API CUsbHidApi {
public:

   // The serial number for the open device
   char  m_SerialNumber[20];

   // These variables define the required lengths for reading and writing
   // the device.
   unsigned short m_ReadSize;
   unsigned short m_WriteSize;

   // These variables define optional interface and/or collection values
search purposes
   // added 3/12/07
   int m_Interface;
   int m_Collection;

   // Constructor
   CUsbHidApi(void);

   // Destructor
   ~CUsbHidApi(void);

   // Get list of devices and their availability.  Caller must supply a
   // pointer to a buffer that will hold the list of structure entries.
   // Must also supply an integer representing maximum no. of entries
   // his buffer can hold. Returns total number stored.
   int GetList(unsigned int VendorID,      // Vendor ID to search  (0xffff
if unused)
               unsigned int ProductID,     // Product ID to search (0xffff
if unused)
               char         *Manufacturer, // Manufacturer
(NULL if unused)
               char         *SerialNum,    // Serial number to search
(NULL if unused)
               char         *DeviceName,   // Device name to search
(NULL if unused)
               mdeviceList  *pList,        // Caller's array for storing
matching device(s)
               int          nMaxDevices);  // Size of the caller's array
list (no.entries)

   // Opens a USB comm link to a HID device. Returns non-zero
   // on success or 0 on failure.  (Individual read and write handles
   // are maintained internally.)  If the caller desires to open
   // a specific HID  device, he must provide one or more
   // specifiers (i.e., vendor ID, product ID, serial number,
   // or device name.  If a successful open occurs, the function
   // returns TRUE.  It returns FALSE otherwise.
   int Open (unsigned int VendorID,            // Vendor ID to search
(0xffff if unused)
```

```
            unsigned int ProductID,     // Product ID to search
(0xffff if unused)
            char         *Manufacturer, // Manufacturer            (NULL
if unused)
            char         *SerialNum,    // Serial number to search
(0xffff if unused)
            char         *DeviceName,   // Device name to search   (NULL
if unused)
            int          bAsync);             // Set TRUE for non-
blocking read requests.

   // Sets an optional device interface ID (e.g., 0) for search purposes.
   // This method is used to pin-point a device to be opened.
   // Use this method when a USB device has multiple interfaces.
   // Must be called prior to calling the Open() method.
   // added 3/12/07
   void SetInterface (int iface);        // Interface (-1 if unused)

   // Returns the device interface ID that was set using SetInterface().
   // added 3/12/07
   int GetInterface (void);

   // Sets an optional collection ID (e.g., 0) for search purposes.
   // This method is used to pin-point a device to be opened.
   // Use this method when a USB device has multiple collections.
   // Must be called prior to calling the Open() method.
   // added 3/12/07
   void SetCollection (int col);        // Collection (-1 if unused)

   // Returns the collection ID that was set using SetCollection().
   // added 3/12/07
   int GetCollection (void);

   // Close the read pipe
   void CloseRead(void);

   // Close the write pipe
   void CloseWrite(void);

   // Read the from the HID device.  The number of bytes read is
   // determined by the input report length specified by the device.
   // Depending on how the device was opened, the call may perform
   // blocking or non-blocking reads.  Refer to Open() for details.
   // On success, the call returns the number of bytes actually read.
   // A negative return value indicates an error (most likely means
   // the USB cable has been disconnected or device was powered off).
   // (Note:  The first byte location is usually a report ID
   // [typically = 0].  The caller must account for this value in
   // the read buffer.)
   int Read(void *pBuf);                 // Buffer for storing bytes

   // Write a report to the HID device.  The number of bytes to
   // write depend on the output report length specified by the
   // device.  Returns number of bytes actually written.  A negative
   // return value indicates an error (most likely means the
   // USB cable has been disconnected or device was powered off).
   // (Note:  The first byte location is usually a report ID
```

```
   // [typically = 0].  The caller must ensure this value is
   // prepended to the buffer.)
    int Write(void *pBuf);              // Buffer containing bytes to
write

   // This function retrieves the lengths of input- and output-reports
   // for the device.  These values determine the I/O lengths for the
   // Read() and Write() functions.  The device must be opened prior
   // to calling this routine.  Alternatively, you can just use m_ReadSize
   // and m_WriteSize.
   void GetReportLengths (int *input_len,    // Pointer for storing input
length
                          int *output_len);  // Pointer for storing output
length

   // This function retrieves the current library version and populates a
string
   int GetLibVersion(LPSTR buf);

   // This function displays the current library version in a standard
message box
   void ShowVersion(void);


