



Michigan Technological University  
*Create the Future* Digital Commons @ Michigan Tech

---

Dissertations, Master's Theses and Master's  
Reports - Open

Dissertations, Master's Theses and Master's  
Reports

---

2004

## Design and implementation of a 3D computer game controller using inertial MEMS sensors

Ali Pezeshk  
*Michigan Technological University*

Follow this and additional works at: <https://digitalcommons.mtu.edu/etds>



Part of the [Electrical and Computer Engineering Commons](#)

Copyright 2004 Ali Pezeshk

---

### Recommended Citation

Pezeshk, Ali, "Design and implementation of a 3D computer game controller using inertial MEMS sensors", Master's report, Michigan Technological University, 2004.  
<https://doi.org/10.37099/mtu.dc.etds/578>

Follow this and additional works at: <https://digitalcommons.mtu.edu/etds>



Part of the [Electrical and Computer Engineering Commons](#)

---

# *Design and Implementation of a 3D Computer Game Controller Using Inertial MEMS Sensors*

By: Ali Pezeshk

Advisor:  
Dr. Brian T. Davis

*Submitted in Partial Fulfillment of  
Requirements for*

*Master of Science Degree in  
Electrical Engineering*

*Electrical & Computer Engineering Department,  
Michigan Technological University*

*December 2004*

---

*Abstract* — Though 3D computer graphics has seen tremendous advancement in the past two decades, most available mechanisms for computer interaction in 3D are high cost and targeted for industry and virtual reality applications. Recent advances in Micro-Electro-Mechanical-System (MEMS) devices have brought forth a variety of new low-cost, low-power, miniature sensors with high accuracy, which are well suited for hand-held devices.

In this work a novel design for a 3D computer game controller using inertial sensors is proposed, and a prototype device based on this design is implemented. The design incorporates MEMS accelerometers and gyroscopes from Analog Devices to measure the three components of the acceleration and angular velocity. From these sensor readings, the position and orientation of the hand-held compartment can be calculated using numerical methods.

The implemented prototype is utilizes a USB 2.0 compliant interface for power and communication with the host system. A Microchip dsPIC microcontroller is used in the design. This microcontroller integrates the analog to digital converters, the program memory flash, as well as the core processor, on a single integrated circuit. A PC running Microsoft Windows operating system is used as the host machine.

Prototype firmware for the microcontroller is developed and tested to establish the communication between the design and the host, and perform the data acquisition and initial filtering of the sensor data. A PC front-end application with a graphical interface is developed to communicate with the device, and allow real-time visualization of the acquired data.

*This project, “Design and Implementation of a 3D Computer Game Controller Using Inertial MEMS Sensors”, is hereby approved in the partial fulfillment of the requirements for the Degree of:*

*Master of Science in Electrical Engineering*

*Department of Electrical & Computer Engineering*

---

*Project Advisor* *Name*

---

*Head of Department* *Name*

---

*Date*

ACKNOWLEDGMENTS:

*I want to express my gratitude to my advisor and mentor, Dr. Brian T. Davis, for his peerless guidance throughout the project. His expertise, eloquence, magnanimity and graciousness have always been an inspiration and a blessing to me. This project would never have been possible, wasn't it for his kind support and encouragement.*

*Thanks to my friend and colleague, Mehdi Imaninejad for his help and thoughtful ideas during the early stages of the development of the project.*

*I also want to thank my family for their support during all years of my studies, and my fiancée, Fatemeh, for her patience during my two years of studies at Michigan Technological University.*

---

# Contents

---

<b>CHAPTER 1</b>	<i>Introduction .....</i>	<i>1</i>
<b>CHAPTER 2</b>	<i>3D Computer Interfaces .....</i>	<i>3</i>
	Mechanical Tracking .....	3
	Magnetic Tracking .....	4
	Acoustic Tracking .....	4
	Vision-based Tracking .....	4
	Inertial Tracking .....	5
<b>CHAPTER 3</b>	<i>Inertial Tracking.....</i>	<i>7</i>
	Frames of Reference .....	9
	Orientation Calculation.....	10
	Position Calculation .....	12
	Limitations .....	13
<b>CHAPTER 4</b>	<i>Electrical Design.....</i>	<i>15</i>
	Part Selection .....	16
	<i>Microcontroller .....</i>	<i>16</i>
	<i>USB Interface .....</i>	<i>17</i>
	<i>The IMU Sensors.....</i>	<i>18</i>
	<i>Other Parts .....</i>	<i>20</i>
	Layout and Assembly .....	20
	<i>PCB Layout.....</i>	<i>20</i>
	<i>Placement of Accelerometers .....</i>	<i>24</i>
	<i>Chassis Design .....</i>	<i>25</i>

	<i>Debugging the Board</i> .....	26
	Feature Summary .....	28
<b>CHAPTER 5</b>	<i>Software Development</i> .....	29
	Game Controller Firmware .....	29
	<i>Initialization</i> .....	30
	<i>3D Game Controller Finite State Machine</i> .....	31
	<i>Timing and sampling</i> .....	31
	<i>Packet Structure</i> .....	32
	<i>Orientation Calculation Revisited</i> .....	34
	<i>Static Acceleration Compensation</i> .....	34
	PC Front-end .....	35
	<i>Version 1: Text Based Interface</i> .....	35
	<i>Version 2: Graphical Interface</i> .....	36
	Software Debugging .....	37
<b>CHAPTER 6</b>	<i>Results &amp; Future Work</i> .....	39
	Results .....	39
	<i>Sensor Data</i> .....	39
	<i>Sensor Saturation</i> .....	41
	<i>Sensor Noise</i> .....	42
	Future Work .....	43
	<i>Hardware Enhancements</i> .....	43
	<i>Position and Orientation Calculation, and Numerical Methods</i> .....	45
	<i>Filtering and Noise Reduction</i> .....	45
	<i>Host Application Development</i> .....	45
	<i>References</i> .....	47

---

# *List of Tables*

---

Table 4.1: Requirements for the microcontroller and the specifications of dsPIC30F2010-20I.....	17
Table 4.2: Requirements for the USB interface and the specifications of FTDI FT245BM.....	18
Table 4.3: IMU specifications.....	19
Table 4.4: Prototype Feature Summary.....	28





---

# *List of Figures*

---

Figure 3.1: Motion of an object in 2-space . . . . .	8
Figure 3.2: Effect of rotation on the accelerometer readings in 3-space. . . . .	10
Figure 3.3: Coordinate system transformation in 3-space . . . . .	11
Figure 4.1: Layout of the Prototype Design: Floorplan. . . . .	21
Figure 4.2: Circuit Board Layout of the Prototype Design: Top Layer. . . . .	22
Figure 4.3: Circuit Board Layout of the Prototype Design: Bottom Layer. . . . .	23
Figure 4.4: Relative Positioning of Accelerometers. . . . .	25
Figure 4.5: Completed Prototype in Chassis . . . . .	26
Figure 4.6: Prototype After Debugging: Bottom View . . . . .	27
Figure 5.1: The 3D Game Controller Finite State Machine, State Diagram. . . . .	31
Figure 5.2: Task Scheduling and Super-Sample Acquisition Timing Diagram. . . . .	33
Figure 5.3: Data Packet Structure. . . . .	33
Figure 5.4: Host FSM, State Diagram . . . . .	36
Figure 5.5: Screenshot of the Graphical Interface . . . . .	38

Figure 6.1: Sample Sensor Reading for Acceleration: Rotation Around Y-axis . . . . .	40
Figure 6.2: Sample Sensor Reading for Angular Velocity: Rotation Around Y-axis . . . .	41
Figure 6.3: Sample Sensor Reading for Temperature . . . . .	41
Figure 6.4: Gyroscope Saturation: Fast Rotation Around Y-axis . . . . .	42
Figure 6.5: Sample Accelerometer Reading: Very Fast Motion Along X-axis. . . . .	42
Figure 6.6: Sample Noise on Sensor Readings . . . . .	43

---

Since the invention of the first computers, various devices have been designed and implemented to serve as an interface between the human user and the machine. It has been only in the past few years, through immense advances in microelectronic fabrication and design process, that 3D computer graphics has become accessible to non-professional computer users.

Real-time 3D rendering and virtual environments are part of most newly released computer games and every year more movies are made which are completely computer generated. Ongoing research on 3D display systems has recently made the first commercial 3D monitors available to the market [SeeReal04] and in the next few years it is expected that every home would have a 3D TV.

However, the computer interfaces have not advanced at the same pace. The main interface to most modern home computers is still the 2D mouse, the same interface used over a decade ago. Current 3D interface designs are mainly targeted towards industrial, motion tracking and virtual reality applications, where budget is typically not a prohibiting factor.

The main motivation for the present work was to design an affordable 3D computer interface with competitive performance, for normal household use. There are various gaming consoles available on the market now, and the trend is towards a more immersive realistic 3D virtual experience. Since a gaming interface has more relaxed requirements compared to a motion tracking device or a 3D modeling software interface, it was selected as a starting point for the design of a new generation of 3D computer interfaces.

---

Researching current available 3D interface technologies and observing their limitations and drawbacks defined the framework in which the new design should reside:

- The design should be operable within virtually any room, meaning there shouldn't be restrictions on how tidy the room is, how much furniture is in the room and what the objects in the environment are made of, how they look and what their colors are.
- The design should require small amount of processing power: Though processing power of computers is increasing day by day, so is the demand of the applications running on them. A highly processor-consuming device will severely limit the context in which the device is used, i.e. the game.
- The design should be operable at distance: The user should be able to stand, sit or lay at whatever distance (with reasonable limits) (s)he finds suitable for playing the game. This is even more important for console gaming, where the visual output is sent to a TV, which is normally a few meters away from the audience.
- The design should require as few components as possible: For a gaming application, the user plays the game at home not at a studio. It is therefore important that (s)he needs not install extra equipment around the house.
- The design should preferably not use electromagnetic waves: Though the effects of various frequency ranges of electromagnetic waves on human body have been subject to intensive research, the results are still highly debated. It is however always better to stay on the safe side and avoid using what is potentially harmful, especially for this case, where the device may be used frequently and for long durations.

By the advances in the design of Micro-Electro-Mechanical-System (MEMS) sensors, new horizons to design of 3D computer interfaces are emancipated. This report presents and elaborates a novel design and implementation of a 3D computer game interface based on inertial MEMS sensors which fits nicely in the above framework.

MEMS devices combine high accuracy, low cost, compactness, and durability, making them an ideal choice for handheld devices. The design goal of this project has been to make a prototype device that could be demonstrated as the proof of concept. The prototype device also unveils the limitations of the design and the challenges one faces before the design can be commercialized.

The idea presented in this work is partially presented as a paper [Pezeshk04] and an invention disclosure is filed with Michigan Technological University's Office of Intellectual Property and Technology Commercialization (IPTC).

# *3D Computer Interfaces*

---

3D computer interfaces generally consist of a movable compartment and a system to track or sense its motion. The detected motion is translated in the software to controls in the virtual environment.

The various ways of tracking the motion of the movable compartment can be categorized into five main methods: mechanical, magnetic, acoustic, vision-based, and inertial. In this chapter these techniques and their requirements will be reviewed, and the advantages and drawbacks of each system will be discussed.

---

## *2.1 - Mechanical Tracking*

The mechanical tracking systems are among the simplest designs. The movable part is physically attached by a number of sliding joints to a fixed frame, which is connected to a base. Sensors are present at the joints which monitor the amount of movement in each direction.

Interfaces using mechanical tracking therefore offer very limited range and take a lot of space. However, since the moving compartment is attached to a base, these devices have the ability to exert force-feedback or limit the motion in some or all directions. These interfaces are also typically highly accurate and updates to the position and orientation can be obtained at much higher rates and with lower latencies [Bowman05].

Since the base is normally fixed on the ground these devices are rather hard to use and severely limit the mobility of the user.

---

## 2.2 - Magnetic Tracking

In magnetic tracking systems, a transmitter device is utilized, which sends out a low-frequency magnetic wave. Special magnetic sensors are mounted on the object being tracked, using which the position and orientation of the object can be extracted, relative to the magnetic transmitter [Bowman05][Foxlin02].

Interfaces using magnetic tracking are very expensive and those of high accuracy typically have a short range of about 4 feet. The accuracy of these devices is in the order of few millimeters in position and tenths of a degree in orientation.

Another disadvantage of these devices is their susceptibility to distortions in their output caused by presence of magnetic (Ferro-/para-/dia-magnetic) objects such as metal surfaces or objects present in the room. These effects could be mitigated to some extent by using some computational overhead and applying algorithms at start-up of the device, but not for a dynamically changing environment.

---

## 2.3 - Acoustic Tracking

The interfaces based on this type of tracking usually use a transmitter on the hand-held compartment of the device. The transmitter emits an ultra-sound wave which is received by three sensors. The position and the orientation of the hand-held device can then be computed from the time it took for a pulse to reach the receiver and via triangulation. The placement of the transmitter and the receivers is interchanged in some designs but the underlying principle is the same [Logitech92].

These devices however have a short range and offer less frequent updates of the location of the hand-held device. These devices are relatively less expensive compared to the magnetic and optical interfaces. The accuracy of these interfaces is also dependent on the presence of objects in the line-of-sight of transmitter to the receiver. Also external noises such as jingling keys or a phone ringing can interfere with the tracking signal and reduce accuracy [Bowman05].

---

## 2.4 - Vision-based Tracking

These tracking systems rely on the light emitted by or reflected from the objects or markers on the objects, and a set of cameras followed by image processing techniques, to find the position and orientation of the object being tracked [Foxlin02].

Interfaces based on this type of tracking are usually very expensive and hard to setup:

- Multiple cameras must be placed in various locations to cover the area of interest in which motion should be sensed.
- The background should be easily distinguishable from the markers placed on the hand-held compartment or objects being tracked. This can be difference in color, pattern, texture, reflectivity or luminance.
- In case of an interface with a hand-held compartment, the hand-held part of the device should also be distinguishable from the user's body and clothing.
- Proper lighting of the environment, where the controller is used, is of extra importance as it drastically affects the performance of the system.

These systems require high processing power to determine the position and orientation in real-time. The requirements for the processing power and hardware increase with the number of cameras used.

The main advantage of these systems is that they are capable of tracking multiple objects in the scene with the same amount of equipment using more sophisticated software and further processing of the visual data from the scene.

---

## *2.5 - Inertial Tracking*

This type of tracking is based on measuring the angular velocity and acceleration of the object being tracked to calculate its position and orientation.

An interface based on this type of tracking has the advantage that it is self-contained, meaning that no devices outside of the object being tracked are required. Due to the nature of measurements, there is also no requirement for transmission of electromagnetic or acoustic waves. This is an advantage as the user won't be subject to radiation which may result in disorders such as cancer caused by prolonged exposure to radiation. Another advantage is that except for sudden temperature changes, which are unlikely in indoors environments, the accuracy of the system is not affected by the environment where the interface is being used. Also, since the sensors are contained within the interface, the motion can be sensed virtually anywhere, as long as it is connected to the host machine.

