USB Virtual Reality HID

Final Report

Students:

Weston Taylor

&

Christopher Budzynski

Project Advisor:

Dr. Malinowski

May 11, 2009

ABSTRACT

Modern video game controllers for the Nintendo Wii and PlayStation 3 use MEMS sensors for motion sensing. However, these controllers sense only basic and simplified motions and many users are not satisfied with them and desire more realism. The purpose of this project is to create a USB Human Interface Device (HID) motion sensing controller using low-cost MEMS inertial sensors. The primary goal is to show that MEMS inertial sensors are capable of supplying sufficient position data to provide a complex interactive experience.

The USB HID controller interfaces with a personal computer and its programs by emulating a USB gamepad. The HID handheld device translates user movements into on-screen actions to provide a realistic and lifelike interactive platform for PC games and other virtual environments. This project shows that MEMS inertial sensors are capable of providing accurate position data suitable for a complex interactive experience. However, it also shows that the abundance of software layers inherent to the USB protocol can cause interpretation problems when sending the position to the PC. In order to fully utilize the inertial sensors’ position data, the software must be written to accept absolute position data from the controller.

TABLE OF CONTENTS

INTRODUCTION

Motion sensing is an important and developing sector of the electrical engineering industry. Microelectromechanical system (MEMS) and other motion sensing devices are being used in the medical rehabilitation of partially paralyzed patients, in vehicles to deploy airbags and detect vehicle rollover, in industrial equipment for balance monitoring, and in motion sensing video game controllers. All of these applications use cost-effective MEMS inertial sensors to detect simple and uncomplicated motions, but some systems use expensive and more accurate inertial navigation sensors to sense absolute position and complex motions. Inertial navigation units (INU), which use costly and extremely accurate inertial sensors, are used in civilian and military aircraft to help stabilization and control during flight. The military also uses them to control missiles and provide more accurate GPS locations for its vehicles.

Currently, the video game industry is moving towards providing users with an increased level of realism and interaction with their virtual environments by providing motion sensing capable controllers. However, only basic motion sensing is provided. For example, the Wii controller cannot detect the difference between a swift flick of the wrist and a full baseball swing, and this disappoints many users who desire a realistic experience. Much research has been dedicated to investigating whether MEMS devices are capable of providing this increased realism at a practical and affordable cost.

GOALS

The overall goal is to use low-cost MEMS inertial sensors to translate user movements into on-screen actions to provide a more realistic in-game experience for the user. Here are our sub-goals to help us attain our overall goal:

- Translate inertial sensor readings into a 3-D position.
- Send 3-D position to PC using USB, emulating a USB gamepad.
- Process data and USB communication using low-cost 8-bit embedded systems.
- All embedded programming done using the C programming language, to provide portability and reusability of code.
- Establish wireless communication between subsystems (ZigBee, Bluetooth, etc.).

PREVIOUS WORK

Figure 1, next page, lists the patents and standards that are applicable to the design of the final product. No patent was found that exactly matched this project's proposed system design but there are multiple patents on similar motion sensing controllers. However, no patent seemed to restrict the final design of our project if it were to become a consumer product after further development. The standards that are listed correspond to the proposed standardized communications protocols that will be utilized in the final design.

Patent/Standard Number	Patent/Standard Description
USB 2.0	Universal Serial Bus (www.usb.org)
IEEE 802.15.4	ZigBee Wireless Network Protocol
US Patent 5139261	Foot-actuated computer game controller serving as a joystick
US Patent 6545661	Video game system having a control unit with an accelerometer for controlling a video game
US Patent 4514600	Video game hand controller
US Patent 6902483	Handheld electronic game device having the shape of a gun
11/313,050 (application number)	Advanced video controller system

Figure 1: Related Patents and Standards

FUNDAMENTAL ANALYSIS

This project's motion sensing HID is essentially a stationary Inertial Navigation System (INS). An INS is a computer based platform that uses motion-sensing devices to continuously calculate velocity and position by integrating the data received from the motion sensors. For our stationary platform, no linear accelerations will need to be integrated because the user's position is assumed to be a fixed location. However, angular accelerations such as pitch and yaw will be very important to the on-screen pointer location. Since gyroscopes measure angular acceleration, a 2-axis gyroscope will be used to measure the pitch and yaw of the HID controller system (roll will be ignored in this project).

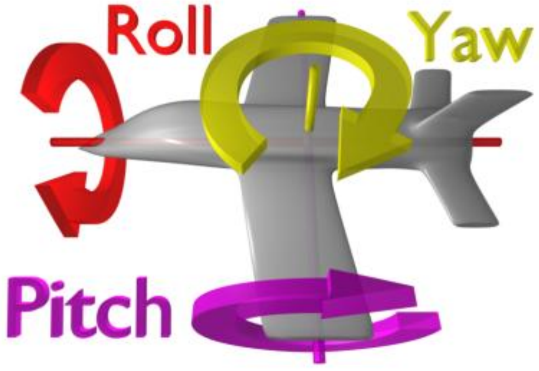


Figure 2: Rotational Axes
Source: ***Wikipedia [2]***

Gyroscopes measure angular acceleration in degrees/second, so in order to find information about the system's angular position the data must be integrated once. This integration will introduce integration drift, which means that small errors in the measurement of angular acceleration will accumulate over time and eventually compound into a large error in position. This position measuring technique is open loop (no position feedback) and the loop must be closed through some sort of augmentation to compensate for the accumulated error. Since our inertial system is operating around a fixed equilibrium position, the linear acceleration components are not being used. This means that true angular measurements can be made using a

2-axis accelerometer (when the system is relatively stable) which can be used to close the loop and compensate for drift on the pitch axis. This technique cannot be used for the yaw axis because it is parallel to the acceleration vector of gravity of the reference system. So in order to make the yaw axis' position measuring system closed loop, we will use an electronic compass that provides absolute heading information. If the compass is not tilt compensated, then yaw axis drift can only be corrected when the HID controller is relatively level in the reference system (parallel to the floor).

