Inertial-Optical Motion-Estimating Camera for Electronic Cinematography

by
Christopher James Verplaetse

B.S., Aerospace Engineering
Boston University
June 1994

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN MEDIA ARTS AND SCIENCES
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1997

Author

Program in Media Arts and Sciences
March 15, 1997

Certified by

Neil Gershenfeld
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by

Stephen A. Benton
Chairperson, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Inertial-Optical Motion-Estimating Camera for Electronic Cinematography

by
Christopher James Verplaetse

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on March 15, 1997
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

Abstract

The IOME Cam (inertial-optical motion-estimating camera) system estimates a video camera's motion using both optical and inertial data. Optical, or vision-based motion estimation, has a number of performance limitations based on the nature of its input: external motions as well as pixel noise, shadows, and occlusions in the optical field cause errors. Similarly, pure inertial motion estimation experiences errors that grow quadratically with time. This thesis examines a system that combines the benefits of both data types and uses each to correct for the other's errors. Motivating applications for a self-motion-estimating video camera, such as graphical and physical modeling of a scene, are discussed. The Hummingbird inertial navigational hardware, designed and built for this project, is also described herein. Additionally, several related proprioceptive devices are presented.

Thesis Supervisor: Neil Gershenfeld
Title: Associate Professor of Media Arts and Sciences

This work was supported by the Things That Think and News In the Future consortia.

Inertial-Optical Motion-Estimating Camera for Electronic Cinematography

by

Christopher James Verplaetse

Acknowledgments

The endeavor to complete this thesis has been a long one and many kind people have helped me along the way.

Above all, I want to extend my gratitude to my advisor and my readers for their guidance, support, and amazing patience. I thank Neil Gershenfeld, my advisor and an inspirational person, for allowing me to take part in his wonderful group, for his constant jet of brilliant and inquisitive ideas and his parallel ability and will to maintain an interest in each member of his team, for his guidance and flexibility, and for his demand for focus. Walter Bender, I thank for helping me with all aspects of my thesis, and for giving me the opportunity to come to the Media Lab and to experience this unique place to learn, teach, invent, and play. I thank Mike Bove for helping me understand many aspects of the IOME Cam project: visual and inertial signals, communications systems, and electical engineering aspects; and for helping me refocus on the realities of the project a number of times.

A good deal of friends have helped me through this project, and other trying issues, during its tenure. Jon Orwant has been a good friend and unofficial advisor/guide around MIT; he made me feel at home at the 'Tute and in the garden. Michelle McDonald, a gifted young woman and a good friend, taught me how to program in X, helped me with presentations, and kept me sane and stable through many Karman streets. Chris Turner, another invaluable colleague and good friend, helped me with PIC programming, debugging electronics, and helped me focus on the big picture. I thank Joshua Wachman for introducing me to the Lab and for his continued friendship and goodwill. I thank Dave Becker for his true friendship, for making me laugh, and always inflating my spirits. Stefan Agamanolis was the Earth's best officemate, making me laugh, keeping me sane, and helping me out with random math and computer questions. I thank Majida Rizk for her friendship, her patience, and for her understanding.

The folks in the Garden and on the third floor were a big part of my MIT experience and I am most thankful for their help and comradeship. Mike Massey has been a friend and has helped me learn about optics and the 70s. Henry Holtzman, Jon Watlington, Shawn Becker, Roger Kermode, Jeff Noris, and Ross Yu have also helped me along the way. I thank Flavia Sparacino for helping develop ideas about motion-base ride movies and for helping me with the title of this paper. Thanks to Erik Trimble for his help with my thesis poster.

The people of the Physics and Media Group have given me a lot of help and support. I thank Joe Paradiso for his friendship and for helping me learn about countless aspects of electronics. Thanks also to Henry Chong, Matt Reynolds, Rehmi Post, Edward Boyden, Bernd Schoner, Tom Zimmerman, Barret Comisky, and Ara Knaian for their various help with electronics and computation; and to Jeremy Levitan for helping with IOME Cam mechanics and machining. I thank Josh Smith for his help with general computation and his brilliant humor and observations. I thank Rich Fletcher for his continual interest, and for discussing applications and future research. I am hyper-thankful for Craig Abler's help with the Hummingbird's pcb layout, which I should mention is the only electronics layout I've ever heard of that returned from the fab house 100 percent correct in one try.

A number of other people at the Media Lab have offered me assistance and opportunities. I thank Linda Peterson, Santina Tonneli, Susan Bottari, Lena Davis, and Felice Napolitano for their help. Thanks to Tod Machover and Teresa Marrin for varied opportunities and help.

I want to thank Nicholas Negroponte and Jerome Weisner for envisioning and creating the MIT Media Lab, a truly magical environment. Thanks to both Glorianna Davenport and Bran Ferren for their guidance and interest in using IOME Cam for an array of electronic cinematographic applications. And thanks to Walter De Brouwer and Riverland for providing the facilities and push to finish this project.

I thank the eleemosynary men of Shfucco, Rob Passaro, B. Ellingwood, and Kazu Niimi, for their support and friendship. I thank my family for their support, patience, and interst. I am also indebted to Michelle Kemper for her friendship, energy, patience, and caring.

Lastly, I will always be grateful to my Mom and Dad for being supportive and interested in what I do and for always believing in me.

Contents

Chapter 1

Introduction

The IOME Cam (inertial-optical motion-estimating camera) system is designed to combine inertial and optical motion-estimating techniques to determine a camera's motion during a video sequence. IOME Cam employs a Kalman filter joint estimator to observe both optical and inertial data types to predict camera motion parameters. While either the optical or inertial data could be used alone for motion-estimation, each method has inherent error types that limit its performance. When used together, however these two data types can actually correct for the other's errors (see Sections 1.3.4 and 1.4.4).

The development of the IOME Cam has occurred concurrently with and has been influenced by the projects associated with the Things That Think (TTT) consortium at the MIT Media Laboratory. One area explored within TTT is that of inertial proprioceptive, or motion-cognizant, devices [Verplaetse, 1996] - devices that *feel* their own motions using inertial sensors. While computer vision has been the sole navigator or motion estimator for cameras, and a good deal of work has been done in this area [Becker, 1996] [AP, 1995] [Teodosio, 1992], a vision estimator's performance is ultimately limited by the lack of information in its input. Video-estimator errors are caused by external motions, lighting, shadows, and unanticipated (unmodelable) camera motions among other things. Introducing inertial sensing, which is not dependent on any visual information, to the camera-motion estimation problem is a logical step.

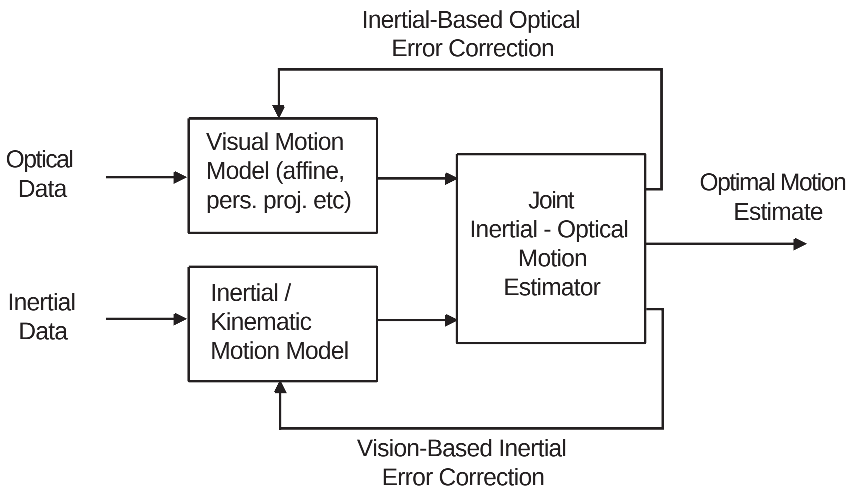

Inertial-Based Optical Error Correction
Optical Data
Visual Motion Model (affine, pers. proj. etc)
Joint Inertial - Optical Motion Estimator
Optimal Motion Estimate
Inertial Data
Inertial / Kinematic Motion Model
Vision-Based Inertial Error Correction

Figure 1.1: Schematic view of the IOME Cam system.

1.1 Prior Art and Science

Camera motion estimation is a well defined and developed problem in the field of computer vision (see Section 1.3.) Combining inertial sensors with video cameras has also been an increasingly active area of research in the past decade, but the majority of related work has been done in the way of image stabilization. Most major video camera manufacturers offer gyrostablized cameras today. There is no evidence, however, of previous work done in the area of instrumenting a video camera with an full inertial measurement unit for recovery of camera motion parameters.

1.2 Motivating Applications

There are a number of motivations for determining a camera's motion during an image sequence. A motion-cognizant vision system such as the one we're discussing allows new and exciting uses of the visual data that was once only useful for playing back as video or for capturing still screen shots.

1.2.1 Salient Stills - 2D Modeling

Still photography and video are both visually effective but each in different visual domains, due to the respective medium of each. Stills offer high resolution, high contrast visual information and are ideally shown on handheld print media. However, as their name implies, they are limited to one still moment in time [1]. Video on the other hand offers dynamic, temporal visual information, with the ability to follow a character or action or to zoom in on a certain visual aspect. Video, however, requires special displays (typically physically constraining compared to a still's handheld media); also video has poor resolution for any one given frame. Salient Stills is a computer program intended to combine the advantages of both video and stills [Teodosio, 1992].

A Salient Still is a high resolition, wide field-of-view 2D image that contains temporal information. The Salient Still is comprised of multiple frames from a video sequence that have been translated, skewed, and rotated, based on the camera motion. A number of photographic and cinematographic examples of Salient Stills are given by [Massey, 1996]. Salient still production is hampered by vision-motion-estimation errors and has been a driving factor in the development of IOME Cam. The purely visual input combined with the linear Affine model being used create limitations to the Salient Still output such as skews, warping, and blurred visual information.

1.2.2 Camera operator gesture recognition

IOME Cam can also be thought of as a parsing tool to be used intermediately between the motions of a camera operator and the segments of a video shoot. The camera's inertial motion datacan be used as meta-information that accompanies, and pertains to, the visual and audio content. As an editing tool, the camera motion data can be analyzed with a gesture-recognition system to identify the operator's intent or to anticipate his/her next

[1]Time-delay and multiple-exposure photography arguably produce still images representing multiple-time-instances.

move. This information can be used to annotate or parse the video into known segments [Davenport].

SmartCams

When used in the capacity described above, IOME Cam would be a sensible input component of Pinhanez's SmartCam system [Pinhanez, 1995]. Currently this system uses computer vision to monitor and recognize actions in the video and to automatically control the camera(s) appropriately [2]. Given the additional information of camera operator intent (or anticipated next move) the SmartCam system could base its camera control on that information as well as the vision-recognized actions.

1.2.3 3D Modeling

Perhaps the most basic and all-encompassing application goal of the IOME Cam system is that of generating 3 dimensional models of scenes. If the camera's 3D motion from frame to frame is known, the motions and geometries of objects on the image plane can be analyzed relative to that camera motion, yielding a geometrical model of the objects and camera. Bove nicely summarizes [Bove, 1989]

> the output of the camera should not be frames but rather a three-dimensional database describing the objects in the scene and their movements through time, as this would permit great data compresion as well as computer-graphics-style operations and interactive modification

Becker's [Becker, 1996] system approaches this goal nicely with purely visual input, achieving 2.5D modeling, but is of course constrained to use with static scenes and other visual

[2] An example is Pinhanez's cooking-show demo (http://pinhanez.www.media.mit.edu/people/pinhanez/): when the vision system senses the onion is being diced, it zooms the camera in on the action. In a likewise manner, an IOME Cam might pan across the room, following a character, automatically yielding the live-shot off to a better-located camera.

limitations. Given a 3D model of a real scene, there are countless ways to modify, enhance, and playback that scene. Some pertinent ones are discussed in the following few paragraphs.

Data Compression

A video scene described in the form of sequential frames (with each frame comprised of numerous pixels) takes up much more bandwidth than a modeled scene. Here, a "modeled" video sequence refers to a scene that represented by a computer database that describes the properties of objects and motions within that scene. The Structured Video projects at the Media Lab have included a copious research in the area of data compression and modeled scenes.

Object-Based Media Presentation and Special Effects

Given the aforementioned modeled scene with segmented objects, it would be possible to manipulate the appearence of (or even presence of) certain objects or characters in a video presentation of that scene. In a sense, the IOME Cam could be used to generate virtual sets [Ferren]. This would enable the same story to be told any number of times using completely different settings, props, or characters. Virtual (computer generated) objects could also be integrated into the scene and made to interact with the real objects. [3]

1.2.4 Motion Capture and Playback

A popular attraction type at theme park and location-based-entertainment venues is the motion-base ride-movie. These "rides" consist of a platform that is suspended in conjunction with a motor system [4] and a display surface. Typically a movie is shown from the viewpoint

[3]This real-virtual character fusion is commonly done in the special effects industry - Jurassic Park is a popular example. However, current motion estimation and scene modeling are done manually frame by frame - a laborious task.

[4]The majority of motion-bases are in a Steward Platform configuration, driven by hydraulic actuators, although pneumatic and electromagnetic actuators (like those built by Aura Systems Inc.) are becoming

of a moving vehicle (i.e. a jet fighter) and as the camera view changes with the motion of the recorded video, the platform on which the viewers are located also moves accordingly. The viewers are made to feel like they are visually and physically experiencing the motions that the camera underwent.

When considering the ride-movie as both a video and motion playback system, IOME Cam seems an ideal input device. In its simplest form it records motion and video; and its inertial data could be made to drive a motion platform time-synched with the video to form an automated ride-movie driver.

1.3 Vision-based motion estimation

Estimating the motion of a video camera has long been one of the problems approached by the computer vision field. Autonomous robot navigation, scene modeling, and the example applications described above are all driving this research. One common technique that has been developed to estimate a camera's motion is through analysis of *optical flow*, or the apparent motion of brightness patterns on a camera's image plane. Another set of techniques, called *feature tracking* [AP, 1995] [Becker, 1996], involve tracing the motion and properties of a set of visual features from frame to frame. Unfortunately, vision-based motion estimators are not fully robust - they are limited by lack of information or mismodeled information in the video, by discontinuities and noise in optical flow and other error sources.