Having these advantages in mind, this type of tracking was seen to be suitable for a 3D game controller system. In the next chapter, details of how this type of tracking works are presented.





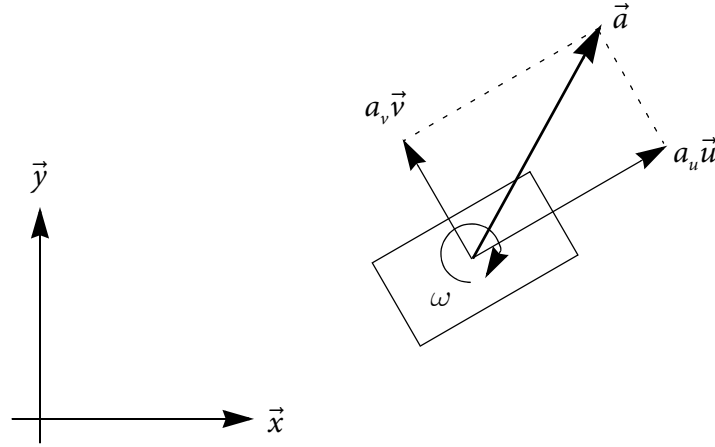
Inertial Navigation Systems (INSs) have been used in navigation systems since 1950s in ships and airplanes [Grewal01]. Presently, INS is used for navigation of missiles, airplanes and satellites. The position and orientation calculated in INS are fairly stable in short periods of time and updates to position and orientation can be made much faster than other methods such as GPS for these applications.

An inertial tracking system is comprised of an Inertial Measurement Unit (IMU) and a processing unit. Calculation of position and orientation is performed by the processing unit, based on the measurement of instantaneous acceleration and angular velocity of an object, which is performed by the IMU. Two types of sensors are used for measurement of these quantities:

- Accelerometers: used to measure the acceleration
- Gyroscopes: used to measure the angular velocity around an axis.

Initially gyroscopes used for sensing and measurement of angular velocity were very heavy and based on the spinning wheel with a heavy mass which limited their usage. Laser based gyroscopes revolutionized the gyroscope industry by eliminating moving parts and offering higher precision.

Accelerometer designs are mainly mechanical and until the past few years, accurate accelerometers were also bulky, heavy and hard to mount. Various types of accelerometers and gyroscopes are discussed in detail in [Lawrence98].



**FIGURE 3.1. Motion of an object in 2-space**

Advances in micro-machining and MEMS technology have allowed sensor sizes to shrink by several orders of magnitude, while retaining their performance, and has considerably reduced production costs. These favorable features, together with their low power consumption, and ease of mounting and integration with other electronic components, makes these sensors ideal for hand-held devices.

In order to track the motion of the Object of Interest (OOI), it is required that components of acceleration and angular velocity in the vector space containing the object location, be measured in a basis system that would span all possible motions of the OOI. For instance if the OOI moves only on a plane, it is required to measure the two components of acceleration on a desired basis in the plane of motion and the angular velocity component around the vector normal to the plane of motion. This is shown in Figure 3.1: The rectangular object has an acceleration vector  $\vec{a}$ , which can be represented in terms of its two components parallel to the basis  $(\vec{u}, \vec{v})$  as  $a_u \vec{u}$  and  $a_v \vec{v}$ . The angular velocity in the plane of motion is represented as  $\omega$ .

As can be seen, the choice of the basis is arbitrary as long as the motion of the object remains in the vector space spanned by the basis. It is however more convenient to use an orthogonal basis for this purpose which could be thought of as a coordinate system.

In the following sections we'll discuss the frames of reference, how to calculate the position and orientation of the OOI from the sensor data and the limitations of inertial tracking systems.

### 3.1 - Frames of Reference

*Definition:* A *Frame of Reference* is a framework that is assumed to be static and motion and position of objects are defined relative to a coordinate system attached to this frame.

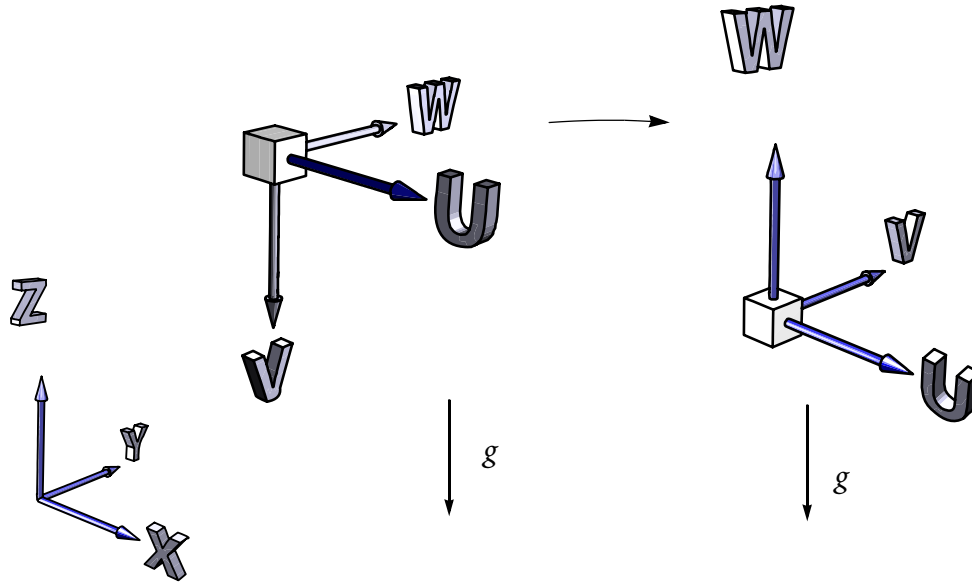
As an example, the frame of reference for a computer user can be considered to be a coordinate system attached to the earth. For this person the rotation of the earth around the sun or around itself is irrelevant since (s)he moves along with the earth and the her/his position defined in this coordinate system is not a function of rotation of the earth.

If the user holds a game controller in her/his hand and moves it around, the motion of the controller can be described in three frames of reference, namely:

- The frame of reference of the earth: In this frame of reference the motion of the game controller is given with respect to the earth. If the room or environment in which the controller is used is not moving relative to the earth, then a coordinate system attached to the room or environment may be used to define the motion of the game controller.
- The frame of reference of user: In this frame, the motion of the controller is described relative to its position from the user, so for instance if the user walks in the room but holds the controller in a constant location relative to herself/himself, the position of the controller will be constant.
- The frame of reference of the game controller: According to this frame of reference, the game controller is static and everything else is moving.

Since the accelerometers and gyroscopes are mounted on the OOI, what the sensors sense are along/around the coordinate system in the local frame of reference of the OOI, though the measured quantity is in earth's frame of reference. This is shown in Figure 3.2, where the OOI turns 90 degrees around the  $x$ -axis of the earth's frame coordinate system. As can be seen in both cases the sensors measure the earth's gravitational acceleration vector,  $g$ , but the values that sensors read, change. Before the rotation, the accelerometer on the  $v$ -axis of the local frame coordinate system senses  $g$ , but after the rotation the accelerometer on the  $w$ -axis of the local frame coordinate system is the one that measures  $g$ .

In this report we assume that the room or place where the controller is used is attached to the ground and thus the frame of reference of earth could be used. If the controller is to be used in a room which itself is moving, e.g. a flight simulation chamber, additional sensors are required which should track the position and orientation of the room with respect to the earth's frame of reference. The position and orientation of the game controller can then be translated into the frame of reference of the room by subtracting the results obtained by the two measurement devices to obtain the relative motions.



**FIGURE 3.2. Effect of rotation on the accelerometer readings in 3-space**

Inertial tracking can be performed using two main different types of IMU mounting:

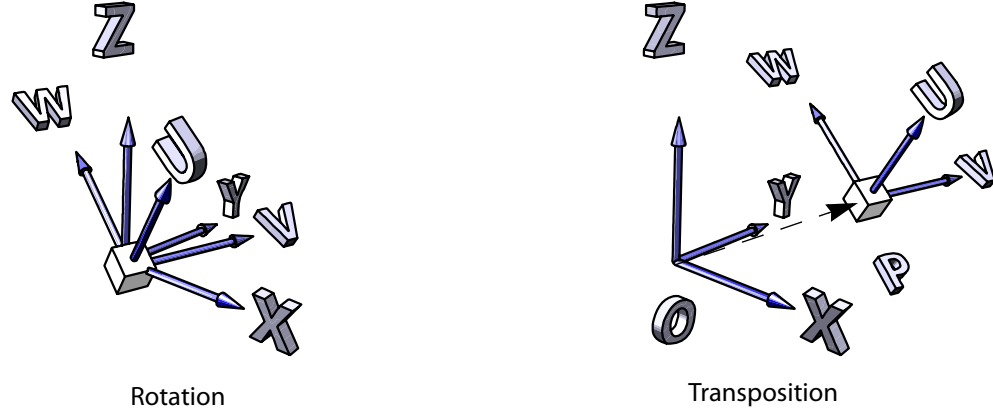
- IMU is mounted in a way such that it does not rotate with the object, so the coordinate system attached to it is always aligned with the coordinate system of the frame of the earth. Examples of these are the gimbal mounted IMUs.
- IMU is mounted directly on the object, so the coordinate system attached to it is always aligned with the coordinate system of the frame of reference of the object. Examples of these are the strap-down IMUs.

In the present work, the second approach is taken, since it results in a more compact design and lower cost.

### 3.2 - Orientation Calculation

The orientation of an object can be represented in various ways. In this section we will introduce the more intuitive representation of *rotation matrices* [Eberly04]. In Chapter 5 however, *quaternions* will be briefly introduced for their better computational efficiency in calculation of orientation [Schneider03].

Consider a solid object in 3-space as shown in Figure 3.3, with the coordinate system attached to its frame of reference as  $(P; \vec{u}, \vec{v}, \vec{w})$  with origin at  $P$ . Further assume that the



**FIGURE 3.3. Coordinate system transformation in 3-space**

frame of reference of interest, in which the motion of the object is defined, has a coordinate system attached to it as  $(O; \vec{x}, \vec{y}, \vec{z})$  with origin at  $O$ .

It can now be seen that the position and orientation of a solid object can be represented by the relative situation of the two coordinate systems. More rigorously, the coordinate system  $(O; \vec{x}, \vec{y}, \vec{z})$  can be transformed into  $(P; \vec{u}, \vec{v}, \vec{w})$  and vice versa, by means of a *transposition* of origin and *rotation* of the axes:

$$(P; \vec{u}, \vec{v}, \vec{w}) = (O + \vec{D}; R[\vec{x}|\vec{y}|\vec{z}]) \quad (\text{EQ. 3.1})$$

where  $\vec{D} = \overrightarrow{OP}$  is the transposition vector,  $R$  is the rotation matrix, and  $[\vec{x}|\vec{y}|\vec{z}]$  is a matrix whose columns are the axes vectors  $\vec{x}$ ,  $\vec{y}$ , and  $\vec{z}$ .

**Definition:** A rotation matrix in 3-space, is a 3-by-3 matrix and defines a transformation in 3-space which retains vector lengths and the angle and the sense of the angle between two vectors.

A rotation matrix, when applied to a coordinate system, i.e. a set of three mutually orthogonal vectors, rotates this system. This means that the unit vectors in the direction of the axes retain their unity length after transform and their orthogonality is preserved.

For an object with dynamics, a rotation matrix,  $R(t)$ , can be defined as a function of time, which at every instant defines the rotation which aligns the reference coordinate system  $(O; \vec{x}, \vec{y}, \vec{z})$  with  $(P(t); \vec{u}(t), \vec{v}(t), \vec{w}(t))$ . Hence, calculating  $R(t)$  in real-time, provides the instantaneous orientation information of the OOI.

It can be proved that  $R(t)$  can be calculated from the OOI's angular velocity vector,  $\vec{\omega} = \omega_u \vec{u} + \omega_v \vec{v} + \omega_w \vec{w}$ , and the initial orientation of the OOI, from the following differential equation [Grewal01]:

$$\dot{R}(t) = \text{Skew}(-\vec{\omega})R(t) = \begin{bmatrix} 0 & -\omega_w & \omega_v \\ \omega_w & 0 & -\omega_u \\ -\omega_v & \omega_u & 0 \end{bmatrix} R(t), \quad R(0) = R_0 \quad (\text{EQ. 3.2})$$

Note that since the measurements are made in the frame of reference of the OOI, the OOI *sees* itself as static and everything else rotating in the *opposite* direction. Therefore, a negative sign is introduced for the angular velocity vector in Equation 3.2. The dependence of the angular velocity on time is removed as shorthand.

### 3.3 - Position Calculation

The acceleration of an object is the second derivative of its position. Though it seems that the position can be readily obtained by doubly integrating the acceleration, this is not the case. As observed in Figure 3.3, since the accelerometers are mounted on the OOI, they measure the acceleration of the OOI as projected on the coordinate system attached to the frame of reference of the OOI, i.e.  $(P(t); \vec{u}(t), \vec{v}(t), \vec{w}(t))$ .

As a result, to calculate the position of the object, it is necessary to map the accelerometer readings back to the coordinate system in which the position is to be calculated, i.e.  $(O; \vec{x}, \vec{y}, \vec{z})$ . The position can then be calculated by doubly integrating the mapped acceleration.

The readings of the accelerometers can be combined as:

$$\vec{a} = a_u \vec{u} + a_v \vec{v} + a_w \vec{w} = a_x \vec{x} + a_y \vec{y} + a_z \vec{z} \quad (\text{EQ. 3.3})$$

where,  $\vec{a}$  is the acceleration vector of the OOI, which is represented with its corresponding components in coordinate systems  $(P; \vec{u}, \vec{v}, \vec{w})$  and  $(O; \vec{x}, \vec{y}, \vec{z})$ .

Noting that:

$$\begin{aligned} \vec{u} &= R_{11}\vec{x} + R_{21}\vec{y} + R_{31}\vec{z} \\ \vec{v} &= R_{12}\vec{x} + R_{22}\vec{y} + R_{32}\vec{z}, \quad R = [R_{ij}]_{1 \leq i, j \leq 3} \\ \vec{w} &= R_{13}\vec{x} + R_{23}\vec{y} + R_{33}\vec{z} \end{aligned} \quad (\text{EQ. 3.4})$$

and substituting in Equation 3.3, we have:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} a_u \\ a_v \\ a_w \end{bmatrix} = R \begin{bmatrix} a_u \\ a_v \\ a_w \end{bmatrix} \quad (\text{EQ. 3.5})$$

Having the acceleration vector mapped into the frame of reference of the earth, assuming that the item is initially at rest, and knowing the initial position of the OOI at time 0, the instantaneous position of the OOI,  $\overrightarrow{OP}(t)$ , can be calculated as:

$$\overrightarrow{OP}(t) = \begin{bmatrix} P_x(t) \\ P_y(t) \\ P_z(t) \end{bmatrix} = \begin{bmatrix} \int_0^t \int_0^\tau a_x(\zeta) d\zeta d\tau + P_x(0) \\ \int_0^t \int_0^\tau a_y(\zeta) d\zeta d\tau + P_y(0) \\ \int_0^t \int_0^\tau a_z(\zeta) d\zeta d\tau + P_z(0) \end{bmatrix} \quad (\text{EQ. 3.6})$$

For a solid object, the position of every point in/on the object is fixed relative to the frame of reference of the object. Therefore, knowledge of the evolution of the coordinate system attached to the frame of reference of the object fully determines the position of every single point on the object. This is however not true for an object which is not solid such as a flexible rubber band.