Finally, in order to measure angles relative to the reference system (the earth's surface), the gyroscope data must be mathematically transformed from the controller's co-ordinate system to the earth's three dimensional reference co-ordinate system. This will involve trigonometric functions and math intensive routines. However, if the roll axis is ignored then the only axis that needs to be transformed is the yaw axis (note: this is a valid assumption because almost all games ignore the roll axis). For example, when the controller is not completely level (has some pitch angle), the yaw axis gyroscope will no longer read the full yaw component. The yaw component is now [yaw / cos(pitch angle)].

SYSTEM DESCRIPTION

This section introduces our system and some of the essential design requirements that we developed.

System Requirements

The first step in the design process was to determine what restrictions and designs limitations we needed to fill. We ran a few experiments to find a human's maximum movement velocity. First, we moved our arms through 90° and 180° without weight or hindrance. We found we could move around 350°/s to 400°/s. When we held something comparable to the controller, we found our speed decreased to about 300°/s. Another concern was the position calculation time of the system. Human reaction time is around 100 ms (10 Hz) so in order to ensure that all movements are captured and no lag occurred between user movement and screen position movement, we needed a system that calculated position at least every 10 Hz. Another consideration we took into account was the average screen refresh rate of video games. Most monitors and televisions refresh at 60 – 80 Hz; therefore, to ensure that our position calculation was current, we wanted to update it at least once for every 80Hz screen refresh (every 12.5ms).

Here is a short list of the system requirements outlined above:

- Inertial sensors must measure up to 400°/s without saturating.
- Position calculation must be faster than the human reaction time (100ms, 10Hz).
- Position calculation and transmission to the PC over USB must be faster than monitor refresh rates (80Hz, 12.5ms).

System Block Diagram

Our project consists of two main subsystems. The first subsystem is the handheld controller which translates the user's movements into a 3-D position. The second subsystem is the USB communication board which sends the position to the host PC. See Figure 3, on the next page, for a basic representation of how data flows through our system.

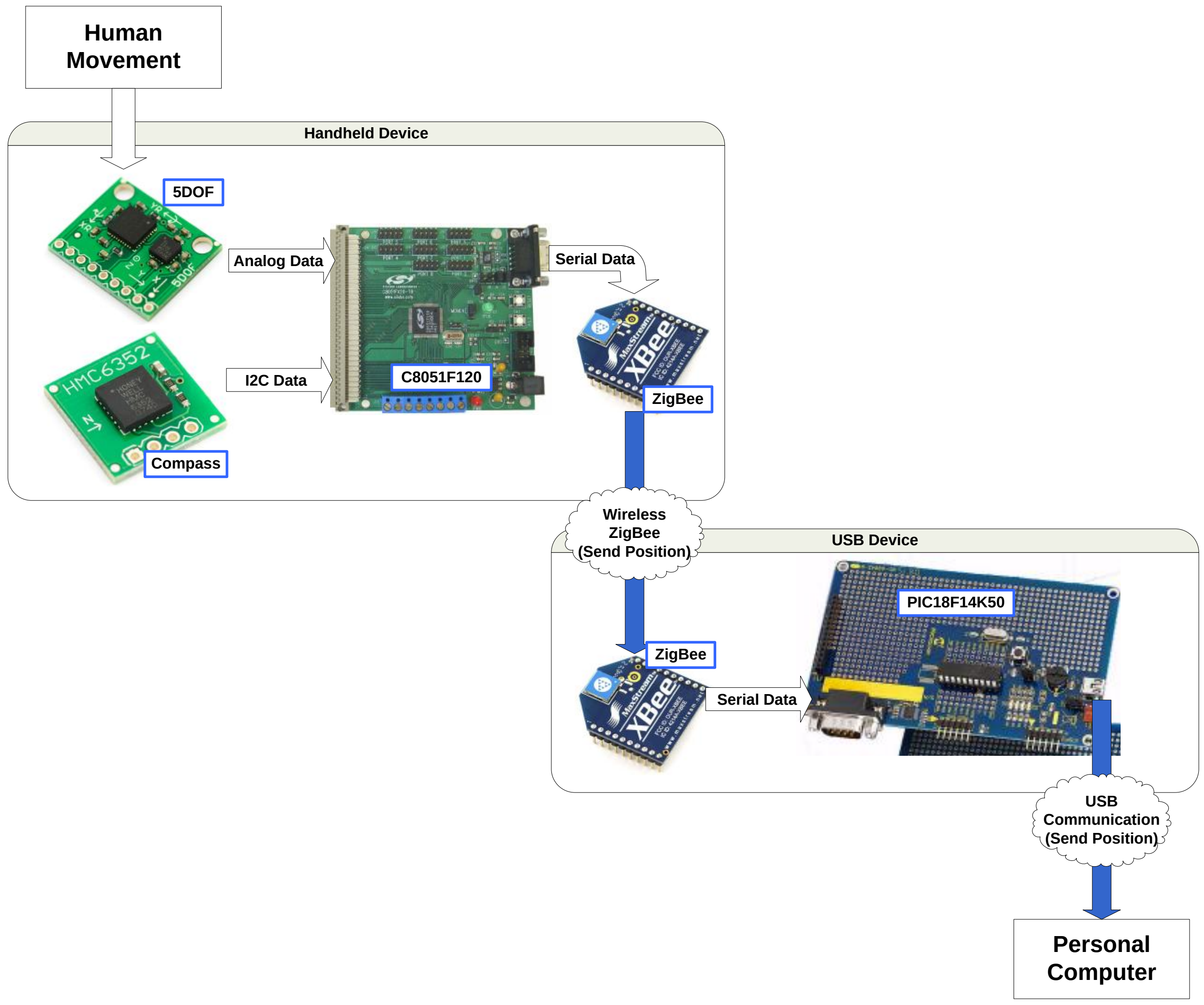


Figure 3: System Block Diagram

The handheld controller takes human movements as an input. The 5DOF inertial sensor (which has a 2-axis gyroscope and a 3-axis accelerometer) and the electronic compass convert those movements into analog / I2C signals for the Silicon Lab's C8051F120's 8-bit microcontroller. The F120 converts the signals into a 3-D angular position and then sends this position to the USB board using serial UART communication. The ZigBee chips convert UART serial transmissions into the ZigBee protocol for easy wireless implementations.