1.3.1 Optical Flow

Complex video sequences, both static and dynamic, may be thought of as a distribution of image intensity that undergoes simple translations when viewed over sufficiently small time steps[Horn, 1986]. This changing image intensity, optical flow, is modeled as a continuous

more common. Another motion-base example is Tim Anderson's Jerkotron, an electric motor, cable, and pulley assembly whose base is moved in pure tension.

function of space and time

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

Given a continuous motion field, I may be expanded using a Taylor series (ignoring higher terms)

$$\frac{dI}{dx}\frac{dx}{dt} + \frac{dI}{dy}\frac{dy}{dt} = -\frac{dI}{dt}$$

The above model accounts only for x and y translations, and not for zoom. The affine transform model does account for zoom in terms of scaling factors.

1.3.2 Affine Modeling

Given the optical flow data from a video sequence, motions of objects on the image plane are related to motions of objects in the real world via one of a number of existing models. The affine model is used by the Salient Still program. That process is described in this section.

The affine model, given in 1.1, provides a linear transform between the image sequences (optical flow) and camera motion

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_x & c_x \\ b_y & c_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a_x \\ a_y \end{pmatrix} \tag{1.1}$$

where: u and v are the optical flow terms,

a_x and a_y are pure transitional terms,

b_x is a scaling factor for x in the x direction,

c_x is a rotation factor for x in the y direction,

b_y is a rotation factor for y in the x direction,

c_y is a scaling factor for y in the y direction,

and x and y are the real world coordinates, x and y.

Note that the z term is not present in equation 1.1; the affine model treats z translations as zooms, with constant changes in the b_x and c_y terms.

1.3.3 Perspective Projection

The perspective projection model (Equation 1.2) is another approach to mapping optical flow to real world motion and is discussed in [AP, 1995] and [Szeliski, 1993] and planned for use in the future versions of Salient Stills. In this nonlinear model,

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \frac{1}{1 + z\beta} \tag{1.2}$$

u and v are the optical flow terms, x, y, and z are the real world motion terms, and β is the inverse focal length.

1.3.4 Vision-based Errors vs. Inertial

Recall that IOME Cam's goal is to improve upon existing camera motion estimation schemes by combining sensor modalities. In vision-based motion estimation, one of the most common and harmful errors is that due to motions in dynamic scenes. When objects are moving on the image plane in an uncorrelated manner to the camera motion, vision techniques will not be able to differentiate between the two types of motion. Shadows and occlusions caused by moving objects will also cause errors such as "false positive" motion estimates. Such optical queues cause vision-based estimators to have "not enough information" and the accuracy of

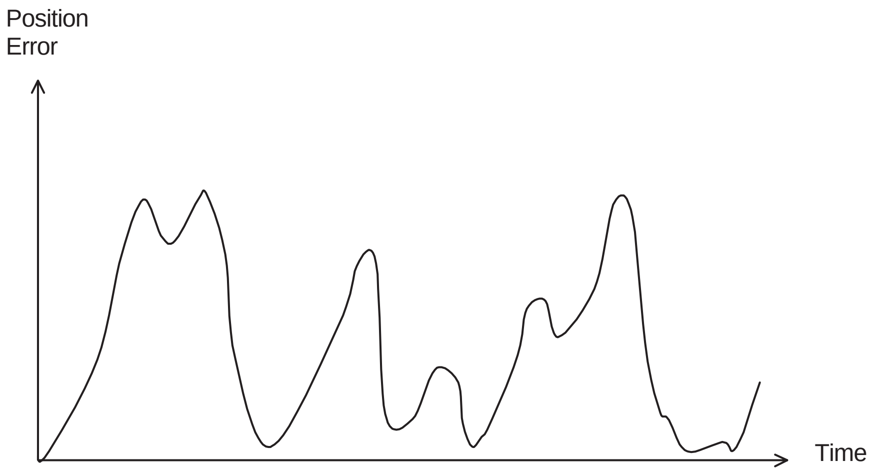


Figure 1.2: Characteristic positional error of optical estimators.

their estimates tend to be sporadic or unstable, with segments of increased positional error (Figure 1.2.) When enough [correct] optical information is available again, the positional error will fall.

Inertial sensing is a good check source for these frame-to-frame error types as it provides excellent short-time motion data. Vision-based estimation is more reliable for long term accuracy. Since inertial navigation methods suffers from positional error drifts, the long-term stability of optical methods can be used for baseline restoration of inertial errors.

1.4 Inertial Motion Sensing & Navigation

The technology of inertial motion sensing is the only source of fully autonomous self-motion sensing and is a logical aid for video camera motion estimation, since it does not require external references and thus won't be hampered by the same error sources as video. A video camera, instrumented with inertial motion sensors, can record its inertial motion data as it records video. Such a system allows for motion estimation to be performed using the inertial data alone, or via a joint estimator, like with IOME Cam, using both inertial and video data.

1.4.1 Motion-Sensing Modalities

Motion sensing is not a new idea. For years, security systems, weapon systems, and medical and entertainment systems have employed various forms of "externally referenced" motion sensing technologies such as infrared, radar [5] , and video. Internally referenced, autonomous motion sensing has also existed for quite some time. Robots, aircraft, automobiles, and other vehicles have sensed and measured their motions for decades, using varying electromechanical sensors as well as inertial sensors.

Most of the motion sensing technologies referred to above are restricted in terms of where and how they are useful. Infrared, radar, and video motion sensing technologies are all "externally referenced", physically removed from the moving object of interest. As a result these sensing modes are subject to occlusions and numerous interferences and noise sources. Although cars and aircraft measure their own motions, their motion sensors are both dimensionally and directionally limited. A car wheel's motion sensor requires the friction of a road and only senses in one dimension; a pitot tube only works for an aircraft traveling forward in familiar atmospheric conditions. Table 1.4.1 summarizes some sensor types, their constraints, and their respective application domains.

A more tractable and generally effective type of motion sensor is the inertial sensor. Used in spacecraft, aircraft, and submarines for years, this type of sensor attaches directly to the moving body of interest and gives an output signal proportional to its own motion with respect to an inertial frame of reference. Two types of sensors comprise inertial sensing: accelerometers and gyroscopes. Accelerometers sense and respond to translational accelerations; gyroscopes sense and respond to rotational rates. Inertial sensors are desirable for general motion sensing because they operate regardless of external references, friction, winds, directions, and dimensions. However, inertial systems are not well suited for absolute position tracking. In such systems, positions are found by integrating, over time, the sensors' signals as well as any signal errors. As a result position errors accumulate. Inertial

[5]Radar is used, of course, for aircraft sensing, but has recently found use in human-machine interfaces. Examples of using radar as a performance space interface include Joe Paradiso's 10/10 Media Lab anniversary space and Steve Mann's dance interface, done in Canada in the 1980s.

Sensory Type	Reference	Example Applications	Limitations
radar	external body in motion	airplane, human tracking	objects need to be in motion
electric field sensing	external body with electric charge	3D mouse, cello bow	limited range
magnetic	transmitter	Polhemus†, FOB††	limited range
GPS	Earth orbitting sattelite network	navigation, more	outdoors, Earth
video	external objects in acceptable visual field	robots, CHI	excess external motions, lighting
Inertial	self	ICBM navigation systems, Hummingbird	errors increase with time
†: *Polhemus refers to the manufacturer of the Ultratrak^{TM} motion sensors*			
††: *FOB refers to Flock of Birds Ultratrak^{TM} by Ascension Technology Corp.*			

Table 1.1: Various motion sensing modalities

systems are most effective in sensing applications involving relative motion, when used in conjunction with another sensor type such as GPS, IR, or video.

1.4.2 Inertial

Inertial navigation describes the techniques and technologies used in measuring the inertial effects of a moving body and using those measurements to deduce the body's time-varying position. Inertial sensing is accomplished with two types of sensors: accelerometers and gyroscopes. Typically, both of these sensors are sensitive to only one axis of motion. Inertial navigation systems (INS) used in aircraft, spacecraft, and other vehicles are usually based on an inertial measurement unit (IMU) that consists of a set of three orthogonal accelerometers and 3 mutually orthogonal gyroscopes. Such a device is sensitive to the full 6 degrees of freedom of motion (3 transitional and 3 rotational).

Compared with other modes of motion sensing, the inertial type is ideal for enabling devices to become proprioceptive, or aware of their own motions and positions. Some example proprioceptive devices are described in section 5.1. Besides the inertial type, the other aforementioned motion sensor types are all limited by being externally referenced (these sensor modes would be ideal if the task at hand was to sense other bodies' motions). This project, however, addresses the problem of estimating the motion of a video camera during a video sequence after video has been recorded; and IOME Cam introduces inertial sensing to the camera motion estimation problem.

1.4.3 Determining Position Inertially

Inertial navigation systems determine position and orientation from the basic kinematic equations for translational and rotational motion. An object's orientation, given a sensed rotational rate, ω, during each time step, dt, is given by

$$\theta = \theta_0 + \omega t \tag{1.3}$$

where θ = orientation angle, and dt = time step. A gyros' output is the rotational rate ω. Similarly, position is found with the translational kinematic equation

$$x = x_0 + v_0 t + \frac{1}{2} a t^2 \tag{1.4}$$

where x = position, v = velocity, and a = acceleration, an accelerometer's output.

It should be noted that IMUs alone can not be used for absolute position tracking. Since an INS calculates position by multiplying an accelerometer's output by t^2, any errors in the accelerometer's output are also multiplied by t^2; accelerometer errors propagate by equation 1.4 (Figure 1.3.) This leads to huge position errors: in just 60 seconds, a one dimensional IMU using an accelerometer with an output noise level of just 0.004 g yields a position uncertainty of about 70 meters. Gyroscope errors increase linearly with time, via equation 1.3, and are therefore typically less 'harmful' than accelerometer errors. Because of their inherent accumulation of absolute positional errors, inertial sensors are much better suited for relative motion sensing/tracking. The accelerometer with 0.004 g noise, gives a positional uncertainty of about 0.2 mm per cycle in a system as slow as 10 Hz; the uncertainty falls to about 80 micrometers per cycle in a 50 Hz system.

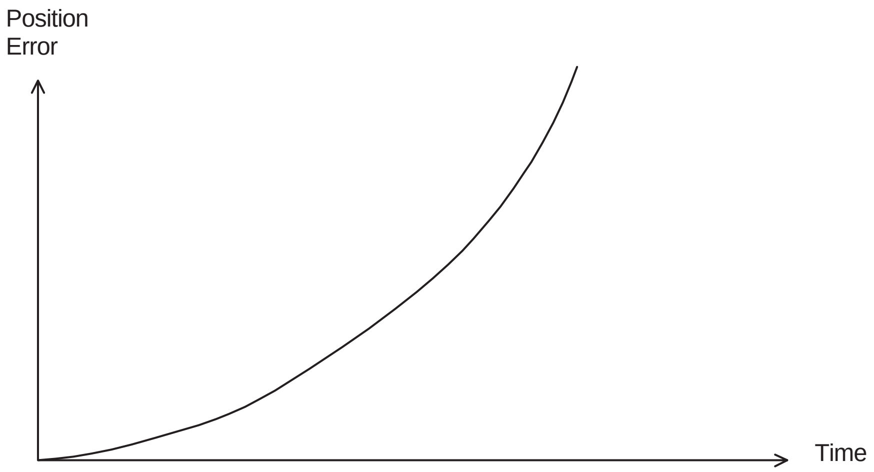


Figure 1.3: Characteristic positional error of inertial estimators.

Pure inertial measurement systems are best suited for relative motion sensing applications or for short-duration position tracking applications. The smart pen application (a pen that knows what it is writing) is an example of a system where absolute position tracking of the pen tip would be desirable, but relative position tracking still allows a highly useful system. Given absolute position tracking, the pen's IMU could essentially store analog 'carbon' copies of what the pen had written. Due to inertial errors, the pen system could never accurately track the pen tip's position on the paper for a useful duration, but in tracking the pen tip's relative motions and continuously checking for verifiable characters, the pen's IMU can recognize written characters and store the corresponding ASCII characters in memory.

Inertial navigational systems suffer most from the inherent buildup of positional errors associated with inertial sensors because INSs need to operate indefinitely for the duration of their 'missions'. For navigation and other applications where system accuracy is more important than system autonomy hybrid inertial motion sensing systems are common. An inertial-optical motion estimator was discussed above in the context of a proprioceptive video camera. Other hybrid inertial systems include inertial-stellar missile navigation systems [Mackenzie, 1990] and inertial-GPS (global position system) airplane guidance systems.

Applications requiring absolute rotation (orientation) tracking and relative translation tracking can accomplish this with a pure inertial system. In such a system, orientation is computed from the gyro outputs as usual - with a slightly growing time dependent error as usual. Provided that the system is at rest occasionally, the accelerometers can accurately sense the orientation of the 1g gravity acceleration vector. Given these occasional gravity-orientation updates, the system can correct its gyro-induced orientation errors for an indefinite duration. This is an example of a "zero velocity update". Note this scheme will only work if the accelerometers being used are DC responsive, that is if they sense constant accelerations.

1.4.4 Inertial errors vs. optical

As mentioned in section 1.4.3, inertial navigation systems have a characteristic positional error drift - the position errors of inertial navigation systems grow quadratically with time. Since INSs use a dead reckoning approach, their errors are continuously compounding. For the case of IOME Cam, optical data can be used to correct for the inertial errors. Vision-based motion estimators are better suited for longer time duration tracking and can therefore aid in baseline restoration of inertial error drift. Given occasional visual queues, absolute position and orientation data may be restored from time to time. Conversely, inertial sensing can prevent false-positive estimates associated with optical analysis of dynamic scenes.

Chapter 2

Inertial Sensing Technologies

The following two sections discuss the technologies of accelerometers and gyroscopes, the primary components of inertial navigation. Care is taken to show example state-of-the-art inertial technologies in terms of commercially avaiable sensors. In each of the following sections, available accelerometer and gyroscope technologies are surveyed according to performance, price, and size characteristics, as applied to the domain of IOME Cam (and other proprioceptive applications.)