### 3.4 - Limitations

Sensors are prone to additive noise on the signals. For reasonable accuracy, the output of the sensors need to be low-pass filtered to remove out-of-band noise. It is therefore desirable to reduced the passband as much as possible to increase the signal to noise ratio. This introduces some limitations on the speed at which the device will be able to track the movements.

Under moderate assumptions it can be assumed that the human motion indeed has a small bandwidth and so the device is practical. The main assumption is that the device does not collide with anything while it is used.

Collisions can also cause sensors to saturate, which is another limiting factor causing failure in tracking. Though cushioning will help reducing the effects of the impact, saturation of sensors is still probable. Saturation in gyroscopes can also happen as a result of fast motion of the device.



Drift in the tracking is mostly referred to as the main limiting factor of an inertial tracking system [Foxlin02]. Errors in the computation and sensor results will accumulate in time, causing drift in the output of the inertial tracker. The drift is caused by two different processes:

- Bias induced drifts: Changes in the bias point of the sensors or incorrect measurement of bias points during calibration of sensors cause drifts in estimated position and orientation. A constant bias error will increasingly accumulate in time (as  $t$  for gyroscopes and as  $t^2$  for accelerometers). These bias drifts are mainly due to the temperature dependence of the sensor response curve.
- Noise induced drifts: The additive Gaussian noise on the sensor data, when integrated forms a random walk process. As a result, though the mean of the estimate is zero, the variance of the estimated position and orientation increases in time.

Drifts of the first kind can be compensated by either carefully keeping the temperature constant using an oven, or careful calibration of the sensors at different temperatures, constantly monitoring the temperature and correcting for the drifts in software by using the calibration data.

Drifts due to noise can also be compensated for by using other types of sensors and/or tracking systems, such as those introduced in previous chapter, running in conjunction with the inertial tracking system, correcting its output at preset time intervals.

---

Prior to design of the prototype device, several different sensors were tested on a Motorola 68HC11 evaluation board [Axiom99] using the Buffalo interface [Axiom03]. This served several purposes:

- To try to verify the concept using a pre-made test-bench.
- To facilitate the selection of proper sensor ranges for the prototype design.
- To determine the required part specifications for the prototype design.

Since the area on which parts could be mounted and connected to the microcontroller was a breadboard, it was not possible to mount all the sensors in the required directions, i.e. rows in the breadboard are interconnected so devices had to be placed perpendicular to the rows, or on the columns. Instead, dual-axis accelerometers and single-axis gyroscopes were tested individually.

Since the Buffalo interface is a serial communication terminal application, it was necessary for the device to transmit ASCII characters which reduced the transmission rate of data at least three times: a byte represents a two digit hexadecimal number and a carriage return or space is required after each value.

The requirement of a power-supply in addition to the cable connecting the device to the computer was another drawback which severely limited the type and extent of motions of the device.

The Analog to Digital Converters (ADCs) on the 68HC11 microcontroller are 8-bit. Since the output of accelerometers does not cover the complete

voltage range of 0-5V, and the least significant bit (LSB) of each sample contains errors, it was evident that a higher resolution is required for sampling the sensor outputs. The other limiting factor was the number of channels that could be sampled by the device and the low sampling rate provided by the microcontroller. Also it was observed that the 2MHz core frequency of the 68HC11 with its multicycle instructions would not be able to handle the workload of processing the data from the sensors.

After these initial tests, it was evident that a prototype device needed to be constructed. In the following sections, we describe how the parts were selected and how the prototype was designed.

---

## ***4.1 - Part Selection***

The design goal for the first prototype was to be able to sample the three gyroscope and three accelerometer outputs at a higher sampling rate than the frequency at which data should be transmitted to the PC to allow filtering of the signals. It was also decided that the first implementation should be using a wired connection to the PC and should be easy to move around for the tests.

### ***4.1.1 - Microcontroller***

The microcontroller needed to meet the criteria described in Table 4.1. The requirements on the packaging are due to current mounting capabilities of ECE department, where surface mount devices cannot be mounted.

Based on these requirements, a microcontroller from the dsPIC30F series, a new family of microcontrollers developed by Microchip, was selected [dsPIC-A]. Since at the time of design, only the highest end and the lowest end of the family were in production and the high-end dsPIC30F6014 comes only in a TQFP80 package, the low-end dsPIC30F2010 was chosen which meets all the requirements and offers some of the desirable features [dsPIC-B]. The specifications of this device are summarized in Table 4.1.

The development environment for dsPIC30F series is the Microchip MPLAB IDE which is available free of charge on the company's web site. The C30 compiler provided by Microchip is also available to compile the C code written for dsPIC series.

Programming is done with the Microchip ICD 2, in-circuit programmer/debugger. This piece of hardware allows faster debugging of the software and hardware by allowing single-stepping of code, addition of break-points and slow-motion animation of the code while the microcontroller is in the circuit with actual stimuli. The device can be interfaced to a USB module using I/O pins or UART serial communication and can be powered through USB bus or an external power supply.

**TABLE 4.1. Requirements for the microcontroller and the specifications of dsPIC30F2010-20I**

Criterion	Required	Desirable	dsPIC30F2010-20I
On chip nonvolatile memory	Flash: Enough to eliminate need for memory components outside the microcontroller	Some EEPROM for settings and calibration data storage	12KB Flash 1KB EEPROM
ADC	At least 6 channels, with at least 10-bit resolution, integrated sample and hold with 10KHz minimum sampling frequency	Simultaneous sampling, ADC buffer memory	6 10-bit ADC channels, simultaneous sampling on maximum of 4 channels, 16 word ADC buffer, up to 500KSPS
I/O pins	Enough number of I/O pins to support transmission to PC, a push button and an LED	I/O pins available for more push buttons	20 I/O pins, 20mA source/sink on each I/O pin
Timers	At least one 16-bit timer	More timers	3 16-bit timers available, 32-bit timer functionality using two 16-bit timers
Programmability	Not requiring expensive programming devices	On-board programmability	On-board programmable, inexpensive programming tools available
Processing power	Fast enough to process the raw data	Single cycle instructions, Special instructions for faster numerical computation	20 MIPS @ 20MHz clock, single cycle instructions, some DSP functionality, fast single cycle 17-bit by 17-bit signed multiplication
Compiler availability	C compiler availability	Floating point libraries availability	Development environment available for free, C30 compiler available for free evaluation, includes standard floating point math libraries
Packaging	LCC, or DIP	DIP	PDIP300
Other		Quadrature encoder for interfacing with a wheel	Integrated RC-oscillator, Integrated quadrature encoder

### 4.1.2 - USB Interface

The Universal Serial Bus (USB) interface was chosen for the communication between the game controller and the host computer for the following reasons:

- Bandwidth: USB offers high data throughput while maintaining low latency, making it suitable for real-time applications.

- Portability: USB ports are available on most computer systems including most of the gaming consoles such as Sony PlayStation2 and Microsoft X-Box.
- Power through USB: The device connecting to a USB host can be powered through the USB connection as long as it does not exceed the maximum power specifications in the standard.

The FT245BM chipset from FTDI was chosen for the interfacing with the microcontroller to deliver USB connectivity [USB-A]. The requirements for the USB interface and the specifications of the FT245BM are summarized in Table 4.2.

Since this interface chip is SMD, a simple DIP module called DLP-USB245M developed by DLP-Design was used [USB-D]. This module incorporates the FT245BM chip together with a serial EEPROM, USB interface front-end specification capacitor and Ferrite beads, and a crystal oscillator.

Since the USB standard specifies that no device is allowed to draw more than 100mA before it is enumerated, a power switch device was used along with the USB interface module [USB-C]. Micrel MIC2025-2BM was used for this purpose. The power from USB interface is fed into this device with the switch off at startup, hence only the USB interface draws current from the USB host. When the USB host has successfully enumerated the device, the USB interface module will send a signal to MIC2025-2BM to release the power to the rest of the circuit. The switch is soft-start, meaning that switching the power on, is not abrupt. This is required in order to meet the specifications of the USB standard.

**TABLE 4.2. Requirements for the USB interface and the specifications of FTDI FT245BM**

Criterion	Required	FTDI FT245BM
Standard Compliance	At least USB 1.0	USB 2.0 Compliant
Ease of use	Not requiring in-depth understanding of the standard	Basic understanding of USB interface suffices
Microcontroller interfacing capability	Convenient to interface with the microcontroller	Seen by the microcontroller as a FIFO buffer through a 13-pin interface
Driver availability	Sample code or driver for Microsoft Windows platform	Drivers available as a DLL file that could be loaded by the user interface program
Power management	Power control at device connection	Can control powering the rest of the circuit after enumeration of the device
Other	Ability to store device identification information	Integrated $I^2C$ connection interface to external serial EEPROM for storage and retrieval of identification data

#### 4.1.3 - The IMU Sensors

There are not many companies developing MEMS accelerometers and gyroscopes. Analog Devices is one of the first and highly acclaimed producers of the MEMS Inertial

Measurement Unit (IMU) sensors. Several sensors from this manufacturer were tested on the Motorola evaluation kit. From these tests the The IMU sensors where selected such that their outputs be analog signals and have a reasonable price. The digital output IMU sensors by Analog Devices used pulse-width-modulation to transmit the output, so the duration the output pulse is high determines the sensor readout value. These sensors where not chosen for the following reasons:

- Accurate timing is difficult: Even in an interrupt driven fashion, finding the pulse-width of 6 sensors is complicated and far from accurate due to synchronization problems and race conditions.
- Sampling limitations: The output is sampled by the sensor and sent out at a much less flexible frequency that is determined by capacitors and resistors added to the circuit, making it difficult to change the design parameters, whereas in the analog output case, the sampling frequency can be controlled in the software.

At design time, Analog Devices only produced gyroscopes in two ranges, so the higher range was chosen [Gyro-A]. 3-axis accelerometers are available from this manufacturer as modules, but they cost over 5 times a dual-axis accelerometer of the same range, hence dual-axis accelerometers were chosen [Accel-A].

The gyroscope package is a Ball-Grid-Array (BGA) and the accelerometer is packaged as a LeadLess-Ceramic-Carrier (LLCC), both of which need special equipment for mounting. As a result the corresponding evaluation boards were used [Accel-B] [Gyro-B]. These modules also integrate a single-pole low-pass filter at the output of the sensors for better signal to noise ratio. These filters also serve as the anti-aliasing filter for the analog to digital converters.

Specifications of the IMU sensors are summarized in Table 4.3.

**TABLE 4.3. IMU specifications**

Sensor Type	Part Name	Quantity Used	Specifications
Accelerometers	ADXL311	2	Dual-axis accelerometer Analog output Range: $\pm 2g$ Noise floor @50Hz cutoff: $10mg$
Gyroscopes	ADXRS300	3	Uni-axis gyroscope Analog output Range: $\pm 300^{\circ}s^{-1}$ Noise floor @ 40Hz Cutoff: $0.6^{\circ}s^{-1}$ Temperature sensor

#### **4.1.4 - Other Parts**

A push-button is needed in most computer interfaces for various types of input from the user such as interaction with the environment. Therefore, it was decided for the design to have at least one push-button. This button could also be used during debugging and development of hardware and software, as a user input. A shielded push-button was selected so that any electrostatic discharge would be grounded and won't damage the other sensitive circuitry. It was also chosen to be omni-directional, meaning that the button could be pushed at an angle and it will still work. Usually switches are used as push-buttons, requiring the force to be exerted perpendicular to the surface of the switch, which makes them rather inconvenient to use.

Using a set of jumpers was thought to add some additional flexibility to the circuit. Since the circuit board was to be placed in a chassis, it would have been difficult to access the jumpers inside, so a right-angle connector was chosen.

A crystal resonator was chosen for supplying the clock signal required by the microcontroller. Compared to a crystal oscillator clock, a crystal resonator does not require power and uses up less area. A 10MHz crystal resonator was selected with HC49-US packaging to take up less space. The dsPIC30F2010-20I microcontroller has a Phase Locked Loop (PLL) to generate the internal core clock which can be 1, 2, 4, or 8 times the frequency of the supplied clock, up to 20MHz. The crystal was therefore chosen to have a resonant frequency of 10MHz, so that the microcontroller could run at core frequencies of 10MHz and its maximum of 20MHz by proper choice of settings for the PLL multiplier [dsPIC-B].

IC sockets are used for all DIP chips, to allow easy replacement and changing of parts without damaging the board pads and/or part pins. This also enables reusing of the expensive sensors in case substantial changes to the board were required such that a new circuit layout needed to be designed and a new Printed Circuit Board (PCB) be made.

---

### **4.2 - Layout and Assembly**

The PCB layout was designed on Eagle Layout version 4.1. The board was made using the milling machine available at ECE department.

#### **4.2.1 - PCB Layout**

Due to limitations of the circuit board fabrication at the department, the layout of the design had to be single layer double sided. A floorplan of the design is shown in Figure 4.1. The top and bottom layers are shown in Figures 4.2 and 4.3 respectively.