USB communication with the host PC was established using Microchip's LPC USB Development Kit. This board receives the position data from the handheld controller subsystem as UART data from the ZigBee chip, and translates it into data that is compatible with the USB protocol. Finally, it sends the position over USB when it is polled by the host PC.

CONTROLLER / INERTIAL SUBSYSTEM

The inertial subsystem is the actual handheld controller that the user manipulates to interact with the game. In this section, we will outline the sensors that we chose and the problems they presented.

Our MEMS Sensors

The final sensor we chose was the IMU 5 Degrees of Freedom from www.sparkfun.com. This sensor came prepackaged with a 2-axis gyroscope (no roll) and a 3-axis accelerometer. The gyroscopes can measure up to 500°/s, which is high enough to measure any movement the controller would experience and small enough that high resolution was available. The 3-axis accelerometer can measure ± 3g, which is sufficient because it only needs to be able to measure gravity (up to 1g) on two of the axes (just used to calculate pitch). Both the gyroscope and the accelerometer had analog outputs with low pass filters already soldered to their output pins.

We did not get the electronic compass working for yaw feedback, but we chose the HMC6352 a cheap electronic compass. It has a 20Hz update rate and communicates using the I2C protocol. It was not tilt compensated so we were unable to use it while the system was not level.

Inertial Sensor Problems and Solutions

While selecting our sensors, we found several projects that used similar technology, and they warned about some common problems associated with MEMS gyroscopes and accelerometers. The three major problems were variations in sensor rate scales, high frequency noise, and offset voltage fluctuations (leading to drift).

The first problem we encountered was variations in sensor rate scales from what the datasheet listed. Scale factor variation was an issue in the gyroscopes because the positive and negative axes read different voltages for the same rotational velocity. These variations lead to an accumulation of error in our position calculation, but only while the sensor was moving. If the sensor was not moving, then scale factor variations did not cause any drift error. In the end, we decided to use what the datasheet listed for the scales and provided position feedback methods to get rid of the accumulated position error.

The second problem was the large noise to signal ratio involved in angular rate sensors. It was suggested that a 100 Hz low pass filter be applied to the analog output to remove noise yet still allow information into the system. Our final sensor package included a pre-designed 80 Hz low-pass filter (LPF) on the gyroscope outputs and a 500 Hz LPF on the accelerometer outputs. These filters eliminated our system's high frequency noise problem. Figure 4, next page, shows the raw analog signal that the F120 was receiving from a stationary gyroscope (after the LPF). The signal is still fairly noisy and it has a voltage offset of about 1954 A/D steps.

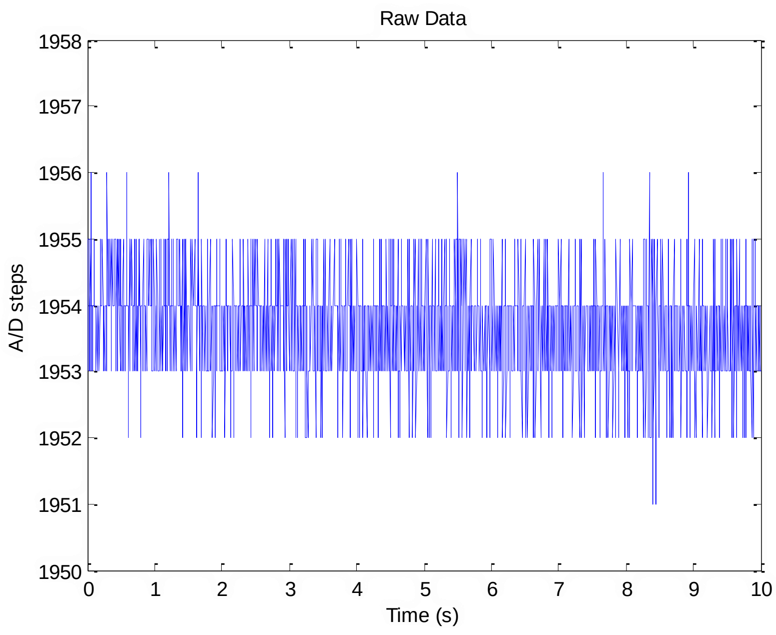


Figure 4: Raw Sensor Data

The final problem was offset, zero-rate, voltage fluctuations that caused the center voltage to vary even if the sensor was not moving. Offset variations caused drift problems for both our gyroscopes and accelerometers. Our solution was to create calibration routines for each axis that accounted for individual variations by taking a 1024 sample average. We also used an adaptive filter that accounted for long term changes in the sensors' zero-rate voltage by computing a running average. The results can be seen in Figure 5 below.

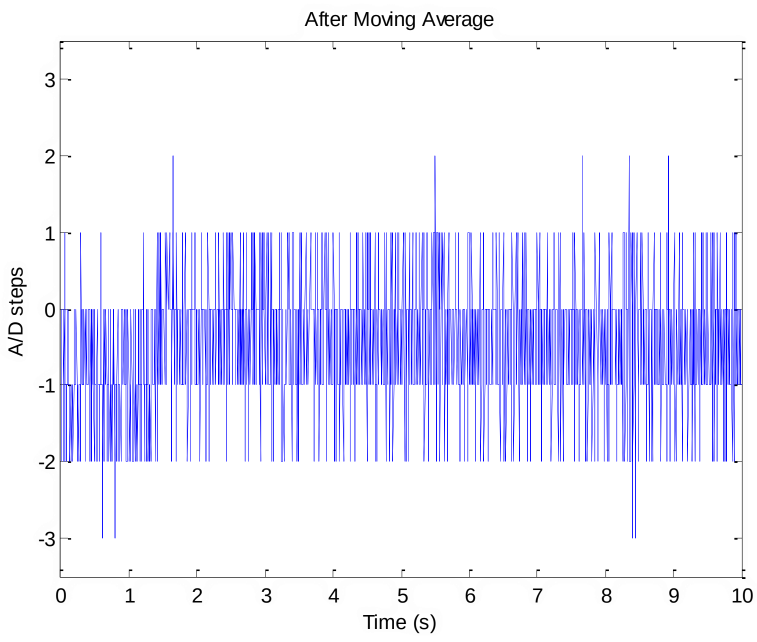


Figure 5: Sensor Data after Calibration and Adaptive Zero-Rate

From Figure 5, we see that the offset is now gone, but the signal is still fairly noisy which would lead to position drift error over time if the data was integrated. To eliminate a majority of this noise, a window filter was used that ignored all data with an amplitude of less than 3 A/D steps, which is equivalent to ±1.6mV or ±0.8°/sec. The signal after the window filter can be seen in Figure 6 below. The signal is now extremely clean and only leads to a drift of approximately 0.1 degrees over the 10 second interval shown.