2.1 Accelerometer Technologies

A number of different accelerometers were considered for use with this project. This section discusses the general operation of the accelerometer and explains some of the major technologies used in accelerometer development.

At its most basic level an accelerometer can be viewed as a classical second order mechanical system; that is a damped mass-spring system with an applied force. When the system undergoes an applied acceleration, the spring is made to stretch or contract by the mass's inertial force. The resultant displacement of the mass or internal force of the spring is the system's output which is proportional to the input acceleration. There are a number of tech-

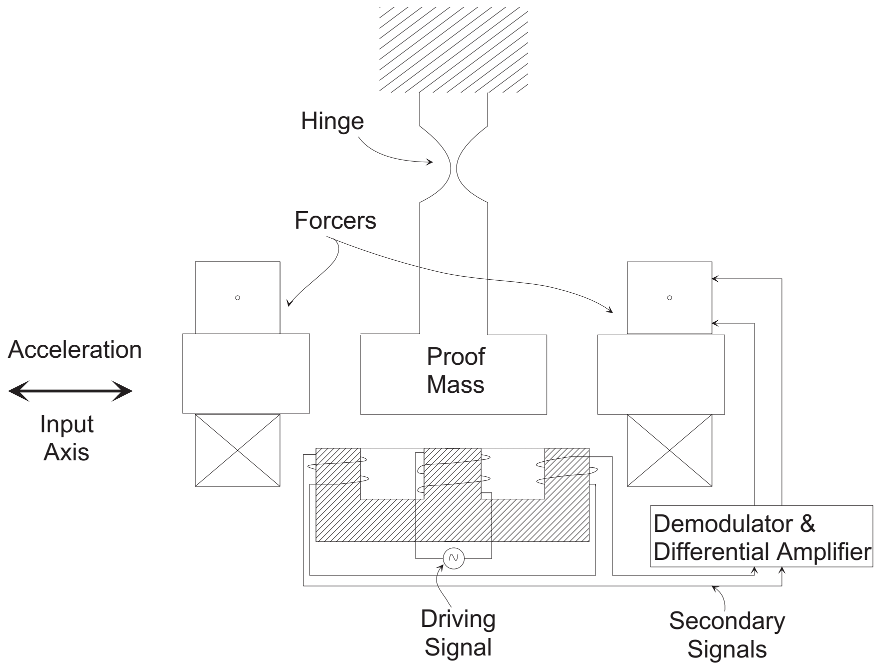


Figure 2.1: Generic pendulous accelerometer

nologies used to implement today's accelerometer designs. In this section, these technologies are described along with the performance and price characteristics of corresponding commercially available acclerometers. See section 2.1.5 for a survey of commercially available accelerometers.

2.1.1 Pendulous Accelerometer

The vast majority of modern accelerometers are of the pendulous type. Although a number of different implementations exist, they all work by similar principles. A generic closed-loop pendulous accelerometer, consisting of a hinge, a proof mass, pickoffs, damping, and a forcer, is shown in Figure 2.1.

Referring to Figure 2.1, a pendulous accelerometer's *pendulosity*, p, will be defined as the proof mass multiplied by the length from its CG to the hinge

$$p = mk.$$

When an acceleration is applied parallel to the input axis, the torque on the output axis is given by

$$T = fk = mak = ap \tag{2.1}$$

where

$f = ma =$ inertial force

$k =$ length from proof mass CG to hinge

$m =$ proof mass

$a =$ applied acceleration

$p = mk =$ pendulosity

This example accelerometer uses a magnetic pickoff that has three separate windings, one primary and two secondary. The pickoff acts as a differential transformer. When the proof mass is centered, under no acceleration, the pickoff's two secondary windings generate the same voltage, yielding no difference. But under an applied acceleration, the proof mass is moved off center and one secondary voltage rises and the other one falls, increasing their voltage difference. The phase of the voltage difference with respect to that of the driving signal will give the direction of the applied acceleration, and the differential voltage amplitude will give the amplitude of the acceleration.

In a closed-loop accelerometer like the example one, the forcer coils' output torque is proportional to the feedback current, the differential circuit's output, $T = k_F i$ (where $k_F =$ scale factor of forcer). Since $a = T/p$ (from Equation 2.1) we see the acceleration can be found as

$$a = \frac{k_F i}{p}. \tag{2.2}$$

2.1.2 Piezoelectric Micromachined Pendulous Accelerometers

One of the main transducer technologies used in pendulous accelerometer design is that of piezoelectricity. A piezoelectric material is defined as a material that develops a distributed

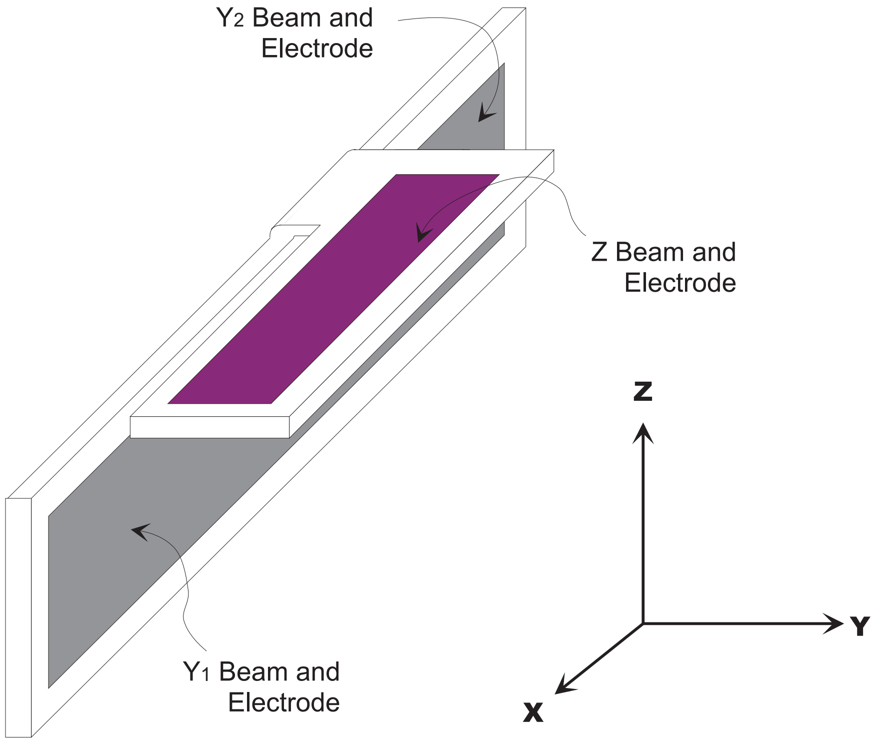


Figure 2.2: AMP's 3dof piezoelectric gyro design

electric charge when pressed or subjected to a force. Piezoelectric materials transform mechanical work input into electrical output and vice versa. A simple piezoelectric pendulous accelerometer consists of a piezoelectric cantilever structure whose center of mass is adequately displaced from its base. When the structure is accelerated it is made to deflect, and when the piezoelectric element deflects it develops a proportional charge differential.

Piezoelectric accelerometers are called *active devices* because they generate their own signals, and theoretically do not need to be powered. Since piezoelectric sensors require physical work to generate an electrical output, they can not respond to steady-state inputs; hence, they are also called AC-response sensors. Most piezoelectric accelerometers will only operate above a certain threshold frequency. Piezoelectric materials are also pyroelectric; that is, they respond to changes in temperature.

AMP Inc. has a line of low cost pendulous piezoelectric accelerometers, including a novel design that senses two translational and one rotational axes of acceleration - all in one

14 pin surface mount package. Figure 2.2 shows the component layout of AMP's sensor. Acceleration in the $\vec{y}$ direction is sensed by the Y_1 and Y_2 beams and electrodes; acceleration in the $\vec{z}$ direction is sensed by the Z beam and electrode. Also rotation about the z-axis is sensed differentially by the translational accelerations of Y_1 and Y_2. These sensors utilize the piezoelectric properties of the polymer polyvinylidene fluoride (PVDF). AMP's acresponding ACH-04-08, has a 3dB frequency response of 7 Hz to 3.3 kHz, and an input range of about ± 2 to $\pm 30g$.

2.1.3 Piezoresistive Micromachined Pendulous Accelerometers

A common type of commercially available low-g pendulous accelerometer is made from micromachined silicon, a piezoresistive material. Piezoresistive materials, like quartz, lead zirconate titanate, and boron, have the property of changing their resistance under physical pressure or mechanical work. If a piezoresistive material is strained or deflected, its internal resistance will change and will stay changed until the material's original position is restored. These accelerometers act as both AC- and DC-response sensors. In order to detect this change in resistance, a power supply is necessary - piezoresistive accelerometers are *passive devices*.

Piezoresistive pendulous accelerometers typically have a proof mass which is suspended by one or more piezoresistive 'cantilever' arms. An applied acceleration will deflect the proof mass and strain the cantilevers, resulting in a resistance change proportional to the acceleration. The resulting resistance change is usually measured via a Wheatstone-type bridge circuit (Figure 2.3(a)). IC Sensors' model 3021 and Entran's EGAX both employ this type of scheme.

Like piezoelectric materials, piezoresistive materials are also temperature-sensitive (they're used in thermistors), which will adversely affect a piezoresistive accelerometer's repeatability. One way to avert this temperature problem is to use the sensor's pendulous arm connected to the proof mass, as a strain gauge arranged in a Wheatstone Bridge circuit (as

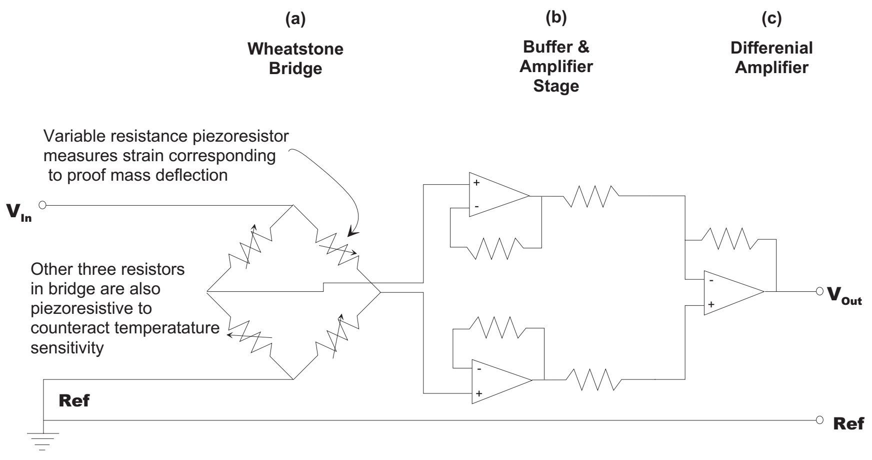

(a)
Wheatstone Bridge
(b)
Buffer & Amplifier Stage
(c)
Differenial Amplifier
Variable resistance piezoresistor measures strain corresponding to proof mass deflection
V_{In}
Other three resistors in bridge are also piezoresistive to counteract temperatature sensitivity
V_{Out}
Ref
Ref

Figure 2.3: Example piezoresistive accelerometer temperature compensation and amplification circuit

described above), and to use the same piezoelectric material as for the other fixed resistors in the bridge. This corrects for temperature sensitivities by canceling the effects from either side of the bridge.

Figure 2.3 shows a simplified example of the typical temperature compensation and amplification circuit used by I.C. Sensors in their piezoresistive silicon micromachined accelerometers. The circuit is divided into three main parts. Part (a), the Wheatstone Bridge, measures the change in resistance of the accelerometer's strain gauge and outputs a corresponding voltage difference. Stage (b) acts as a buffer to minimize the current drawn from the bridge and acts to amplify the bridge's signals, increasing the voltage difference. The differential amplifier, part (c), then measures the final voltage difference and delivers the output voltage.

2.1.4 Capacitive Micromachined Pendulous Accelerometers

Perhaps the most common type of consumer accelerometer is the capacitive micromachined pendulous accelerometer. Accelerometers with capacitive sensing elements typically use the proof mass as one plate of a capacitor and the base as the other. When the sensor is accelerated, the proof mass tends to move and the voltage across the capacitor changes; this change in voltage corresponds to the applied acceleration. These sensors may be operated open-loop or closed-loop.

Capacitive accelerometers generally have higher sensitivities than piezoresistive models - the two piezoresistive accelerometers listed in Table 2.1.5 have sensitivities of about 10 mV/g and the capacitive accelerometers in the same Table have sensitivities an order of magnitude higher.

The basic capacitive micromachined accelerometers are made in much the same manner as the piezoresistive pendulous accelerometers discussed in Section 2.1.3. Silicon Microstructures' model 7130 $\pm 10g$ capacitive micromachined accelerometer has a frequency response of DC to 500 Hz and maximum noise floor of approximately 0.1 g.

2.1.5 Comparison of Accelerometer Technologies

In terms of the motion-estimating camera and the other self-motion-sensing applications described in Section 5.1, it is evident that the accelerometer market is approaching the desired size, price, and performance. Table 2.1.5 surveys representative commercially available accelerometers.

2.2 Gyroscope Technologies

Gyroscopes are the rotational-motion inertial sensors and are available in a variety of types. The model being used for this project is Murata's Gyrostar. This gyro has an input range of ± 90 deg/sec and a resolution of about 1 deg/sec. The Gyrostar employs piezoelectric vibrating elements and operates much like the classical double tuning fork gyro, with one axis for driving and the one axis for sensing. Both of these gyro types cause an oscillating driving motion along one body-fixed axis. When the sensor is made to rotate, the driven element undergoes Coriolis accelerations in a direction that is orthogonal to the axes of

Make/Model	Type	Input Range [g]	3 dB Frequency Response [Hz]	Output Noise @ 12Hz [g]	Price Range [US$]
AMP ACH-04-08	piezoelectric	±2 to ±30	7 to 3300	0.02	25-50
Entran EGAX	piezoresistive	0 to ± 10	0 to 240	0.00013	500
IC Sensors 3021	piezoresistive	0 to ± 10	0 to 400	.33 mg	100
Silicon Micro-structures 7130	capacitive	0 to ± 10	0 to 500	0.1	100
Silicon Designs 1210	capacitive	0 to ± 10	0 to 800	0.002	100
Analog Devices ADXL05	differential capacitive	0 to ± 5	0 to 20 †	0.002	15-30
Analog Devices ADXL50	differential capacitive	0 to ± 50	0 to 20 †	0.13	15-30
†: *These sensors have customized bandwidth* ††: *Includes linearity, hysteresis, & repeatability*					

Table 2.1: Summary of Selected Accelerometers

rotation and of the driven motion. The subsequent Coriolis-driven motion is picked up by the sensing elements and is proportional to the applied rotation.