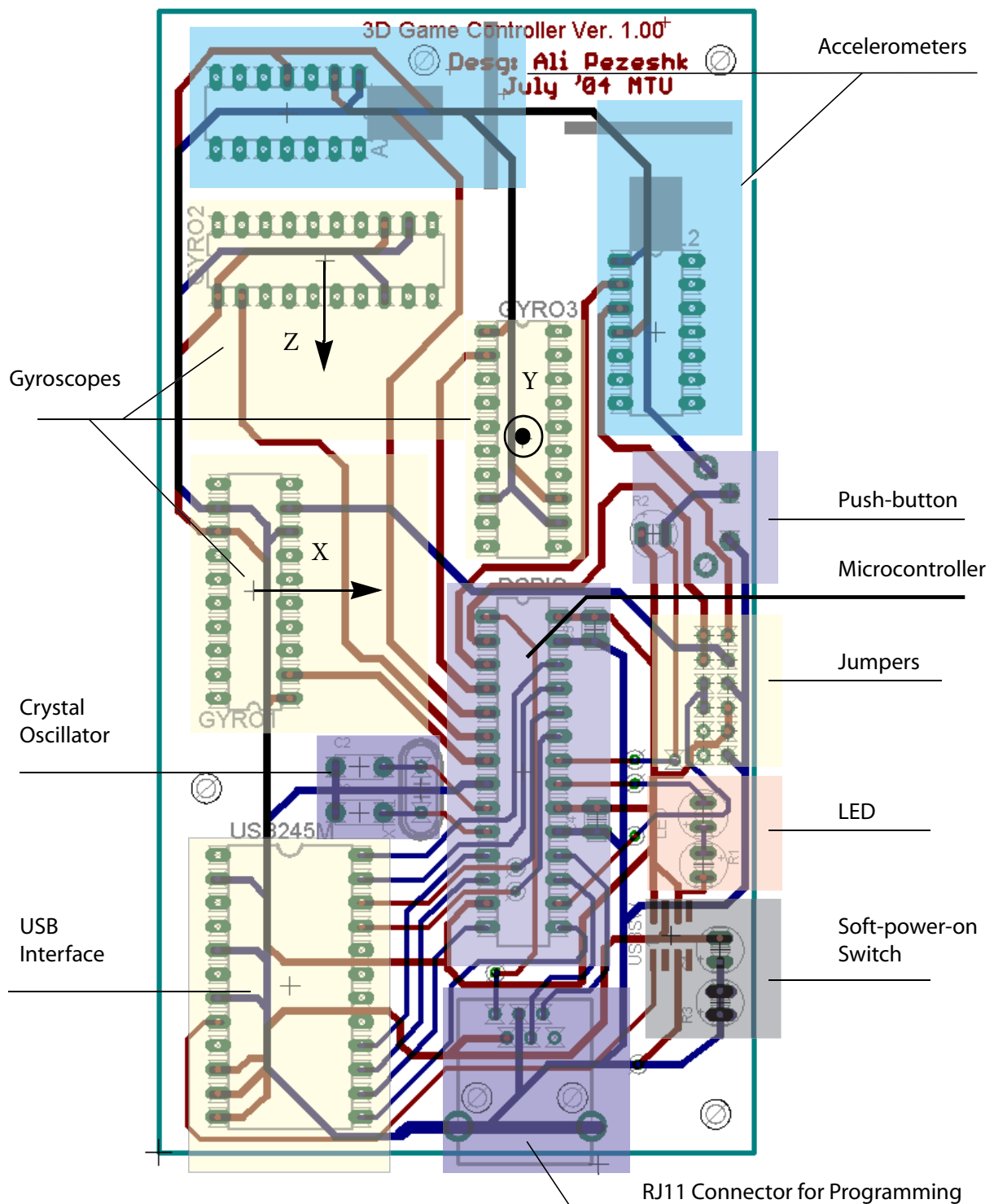
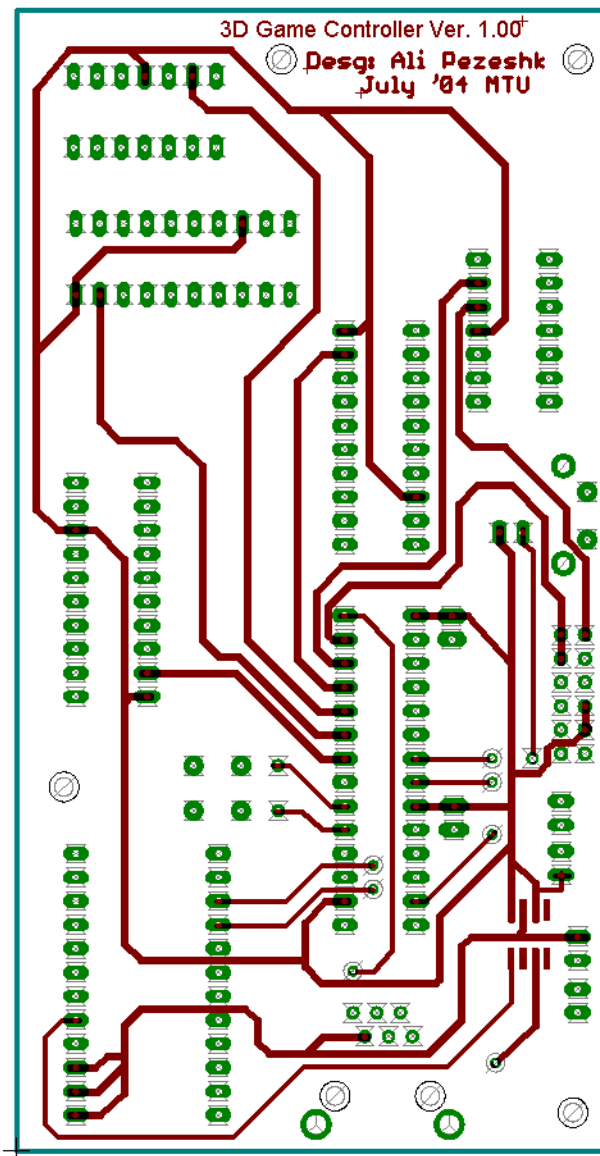


FIGURE 4.1. Layout of the Prototype Design: Floorplan

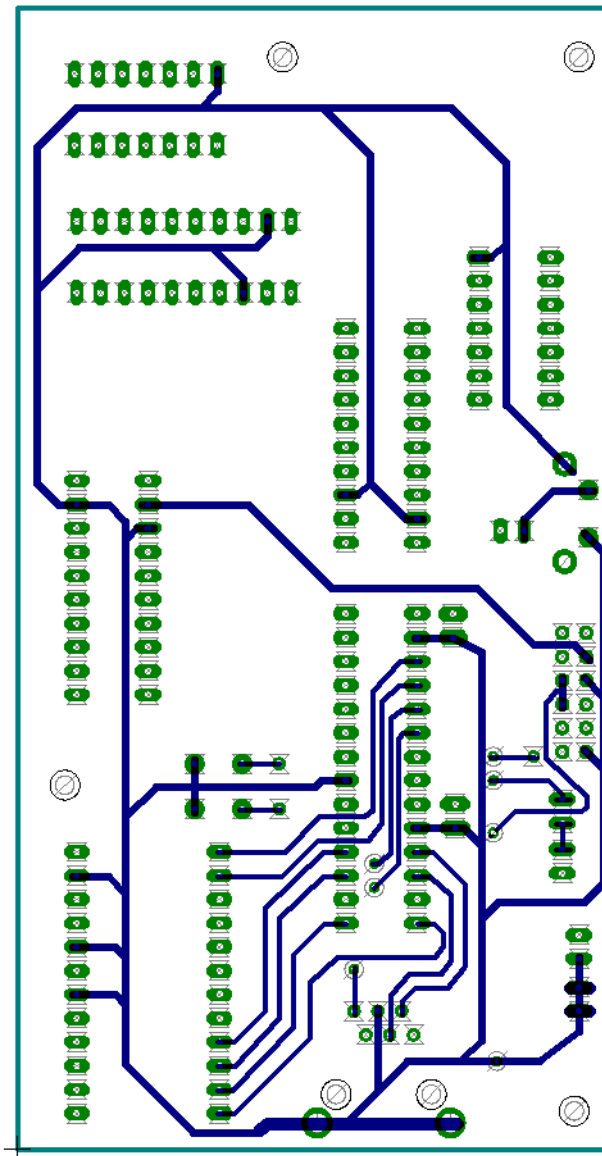




**FIGURE 4.2. Circuit Board Layout of the Prototype Design: Top Layer**

The sensors are placed on the upper area of the board with no other traces except sensor outputs and supply allowed to run through this area for signal integrity. All ground lines are routed at the bottom layer, while all supply lines are routed on the top layer. Sensor supply lines are all routed over ground lines to suppress high-frequency noise.

The jumpers and the push-button are placed on the right-most edge of the board for convenient access for a right-handed person.



**FIGURE 4.3. Circuit Board Layout of the Prototype Design: Bottom Layer**

From the six jumpers available, two are used for possible external power source connection, two are used as an input setting that can be high or low, and the final two are used to switch manually between an IMU output and a temperature sensor on one of the gyroscopes.

The connectors are located at the lower edge of the board to allow convenient connection of the USB cable as well as the in-circuit-programming cable.

Several decoupling capacitors as well as an LED are also incorporated. The LED is specifically useful for early debugging stages and also for tracking fast changing conditions that cannot be caught by the in-circuit-debugger.

The microcontroller only has 20 I/O pins from which one pin is used by the crystal oscillator interface, 6 pins are used as analog inputs for data acquisition from IMU sensors, and two are used for programming/in-circuit-debugging. The USB interface requires a minimum of 12 pins for full byte mode operation which were not available. Instead the circuit is designed in such a way that only half a byte or a nibble of data is transmitted or received at a time. Hence, only 8 pins were dedicated to the USB communication, leaving 3 pins free to be used by the LED, the push-button and a input from the jumpers which can be set to high or low.

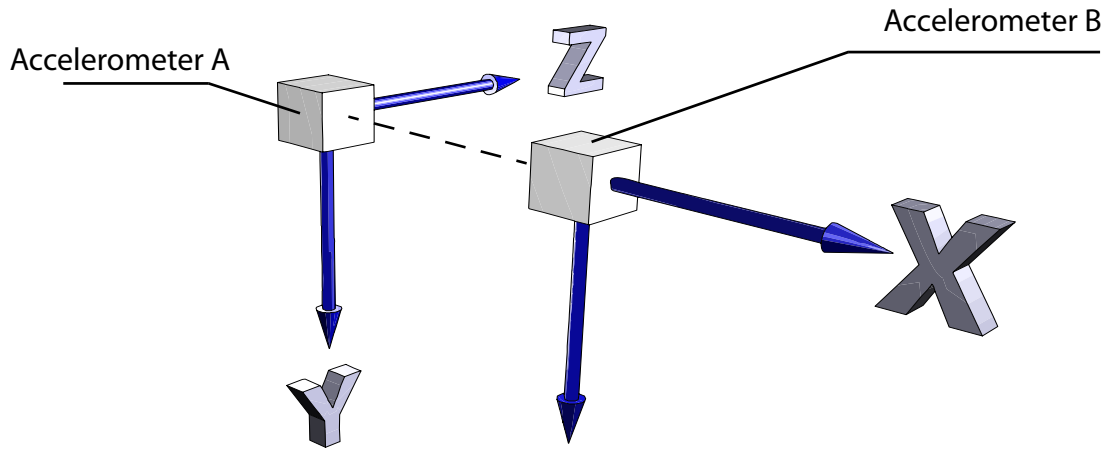
#### **4.2.2 - Placement of Accelerometers**

Since the angular velocity of every point on a solid object is the same, the placement of gyroscopes on the board could be at any arbitrary location. This is however not the case for the accelerometers. Since a 3-axis accelerometer is not used, it is crucial for the sensed data off the accelerometers to be the acceleration components of a single point on the object. Unless extra care is taken in placement of the accelerometers, knowledge of the exact distance between the sensors, and angular acceleration between the two accelerometers and a lot of computational overhead in the software is needed to compensate for induced errors.

Figure 4.4 shows the placement of accelerometers relative to each other. Accelerometer A is sensing the acceleration on  $y$  and  $z$  axes. Accelerometer B is aligned in such a way that its  $x$ -axis lies fully on the  $x$ -axis of accelerometer A. Since the distance between the two sensors is fixed, there is no relative acceleration due to linear motion in the  $x$ -direction. Also, since the two sensors share the same sensing axis, the relative acceleration due to rotations falls completely in the  $y$  and  $z$  directions. This is because the radius of the sphere is orthogonal to the surface of the sphere and rotational acceleration about a point is a tangent to the sphere centered at that point. So, the sensed acceleration in the  $x$ -direction is exactly the same value as would be read by sensor A if it was a 3-axis accelerometer.

This has been taken into account when placing the accelerometers on the board. As a result, two vertical-mount sockets were used to align the accelerometers as shown in Figure 4.1.

Vertical mount sockets are also used for placement of gyroscopes so that their measurement axes are parallel to those of the same coordinate system used by the accelerometers.



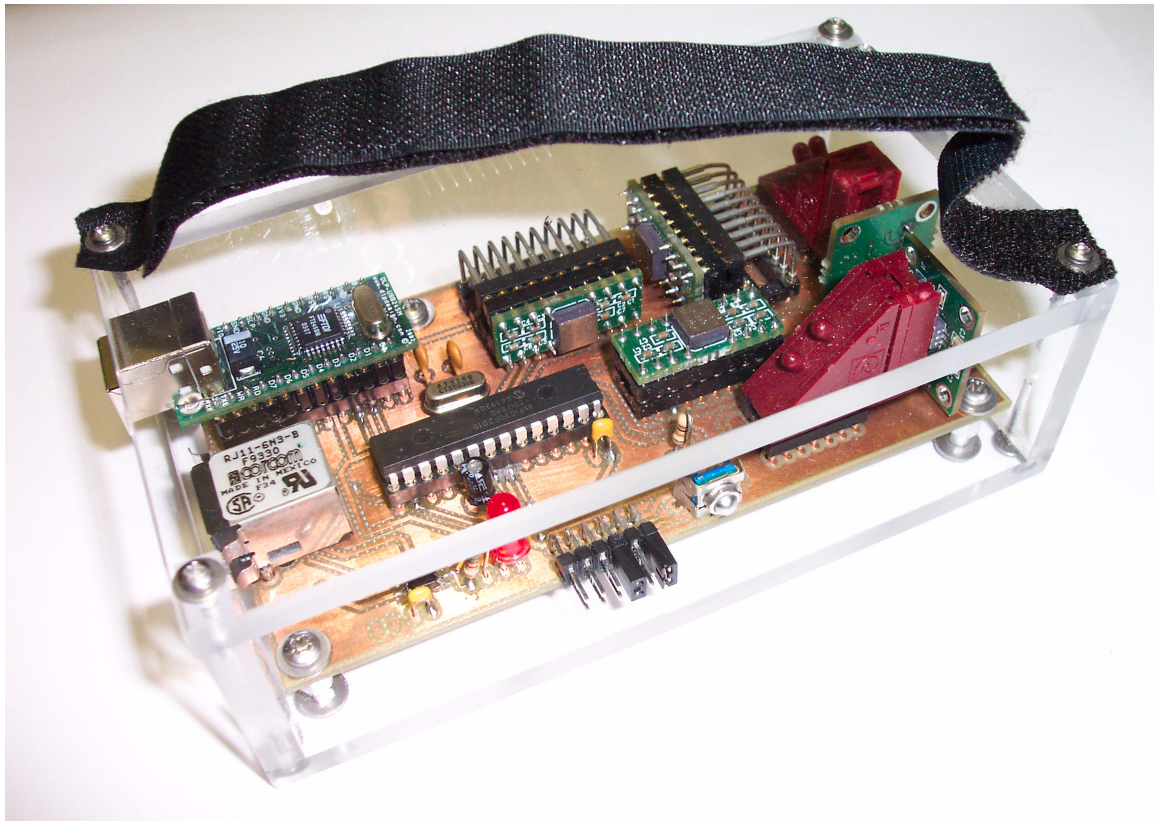
**FIGURE 4.4. Relative Positioning of Accelerometers**

### 4.2.3 - Chassis Design

A chassis was designed to enclose the prototype board having the following in mind:

- **Transparency:** The chassis was decided to be transparent to conveniently allow checking if any parts are loosely connected on the board. As a result the case was made out of Plexiglas.
- **Easy disassembly:** Since the device is a prototype, during the course of debugging it is necessary to be able to fully remove the board from the case. The case is all fit together with screws to allow full or partial disassembly of the chassis.
- **Tight board grip:** Since the sensors used are sensitive to shock and vibration, it is important that the board does not wobble in the chassis. The board is thus, fixed with three screws to bolts mounted on the bottom of the case.
- **Convenient access to ports, jumpers and push-button:** For convenient access, the lower end and right side of the chassis have open windows for ease of access to the push-button, jumpers and the USB and RJ11 ports.
- **Easy grip:** A band of Velcro was added after the chassis was designed to prevent the device from accidental falling off users hand and better grip. This band can be resized to fit with different hand sizes.