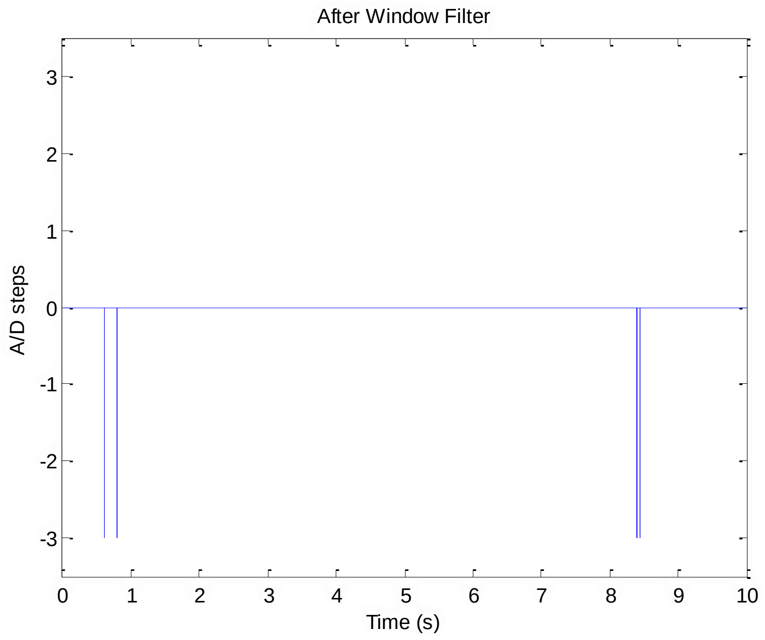


Figure 6: Sensor Data after Window Filter

POSITION CALCULATION

One of the most important goals of the project was to calculate a position in 3-D space using our MEMS inertial sensors.

Calculation Theory

For our design, we decided to calculate the absolute position of the controller, because it would allow the game to position the cursor exactly where the player aimed the controller. Since video games only allow the user to move the cursor in two directions, pitch and yaw, we only needed to calculate two absolute position angles. Gyroscopes are designed to measure rotational velocity, but we need to calculate our angular position. To do this, we needed to perform an integration on the velocity data measured by the sensor. We analyzed several approaches to taking a software integration. Initially, we looked at the different z-transforms of a mathematical integration. Next, we looked at the mathematical proofs and discrete representations for integrations. Eventually, we decided to use the Backward Rectangular Rule (Figure 7 next page) because it provided the simplest equation. Our equation approximates an integration by summing the rate data divided by the time interval between measurements. This equation is widely used in control applications, and the other more complex equations did not add any extra capabilities to our design.

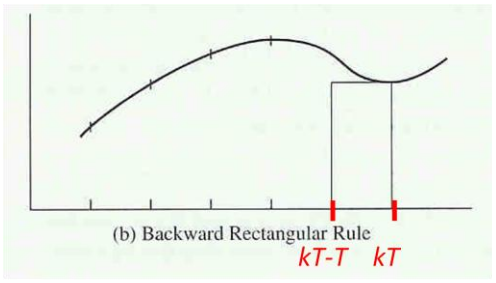


Figure 7: Backward Rectangular Rule
Source: ***www.control.aau.dk/~jsat/Teaching/AnaDig1/AnaDig1mm3.ppt***

Feedback Theory

Our gyroscopes accurately calculated position from the controller's movement, but because the noise was accumulated in our summation alongside the pure data, our position drifted slowly over time even while the sensor was at rest. To fix this problem, we needed to find a way to calculate our angular position that did not rely on angular velocity information from the gyroscopes. We needed a method to determine absolute pitch and yaw angles using forces that were external to our system. For pitch we decided to use Earth's gravity as a constant, and for yaw we used Earth's magnetic field. Accelerometers are capable of measuring an absolute pitch angle when a system is at rest, because it can measure how gravity is affecting its axes. Therefore, we used two axes of an accelerometer to calculate an absolute pitch angle with an arc tangent function. For yaw we used a compass, which would read exactly where the controller was aimed, and could correct the position. However, the compass readings could only be used if the system was relatively level, parallel with the earth's surface.

Position Calculation in Software

We decided to write all of our embedded software in C for clarity and portability of code. Our sensor platform had two main sections of code; the first was serial communication with the USB microprocessor, and the other was the 12-bit analog-to-digital converter (A/D) interrupt. We also had to calculate constants and conversion factors in order to be able to perform the software integrations. The actual code is not shown below, but we provided all of the code on a CDROM submitted with the report.

Main Function Loop

Figure 8, on the next page, shows the flowchart for the main() function loop. The program starts with its initialization routines. Next, the program checks for a flag that is set inside the interrupt that indicates when all sensors have been read and the position calculation has been completed. If the flag is set, the position data is converted to its ASCII equivalent and placed in the UART buffer to be sent to the USB board. Interrupts are disabled for a short period of time while the data is being collated to ensure that all the data is from one position calculation instead of two separate ones. The program then sends the data over UART and waits until the next position calculation is complete.