This section discusses the general operation of the gyroscope, the rotational-motion inertial sensor. There are two main branches of gyroscopes; *mechanical* gyros that operate using the inertial properties of matter, and *optical* gyros that operate using the inertial properties of light. The operation of several types of mechanical gyros are discussed and the characteristics of representative commercially available gyros are covered. The younger branch of optical gyros is briefly mentioned as they are typically more expensive than mechanical gyros and are currently developed primarily for navigational applications.

Original gyro designs, called *gimbaled systems*, were based on the preservation of rotational momentum and consisted of a spinning disk or *rotor* connected to the moving body of interest by low-friction gimbals. When the body underwent rotation but the spinning rotor maintained its original orientation (preserving its angular momentum). As gimbaled systems progressed, because they were entirely mechanically based, they became increasingly intricate, mechanically complex, and expensive. In recent decades, technology has advanced more in the way of electronics and solid state technology than in mechanics; similarly gyroscope designs also moved away from their complex mechanical foundation. Today's gyro designs are almost exclusively of the *strapdown* type, made with few to no moving parts, and without gimbals and spinning rotors.

Assuming that the gyro's rotor is made to spin at a constant rate, ω it will have an angular momentum,

$$H = I\omega.$$

If the gyroscope is rotated about an axis perpendicular to its angular momentum vector at a rate, Ω, a resulting torque will develop,

$$T = \Omega \times H$$

in a direction orthogonal to both Ω and H. This torque will cause the disk to precess, and it is that precession which is measured to give the rotational rate. Closed-loop or 'force feedback' gyroscopes supply a forcing counter-torque, acting against the precession-causing torque. The output signal for closed-loop gyros is the current driving the counter-torque forcers.

2.2.1 Vibrating Gyroscopes

All vibrating gyroscope designs make use of Coriolis acceleration to sense rotation rates. This section describes several forms of the vibrating gyro, including the tuning fork gyro and piezoelectric vibrators.

When an object rotates and has a varying radius about its axis of rotation, it undergoes a Coriolis acceleration in the direction tangential to the rotation

$$a_c = \dot{r}\dot{\theta}\hat{e}_\theta = \dot{r}\omega\hat{e}_\theta \tag{2.3}$$

There is an associated Coriolis force acting in the same direction (normal to the direction

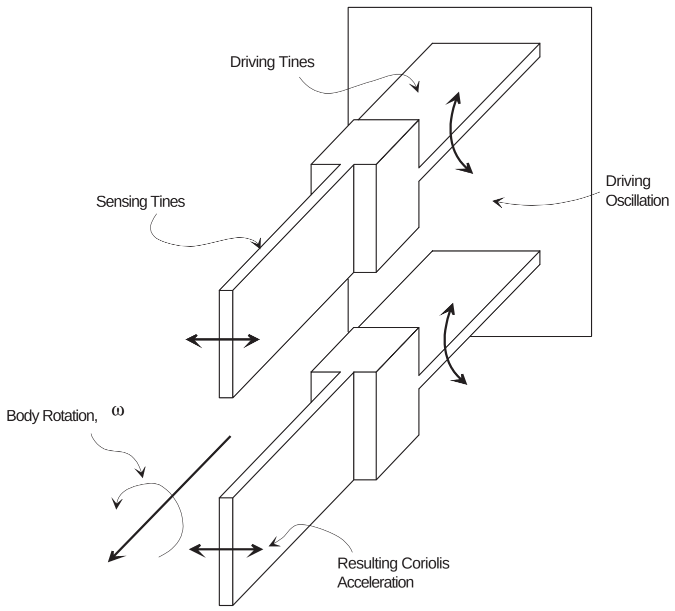


Figure 2.4: Schematic of a tuning fork gyro.

of the varying radius). The Coriolis acceleration and force are proportional to the rotation rate.

2.2.2 Tuning Fork Gyroscope

A schematic of a simple tuning fork gyro is illustrated in figure 2.4. This figure shows a 'double tuning fork' - a structure with two pairs of tines. The two tines in each pair have the same orientation. The double-tines are made to oscillate antiphase, which yields no net motion, but provides a varying radius about the input axis. When a tuning fork gyro is made to rotate about its input axis, its tines undergo sinusoidally varying Coriolis forces in the direction normal to the tines' driven motion. When the tines are subjected to these Coriolis forces, they oscillate in the same direction as the forces. These oscillations are detected by the gyro's sensing elements. Tuning fork gyros may use piezoelectric, piezoresistive, magnetic, or other types of sensing elements.

Systron Donner's GyroChip line of gyros are based on a double tuning fork design. The GyroChip II, is a \$1000 $\pm$100 deg/sec gyro with bias stability $<$0.05 deg/sec and output

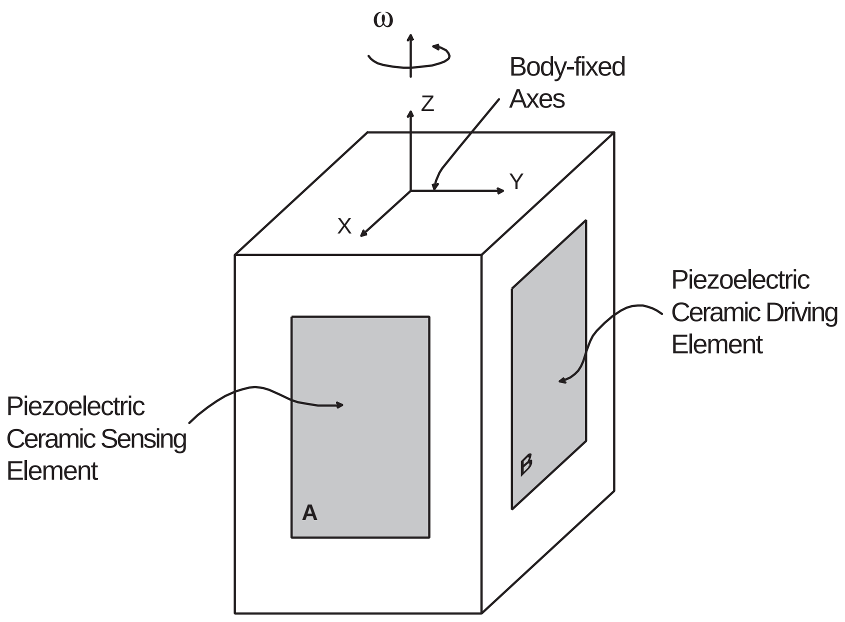

ω
Body-fixed Axes
Z
Y
X
Piezoelectric Ceramic Driving Element
Piezoelectric Ceramic Sensing Element
A
B

Figure 2.5: A Generalized Vibrating Gyro with Rectangular Cross Section

noise <0.15 deg/sec. Watson Industries' tuning fork gyro, Model ARS-C132-1A, is a ±100 deg/sec gyro with bias stability <10 deg/sec. This gyro sells for about $700.00.

2.2.3 Piezoelectric Vibrating Gyroscope

The principles of operation of the tuning fork gyro can also be used to describe the operation of the more general vibrating gyro shown in Figure 2.5. This device, with a rectangular cross section, has piezoelectric ceramic elements on two of its sides, one for driving and one for sensing. The driving element, B, is driven with a periodic electrical signal, and made to oscillate in the $\vec{y}$ direction. The sensing element, A, is sensitive to motions aligned with the $\vec{x}$ axis, and thus does not respond to B's driving signal. When the gyro is made to rotate, at a rate ω, about its $\vec{z}$ axis, the oscillating element B develops Coriolis effects in the $\vec{x}$ direction. The amplitude of oscillation due to the Coriolis forces is proportional to ω and is sensed by the piezoelectric sensing element A.

Make/Model	Type	3 dB Band-width [Hz]	Output Noise @ 12Hz [deg/s]	Bias Stability	Price Range [US$]
Murata Gyrostar	Vibrating Piezoelectric	0 to 7	0.45	0.9deg/10min	80-300
Systron Donner GyroChip II	Double Tuning Fork	0 to 50	0.17	0.05 deg/sec	1000
Watson Ind. ARS-C132-1A	Tuning Fork	0 to 50	0.05	<10 deg/sec	700-800
Hitachi HGA-V	IFOG	Not Available	Not Available	5.0 deg/$\sqrt{hr}$	1500
Hitachi HGA-D	IFOG	Not Available	Not Available	1.3 deg/$\sqrt{hr}$	1250

Table 2.2: Summary of Selected Gyroscopes

2.2.4 Other Gyroscopes

Aside from the various vibrational types, the main gyroscopes in development are optical gyros. Optical gyroscopes operate based on 'The Sagnac Effect'. These sensors use two light waves, traveling in opposite directions around a fixed path. When the device is rotated, the light wave traveling against the rotation direction will complete a revolution faster than the light wave traveling with the rotation. This effect is detected by means of the phase difference in the two light waves. The ring laser gyro (RLG), zero lock-ring laser gyro (ZLG), and the interferometric fiber optical gyro (IFOG) are the main types of optical gyros currently being developed.

Some examples of commercially available optical gyros are Hitachi's IFOG models HGA-V and HGA-D. Compared with the previously mentioned vibrational gyros, these IFOGs are fairly large and expensive, but they exhibit superior bias stability (see Table 2.2.5).

2.2.5 Comparison of Gyroscope Technologies

Table 2.2.5 gives a survey of commercially available gyroscopes in terms of performance characteristics, price, and size.

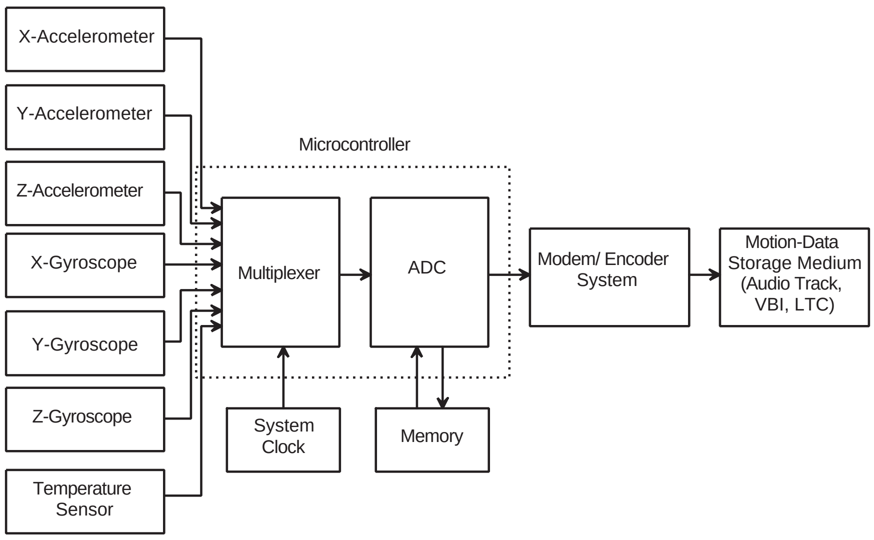


Figure 2.6: Schematic of Inertial Measurement Unit (IMU)

2.3 IMUs and Inertial Navigation Coordinate Systems

A schematized inertial measurement system for a general proprioceptive device is shown in Figure 2.6. This systems consists of a set of sensors whose signals go through an analog-to-digital converter to a microcontroller. The sensors include accelerometers and gyroscopes as well as a temperature sensor (because most inertial sensors' signals are temperature dependent) and any other sensors called for by a given application. The microcontroller either stores the sensor data for later use or it performs some type of real-time analysis and invokes the appropriate output.

Several types of computation and analysis may be performed with the inertial sensors' data by the system's microcontroller. The most basic microcontroller computational function is to estimate motion and position with equations 1.3 and 1.4. A more sophisticated method for estimating motion and position is to use a Kalman filter state-estimation algorithm. Once the system's time-dependent motions and positions are estimated, a pattern recognition scheme such as a neural network, hidden Markov model, or matched filter may be performed with that motion data. These pattern recognition schemes are useful for identifying certain segments of a system's motion. Those motion segments might be caused

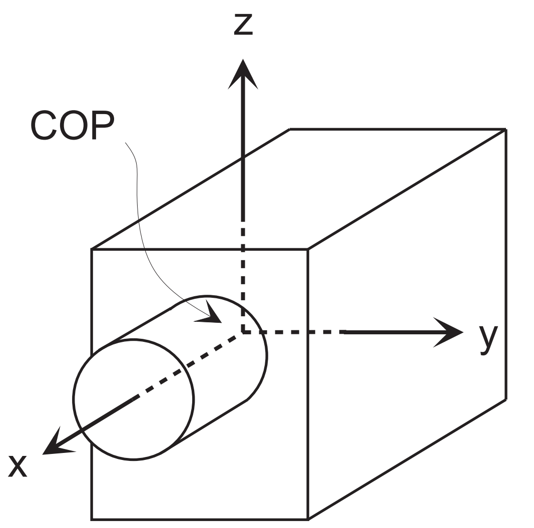


Figure 2.7: Camera-fixed coordinate system

by a baton moving through the upbeat of a conducting gesture, a pen signing its user's signature, or a pair of dancing shoes stepping through a samba.

After estimating the device's motion and position and recognizing any appropriate patterns, the system's microcontroller may accordingly store system state data, activate output media, or communicate with external devices.

2.3.1 Camera-referenced coordinate system

As discussed in section 2.3, an inertial navigation system's sensors sense motion with respect to a body-fixed coordinate system. For the case of IOME Cam, the camera-fixed coordinate system is shown in the figure 2.7.

The origin of IOME Cam's body-fixed axis is located at the center of projection (COP) of the camera. This is done to expedite the joining of the inertial and optical data. Since the inertial sensors themselves can not be positioned at the center of projection, the offset of each needs to be noted and appropriately accounted for in subsequent calculations.