The fully assembled prototype in the chassis is shown in Figure 4.5.



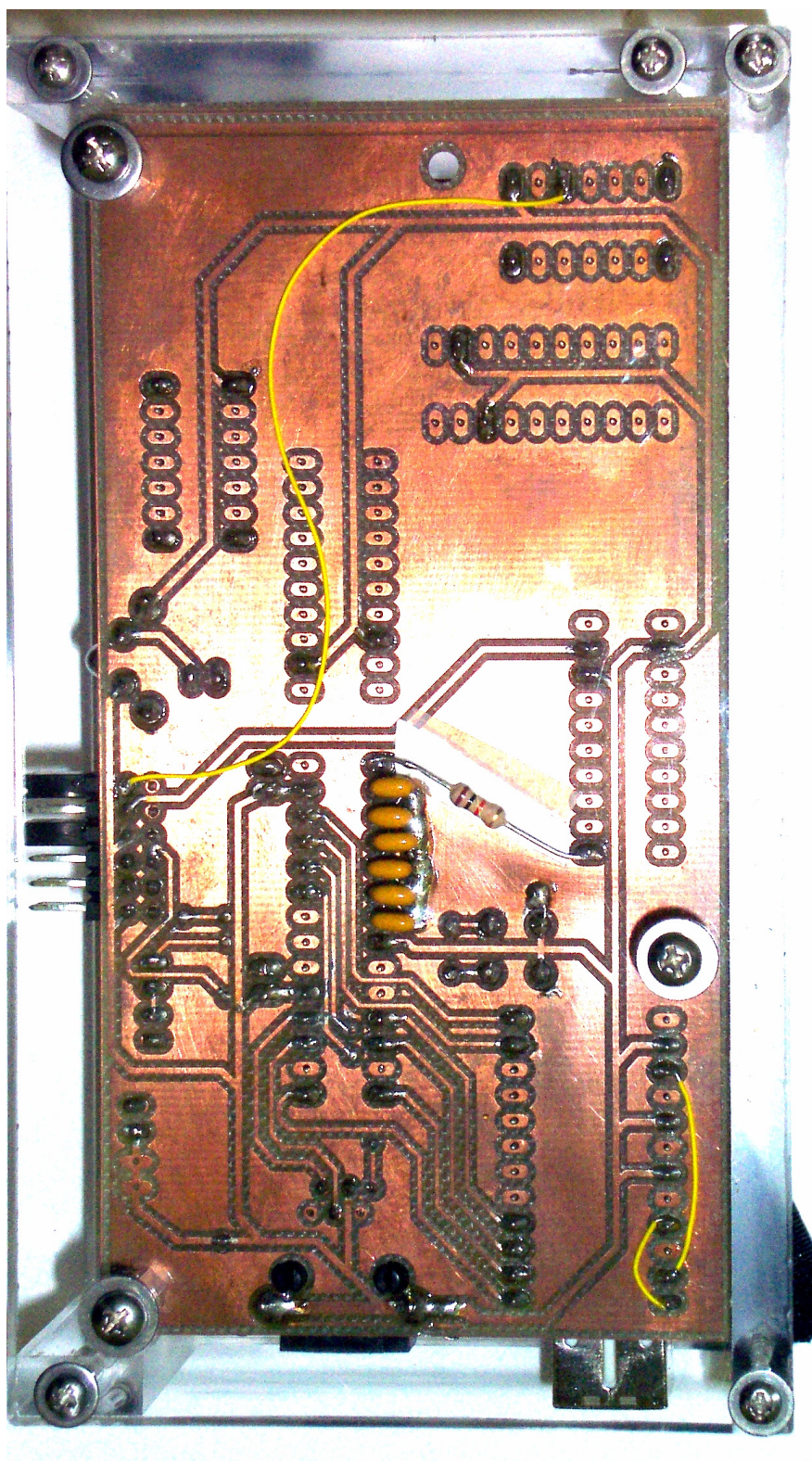
**FIGURE 4.5. Completed Prototype in Chassis**

#### 4.2.4 - Debugging the Board

After soldering the parts and assembly, the device was tested for functionality of different compartments during which, various bugs in the design and assembly were diagnosed and fixed. These include the following:

- Missing connections on the USB interface: two pins on the interface were not connected to supply. These connections were added by thin wires seen in Figure 4.6.
- Oscillator problems: The oscillator on the board, providing the clock signal of the microcontroller was not initially working properly. Since the microcontroller has an internal RC-oscillator, which could be programmed to supply system clock, the functionality of the push-button, LED, sensors and USB connection were tested, using this feature. However, since the RC-oscillator runs at 8MHz and suffers from much higher jitter compared to a crystal oscillator, it was considered essential that the problem be addressed properly. Changing the capacitors of the Pierce configuration, fixed the problem.
- Incorrect connection of acceleration output: The left hand side accelerometer in Figure 4.1 has to have both of its outputs connected to ADC inputs for compliance with





**FIGURE 4.6. Prototype After Debugging: Bottom View**

Figure 4.4. This was fixed by cutting the wrong trace and connecting the correct output to ADC using a thin wire as depicted in Figure 4.6.

- Noise issues: Since the sampled IMU sensor output signals were seen to be noisy, a 100nF capacitor was added at the analog input pin of the microcontroller for each sensor output. Since the ADC of the microcontroller is a charge pumped device and a single converter is shared by multiplexing among all input channels, these capacitors facilitate a fast charge up of the internal capacitor, hence improving the signal to noise ratio of the samples. These capacitors also remove high frequency noise that might be included on the signals. These can be seen as a row of capacitors in Figure 4.6.

### 4.3 - Feature Summary

A summary of the features of the device is given in Table 4.4

**TABLE 4.4. Prototype Feature Summary**

Property	Specification
Connectivity	Wired. USB 2.0 Compliant
Power Source	Bus powered. High-power USB device.
Sensor Ranges	Accelerometers: $\pm 2g$ Gyroscopes: $\pm 300^\circ s^{-1}$
Firmware Upgrading	On board programmable
Other Units	LED Push-button Temperature sensors Jumpers for external supply connection, Microcontroller input and manual switching between an IMU sensor output and the temperature sensor.
Cost	\$350
Dimensions (W x L x H)	3" x 5.25" x 1.75" (7.6cm x 13.3cm x 4.5cm)

# *Software Development*

---

Since the game controller has to communicate with a host machine, the software was developed in parallel on both the controller and the PC. The code written for the microcontroller on the game controller, in brief, is responsible for data acquisition, filtering and transmission. The PC software front-end, on the other hand, has to receive the data sent to it from the game controller, and present them in a reasonable form to the user.

Almost all of the code for the microcontroller is written in C, with the exception of few instances where use of assembly language was unavoidable. The code was developed in the Microchip MPLAB IDE version 6.5 and compiled using the C30 compiler. The code has evolved from a set of test programs for initial debugging to the final version which performs the designated tasks.

The PC used as the host machine for the game controller is chosen to be a Microsoft Windows based machine since the drivers for USB interface were mainly designed for this platform. The code for the PC front-end is written in Visual C++, using Microsoft Visual Studio version 6.0.

In the following sections, details of implementation of the code on both the game controller and the host are discussed.

---

## *5.1 - Game Controller Firmware*

The game controller firmware can be divided into the following sections:



- a) Initialization: Initializes and sets the appropriate settings for the microcontroller.
- b) The 3D game controller Finite State Machine (FSM): This FSM is used to establish the connection with the host.
- c) Task scheduler interrupt: A timer controls and triggers the main events that need to be addressed such as transmission of data packets to the PC or monitoring if the PC host is still listening after a connection is established.
- d) Sampling interrupt: A second timer is used to trigger sampling of IMU sensor outputs, to ensure a fixed sampling rate with equidistant samples. The sampling interrupt is also responsible for initial filtering and unloading the ADC buffer.

### **5.1.1 - Initialization**

On power-on-reset, an initialization routine is executed which carries on the following tasks:

- i) Sets the tri-state selectors for I/O pins of the microcontroller:
  - Pins that are connected to input devices such as push-button are configured as input.
  - Pins connected to output devices such as the LED are configured as outputs.
  - Bidirectional pins such as data-bus connected to the USB interface, are configured as inputs.
- ii) Sets the ADC settings:
  - The six pins connected to the IMU sensor outputs are configured as analog input pins.
  - Positive and negative range of the ADC signals are selected to be 5V supply and ground, respectively.
  - Simultaneous sampling and alternative sampling are activated to sample channels 0, 1, 2, and 3 in phase A and sample channels 2, 3, 4, and 5 in phase B.
  - The ADC buffer is split in two halves to be used as a flip-flop buffer.
  - The ADC interrupt is set to be triggered every two samples, i.e. whenever phases A and B are completed once, resulting in all 6 sensor samples to be taken once.
  - The ADC unit is configured to use the system clock and to be triggered by timer 3 with automatic sampling.
- iii) Initializes the main event timer:
  - The clock source and divisor, and the count value for timer 1 are set so that the counter ticks at 400Hz.
- iv) Initializes the ADC timer:
  - The clock source and divisor, and the count value for timer 3 are set so that the counter ticks at 50KHz resulting in a sampling rate of 25KHz.
- v) Hooks the interrupt handler routines for the main timer and ADC interrupt in the interrupt vector table.
- vi) Starts the ADC unit and the main timer.

### 5.1.2 - 3D Game Controller Finite State Machine

After initialization, the controller will wait for a connection request from host. After reception of this request, the controller will handshake with the PC front-end software and upon successful handshake will enter the active mode where data packets are transmitted to the PC. Figure 5.1 depicts the 3D game controller FSM.

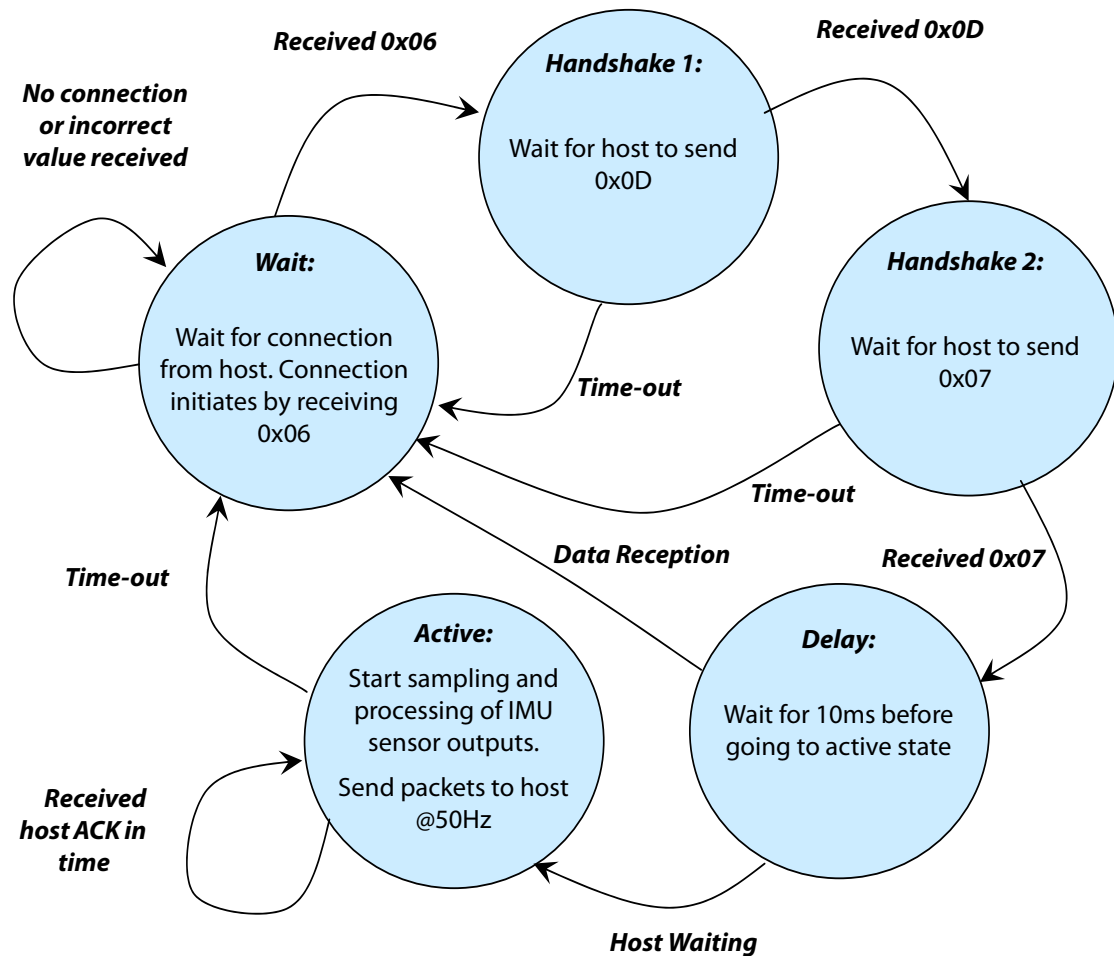


FIGURE 5.1. The 3D Game Controller Finite State Machine, State Diagram

### 5.1.3 - Timing and sampling

Updates on device status and location data are transmitted to the PC at 50Hz. However, to allow filtering, better performance and satisfying Shannon's sampling theorem, four samples are taken in each transmission interval, i.e. sampling rate is 200Hz.

A simple task scheduling algorithm is developed by having 8 time slots for tasks to be performed:

- i) In time slots number 1, 3, 5, and 7, data samples are taken.
- ii) In time slot number 2, a new data packet is sent to the host machine. It is also checked if the connection to the host is still alive, i.e. if any acknowledgment from host has been received within a limited time interval.
- iii) In time slots number 4, 6, and 8, the device monitors the status of inputs, i.e. the push-button and the jumpers, performing debouncing.

To carry out this task, Timer 1 on the microcontroller is programmed as the main event timer with a ticking frequency of 400Hz which is 8 times the packet transmission frequency of 50Hz. The timer interrupt handler routine schedules the tasks as described above.