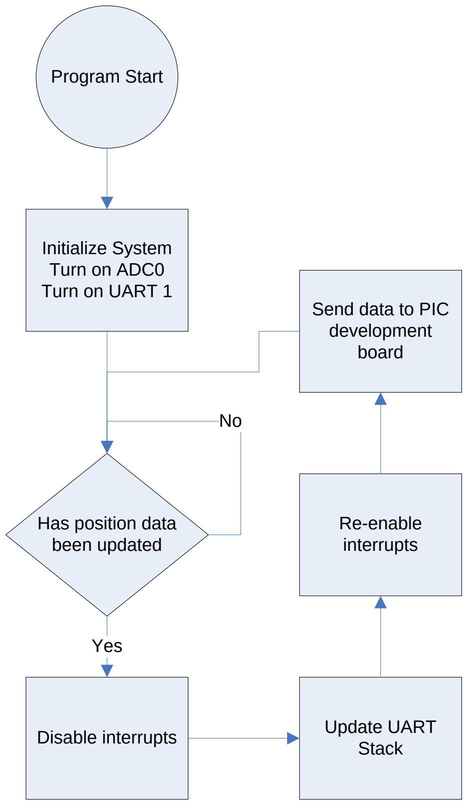


Figure 8: Main Loop Flowchart

A/D Interrupt

The A/D interrupt routine contains all of our calculations for position as well as our feedback error corrections. The 12-bit A/D on the Silicon Lab's F120 board has 8 analog inputs, and we used four of them by continuously switching between channels on every interrupt. The interrupt routine runs at 800 Hz, but only one channel is collected each time so all of the sensor data is collected and the position is calculated at 200 Hz (every 5ms).

The design of the A/D interrupt was complex because we had to ensure that the data was collected in the correct order so that all functions had the proper data to complete calculations. Figure 9, on the next page, is the flowchart for our A/D interrupt routine. The first data collected was gyroscope data, specifically pitch, so subsequent functions would be able to use pitch position data to fix yaw frame of reference. The second data collected was accelerometer and compass information for feedback. The data from the accelerometers is saved until the next cycle. After all the data is collected, the program checks if the controller was still for 10 cycles. If so, the newest accelerometer and compass data is collected and used to correct the drift error that the gyroscope data has accumulated. Once all the data is collected and the position is calculated, the program exits the interrupt and returns to the main module.

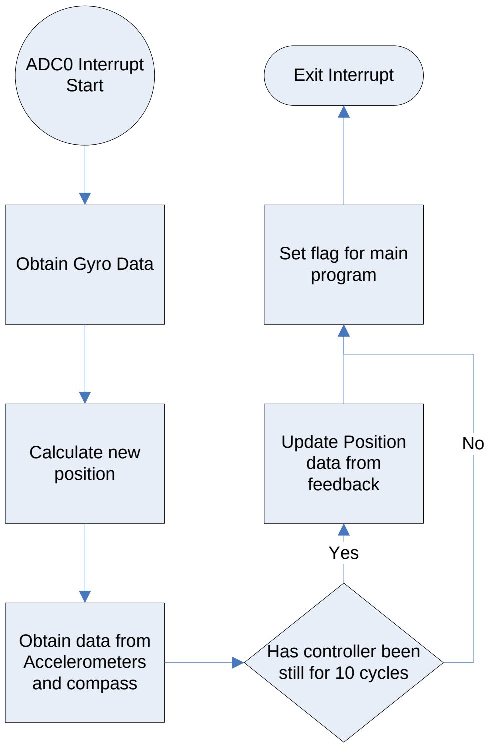

ADC0 Interrupt Start
Obtain Gyro Data
Calculate new position
Obtain data from Accelerometers and compass
Has controller been still for 10 cycles
Yes
No
Update Position data from feedback
Set flag for main program
Exit Interrupt

Figure 9: A/D Interrupt Flowchart

Software Integration

To use the gyroscopes properly and calculate a position we needed to find two calculations. The first was the constant used to convert from sensor voltage to °/sec/step. Figure 10 below shows the constants we combined to find the proper conversion factor. On the left is the conversion from the 12-bit A/D with a reference voltage of 3.3 V. Next is the scale factor from the gyroscope's data sheet. The two values on the right are the final constants used in the code.

$$\frac{3.3\ \text{V}}{4096\ \text{steps}} \text{X} \frac{1°/\text{s}}{2.0\ \text{mV}} = \frac{1500°/\text{s}}{4096\ \text{steps}} = .402832$$

Figure 10: Conversion Constant – Voltage to °/sec/step

The second calculation we needed was the integration to find position from the angular velocity that the gyroscopes measure. Figure 11, next page, shows the mathematical formula we used to approximate an integration in software. The indefinite integral on the left is the exact equation, but it is impossible to use on a microprocessor with sampled data. So we used an approximation known as the backward rectangular rule that adds each new sample to a sum and multiplies it by the time between samples. Because we used the timer to sample the sensor at 200 Hz, Δt is .005, and multiplying Δt with the conversion constant produces 1/496. The final result on the right shows that the value added to the sum is the data sample divided by 496.

$$\int .4028(\mathrm{ADC})dt \approx \sum .4028(\mathrm{ADC})(\Delta t) = \frac{\mathrm{ADC}}{496.485} + \mathrm{ADC}_{\mathrm{sum}}$$

Figure 11: Integration and Summation Representation

Throughout most of the project, we were running the Silicon Labs F120 board at 98MHz to ensure that the trigonometric and position calculations would be completed before the next interrupt. However, at the end we ran tests to see how much of the CPU time was actually being used and if we could slow the clock speed to save power. We discovered that the position calculation did not take very much CPU time and we were able to successfully turn the clock down to 6MHz (see lab book for more details).

USB SUBSYSTEM

The sections below describe how we used USB to send our 3-D position to the computer and the problems we encountered.

Basics

We decided to use USB to communicate our position data to the computer for several reasons. First, USB is a widely used standard and it comes on all modern personal computers and video game systems. Also, most PC video game controllers use USB because it allows them to use Window's standard device drivers to communicate with video games.

We decided to create a USB gamepad which is part of the Human Interface Device (HID) class. We could have setup our system to emulate both a mouse and a keyboard, but if we had done this movement of the controller would have led to movement of the mouse even when a video game was not running. By emulating a gamepad, the cursor location is only altered by the controller inside the video game's virtual environment.

The USB HID device class is a very flexible class. The developer of a HID device gets to decide the data that he or she will send to the computer. This class uses what is called a report descriptor to tell the computer what data it will send and how the computer should use it. For our report descriptor, we told the computer that we would be sending Rx, Ry, and 6 buttons. Rx corresponds to rotation around the computer's x-axis, which is pitch for us, and Ry is rotation around the computer's y-axis, which is our yaw. Our report descriptor is shown in Appendix C.