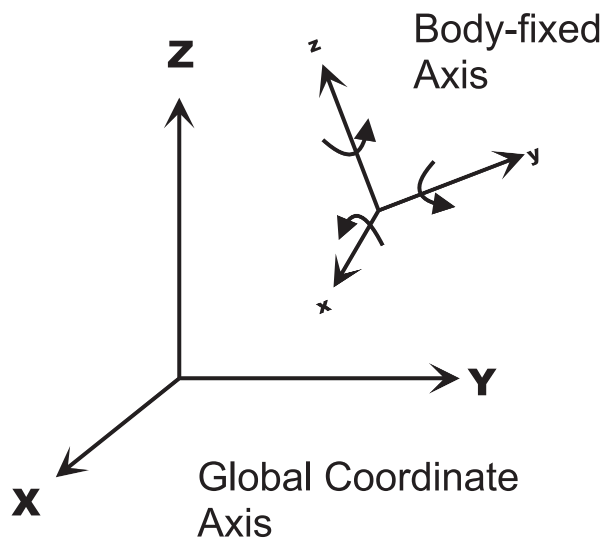


Figure 2.8: Global and body-fixed coordinate systems

The purpose of any navigational system is to determine an object's location or motion with respect to a known (fixed or global) coordinate system.[1] In the host of non-inertial, externally referenced, motion-sensing modes discussed earlier, it was often the case that this coordinate system fix came from the external reference itself (IR and radar are examples.) Inertial navigation systems require an additional step in resolving the fixed coordinate system. Since inertial sensors sense motion with respect to the moving body's reference, a coordinate transformation between the two axes is necessary.

Figure 2.8 illustrates the two coordinate systems. It should be noted that due to the necessary coordinate transformation the IOME Cam system is not able to be modeled as linear. The fact that this transformation introduces a nonlinearity necessitates deviation from the standard linear Kalman filter: an Extended Kalman filter will be necessary.

[1]This coordinate system may be Earth-fixed for vehicular travel, or based on the location of a desktop Polhemus' transmitter, for example.

Chapter 3

Design of IOME Cam

The design of the IOME Cam system largely breaks down into its hardware inertial measurement and data modulation unit and its software demodulator and motion estimator. The system's hardware, consisting of an inertial measurement unit (IMU) and its associated data modulator, interfaces with a video camera. [1] IOME Cam's software consists of a demodulator for the inertial data and a joint motion estimator whose input is the data from the demodulator (the IMU data) along with the output from a vision-based motion estimator .

3.1 IOME Cam's IMU: The Hummingbird

The main hardware component of this project is the inertial measurement unit, which is to be fully designed and fabricated as part of the thesis work. The IMU is of the *strapdown* type, consisting of motion sensors, a microcontroller, a temperature sensor, and a data encoding system (see Figure 2.6). Three orthogonal accelerometers are used to sense translational acceleration and three gyroscopes sense rotational motion. A temperature sensor is needed because the outputs of both accelerometers and gyros are temperature sensitive.

[1]The camera being used for this project is the Sony Betacam SP. All specifications can be found in [Sony].

After the inertial sensors, the next components of the IMU are an analog- to-digital converter (ADC) and a multiplexer. The Hummingbird's on-board microcontroller has these components continuously cycle through each of the inertial sensors, sampling each sensor's analog output signal, converting that signal to digital and then multiplexing those signals onto one data stream. The multiplexed data stream is sent from the microprocessor to the transmitter portion of the circuit. The transmitter encodes the data onto one of the camera's spare data storage media (the version of IOME Cam built for this project used one of the auxiliary audio tracks.) The bulk of this post-sensor work can be accomplished by a microcontroller like one in Motorola's 68HC11 line or Microchip's PIC microcontrollers (the PIC16C71 is used for this work.)

Aside from the electronics-related hardware, a fair amount of physical hardware has been designed and fabricated for this project, including housing and mounting structures. Care must be taken to locate the inertial sensors so that they measure the "correct" axes of input. The camera motion estimation problem attempts to estimate the motion of the camera's *image plane* and the IMU's physical layout must be designed accordingly or accounted for in the analysis and computation. Additionally, an ideal IOME Cam IMU is a removable component, one that can be plugged into a variety of cameras.

3.2 Hummingbird's Components

The Hummingbird IMU consists of roughly one hundred parts. Characteristics and use of the primary components are given in the following section. A detailed parts list is given A.1.

3.2.1 Gyroscope - Murata Gyrostar

A design similar to the rectangular cross section vibrating gyro described in 2.2.3, but with improved piezoelectric conversion efficiency [Nakamura, 1990], is a vibrating gyro with an

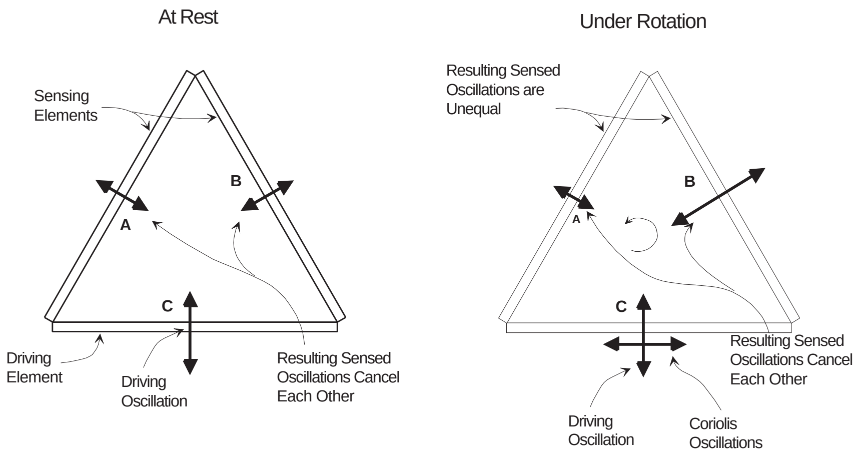

At Rest
Sensing Elements
A
B
C
Driving Element
Driving Oscillation
Resulting Sensed Oscillations Cancel Each Other
Under Rotation
Resulting Sensed Oscillations are Unequal
A
B
C
Resulting Sensed Oscillations Cancel Each Other
Driving Oscillation
Coriolis Oscillations

Figure 3.1: An example piezoelectric vibrating gyro: Murata's Gyrostar.

equilateral triangle cross section. Murata Electronics Corporation's new gyroscope, the Gyrostar, or ENC05E, employs this design. Figure 3.1 shows cross-sectional views of the gyro while at rest (a) and while rotating (b). The triangular cross section design uses three piezoelectric ceramic elements, one attached to each outer wall. Of the three elements, one is a driving element, C, and two are sensing elements, A and B. The output signal of this device is the difference between A's signal and B's signal.

$$output(t) = a(t) - b(t)$$

When the gyro is at rest, element C's driven motion is imparted to the sensing elements, A and B, in a direction that is aligned with the axis of symmetry of A and B. Therefore, while at rest, A and B are *seeing* the same signal and the output is zero.

When the gyro rotates about its input axis, the driving element C, experiences a periodical Coriolis acceleration in the direction perpendicular to both the driving motion and the rotation vector. With the introduction of Coriolis forces to element C, the net motion that C imparts onto A and B is not aligned with A's and B's axis of symmetry. In this case, A and B receive differing signals, and the sensor's nonzero output is proportional to the rotation rate.

The ENC05E operates by actuating and sensing physical vibrations (acoustical pulses) and therefore is susceptible to errors in the presence of applied vibrations in the given frequency range. An easy error would be to use two identical ENC05Es on the same physical surface - the driving pulses of one would wrongly affect the sensing pulses of the other. For this reason, the sensors are available in two oscillation frequencies, 25.0 kHz and 26.5 kHz. Hummingbird uses the two different frequencies to avoid errors.

Hummingbird Gyro Noise

The Murata gyroscopes will experience noise signals due to induced sound waves at their triangular prisms' corresponding resonant frequencies. To get an idea for a typical IOME Cam gyroscope noise floor, consider the signal characteristics for an average pan motion of about 30 degrees. Given the ENC05's mean noise at 12 Hz of 0.45 deg/sec and IOME Cam's update rate of 90 Hz, IOME Cam will experience a milidegree noise floor in terms of angle which is a signal to noise ratio of about 3.64.

3.2.2 Accelerometer - Analog Devices ADXL05

The accelerometers being used in this project, Analog Devices' ADXL05, are made of micromachined silicon and use a differential capacitive sensing element. These sensors sense an acceleration range of ± 5g with resolutions near 0.01g. A simple schematic is shown in Figure 3.2. The ADXL05 has a silicon beam proof mass which is tethered at each end and has comb-like "fingers" extending perpendicularly from both sides of the beam's length. Each of these fingers is located centrally between two fixed capacitive plates when the sensor is at rest. As the sensor is accelerated the proof mass deflects and the fingers move from the centers of the fixed capacitive plates. A difference in capacitance is sensed at the fingers and this signal is proportional to the applied acceleration.

A variation of the general capacitive accelerometer described above is implemented by Analog Devices. Analog Devices' ADXL50 and ADXL05 accelerometers use differential capacitive sensors, consisting of independent fixed plates and movable "floating" central plates that deflect in response to changes in relative motion (Figure 3.2). Under acceleration,

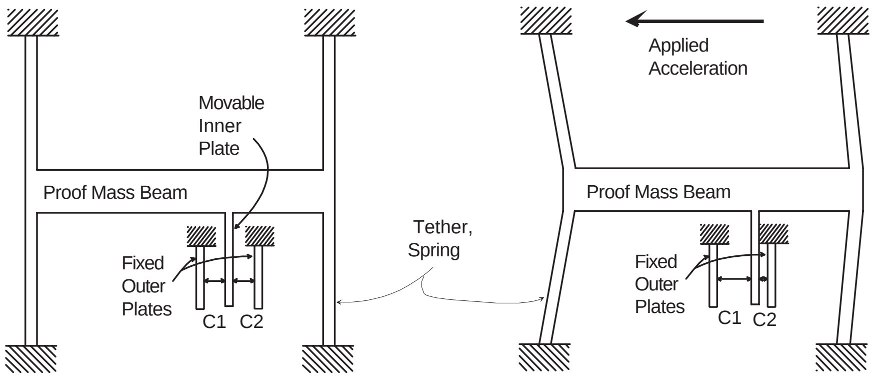


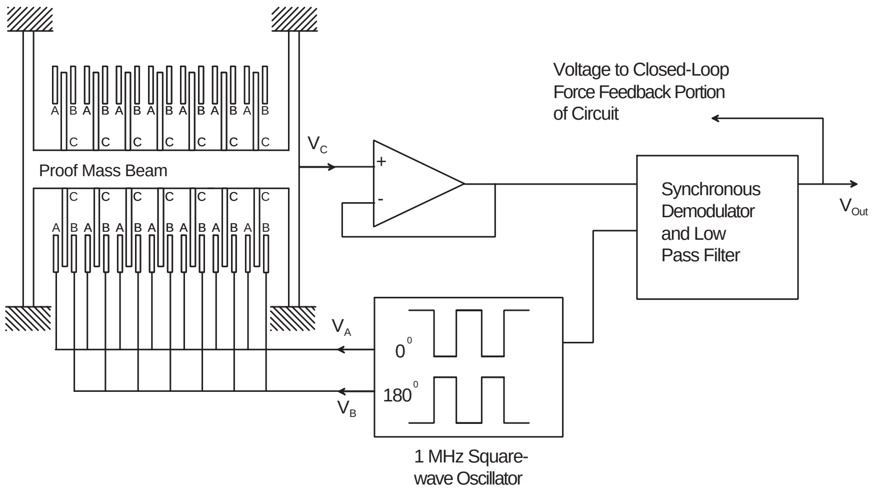
Figure 3.2: Analog Devices' ADXL05 (simplified figure): at rest and under applied acceleration.

the deflection of the center plate will cause a difference in the capacitances of the two capacitors on either side of the plate.

Analog's actual sensor consists of a large series of the differential capacitors mentioned above in a comb-like shape (Figure 3.3). This comb-like structure acts as the sensor's proof mass and its 'fingers' are etched from silicon. The center plate of the differential capacitor is a movable element connected to the proof mass. Referring to Figure 3.3, under zero-acceleration, the center fingers, C, remain equidistant from the outer fingers, A and B. When an acceleration is applied, the A-C and B-C distances become unequal and therefore the A-C and B-C capacitances become unequal.

Signal Conditioning of the ADXL05 Differential Capacitance Accelerometers

The differential capacitances across the capacitors' fingers are detected by the signal conditioning portion of the accelerometer. Referring again to Figure 3.3, we see that a 1-MHz oscillator applies square waves V_A and V_B to the A and C plates respectively. V_A and V_B are *complimentary* - equal in amplitude and 180 degrees out of phase. When at rest (A-C and B-C capacitances are equal), the center plates, C, detect equal amounts of waveforms V_A and V_B. These two waveforms are coupled at C and cancel each other, resulting in zero-amplitude V_C. When an acceleration causes varied A-C and B-C capacitances, the



Voltage to Closed-Loop Force Feedback Portion of Circuit
A
B
C
Proof Mass Beam
V_C
+
-
Synchronous Demodulator and Low Pass Filter
V_{Out}
V_A
0^0
180^0
V_B
1 MHz Square-wave Oscillator

Figure 3.3: Schematic Illustration of ADXL05.

waveforms V_A and V_C change in amplitude. For example if the A-C capacitance increases and the B-C capacitance decreases, the amplitudes of the corresponding waveforms do the same. When coupled at the center plate, V_A and V_B result in a square waveform V_C whose amplitude and phase are related to the amplitude and direction of the applied acceleration.

After passing through a buffer, square waveform V_C enters the synchronous demodulation portion of the sensor's circuit. *Synchronous demodulation* is the process of convolving a frequency-domain *modulated* signal with the frequency-domain *modulation* signal in order to recover the original signal. The ADXL05's phase-sensitive demodulator receives a 1-MHz square waveform from the oscillator - the same square waveform that the oscillator sends the A capacitor plates. The phase of signal V_C determines the direction of applied acceleration: if V_C is in phase with V_A then C has moved towards A, and if V_C is in phase with V_B then C has moved towards B.