In order to further improve the quality of samples and to reduce noise, 16 samples at much higher rate are taken and summed to form a *super-sample*. Timer 3 on the microcontroller is set to trigger the ADC conversions at 50KHz. Since the microcontroller samples only 4 channels simultaneously, it takes two sample/conversion cycles for it to sample all the 6 sensor outputs, as a result a sampling frequency of 25KHz is achieved. Since this is about 500 times the cut-off frequency of the analog filters at the output of IMU sensors, these samples should ideally read the same value. So, any variations in the read-out values of samples taken from each sensor output are due to noise.

By averaging these 16 sample values, a four-fold reduction in the variance of the noise can be achieved. Alternatively, by summing the samples, it is as if the resolution of the ADC is increased by 4 bits.

Figure 5.2 depicts the timing diagram of the task scheduler and the acquisition of a super-sample. In sampling time slots, the main event timer interrupt handler enables the sampling interrupt. The sampling interrupt takes 16 samples, unloads the ADC buffer, and performs the summation. After 16 samples are taken, the interrupt handler disables the interrupt and sets a flag showing that super-sample acquisition has completed.

#### **5.1.4 - Packet Structure**

The packet structure is depicted in Figure 5.3. The first 112 bytes of the packet are reserved for future development, where the actual position and orientation data are calculated by the microcontroller on the game controller. It should be noted that the sizes of all fields are twice the actual size of the data transmitted in each field due to nibble transmission in current implementation. Each byte in the packet is of the form 0xUD. U is the high nibble and is all ones, while D is the low nibble and contains the transmitted data.

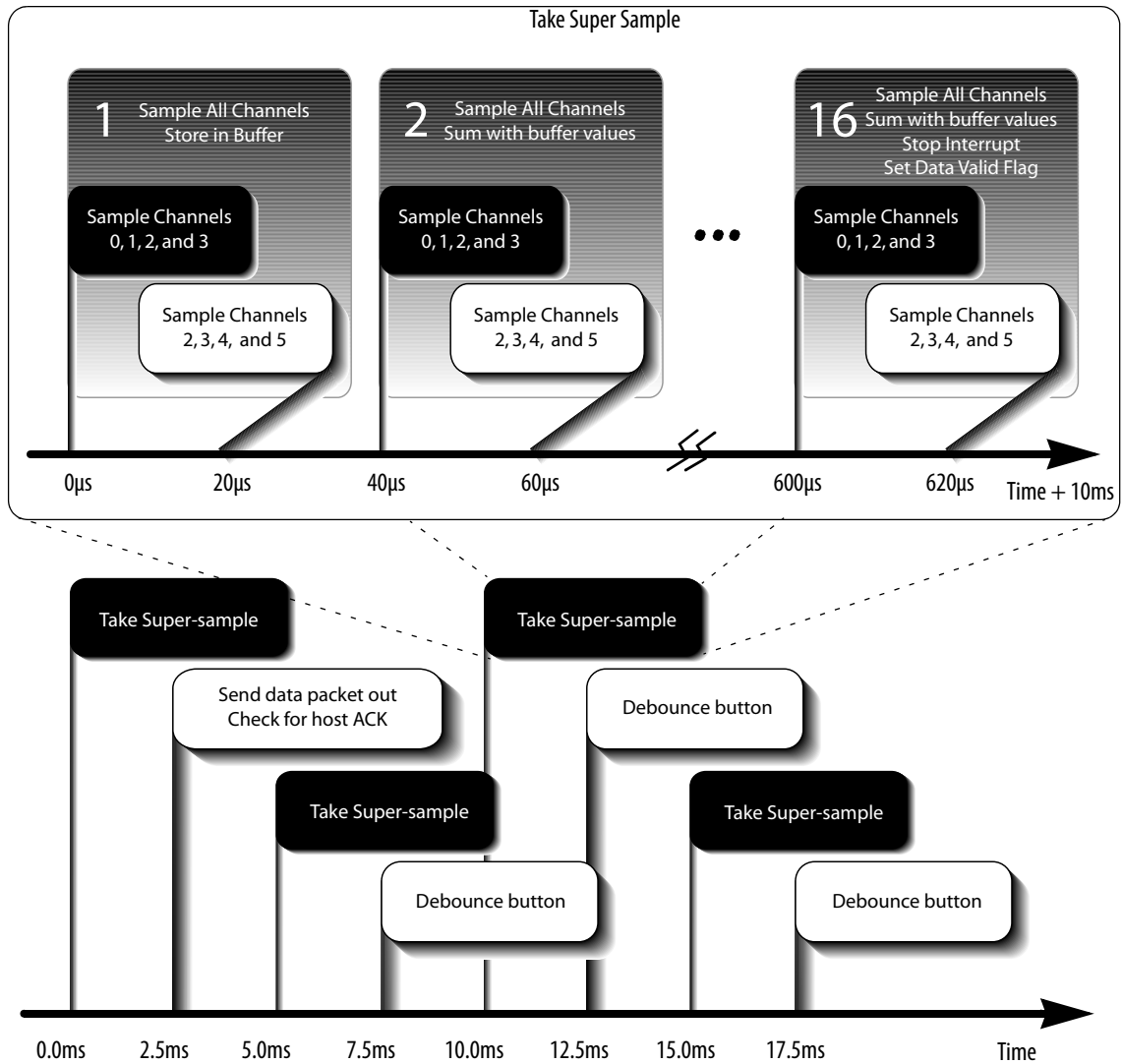


FIGURE 5.2. Task Scheduling and Super-Sample Acquisition Timing Diagram

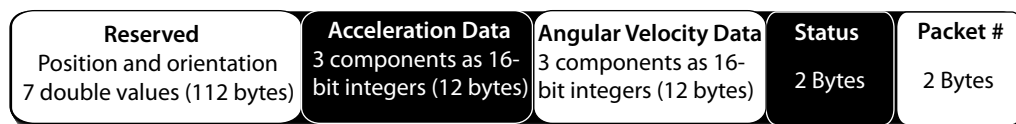


FIGURE 5.3. Data Packet Structure

### 5.1.5 - Orientation Calculation Revisited

The rotation matrix representation of the orientation requires that 9 elements of the matrix be calculated at all times. It is also difficult to compensate for numerical (round-off) error accumulation.

Another approach to representing the orientation data, is to use an extension of complex numbers, called *quaternions*. This approach is widely used in 3D computer graphics software to represent rotations in 3D [Eberly04]. Quaternions are also proven to be more accurate than rotation matrices in calculation of orientation [Titterton97].

The orientation of an object can be represented by a unit length quaternion

$q = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)(iq_i + jq_j + kq_k)$ , where  $(q_i, q_j, q_k)$  is a unit vector parallel to the axis of revolution and  $\theta$  is the angle of rotation [Schneider03].  $i, j$ , and  $k$  satisfy the Hamilton equation:

$$i^2 = j^2 = k^2 = ijk = -1 \quad (\text{EQ. 5.1})$$

The equivalent to Equation 3.2 will then be [Titterton97]:

$$\frac{dq(t)}{dt} = \frac{1}{2}\omega(t)q(t), \quad \omega(t) = \omega_x i + \omega_y j + \omega_z k \quad (\text{EQ. 5.2})$$

Drifts due to machine round-off can be to some extent compensated for, by normalizing the length of the quaternion  $q(t)$  to unity at all times.

Having this in mind, the storage space in the packet dedicated to storage of orientation data is set to four double numbers. These four numbers will be the real and imaginary components of the quaternion  $q(t)$ .

### 5.1.6 - Static Acceleration Compensation

The accelerometers used in the IMU of the prototype also sense the static acceleration of the Earth's gravity. As a result, to calculate the position, this static acceleration has to be removed from the sensor data.

This could be done in hardware by adding series capacitors between sensor outputs and the ADC to block the DC part of the signal. However, additional circuitry would have been needed to add a reference offset DC voltage to these signals to bring them into the 0-5V range of the ADC. The static acceleration data can also be processed to calculate the inclination of the controller, when it is moving slowly. Using DC-blocking capacitors would completely remove the static acceleration and as a result the inclination data can no longer be calculated this way, so this method was not used.

The second solution is to compensate for the static acceleration in software. This can be done by using the orientation data obtained from gyroscope readings to project the gravitational acceleration,  $g$ , onto the coordinate system used by the accelerometers. As a result, an estimate of the static acceleration component in the readings of each accelerometer can be obtained. This component can then be simply subtracted from the ADC reading of the sensor data to result in the dynamic acceleration, which can be used to calculate the position.

---

## 5.2 - PC Front-end

The front-end is an application running on the host machine, that is used for testing the 3D game controller prototype. For debugging purposes, the front-end must have an interface with the user to allow study of the prototype functionality. This is in contrast with a device driver which does not have a user interface and communicates with other programs to give service to them.

The PC front-end was developed in two phases. In the first phase, the software was developed with a text only interface. In the second phase, a graphical interface was added, which allows real-time visualization of the data.

### 5.2.1 - Version 1: Text Based Interface

A console based Windows application was developed in this phase of work. The D2XX Dynamically Linked Library (DLL) supplied by FTDI is used for USB communication [D2XX]. The software is composed of the following sections:

- d) Initialization
- e) Host FSM
- f) Data Reception and Output

During initialization phase, the USB communication routines are loaded from the DLL file. The transmission and reception buffers are then cleared and the USB interface time-outs are set. It is then checked to see if matching devices are connected to the USB port for handshaking to begin.

The host FSM state diagram is shown in Figure 5.4. This FSM is responsible for establishing the connection with the game controller. When the connection is established, data packets are received from the game controller and translated and processed to retrieve the carried information. The data are then output on the screen as hexadecimal numbers for sensor data, and as symbols for status of the controller.

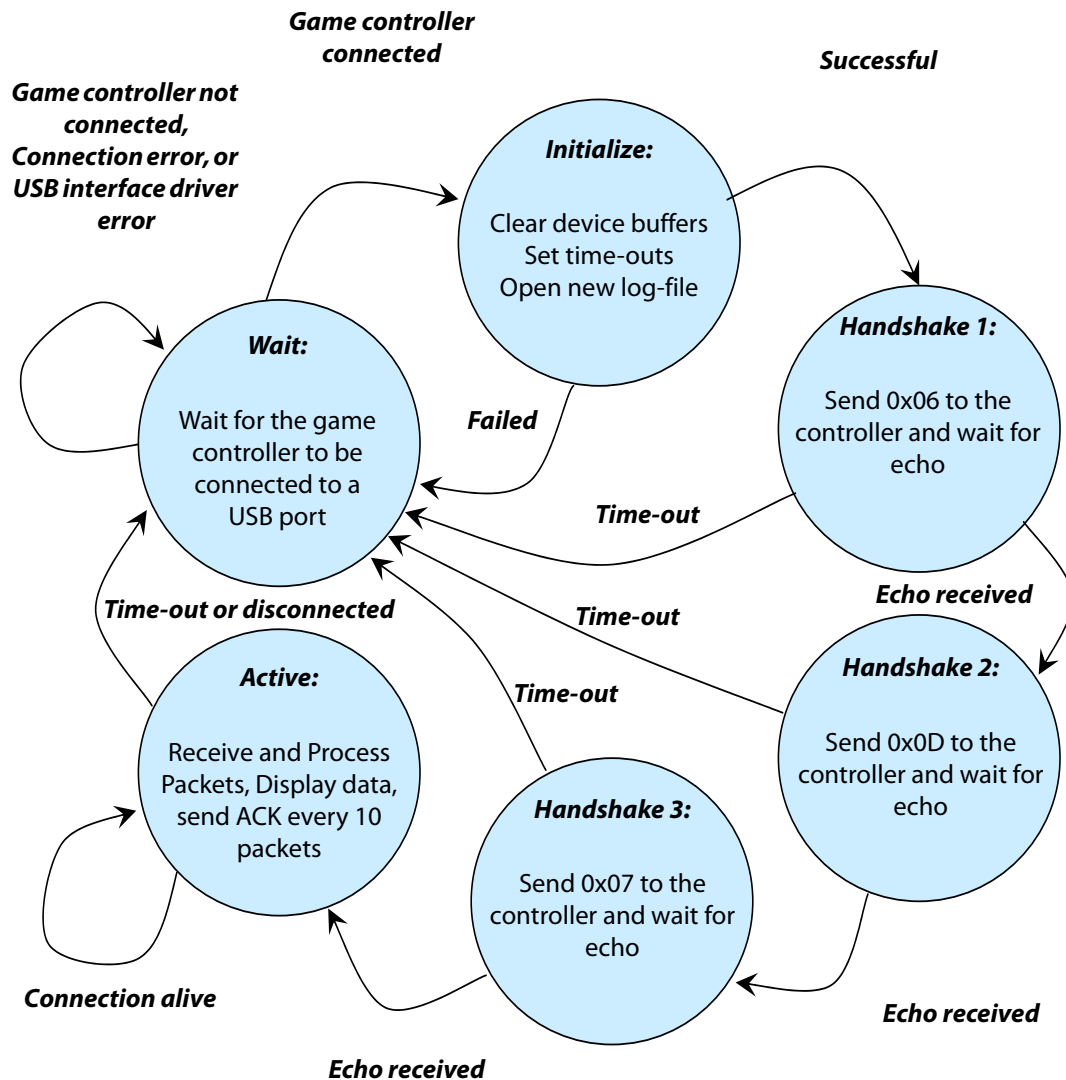


FIGURE 5.4. Host FSM, State Diagram

The status bits show if the button is pressed, where the input jumper is placed, if there has been a clock failure on the game controller, and if an acknowledgment from host is received. These states are represented in the output screen as letters B, J, F, and A respectively.

### 5.2.2 - Version 2: Graphical Interface

The second version of the PC front-end is developed as a Windows Win32 Application. The code was developed using Microsoft Foundation Classes (MFC) [MFC]. A console

window was created in addition to the main frame of the application, where Version 1 output is sent out to be displayed.

Tasks performed in the frame and application instance classes must run in a short period of time, otherwise Windows considers the application as non-responding. As a result, Version 1 code could not be run in any of these classes, since it runs constantly due to its data polling nature. There are two ways to overcome this problem: using a separate running thread for Version 1 code, or using a timer to clock running of this code.

A multimedia timer was used to function as a clock trigger for the host FSM of Version 1 code. Version 1 was therefore wrapped as a separate class which is instantiated in the main application class. It was seen that the periods of less than 50ms for the multimedia timer, cause the system to freeze. Since data packets are sent at a rate of 50Hz by the controller, the code for host FSM of Version 1 was modified to clear the reception buffer by reading as many packets as possible on each entrance to the read function.

For generation of plots an open-source library named Plot Graphic Library (PGL) was used [PGL]. This library in terms uses a set of classes developed by Microsoft in their Windows Software Development Kit (SDK), named as Graphics Device Interface Plus (GDI+) [GDI+].

A set of pointers to plot regions and the graphic interface class instance were added to the class definition of the application. These pointers were then initialized in the application start-up member function.

A circular buffer class was developed to store a history of data received from the game controller for display in plots. Pointers to circular buffers were added to the Version 1 class properties and initialized in the class constructor. Upon reception of a packet, its data is stored in the circular buffers, where it can be later used by the plot routine.