Problems

USB presented us with many problems throughout this project. From our initial research, we believed that it would be possible to send an absolute position to the computer. However, as we began testing this theory, we quickly discovered that an abundance of software layers was distorting the absolute position and the way it was being interpreted. Figure 12, next page, is a visual representation of all the software layers that the position data must travel through.

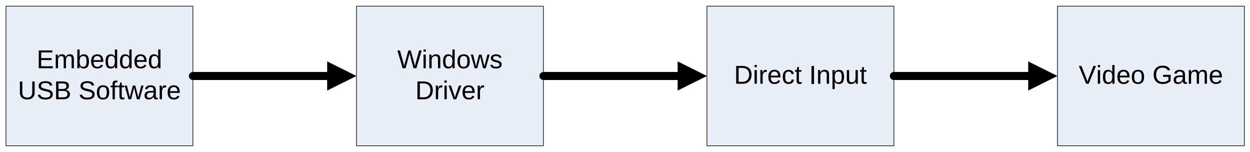


Figure 12: Software Layers

Eventually, we discovered that Direct Input was transforming the absolute position data into its derivative. Instead of sending an absolute position, it was sending the absolute number of units that we wanted to move from our current location. We made some adjustments to our software to handle this, but the problem was that we no longer had any feedback from the computer as to where the cursor was positioned. Without this feedback, there was no way to correct the error that was accumulated by sending the derivative of absolute position.

Another problem we encountered was that Direct Input only polled the Windows driver for our gamepad position data every 50ms. However, our Microchip USB board device was being polled by the computer every 10ms, so we had to program our USB board to send the same position 5 times in a row to ensure that the computer received the new position change. This also significantly increased the delay of the system since the on-screen cursor location could not update faster than 20Hz (50ms).

The final problem was that Direct Input has a resolution of only about 1.75 degrees, which is much less than our position data with a resolution of 1 degree. If the position change was less than this amount, Direct Input would ignore the cursor position movement. Because of this, we had to implement a routine in our USB code that accumulated the change in cursor position data until this threshold was met, and then we would send the new position to the computer.

Software

The USB module was another complex program that required a very specific series of events. The flowchart, Figure 13 next page, shows the steps used to send the data to the PC. Once the system is initialized, the program begins the standard USB communication through Microchip's USB API. Because the API runs in the background, the user only has to update the USB buffers and call a function that sends those buffers over USB. In order to properly collate the data, the program runs through a state machine that collects the bytes sent over UART, and converts the ASCII values back to integers. Then the program calculates the derivative for pitch and yaw and updates the USB buffers accordingly. A temporary buffer is updated each time the program runs, but the USB buffers only change if the previous position data has been sent to the PC five times.

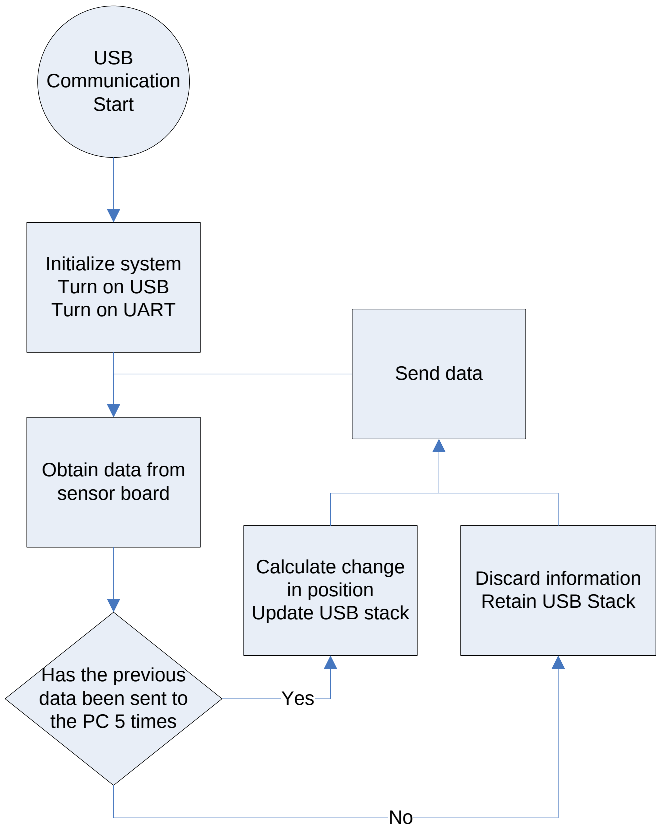


Figure 13: USB Flowchart

WIRELESS (ZIGBEE)

One of the goals of the project was to establish wireless communication between the handheld controller and the USB board. The wireless functionality provided freedom to the controller and the user by eliminating all of its wires.

The XBee chips that we used converted UART communication data into the ZigBee protocol. This conversion was completely transparent to the controller and USB boards. To the subsystems, it appeared as if they were communicating over a UART cable at 9600 baud. This allowed us to quickly and easily establish a wireless communication method.

We only encountered one problem. Initially, we wanted to change the XBee chips' default setting of 9600 baud to the 115200 baud that we had been using all semester. We were unable to correctly reconfigure the XBee chips' settings, but we realized that 9600 baud was sufficient for our communication needs. At 9600 baud, our communication would be completed within 10ms, and Direct Input was only accepting position updates every 50ms.

ANALYSIS OF RESULTS

Our project was a success because we were able to show that low-cost MEMS inertial sensors and 8-bit microcontrollers are capable of providing accurate 3-D position data. We were able to design a system that had no drift at rest, and accurate position to within a degree for both pitch and yaw. We also confirmed that even if drift becomes noticeable, absolute information can be used as feedback to eliminate accumulated error.

Our inertial subsystem was working correctly as described above, but we had problems sending this position data to the video game. We could send it to the computer easily, but the abundance of software layers was causing the position data to be translated incorrectly. We were eventually able to overcome this obstacle by sending the derivative of the absolute angular position, but by doing this we lost all positional feedback from the computer and video game. The computer would accumulate position error over time inside the video game, even though the controller was still sending correct position data.