The demodulator acts as a *lock-in amplifier*, it demodulates the modulated signal and applies a low-pass filter. The output signal is an analog voltage proportional to the acceleration. This signal goes to the sensor's output and to the force-feedback closed-loop portion of the circuit.

3.2.3 Other System Components

The primary other system components are the PIC 16C71 microcontroller, a multiplexer, and a RS232 transceiver. These are all standard electronic components and are listed in the "parts list" appendix of this document.

3.3 Modulation / Demodulation Strategies

IOME Cam stores its inertial motion data synchronously with the video and audio on one of the camera's spare audio tracks. The motion data is recovered from the digitized audio signal in post production/analysis. IOME Cam therefore uses a hardware modulator and software demodulator for its inertial data. For the modulator, the Hummingbird converts its IMU data output (which is in the form of a multiplexed DC voltage waveform) to an appropriate audio level signal. The audio signal is then sampled [2] and is demodulated and converted into bits and numerical values via a software demodulator (performed in Matlab.)

A number of modulation schemes, including several each of frequency division multiplexing (FDM) and time division multiplexing (TDM), were considered in designing this version of IOME Cam. A discussion of the modulation and demodulation issues related to IOME Cam follows.

3.3.1 IOME Cam's Data Characteristics

In considering IOME Cam's motion data modulation and demodulation systems, it is important to characterize the important data signals and channels. The data signals of interest consist of the outputs of the accelerometers and gyroscopes as well as the IMU's temperature sensor. Characteristics of data channel are defined by the Hummingbird's modulation

[2]Sampling of the audio signal is done at 44.1kHz, which is far greater than the nyquist rate of any of the inertial signals.

components: one PIC microcontroller and one transformer, and the Beta Cam's spare audio track.

Data characteristics of both the accelerometers and gyroscopes are determined by the nature of their motion sensing applications. The required sensing capabilities for a motion-sensing camera can be estimated by looking at the rates of movement of typical camera maneuvers. Camera movement rates were experimentally found by monitoring the pan and tilt motions of a handheld video recorder throughout a series of shooting sequences. An average rotational rate of about 36 deg/sec was observed. Pan rates varied from near zero deg/sec up to about 60 deg/sec. These rotational rates determine the input range for gyroscopes used in a motion sensing camera.

Characteristic camera motion frequency can be estimated as that of human head motion. Head motion frequency averages about 3.5 Hz and rolls off around 8 Hz [FD, 1994]. Giving each inertial sensor as much as 50 Hz of bandwidth for each signal should well cover the necessary requirements.

For thoroughness' sake, the temperature sensor is allocated the same 50 Hz of bandwidth, although meaningful changes in temperature will most likely not happen faster than once per few minutes.

Data channel capacity is no problem considering the available bandwidth of about 12.5 kHz for the Sony Betacam SP.[3] The PIC microcontroller also poses no limitations for data communications considering its 8 to 16 MHz processor speed and 0.25 s to 0.5 s instruction speeds.

[3]This bandwidth is determined from the Recorder's 3 dB frequency response, 50Hz to 12.5 kHz. See [Sony] for more details.

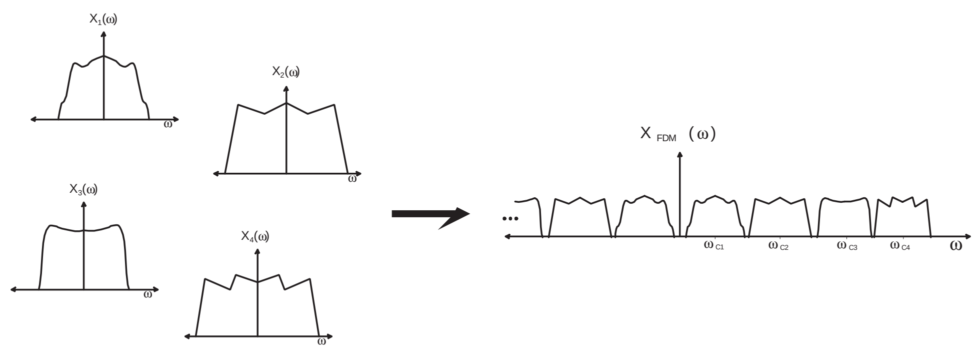


Figure 3.4: Schematized view of FDM (AM).

3.3.2 Frequency-division-multiplexing - FDM

Frequency-division-multiplexing refers to a type of modulation in which several band-limited data signals are placed on to one wide band signal by each being modulated by a carrier of different frequency (Figure 3.4.) Accordingly, such systems are often found in cases where the data channel has a large bandwidth capacity compared to the bandwidth of the data signal(s). For our case, the audio track of the Sony Betacam SP has a bandwidth of approximately 12.5 kHz IOME Cam's data consists of seven 50Hz band-limited signals, therefore the total data bandwidth is about 350Hz. Since the required bandwidth is about an order of magnitude less than the available bandwidth, FDM is a reasonable modulation strategy for IOME Cam's data storage.

The common commercial radio transmission methods of AM (amplitude modulation) and FM (frequency modulation) are both FDM types and were considered for use in developing this system.

Amplitude Modulation

AM was perhaps the most intuitive FDM modulation scheme considered for IOME Cam. It was attractive because of the high available bandwidth capacity. There were several amplitude and frequency-related detractors from using AM modulation however.

One problem was caused by the Beta Cam's built in automatic gain control (AGC). The AGC's function is essentially to normalize the amplitude of the input signals to the camera's audio tracks - an obvious obstacle to amplitude modulation. The camera did have an option for disabling the AGC and manually adjusting the audio signal gain but the physical design and interface of the corresponding knob gave no precise control, prevented repeatable gain settings, and was obviously not designed for allowing storage of discrete electronic data signals.

The other major obstacle to building the AM data modulation system was in the software demodulation stage. Given seven different, and probably not precise, modulation frequencies implemented in hardware, the task of synchronously demodulating each of them is formidable.

Another consideration to keep part-count, and therefore size, down, was that number of components needed to modulate and multiplex the seven different signals. It was most desirable to keep all modulation tasks within the microcontroller.

Frequency Modulation and Phase Modulation

Where AM uses a modulating signal to vary the amplitude of carrier signal, the phase and frequency of the carrier signal may also be altered for use in modulation. In FM modulation, the frequency of the modulated signal varies around a central carrier frequency in proportion to the amplitude of the input signal. Likewise, PM varies the phase of the carrier sinusoid around a central value in proportion to the input signal.

Demodulation of an FM signal can be performed using a phase-locked loop. PLLs lock on to an input signal's phase, calculate the difference between the expected phase and the actual phase of the input signal which results in an output error signal. This output error signal varies coincidentally and in the same manner as the changes in frequency of the modulated signal: the output signal is proportional to the data signal.

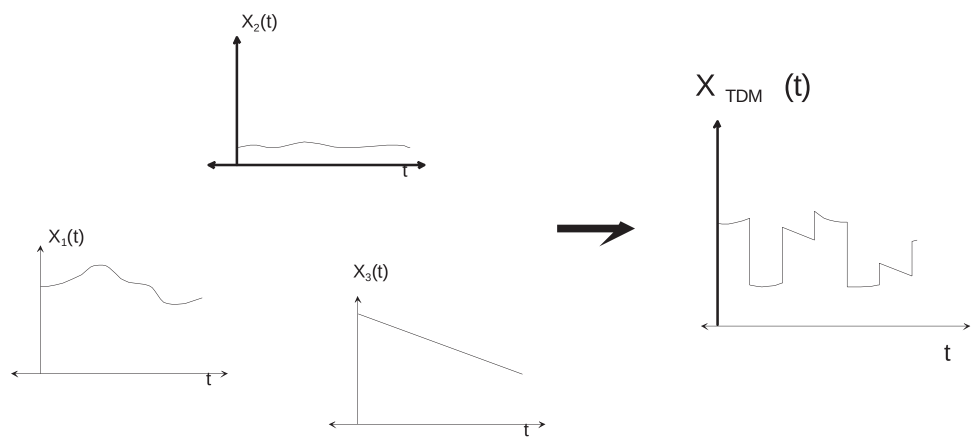


Figure 3.5: Schematized view of TDM (AM with pulse train carriers.)

3.3.3 TDM

Considering the limitations and obstacles of the FDM modulation schemes, several time-domain-multiplexing schemes were considered. As its name implies, time-domain-multiplexing communicates its multiplexed data streams through shared segments of the time spectra (Figure 3.5) (i.e. only one data signal is transmitted at one time.) Intuitively, TDM modulation is the simplest to implement. The main issues to be concerned with for IOME Cam are the necessary and allowable data rates.

Since the Hummingbird's microcontroller has only digital (TTL) output capabilities and considering computer interface purposes it made most sense to consider a digital output TDM scheme (unlike the analog one, AM, depicted in Figure 3.5.)

Phase shift keying

The digital TDM modulation scheme that was chosen for use with this project is called phase shift keying (PSK). In PSK, a periodic carrier signal stores information in it's phase. Shown in Figure 3.6 with a pulse train carrier, a 1-bit is represented in (a), a 0-bit is a "flipped" 1-bit in (b), and an example 8 bit byte is shown in (c). Phase shift keying was an attractive solution to the modulation problem because it utilizes TTL digital output,

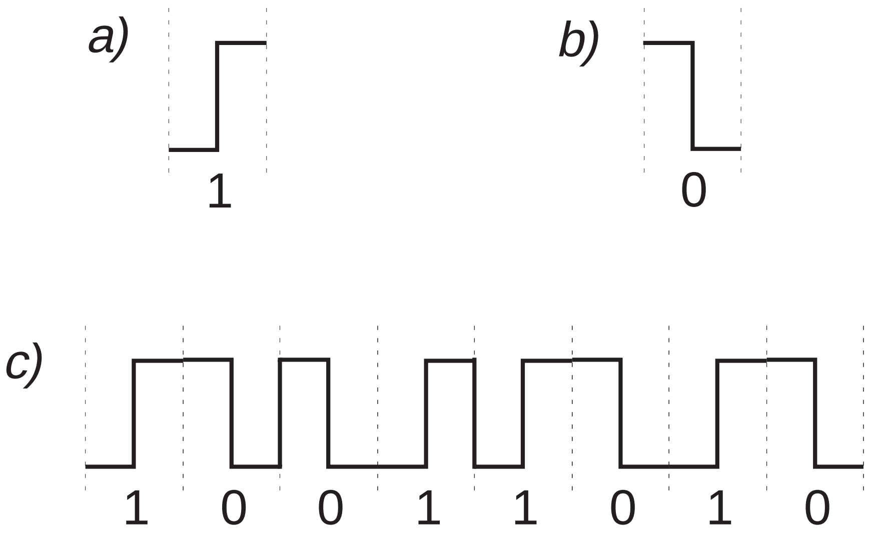


Figure 3.6: Schematized view of PSK: (a) 1-bit, (b) 0-bit, (c) 8 bits: 10011010

it lends itself well to time- domain-multiplexing without need for complex synchronous demodulation techniques, and its operation is not functionally dependent on gain control of the data channel.

Essentially the only consideration for implementation of the PSK modulator was the desired data rate and the data storage channel's bandwidth. Since it is desirable to have inertial data over-sampled compared to optical data (because of inertial data's time-dependent nature) and since the NTSC video signal is updated at 30 Hz, the target update frequency for the IMU was set at 90 Hz. An image of an actual IOME Cam PSK modulated data waveform can be seen in Figure 3.7.

Recalling the 3 dB bandwidth of the audio track is roughly 0 to 12 kHz, it is safe to assume gaussian-type curve and that the optimal audio frequency is about 6 kHz. A 6 kHz PSK carrier has a period (bit length) of about 166 s. Accounting for 8 data bits, 1 start bit, one stop bit, and 7 bytes per frame, each frame consists of 70 bits. The frame length is 11.67 ms, which yields a frame update rate of about 85.7 frames/second - quite close to the desired 90 frames/second (90 frames per second is achieved with a 6.3 kHz carrier.)

While the inertial information update frequency of 90 Hz is well above the bandwidths

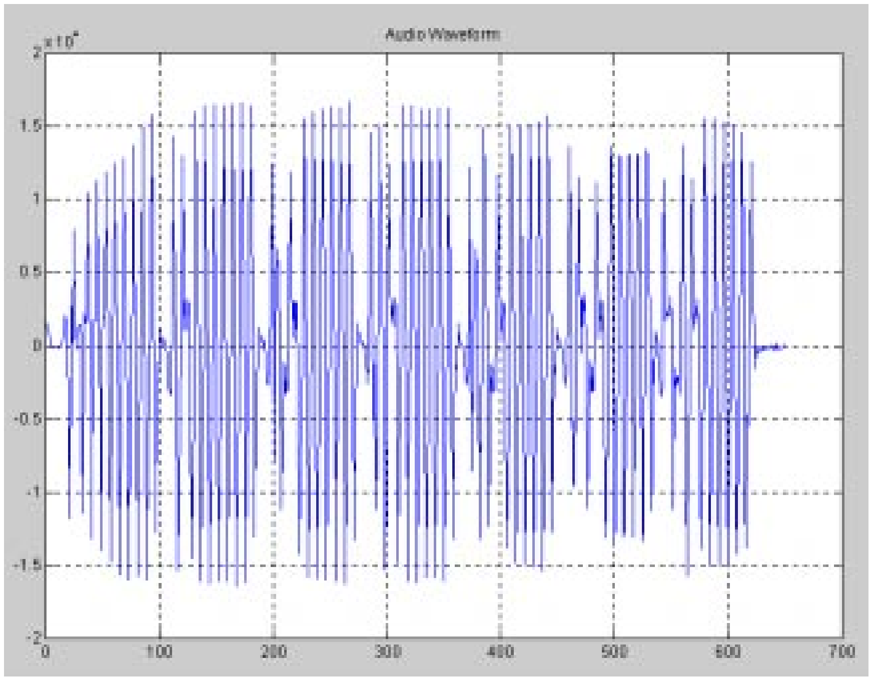


Figure 3.7: Raw PSK audio signal

of each of the motion signals, the estimated errors per frame for each data type can be calculated given the duration of non-information per frame. Each frame consists of 7 bytes, so a lag of 6 bytes, or 9.96 ms exists between each sensor's update. An estimate of each sensor's frame-to-frame error may be found with the corresponding kinematic equations. For the accelerometers (with 0.002 g noise)

$$dx = \frac{1}{2} 0.002g \left(\frac{9.8m/s^2}{1g} \right) (9.96ms)^2 = 1.0$$

The gyro, with 0.45 deg/sec noise gives frame-to-frame angular noise of

$$da = 0.45deg/sec\,(9.96ms) = 0.0045deg$$

.