The multimedia timer used for clocking the host FSM also calls the plot routine after every 10 timer ticks [MMTimer]. The plot routine reads the circular buffer contents, stores them in corresponding plot data buffers, rescales each plot, and redraws the plot window with new data.

Super-sample data received from the game controller for all the three accelerometer and the three gyroscope readings are plotted on separate axes. Each plot shows evolution of the quantity being displayed for the past 8 seconds. A screenshot of the graphical interface is shown in Figure 5.5.

---

### 5.3 - Software Debugging

The software for the microcontroller as well as the host front-end have gone through considerable changes during development. Debugging of the code has formed the final





**FIGURE 5.5. Screenshot of the Graphical Interface**

FSMs presented the prior sections and fundamental modifications in the way data is acquired, processed and transmitted.

Debugging of code was done at early stages on the primitive tasks of both sides, such as data transmission and reception. Data acquisition was at first done by sampling IMU sensor outputs at the maximum rate possible and transmission of the samples without any processing to the host. This caused USB channel flooding, causing loss of data. Another problem with this method was the lack of a robust packet structure and connection monitoring. As a result, it was unclear which sensor reading the received data is representing.

The interrupt driven approach on the game controller addressed the flooding problem. The improved packet structure with the added packet number field and acknowledgments from both sides allowed detection of packet losses and an overall more reliable connection.

# *Results & Future Work*

---

Using the graphical version of the PC front-end, a number of tests on the prototype were performed. These tests, their results, and a brief discussion on each test are presented in this chapter.

The remainder of the chapter is dedicated to future improvements envisioned for the system in order to make it more robust, versatile, user friendly, and ultimately ready for commercialization.

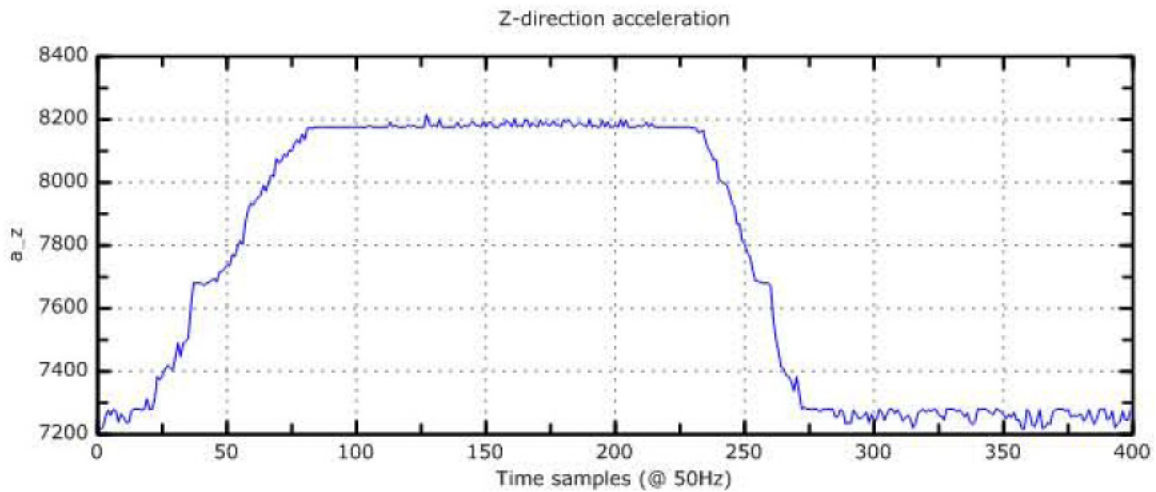
---

## *6.1 - Results*

Using the second version of the PC front-end a number of tests were performed to validate the assertions in former chapters. Rotation around each of the three axes were performed at various speeds to check the response of the gyroscopes. Motion along each of the three axes was performed to check the response of the accelerometers. The functionality of the temperature sensor was also tested.

### *6.1.1 - Sensor Data*

The accelerometers used in the prototype measure the static acceleration due to the Earth's gravity, proportional to  $g$ . Therefore, the first tests were to check if each accelerometer is functioning properly, by rotating the accelerometer so that its measuring axis will be in the direction of  $g$ , perpendicular to  $g$ , and in its opposite direction. The results for a sample motion data is shown for the  $z$ -



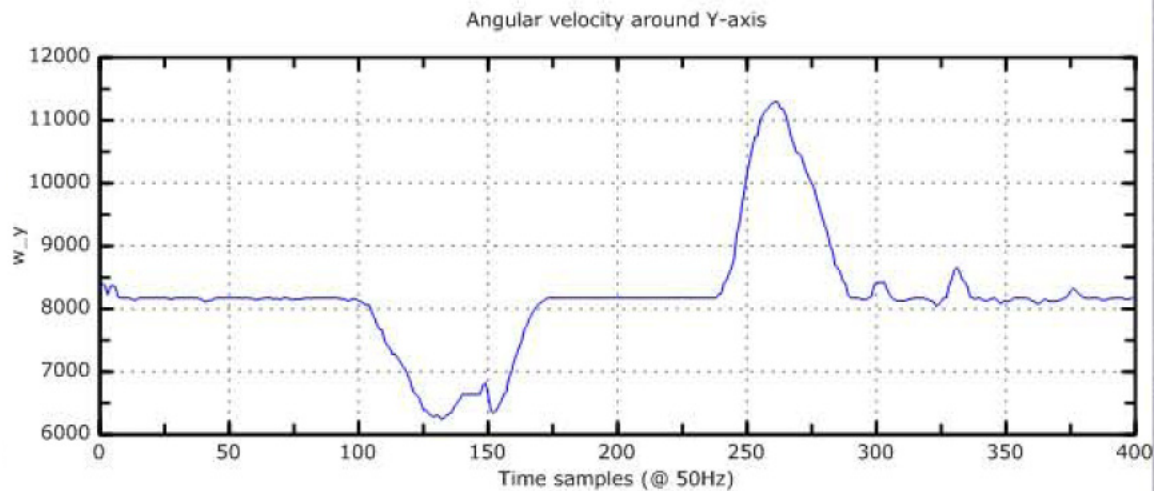
**FIGURE 6.1. Sample Sensor Reading for Acceleration: Rotation Around Y-axis**

direction accelerometer in Figure 6.1. The device was held upright at the beginning, so that the  $z$ -direction accelerometer would see  $-g$ . The controller was then rotated 90 degrees so that the  $z$ -direction accelerometer would be perpendicular to  $g$ . The device was then rotated 90 degrees in the opposite direction to come back to its original state. As can be seen, the accelerometer is working properly, with outputs of  $-g$  ( $\sim 7250$ ), 0 ( $\sim 8192$ ), and again  $-g$  ( $\sim 7250$ ). Note that the sensor output after super-sampling is an unsigned value between 0 and 16383 but the quantity being measured is signed. From these data, an approximate estimate for the coefficients in a linear model for the accelerometers can be calculated. These calculations, however have limited accuracy since no precise reference or mechanism is used to make sure that the accelerometer axes is indeed exactly in the direction of  $g$  or perpendicular to it.

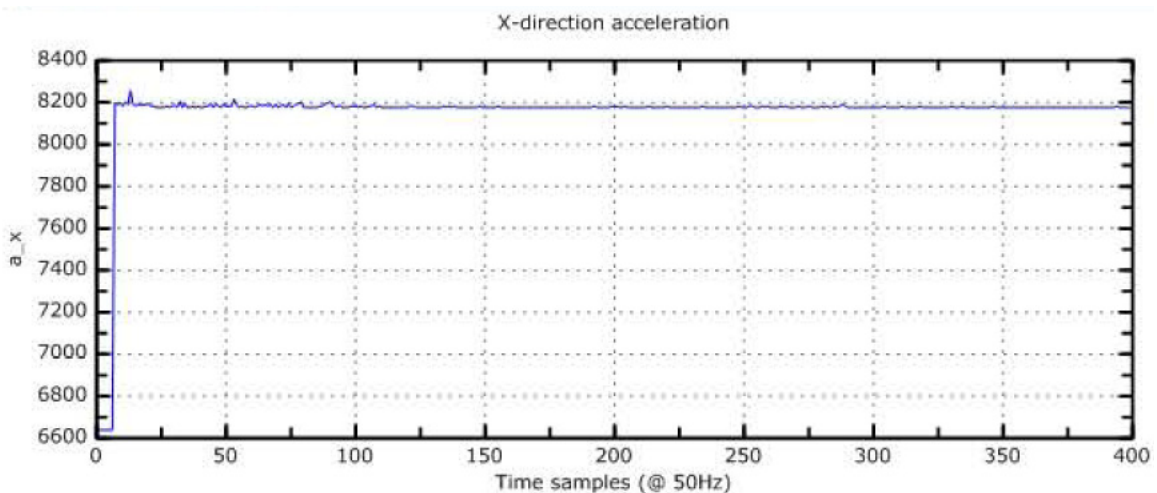
It should be noted that these results are slightly different on different hosts, since the analog supply voltage for the ADC and the sensors, taken from the USB bus, varies from machine to machine.

Next, the functionality of the gyroscopes were tested by rotating the game controller around the axes of each of the gyroscopes, one at a time. A result for a sample motion for one of the gyroscopes is shown in Figure 6.2. The game controller was held at rest for some time, then rotated 90 degrees clockwise, stayed at rest there for a moment, and then returned to initial orientation. As can be seen, the gyroscope is working properly. The irregularities seen in the output are mainly due to nonuniformity of the motion of human operator.

The temperature sensor was tested by manually switching the input of the first channel of the ADC to temperature sensor output by switching the respective jumper connection. As can be seen in Figure 6.3, the sensor is working properly. The step-like behavior at the



**FIGURE 6.2. Sample Sensor Reading for Angular Velocity: Rotation Around Y-axis**

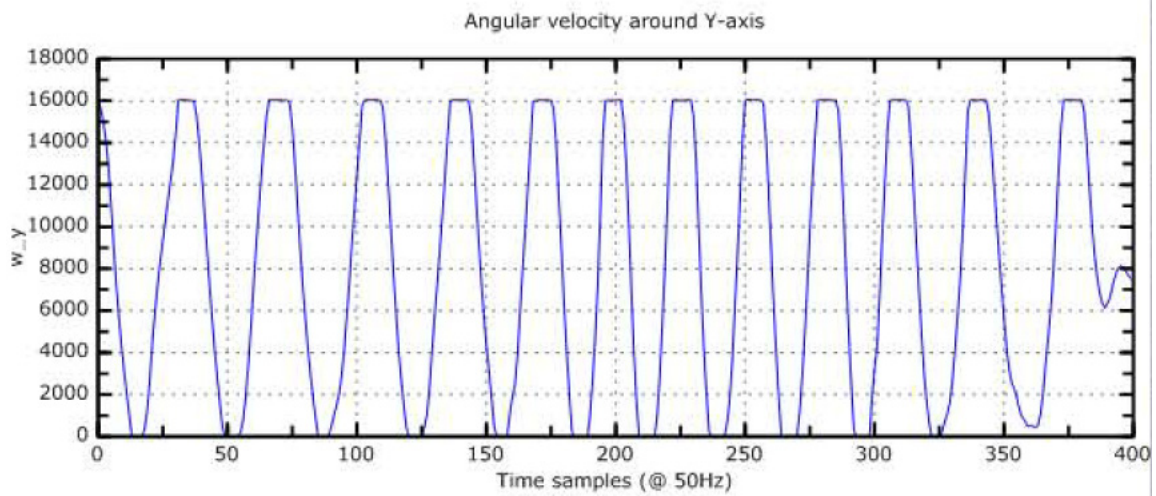


**FIGURE 6.3. Sample Sensor Reading for Temperature**

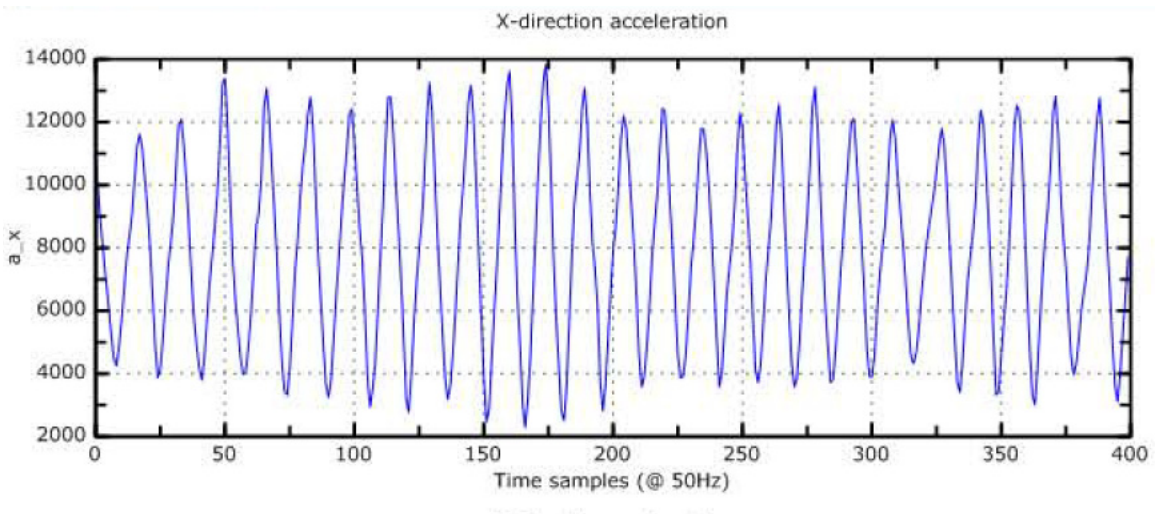
beginning of the waveform is due to the manual switch from accelerometer sensor output to temperature sensor output.

### 6.1.2 - Sensor Saturation

To test the coverage range of sensors, the game controller was moved and rotated at different speeds. As can be seen in Figure 6.4, the gyroscopes will saturate once the prototype is rotated very fast. The range can be extended by adding a resistor to the circuit [Gyro-C], but this circuit modification will sacrifice the sensitivity and readout resolution of angular rate data.