// Private (internal) declarations
private:

   // This function reads an input report from an open device.
   // The call will block until any number of bytes is retrieved,
   // up to the maximum of nBytesToRead.  On successful completion,
   // the function returns an integer representing the
   // number of bytes read (usually the input report length
   // defined by the device). (Note:  The read buffer must
   // be large enough to accommodate the report length.  This
   // number is located in m_ReadSize.)
   int ReadSync(void *pBuf);             // Buffer for storing bytes

   // Read input report from an open device.  If no data
   // is currently available, the call will not block,
   // but will return 0.    On successful completion,
   // the function returns an integer representing the
   // number of bytes read (usually the input report length
   // defined by the device).  The function returns -1 if
   // disconnect is detected.  (Note:  The read buffer must
   // be large enough to accommodate the report length.  This
   // number is located in m_ReadSize.)
   int ReadAsync(void *pBuf);             // Buffer for storing bytes

   // Read/write handles.  While a single handle could suffice for
   // both operations, 2 handles have been created to allow the client
   // application to utilize separate threads for reading and writing.
   HANDLE ReadHandle;
   HANDLE WriteHandle;
};

#endif // #ifndef __USBHID_API_H__
```

7.3 Quaternion to Euler Angles – C Code

Often it is desirable to have outputs in Euler angles (yaw, pitch, and roll), rather than as quaternions. The sample C Code below provides an example for how to do this conversion.

```
#include "stdio.h"
#include "stdlib.h"
#include "math.h"

#define PI     3.14159265

/*
This routine converts the q0 through q4 from the SpacePoint
Fusion to Yaw, Pitch, and Roll.
*/

struct ypr
{
   float yaw;
   float pitch;
   float roll;
};

//Inputs:      Pointer to quaternion array q from the device
//Outputs:     Struct ypr containing Yaw, Pitch, and Roll results.
struct ypr q2ypr(float * q)
{
   struct ypr result;
   float sqx, sqy, sqz, sqw;

    sqx = q[0]*q[0];
    sqy = q[1]*q[1];
    sqz = q[2]*q[2];
    sqw = q[3]*q[3];

    result.yaw = (180/PI)*atan2(2.0 * (q[0]*q[1] + q[2]*q[3]),(sqx - sqy -
sqz + sqw));

   result.pitch = (180/PI)*asin(-2.0 * (q[0]*q[2] - q[1]*q[3]));

   result.roll = (180/PI)*atan2(2.0 * (q[1]*q[2] + q[0]*q[3]),(-sqx - sqy
+ sqz + sqw));

   return result;
}

int main(void)
{
   //
   //4 elements for the quaternions read from the SpacePoint Fusion
   float q[3];
   //
   //Struct for the result
   struct ypr result;
```

```
    //
    //example input
    q[0] = -0.0055220;
    q[1] =      0.0278969;
    q[2] =      0.9983865;
    q[3] =      0.0491494;

    result = q2ypr(q);

    //example result
    //Yaw:    174.385117
    //Pitch:  0.788897
    //Roll:   3.162396
    printf("Yaw:\t%f\r\n", result.yaw);
    printf("Pitch:\t%f\r\n", result.pitch);
    printf("Roll:\t%f\r\n", result.roll);

    return 0;
}
```

7.4 Quaternion to Euler Angles – Matlab

Often it is desirable to have outputs in Euler angles (yaw, pitch, and roll), rather than as quaternions. The sample Matlab code provides an example for how to do this conversion.

```
function [y,p,r] = q2ypr(q)
% Q2DCM(Q) converts a quaternion into heading(y), pitch(p) and roll(r)
% Inputs: q1 through q4 from SpacePoint Fusion (in that order)
% Ouputs: Y,P,R

%example
if nargin<1
     q(1) = -0.0055220;
     q(2) =  0.0278969;
     q(3) =  0.9983865;
     q(4) =  0.0491494;
end

     sqx = q(1)*q(1);
     sqy = q(2)*q(2);
     sqz = q(3)*q(3);
     sqw = q(4)*q(4);


%example result
%
%y =
%    1.743851191837815e+002
%p =
%   0.788896841796396
%r =
%   3.162395846657551
```

```
    y = (180/pi)*atan2(2.0 * (q(1)*q(2) + q(3)*q(4)),(sqx - sqy - sqz +
sqw));

    p = (180/pi)*asin(-2.0 * (q(1)*q(3) - q(2)*q(4)));

    r = (180/pi)*atan2(2.0 * (q(2)*q(3) + q(1)*q(4)),(-sqx - sqy + sqz +
sqw));

end
```