3.3.4 Demodulator

The PSK demodulator for IOME Cam was developed entirely in software. In practice, three elements were used in converting the audio waveform to binary data. A garden variety audio digitizer was used to sample the audio signal at 44.1 kHz as a 16 bit stereo signal in .wav format. Goldwave software was then used to convert .wav to .snd (ASCII.) Finally a software demodulator was developed in Matlab.

The demodulation algorithm was simply to identify the interframe synch "bytes" and then to read the 70 10-bit bytes in between. Byte reading was accomplished by iteratively time-shifting one bit period length and analyzing the magnitude of the second half of each bit carrier. An initial normalization component also was implemented in an attempt to undue the signal damage incurred by the transformer and any other un intentional filtering. An image of the demodulated "bits" from the signal in Figure 3.7 can be seen in Figure 3.8.

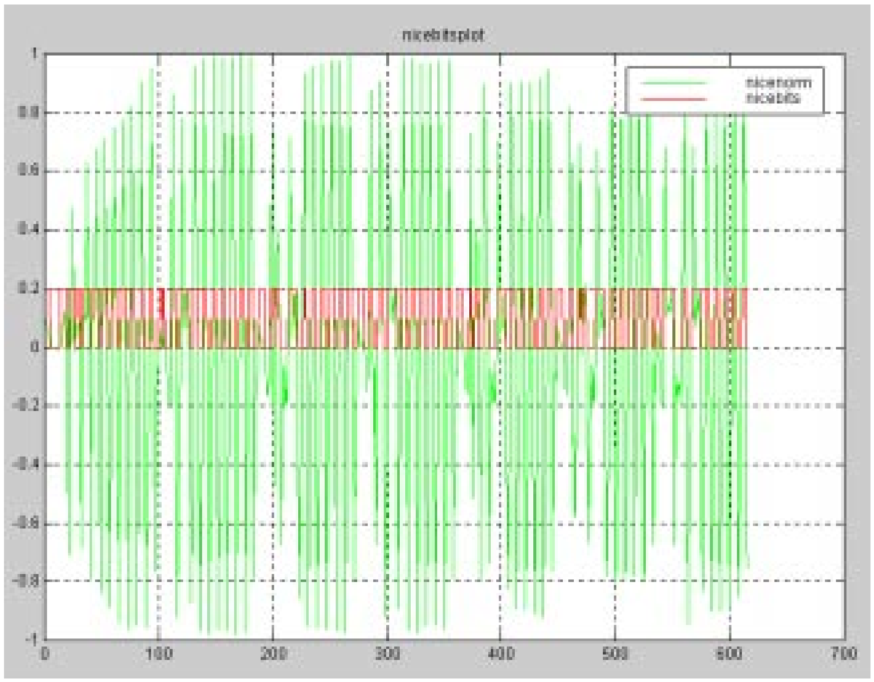


Figure 3.8: Audio signal and demodulated bits

3.4 Improving on the Hummingbird

There are a good number of engineering aspects of the Hummingbird that call for improvements, should the device be considered for more serious modeling application and such. Namely, the fabrication, component placement, and design would need to be carried out with more precision. The camera mounting connection is another necessary area for improvement.

One of the Hummingbird's biggest advantages is its ease of use and application-based adaptibility. For general purpose proprioceptive applications, the Hummingbird is an ideal prototyping tool considering its ready-to-use nature and its built-in user-programmable gain and bandwidth settings.

Several other striking needs for improvement of the Hummingbird IMU are its size and speed. Currently the device is about the size of a computer mouse and needs to approach the size and form factor of a matchbook, considering today's surface mount components and minimal area connectors. IMU speed is another current limiting factor. Considering the data channel it was designed to work with, transmission of around ten 8-bit data streams at around 100 Hz suffices. However, such a system should be capable of real-time analysis and storage of such data and transmission of higher bandwidth information.

Chapter 4

The joint motion estimator: the Kalman Filter

This thesis sets out to produce a joint motion estimator, combining inertial and visual motion estimation techniques. This can be accomplished by combining an optical motion estimator, such as those based on optical flow or feature tracking (Section 1.3), with inertial motion data by use of Kalman filtering techniques. The inertial motion data, when transformed to be represented in terms of the image plane via a transform model like equation 1.1 or 1.2, becomes readily accessible for comparison or joining with other (optical) motion terms.

4.1 Characteristics of the Kalman Filter

The Kalman filter came about as an extension of the Wiener solution for obtaining an optimal estimate of a system's state [Kalman, 1960]. That is, the Kalman filter is used to produce an unbiased, minimum variance, consistent estimate of a state vector, $\vec{x}$, based on a set of measurements, $\vec{y}$, in a system that can be described by the two governing equations:

The extended Kalman filter is implemented by recursively computing updates [1] to the state vector $\vec{x}$ and the state error covariance matrix $\mathbf{P}$, given initial estimates of these two quantities. For our system, updates occur in two ways: temporal updates, based on inertial information, and observational updates, based on optical information. When accurate optical motion information, or measurements, are available, they are used for the estimator. For the times when optical information is available with a higher error probability, the inertial information is used for the estimator (in a *dead-reckoning*-like manner.)

$$\vec{x}_k = \mathbf{A}_{k-1} \cdot \vec{x}_{k-1} + \vec{\eta}_k \tag{4.1}$$

and

$$\vec{y}_k = \mathbf{B}_k \cdot \vec{x}_k + \vec{\epsilon}_k \tag{4.2}$$

where

$\mathbf{A}$ and $\mathbf{B}$ are matrices describing the relationships between observed and state vectors,
$\vec{\eta}$ is Gaussian random forcing function (noise) with error covariance matrix $\mathbf{Q}$
$\vec{\epsilon}$ is the measurement noise with error covariance matrix $\mathbf{R}$

The two noise sources, $\vec{\eta}$ and $\vec{\epsilon}$, are considered white noise, that is their values at some time k is completely independent of their values at any other time k and that they are independent from all other variables being considered. The errors are assumed to be zero-mean random vectors, uncorrelated in time, but possibly having correlations among their components. They are expressed in terms of covariance matrices $\mathbf{Q}$ and $\mathbf{R}$

[1]The updates for this system occur at a rate of 90Hz, which provides inertial data at three times the rate of optical data (for NTSC signals). For PAL or other video signal types, the IMU speed could be altered to maintain the 3:1 inertial to optical data ratio.

$$\mathbf{Q} = \langle \vec{\eta}\vec{\eta}^{\mathrm{T}} \rangle \tag{4.3}$$

$$\mathbf{R} = \langle \vec{\epsilon}\vec{\epsilon}^{\mathrm{T}} \rangle \tag{4.4}$$

The error covariance matrices are used in the iterative filter to compute current values for filter weights.

The Kalman filter is a recursive procedure, efficient with computational resources and memory. It uses the result of the previous step to aid in obtaining the desired result for the current step. This is a key difference from the Wiener (weighting function) approach, which requires arithmatic operations on all the past data. Kalman's solution also lends itself well towards discrete-data problems, and time-variable, multiple input/output problems (which navigation is an example of). The camera navigation problem, with both optical and inertial data, is a prime candidate for use of the Kalman filter estimator.

4.2 Filter Iteration and Propagation

For a system defined by equations 4.1 and 4.2 the recursive Kalman filter propagates in the following manner. Given initial estimates for the state $\vec{x}_{k|k-1}$ and error $\mathbf{P}_{k|k-1}$, the steps are the following.

• Make estimate of the new observable - This estimated value uses all state data up to the present time.

$$\vec{y}_{k|k-1} = \mathbf{B}_k \cdot \vec{x}_{k|k-1}$$

• Obtain new measurement for the observable - This observable vector may be of variable length, depending on which measurements are available.

$$\vec{y}_k$$

• Compute the Kalman gain matrix - The Kalman gain matrix includes the weighting factors of each observable for future state predictions. It is this matrix which will alternate between the inertial and optical data, based on their respective contributions to the error covariance.

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{B}^{\mathrm{T}}(\mathbf{B}_k\mathbf{P}_{k|k-1}\mathbf{B}_k^{\mathrm{T}} + \mathbf{R}_k)^{-1}$$

• Make optimal estimate of the new state - This value will be the best estimate of the state and will be used for predicting the next state as well as for estimating the next observable.

$$\vec{x}_{k|k} = \vec{x}_{k|k-1} + \mathbf{K}_k(\vec{y}_k - \vec{y}_{k|k-1})$$

• Update the error matrix - Using new Kalman gain matrix

$$\mathbf{P}_{k|k} = (1 - \mathbf{K}_k\mathbf{B}_k)\mathbf{P}_{k|k-1}$$

• Make prediction of the new state - Make predictions for next iteration.

$$\vec{x}_{k+1|k} = \mathbf{A}_k \cdot \vec{x}_{k|k}$$

• Predict the new error - Typically, the magnitude of $\mathbf{P}$ will increase with inertial (temporal) updates and will decrease with optical (observational) updates.

$$\mathbf{P}_{k+1|k} = \mathbf{A}_k\mathbf{P}_{k|k}\mathbf{A}_k^{\mathrm{T}} + \mathbf{Q}$$

The above steps are then repeated. Given this iterative method, only the data from the previous step, plus any new (observed) information, is necessary to reach the next iteration(s).

4.3 Controlling Errors with the Kalman Filter

The strength of the Kalman filtering comes from its selective weighting of a system's inputs. The Kalman gain matrix $\mathbf{K}$, used for finding the new state $\vec{x}_{k|k}$ and error matrix $\mathbf{P}_{k|k}$, dynamically changes the weighting it gives to each input type based on the system's current

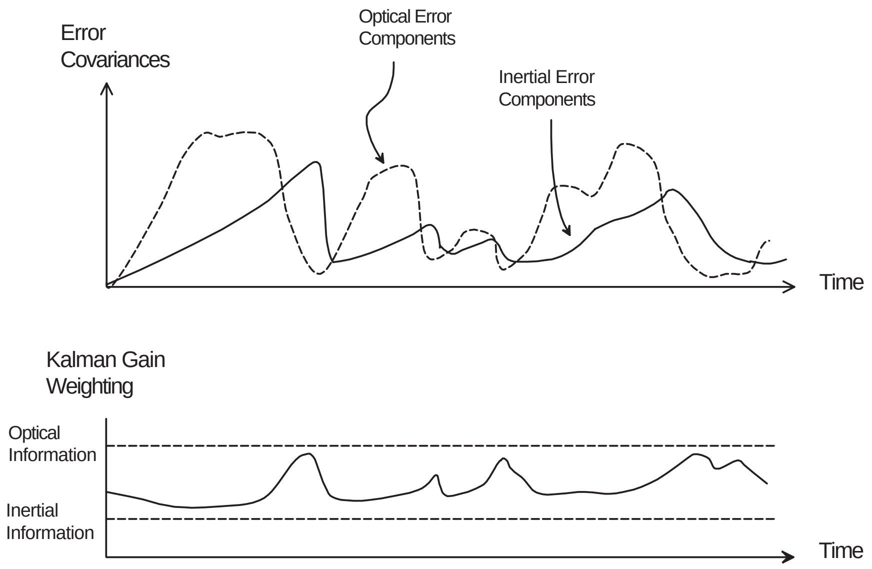


Figure 4.1: Kalman gain matrix dynamically changes weights of input types based on their contributions to the error covariance matrix.

error covariance **P** whose internal components are functionally dependent on the accuracy of the inertial and optical information.

This functionality can be qualitatively seen in Figure 4.1. It is shown that the components of the error covariance due to inertial data and optical data continuously rise and fall during the segment. The Kalman gain weighting adjusts accordingly, yielding a minimum variance estimate of the state vector.

Figure 4.2 shows the corresponding positional errors for each data type alone accompanied with the resulting optimal-estimate error contribution, which is better than either of the two alone.

4.4 IOME Cam's System Model and State Space

To estimate a camera's motion, the relevant variables for the state vector are those associated with translation and rotation with respect to a set of global coordinates. The

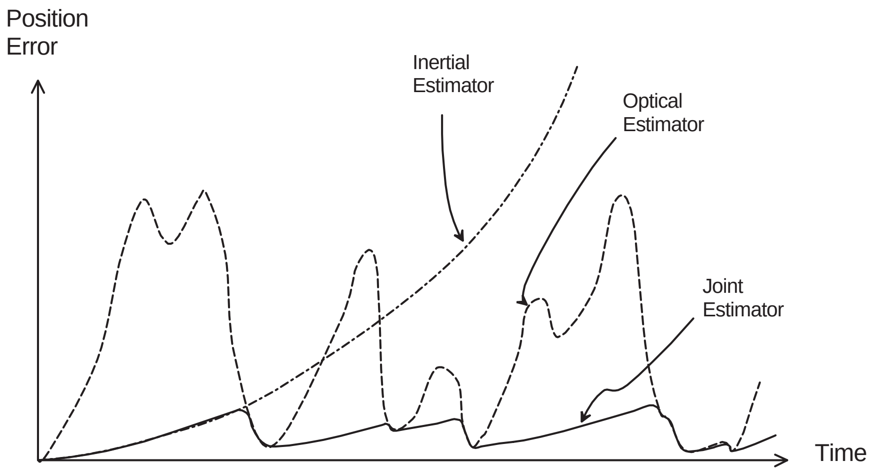


Figure 4.2: View of positional errors: inertial, optical, and joint (using Kalman filter.)

components of the state vector $\vec{x}$ for our system are defined as

- x_1: x-position, P_x, (with respect to global coordinates)
- x_2: x-velocity, V_x, (with respect to body-fixed coordinates)
- x_3: x-acceleration, A_x, (with respect to body-fixed coordinates)
- x_4: y-position, P_y, (with respect to global coordinates)
- x_5: y-velocity, V_y, (with respect to body-fixed coordinates)
- x_6: y-acceleration, A_y, (with respect to body-fixed coordinates)
- x_7: z-position, P_z, (with respect to global coordinates)
- x_8: z-velocity, V_z, (with respect to body-fixed coordinates)
- x_9: z-acceleration, A_z, (with respect to body-fixed coordinates)
- x_{10}: Roll, ϕ, (angular position about body x-axis with respect to global coordinates)
- x_{11}: Roll rate, ω_x, (rate of rotation about body x-axis)
- x_{12}: Pitch, θ, (angular position about body y-axis with respect to global coordinates)

- x_{13}: Pitch rate, ω_y, (rate of rotation about body y-axis)
- x_{14}: Yaw, ψ, (angular position about body z-axis with respect to global coordinates)
- x_{15}: Yaw rate, ω_z, (rate of rotation about body z-axis)

Note that the terms associated with camera orientation, that is roll (ϕ), pitch (θ), and yaw (ψ), must be defined in terms of the global coordinate system via the coordinate transformation introduced in section 2.3. This step introduces a nonlinear characteristic to our Kalman filter and is described in detail in section 4.8.