**FIGURE 6.4. Gyroscope Saturation: Fast Rotation Around Y-axis**



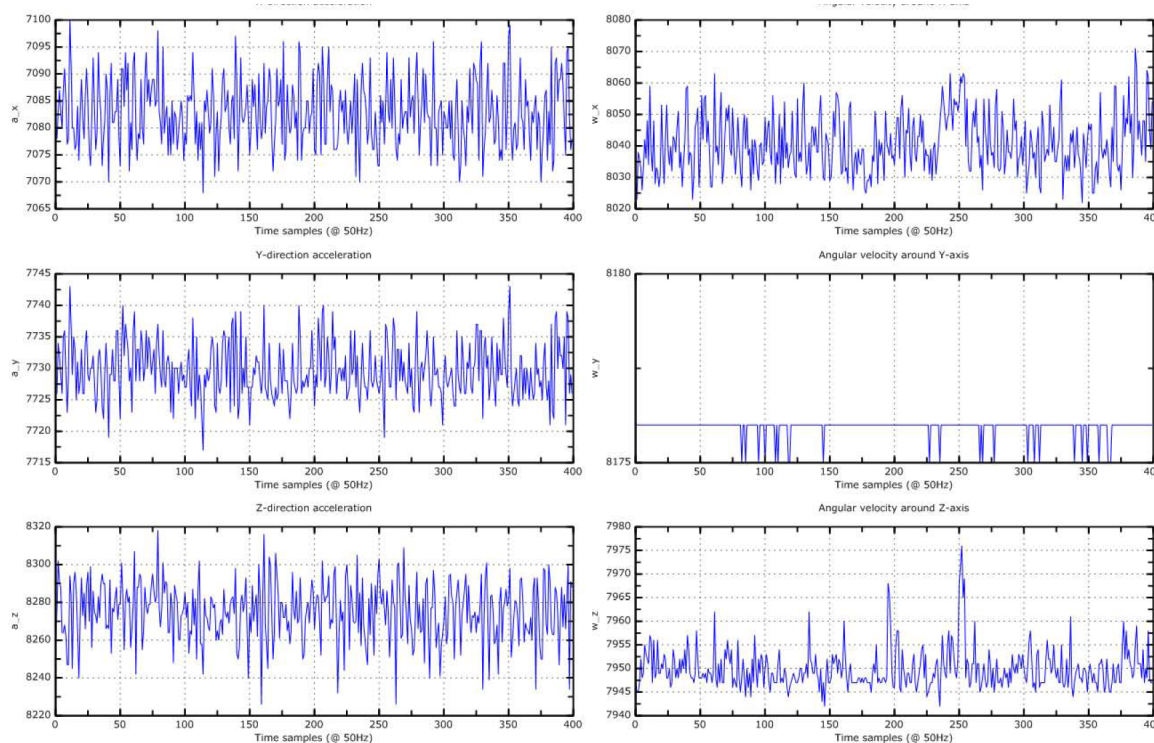
**FIGURE 6.5. Sample Accelerometer Reading: Very Fast Motion Along X-axis**

Similar tests with linear motion were performed, to find the coverage range of accelerometers. However, as can be seen in Figure 6.5, the accelerometers do not saturate even at very fast motion. Saturation of accelerometers was only seen during collisions of the game controller with a hard surface.

### 6.1.3 - Sensor Noise

Using autoscaling of limits of the graphs, it was easy to see the noise on the readings. Since the pattern and variance of the noise was seen to be different when the game controller was connected to different machines, it can be deduced that the supply voltage from the USB is





**FIGURE 6.6. Sample Noise on Sensor Readings**

a source for the noise. Another observation was that, depending on the host machine the system is connected to, some sensors may show a noise level below the precision of the ADC. A sample of such behavior is shown in Figure 6.6, where angular velocity around  $y$ -axis shows almost no noise. The device was at rest in this test.

## 6.2 - Future Work

The future improvements to the system can be categorized into the following branches:

- Hardware enhancements
- Implementation of the position and orientation calculation and numerical methods
- Filtering and noise reduction techniques
- Host Application development

### 6.2.1 - Hardware Enhancements

The design can be significantly improved using a better microcontroller. A higher number of I/O pins will allow full-byte USB communication compared to the current nibble-transmission. This results in twice the current bandwidth efficiency. Additional I/O pins will also allow more push-buttons to be added to the design, which can significantly

improve productivity and usability. For instance, a push-button can be used to indicate that the game controller is just being relocated physically and its position in the virtual environment should be preserved. This is similar to picking up the mouse and putting it somewhere more convenient for the user. Another button can be used for changing the view-point rather than manipulating objects in the virtual environment. Also, a commercial game controller has to have a number of push-buttons for various controls required by a computer game.

More I/O pins also allow analog multiplexers to be added to the design to change the range of the sensors by software control. For instance, the gyroscope output could be selected to be in its normal range, while not near saturation, and to change to a wider range by digitally bringing a resistor into the circuit. This way, both wider range and high accuracy can be achieved. A hysteresis curve for switching can be used to reduce the number of switches and result in a more reliable performance.

Higher number of ADC channels, allows temperature to be sensed and additional compensations for the sensor parameter drifts to be made in the software. More ADC channels also allows having two sensor ranges. The high range sensor can be used during saturation of the low range sensors for device stability, while the low range sensors provide more accurate readings.

More RAM on the microcontroller allows storage of a longer history of data and enables better filtering schemes to be implemented on the device for higher performance. Additional RAM also allows more sophisticated numerical methods and higher precision floating point arithmetic to be used.

The Microchip dsPIC30F4011-30I is a good candidate and has 9 ADC channels, 30 I/O pins, 2KB RAM, and a processing power of 30 MIPS [dsPIC-C]. Compared to the current microcontroller used in current prototype, these are 50%, 50%, 400%, and 50% more, respectively. Since these devices belong to the same family, the code is fully portable, which reduces the prototyping time significantly.

Addition of a magnetic compass to the design will allow compensation for drifts in the orientation estimates. The readings of this sensor together with the inclinations sensed by the accelerometers even at relatively low frequencies, e.g. every few seconds, give an estimate of the orientation that can be used to correct the drifts in orientation estimates.

A wireless communication module, e.g. for IEEE 802.11, can replace the USB communication interface, to allow untethered connection to the host system. A power supply or battery needs to be added to the design in this case.

As shown early in this chapter, the sensor and ADC power supply require further filtering, to minimize their effects on the performance of the system. It might as well be necessary to use a different power source than other than the USB bus for analog signals to ensure better signal to noise ratio in the sensor readings.

Adding a wheel to the design and using the quadrature encoding capabilities of the microcontroller would allow on-the-fly scaling of the position resolution of the system. This is especially helpful if an accurate small motion is required. In this case, the wheel can be turned in the direction of increasing resolution, so that a large movement is needed physically, to create a small scaled version of the movement in the virtual environment.

### ***6.2.2 - Position and Orientation Calculation, and Numerical Methods***

Code for calculation of position and orientation calculation can be developed based on the discussions in Chapter 3 and Chapter 5. It should be noted however, that the derivations presented in the aforementioned chapters all assumed that the data read from sensors is converted to actual values. Calibration of sensors in order to find sensor response curves is a prerequisite to this assumption.

Numerical methods to be used for calculations need to be further studied. For instance, the performance of the system might be better if a numerical integration method [Akai94] is used instead of a numerical ODE solver [Shampine94]. The stability of the method and its accuracy, given the constraints of the system on sampling, together with its computational complexity are among the factors that need to be considered in implementation of the position and orientation calculation [Higham02].

### ***6.2.3 - Filtering and Noise Reduction***

Filtering of the sensor readings directly affects the performance of the system. Though a digital filter with constant coefficients can reduce the noise in the system to some extent, an adaptive filter delivers superior performance [Haykin01].

Kalman filters have been widely suggested for INS applications [Grewal01] [Shin04]. An attractive feature of these filters is their high tolerance to imperfections in the design model and their ability to work with non-stationary processes. This is specifically the case for the game controller, where the input motion of the user is non-stationary.

The Kalman filter approach can be further improved using maneuver detection techniques [Gustafsson00]. Since the human motion is not continuously at the same pace, there are times that the motion is relatively slow and times that the motion is fast. When the motion is slow, a filter with smaller bandwidth can be used and more noise can be rejected. Detection of when the motion changes is required for this purpose and is referred to as maneuver detection in navigation systems.

### ***6.2.4 - Host Application Development***

The choice of the host system determines how the USB connection should be established by the software, how the data should be presented to the user application, and how the data can be visualized.



After the host system has been selected, it is possible to develop a driver that communicates with the game controller and relays the data received from it to the user application. Applications can then be developed using this driver which would allow virtual manifestation of an object which can be controlled by the user. The controller can be used in three different modes:

- Virtual world navigation mode: In this mode the trends of motion, i.e. up, down, etc., can be used to navigate the character in the virtual environment. This is especially useful when the character is moving in a big terrain, whereby using this mode relieves the user from running in a field of the same size as the virtual terrain and significantly mitigates the effects of the drift in results. Different levels of speed of motion can be differentiated based on how far the user has moved the game controller in a direction. For instance if the controller is moved forward in the direction of line-of-sight axis, a little bit, the character in the virtual environment starts walking slowly in forward direction. If the controller is moved farther forward, the character starts walking and if the controller is moved even farther the character starts running.
- Viewpoint control mode: In this mode the motion of the game controller can be used to control the viewpoint of the user to the virtual environment. Movement along the line-of-sight axis can be used as a zoom-in/zoom-out feature.
- Object manipulation mode: In this mode the motion of the controller directly controls the motion of an object in the virtual environment. For instance the user can control a light saber in the virtual environment as if (s)he is holding one in her/his hand by moving and turning the game controller in her/his hand.

A host application and driver can also be developed on a gaming console. The availability of the Linux Development Kit for Sony PlayStation 2, makes this console a good candidate for the development of the host application and driver.

---

# References

- 
- [Accel-A] Analog Devices: *Low-cost, Ultracompact  $\pm 2g$ , Dual-axis Accelerometer*, 2003.
- [Accel-B] Analog Devices: *Dual-axis Accelerometer Evaluation Board ADXL311EB*, 2003.
- [Akai94] Terrence J. Akai: *Applied Numerical Methods for Engineers*, John Wiley & Sons, 1994.
- [Axiom99] Axiom Manufacturing: *CME11E9-EVBU Development Board*, Reference Manual, 1999.
- [Axiom03] Axiom Manufacturing: *Buffalo Monitor for HC11 Development Boards*, Reference Manual, 2003.
- [Bowman05] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola Jr., Ivan Poupyrev: *3D User Interfaces, Theory and Practice*, Addison Wesley, Pearson Education Inc., 2005.
- [D2XX] Future Technology Devices International (FTDI) Ltd.: *D2XX Programmer's Guide Version 2.01*, 2002.
- [dsPIC-A] Microchip Technology Inc.: *dsPIC30F Family Reference Manual, High Performance Digital Signal Controllers*, 2004.
- [dsPIC-B] Microchip Technology Inc.: *dsPIC30F2010 Data Sheet, 28-pin High Performance Digital Signal Controllers*, Preliminary Version E, 2004.
- [dsPIC-C] Microchip Technology Inc.: *dsPIC30F4011/4012 Data Sheet, High Performance Digital Signal Controllers*, Advance Info. Version B, 2004.
- [Eberly04] David H. Eberly: *Game Physics*, Morgan Kaufmann Publishers, Elsevier Science, 2004.

- [Fastrak] Polhemus: *Fastrak, The Fast and Easy Tracker*, Product Brochure, 2004.
- [Foxlin02] E. Foxlin: "Motion Tracking Requirements and Technologies", *Handbook of Virtual Environments: Design, Implementation, and Applications*. K. Stanny (Ed.), Lawrence Elbraum Associates, pp. 163-210, 2002.
- [GDI+] Microsoft Corp.: *GDI+*, MSDN Reference, Available online at: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdicpp/GDIPlus/GDIPlus.asp>
- [Geen04] John A. Geen: "Progress in Integrated Gyroscopes", *Proc. IEEE Position Location and Navigation Symposium (PLANS)*, pp. 1-6, 2004.
- [Grewal01] Mohinder S. Grewal, Lawrence R. Weill, Angus P. Andrews: *Global Positioning Systems, Inertial Navigation, and integration*, John Wiley & Sons, 2001.
- [Gustafsson00] Fredrik Gustafsson: *Adaptive Filtering and Change Detection*, John Wiley & Sons, 2000.
- [Gyro-A] Analog Devices Inc.:  *$\pm 300^\circ/s$  Single Chip Yaw Rate Gyro with Signal Conditioning*, 2004.
- [Gyro-B] Analog Devices Inc.:  *$\pm 300^\circ/s$  Single Chip Yaw Rate Gyro Evaluation Board*, 2003
- [Gyro-C] Harvey Weinberg: *Modifying the Range of the ADXRS150 & ADXRS300 Rate Gyros*, Application Note AN-625, Rev. 0, Analog Devices, 2003
- [Haykin01] Simon Haykin: *Adaptive Filter Theory*, 4th Ed., Prentice Hall, 2001.
- [Higham02] Nicholas J. Higham: *Accuracy and Stability of Numerical Algorithms*, Society of Industrial and Applied Mathematics (SIAM), 2002.
- [Kim04] Anthony Kim, M. F. Golnaraghi: "A Quaternion-Based Orientation Estimation Algorithm Using an Inertial Measurement Unit", *Proc. IEEE Position Location and Navigation Symposium (PLANS)*, pp. 268-272, 2004.
- [Lawrence98] Anthony Lawrence: *Modern Inertial Technology, Navigation, Guidance, and Control*, 2nd Ed., Springer Verlag, 1998.
- [Logitech92] Logitech: *3D Mouse and Head-Tracker, Technical Reference Manual*, 1992.
- [MFC] Microsoft Corp.: *MFC Development Using Microsoft Visual C++ 6.0*, Microsoft Press, 2000
- [MMTimer] Leslie Sanford: *Wrapper Class for Multimedia Timer Functions*, Available at the Code Project website: [http://www.codeproject.com/audio/mult\\_media\\_timer.asp](http://www.codeproject.com/audio/mult_media_timer.asp)
- [Pezeshk04] Ali Pezeshk, Mehdi Imaninejad: "A 3D Computer Game Controller: Design and Application", *Proc. SIGGRAPH*, Los Angeles, 2004.

- 
- [PGL]** Jonathan de Halleux: *Plot Graphic Library*, Available at The Code Project website: <http://www.codeproject.com/miscctrl/pgllib.asp>
- [Schneider03]** Philip J. Schneider, David H. Eberly: *Geometric Tools for Computer Graphics*, Morgan Kaufmann Publishers, Elsevier Science, 2003.
- [SeeReal04]** SeeReal Technologies: *Autostereoscopic 3D Display, 'C' Product Line, Operating Manual*, 2004.
- [Shampine94]** Lawrence F. Shampine: *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, 1994.
- [Shin04]** Eun-Hwan Shin, Naser El-Sheimy: "An Unscented Kalman Filter for In-motion Alignment of Low-cost IMUs", *Proc. IEEE Position Location and Navigation Symposium (PLANS)*, pp. 273-279, 2004.
- [Titterton97]** D. H. Titterton, J. L. Weston: *Strapdown Inertial Navigation Technology*, IEE, Peter Peregrinus Ltd., 1997.
- [USB-A]** Future Technology Devices International (FTDI) Ltd.: *FT245BM USB FIFO (USB - Parallel) I.C.*, Version 1.4, 2004
- [USB-B]** Future Technology Devices International (FTDI) Ltd.: *Data Throughput, Latency, and Handshaking*, Application Note, 2004
- [USB-C]** Future Technology Devices International (FTDI) Ltd.: *FT232BM and FT245BM Power Control and Pin States*, Application Note, 2004
- [USB-D]** DLP Desgin Inc.: *DLP-USB245M User Manual*, 2002