CONCLUSION

We hope that this project is an inspiration for future senior projects to continue perfecting our motion sensing controller. We believe that the software layer problems could be fixed by writing a specialized Windows driver that is designed to properly collect and transmit absolute position data. Also, it may be necessary to create patches to video games or Direct Input to ensure that they use the absolute position data correctly. Finally, our current system does not allow for spatial movement throughout the environment. Integrating a footpad subsystem would add much needed functionality that allows the user to walk throughout the virtual environments while they use our controller to look around. Our project provided a major step towards creating a fully interactive system.

REFERENCES

[1] “C8051F34x Data Sheet,” Silicon Labs,
https://www.silabs.com/products/mcu/usb/Pages/C8051F34x.aspx

[2] “Inertial Navigation System,” Wikipedia,
http://en.wikipedia.org/wiki/Inertial_navigation_system

[3] “Inertial Sensors and Systems an Introduction,” GeneSys Engineering Department,
http://www.genesys-offenburg.de/genesyse.htm

[4] EE565 Fall 07 Lectures Notes 20 – 24, D. Schertz, Bradley University

[5] Device Class Definition for Human Interface Device (HID) Version 1.11
http://www.usb.org/developers/hidpage/

[6] HID Usage Tables Version 1.12
http://www.usb.org/developers/hidpage/

[7] Churavy, C., et al. “Effective Implementation of a Mapping Swarm of Robots.” IEEE Potentials July/Aug. 2008: 28-33.

[8] “Sensor Comparison.” Motus Bioengineering. 22 Feb. 2009
http://www.motusbioengineering.com/

APPENDIX A: EQUIPMENT LIST

Figure A-1 is a list of materials used in the project:

Equipment	Part Description	Quantity	≈ Cost	Supplier
Personal Computer	With USB and Half-Life 2	1	$0.00	Personal Laptop
USB Board	Microchip LPC USB Dev. Kit	1	$60.00	www.microchip.com
Main Board	Silicon Labs C8051F120 Dev. Kit	1	$99.00	In Lab
Gyro + Accelerometer	IMU 5 Degrees of Freedom SEN-00741	1	$100.00	www.sparkfun.com
Electronic Compass	Compass Module - HMC6352 SEN-07915	1	$60.00	www.sparkfun.com
Level Converter	Logic Level Converter BOB-08745	1	$2.00	www.sparkfun.com
Wireless / ZigBee	XBee 1mW Chip Antenna WRL-08664	2	$25.00	www.sparkfun.com
		Total Price:	$346.00	

Figure A-1: Equipment List

APPENDIX B: HARDWARE SCHEMATICS

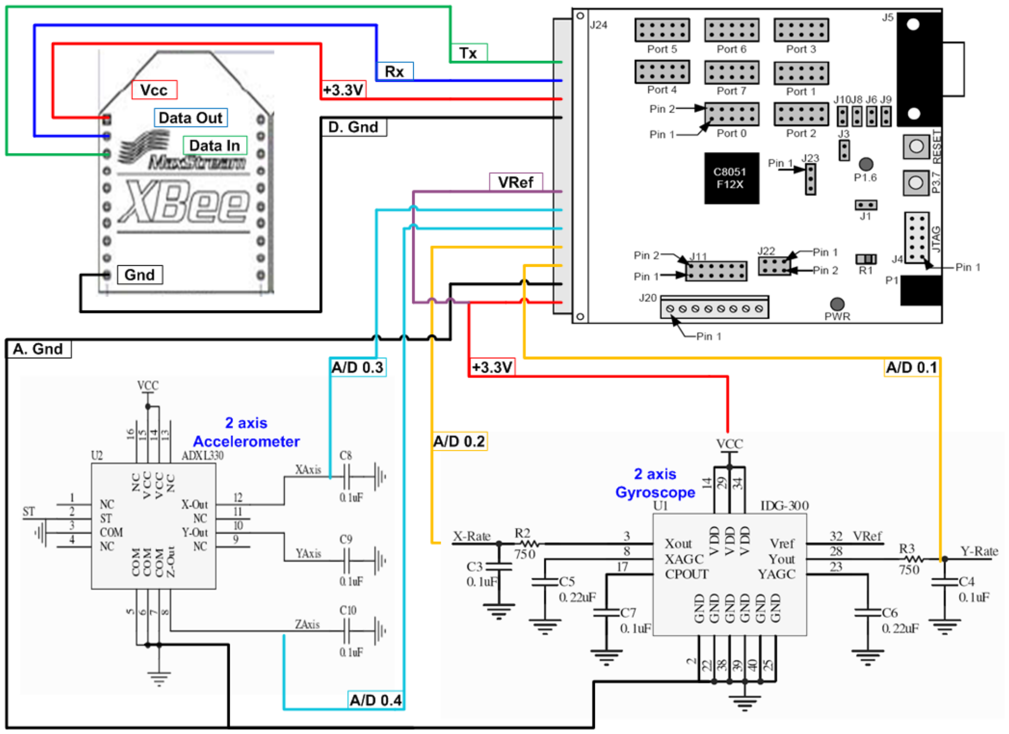


Figure B-1: Inertial Board Schematic

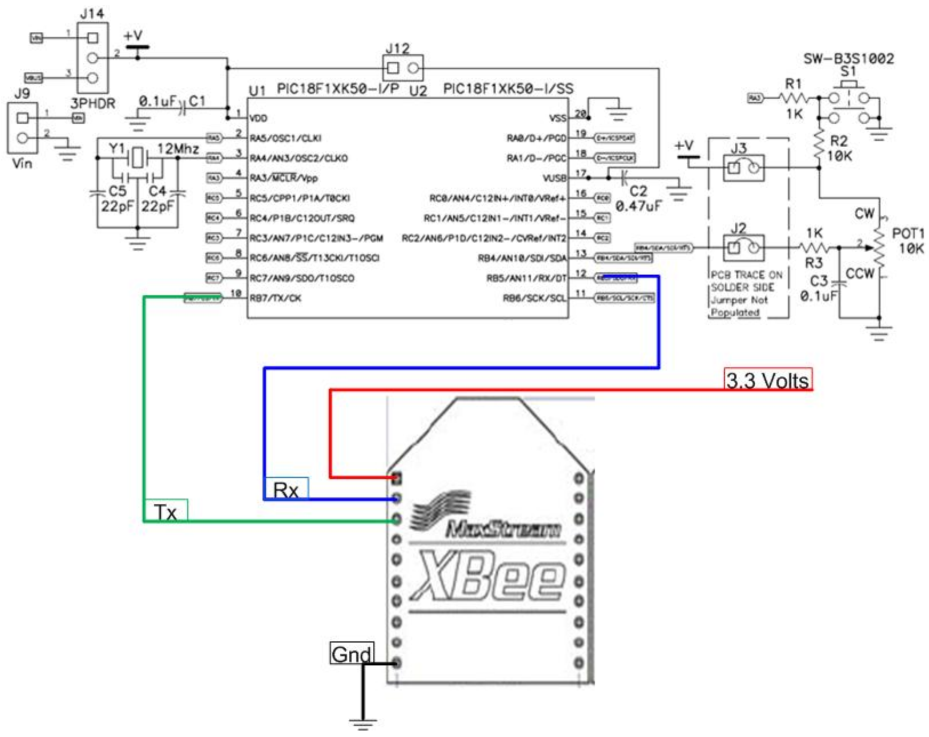
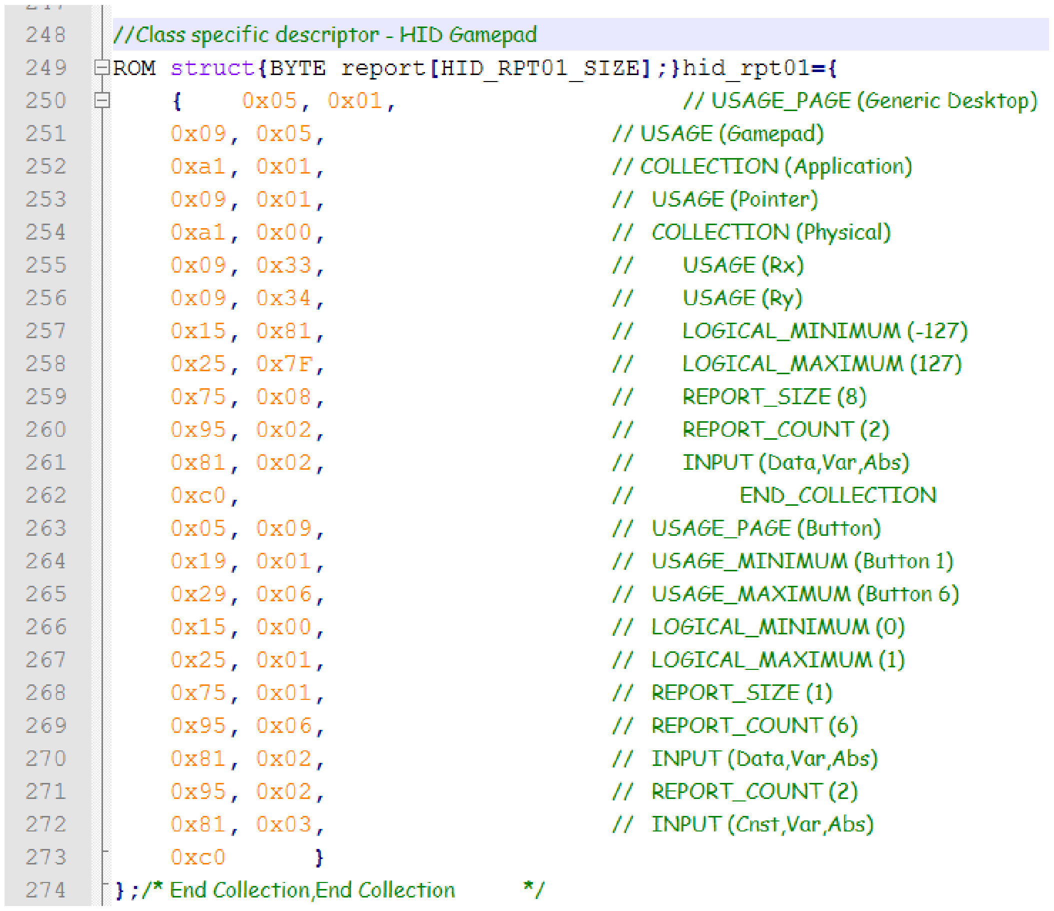


Figure B-2: USB Board Schematic

APPENDIX C: REPORT DESCRIPTOR

Figure C-1 is the report descriptor that we used for our USB HID Gamepad.



```
//Class specific descriptor - HID Gamepad
ROM struct{BYTE report[HID_RPT01_SIZE];}hid_rpt01={
    {     0x05, 0x01,                        // USAGE_PAGE (Generic Desktop)
    0x09, 0x05,                        // USAGE (Gamepad)
    0xa1, 0x01,                        // COLLECTION (Application)
    0x09, 0x01,                        //   USAGE (Pointer)
    0xa1, 0x00,                        //   COLLECTION (Physical)
    0x09, 0x33,                        //     USAGE (Rx)
    0x09, 0x34,                        //     USAGE (Ry)
    0x15, 0x81,                        //     LOGICAL_MINIMUM (-127)
    0x25, 0x7F,                        //     LOGICAL_MAXIMUM (127)
    0x75, 0x08,                        //     REPORT_SIZE (8)
    0x95, 0x02,                        //     REPORT_COUNT (2)
    0x81, 0x02,                        //     INPUT (Data,Var,Abs)
    0xc0,                              //          END_COLLECTION
    0x05, 0x09,                        //   USAGE_PAGE (Button)
    0x19, 0x01,                        //   USAGE_MINIMUM (Button 1)
    0x29, 0x06,                        //   USAGE_MAXIMUM (Button 6)
    0x15, 0x00,                        //   LOGICAL_MINIMUM (0)
    0x25, 0x01,                        //   LOGICAL_MAXIMUM (1)
    0x75, 0x01,                        //   REPORT_SIZE (1)
    0x95, 0x06,                        //   REPORT_COUNT (6)
    0x81, 0x02,                        //   INPUT (Data,Var,Abs)
    0x95, 0x02,                        //   REPORT_COUNT (2)
    0x81, 0x03,                        //   INPUT (Cnst,Var,Abs)
    0xc0       }
};/* End Collection,End Collection      */
```

Figure C-1: USB Gamepad Descriptor