4.5 System Update

When maximal measurement data is available the entire state space can be updated and/or estimated in one step. In such a case the following full state-space update step is taken.

$$\begin{pmatrix} P_x \\ V_x \\ A_x \\ P_y \\ V_y \\ A_y \\ P_z \\ V_z \\ A_z \\ \phi \\ \omega_x \\ \theta \\ \omega_y \\ \psi \\ \omega_z \end{pmatrix} = \begin{pmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 & & & & & & & & & & & & \\ & 1 & \Delta t & & & & & & & & & & & & \\ & & 1 & & & & & & & & & & & & \\ & & & 1 & \Delta t & \frac{1}{2}\Delta t^2 & & & & & & & & & \\ & & & & 1 & \Delta t & & & & & & & & & \\ & & & & & 1 & & & & & & & & & \\ & & & & & & \cdot & \cdot & \cdot & & & & & & \\ & & & & & & & \cdot & \cdot & & & & & & \\ & & & & & & & & \cdot & & & & & & \\ & & & & & & & & & \cdot & \cdot & \cdot & & & \\ & & & & & & & & & & \cdot & \cdot & & & \\ & & & & & & & & & & & \cdot & & & \\ & & & & & & & & & & & & 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ & & & & & & & & & & & & & 1 & \Delta t \\ & & & & & & & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} P_x \\ V_x \\ A_x \\ P_y \\ V_y \\ A_y \\ P_z \\ V_z \\ A_z \\ \phi \\ \omega_x \\ \theta \\ \omega_y \\ \psi \\ \omega_z \end{pmatrix}$$

In other cases, this step updates the required state values. Which components are updated is dependent on the measured information (discussed in the next section.)

4.6 Observables

New IMU data is available to our system at a rate of 90 Hz (see section 3.3) while optical-data occurs at 30 Hz. Therefore, for our 90 Hz update rate system, the comprehensive measurement vector, $\vec{y}$, occurring a third of the time is given by

$$\vec{y} = \begin{bmatrix} a_x \\ a_y \\ a_z \\ \omega_x \\ \omega_y \\ \omega_z \\ af_{xx} \\ af_{yy} \\ af_{xy} \\ af_{yx} \\ b_x \\ b_y \end{bmatrix} = \begin{bmatrix} \text{x-acceleration} \\ \text{y-acceleration} \\ \text{z-acceleration} \\ \text{roll rate} \\ \text{pitch rate} \\ \text{yaw rate} \\ \text{affine scaling parameter} \\ \text{affine scaling parameter} \\ \text{affine shear parameter} \\ \text{affine shear parameter affine x-translation parameter} \\ \text{affine y-translation parameter} \end{bmatrix}$$

When only the inertial data is available, the measurement vector has less terms and the update step matrices, **A** and **B**, are updated accordingly.

4.7 Modeling System Noise and Uncertainties

The noise and uncertainty of IOME Cam's components show their affect in the values of the error covariance matrices **Q**, **R**, and **P**. The following sections discuss how these error sources are modeled.

4.7.1 Accelerometer Uncertainty

Uncertainty of system input signals from accelerometers has a number of potential causes. The ADXL05 type sensors used for this project, like all other accelerometers, suffer from the following: non-linear sensitivity response to various input frequencies [2] , change in noise as a function of temperature [3], and change in sensitivity as a function of temperature. The latter is accounted for in the design of the Hummingbird - a temperature sensor is located on-board so that accelerometer sensitivity can be dynamically calculated.

4.7.2 Gyroscope Uncertainty

Similar to the accelerometers, the Gyrostar gyroscopes being used for this project suffer from uncertainties and noise due to temperature dependencies since their active elements are piezoelelectric. Using vabratory pulses as their actuating and sensing mechanisms, these sensors are also vulnerable to oscillatory interferences that are present at the resonant frequency or its harmonics.

4.7.3 Random Forcing Function System Noise

Possible random forcing-function noise will appear in this system due to unmodelable dynamics of motion of the inertial sensors. Such motion can be caused by the human camera-operator but also can be due to internal sensor errors such as cross-coupling, vibropendulous errors, and hysteresis. These uncertainties must be initially estimated. For this system, their contributions have been considered negligible.

[2]This error will have minimal effect. The ADXL05 has a nearly linear "normalized sensitivity vs. frequency" response up to about 200 Hz, which is well above the limit of human motion.

[3]Micromachined silicon is temperature sensitive and experiences thermal hysteresis effects.

4.8 Recovering rotation and orientation

4.8.1 Orientation From Gyroscopes

The camera's position and orientation with respect to a global coordinate system are desired quantities in our system's state. The IMU's sensors sense motion with respect to body-fixed axes so a coordinate transformation is necessary to yield useful motion data.

To determine orientation, both the IMU's gyroscopes and accelerometers can be used. Figure 2.8 shows two sets of rectangular axes, one in the Global coordinate system (for the system state vector) and one in a body-fixed coordinate system (for the IMU motion sensors). It can easily be shown [Boas, 1983] that one set of axes can be written in terms of the other by

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \\ l_3 & m_3 & n_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

where l_1 = the angle between the x and x' axes = $\cos\angle(xx')$, $l_2 = \cos\angle(xy')$, $l_3 = \cos\angle(xz')$, $l_2 = \cos\angle(xy')$, $m_1 = \cos\angle(yx')$, $m_2 = \cos\angle(yy')$, $m_3 = \cos\angle(yz')$, $n_1 = \cos\angle(zx')$, $n_2 = \cos\angle(zy')$, and $n_3 = \cos\angle(zz')$.

For the case of our IMU, given a sufficiently fast update speed [4], the l_i, m_i, and n_i cosine terms can be found directly from gyroscope outputs, since the associated angles of rotation per clock cycle can be estimated as the rotation rate multiplied by cycle time

$$l_1 = \cos\angle(xx') = \cos([(\omega_y dt)^2 + (\omega_z dt)^2]^{\frac{1}{2}})$$

$$l_2 = \cos\angle(xy') = \cos(90 + \omega_z dt) = \sin(\omega_z dt)$$

$$l_3 = \cos\angle(xz') = \cos(90 - \omega_y dt) = -\sin(\omega_y dt)$$

[4]The angles subtended per IMU system-clock tick should be several orders of magnitude smaller than the state vector angles

$$m_1 = \cos\angle(yx') = \cos(90 - \omega_z dt) = -\sin(\omega_z dt)$$

$$m_2 = \cos\angle(yy') = \cos([(\omega_z dt)^2 + (\omega_x dt)^2]^{\frac{1}{2}})$$

$$m_3 = \cos\angle(yz') = \cos(90 + \omega_y dt) = \sin(\omega_x dt)$$

$$n_1 = \cos\angle(zx') = \cos(90 + \omega_y dt) = \sin(\omega_y dt)$$

$$n_2 = \cos\angle(zy') = \cos(90 - \omega_x dt) = \sin(\omega_x dt)$$

$$n_3 = \cos\angle(zz') = \cos([(\omega_x dt)^2 + (\omega_y dt)^2]^{\frac{1}{2}})$$

This type of orientation recovery was implemented with the Hummingbird for demonstration purposes and related signal-analysis and avatar-rendering code are available in Appendix E and from the Hummingbird web site.[5]

4.8.2 Orientation From Accelerometers

For low-acceleration systems, a triad of accelerometers may also be used to measure, or "watch", gravity to ascertain object orientation, that is roll and pitch angles with respect to the global coordinate system. Such a scheme would yield one ambiguous degree of freedom, however - that being object "heading" with respect to the gravity vector.

[5]The Hummingbird web site is located at http://physics.www.media.mit.edu/projects/hummingbird/ and is a source of related C/C++ code and OpenGL code for avatar control, assembly code for on-board control, and gerber files for board manufacturing.

Chapter 5

Related Work - Inertial Proprioceptive Devices

The IOME Cam one of a number of motor-cognizant devices being researched and developed in a common body of work. Some of the related applications and motivations are described in this chapter as well as the notion of making dumb devices into self-aware ones. Inertial proprioception is also pointed out for its autonomy-enabling virtue as compared to all other (externally referenced) motion sensing types.

As technology redirects *intelligence* away from the desktop and into everyday objects, common devices such as appliances, clothing, and toys are given computational sensing and communication abilities. This technological movement is exemplified in such research initiatives as the MIT Media Lab's *Things That Think* projects and Xerox PARC's concept of *Ubiquitous Computing.* While much of the associated work centers around devices that sense and respond to the motion, presence, or state of people and objects in their surroundings (examples include 3D mice, smart tables, and smart coffee cups), this paper focuses on devices that have a sense of *themselves*, particularly a sense of their own motions. Embedded with inertial sensors, these devices are capable of autonomously sensing their own motions and orientations and reacting accordingly. As a result, they are called *inertial proprioceptive devices.*

Devices with this self-motion-sensing ability can monitor their motions and respond to them. Consider a handheld personal digital assistant (PDA) containing inertial sensors. Such a device could allow its user to move through complex information spaces by physically moving or tilting the PDA in the corresponding direction. To go a step further, an inertial sensing user-controlled device with a sense of its own functionality could assess its state and give its user appropriate feedback. For example, a baseball bat could give batting tips, or juggling balls could teach a novice to juggle.

Motion sensing is not a new idea. For years, security systems, weapon systems, and medical and entertainment systems have employed various forms of "externally referenced" motion sensing technologies such as infrared, radar, and video. Internally referenced, autonomous motion sensing has also existed for quite some time. Robots, aircraft, automobiles, and other vehicles have sensed and measured their motions for decades, using varying electromechanical sensors as well as inertial sensors.

Most of the motion sensing technologies referred to above are restricted in terms of where and how they are useful. Infrared, radar, and video motion sensing technologies are all "externally referenced", physically removed from the moving object of interest. As a result these sensing modes are subject to occlusions and numerous interferences and noise sources. Although cars and aircraft measure their own motions, their motion sensors are both dimensionally and directionally limited. A car wheel's motion sensor requires the friction of a road and only senses in one dimension; a pitot tube only works for an aircraft traveling forward in familiar atmospheric conditions.

A more tractable and generally effective type of motion sensor is the inertial sensor. Used in spacecraft, aircraft, and submarines for years, this type of sensor attaches directly to the moving body of interest and gives an output signal proportional to its own motion with respect to an inertial frame of reference. Two types of sensors comprise inertial sensing: accelerometers and gyroscopes. Accelerometers sense and respond to translational accelerations; gyroscopes sense and respond to rotational rates. Inertial sensors are desirable for general motion sensing because they operate regardless of external references, friction,

Sensor Type	Date	Bias Stability [deg/hr]	Size [in^3]	Price [$U.S./axis]
Electrostatic Gyro (ESG), Rockwell†	1970s	0.02 (1 naut.m/hr)	50-100	17000
Expected Near-Term†† navigation & military gyros	1990s	0.02 (1 naut.m/hr)	10 - 20	5000 - 10000
Expected Near-Term†† general consumer gyros	1990s	10	0.01 - 1.0	1 - 10
†: ESG references include [Schwarz, 1976] and [Mackenzie, 1990] *††: With reference to [BKE, 1994]*				

Table 5.1: Cost, Size, and Performance of Selected Inertial Sensors from 1970s to 1990s

winds, directions, and dimensions. However, inertial systems are not well suited for absolute position tracking. In such systems, positions are found by integrating, over time, the sensors' signals as well as any signal errors. As a result position errors accumulate. Inertial systems are most effective in sensing applications involving relative motion.

Until recent years, inertial sensors have only found use in the few fields mentioned above, since their cost and size have traditionally been quite prohibitive (see Table A.1). Since their inception, these sensors have largely been complex and expensive electromechanical devices. Accelerometers have been made of relatively large mechanical proof masses, hinges, and servos; gyros have been built with multiple mechanical gimbals, pickoffs, torquers, and bearings. Recent advances in microelectromechanical system (MEMS) technologies have enabled inertial sensors to become available on the small size and price scales associated with commonplace devices like consumer appliances. These advances are largely a result of batch processing techniques developed by the time-keeping and microelectronics industries [BKE, 1994].

5.1 Example Proprioceptive Applications

Motion sensing *of* common objects such as shoes and pens has long existed in one form or another. Treadmills have measured people's walking speeds and distances. PDAs sense the path of a pen tip as a user writes on them. And computer programs analyze optical flow of digitized video to infer camera motion. Each of these forms of motion detection requires an externally displaced device to actually sense motion.

Inertial sensors do not require external references, and since they are becoming inexpensive and smaller in size, they offer a new means of autonomous motion detection for devices that have long been dependent on external references (ie. shoes and treadmills). Both the automobile and computer industries have quickly found uses for inertial sensing. In the automotive market, car navigation and airbag control are the main inertial applications; the consumer computer market is seeing new input devices that can be used in three dimensional space like inertial mice and head trackers for virtual reality. The inertial market for these two industries is estimated to be in the range of $4 billion a year over the next several years.[BKE, 1994]

Current work at the MIT Media Lab is focused on giving ordinary devices autonomous motion sensing capabilities, via inertial sensing, so that as pens write, shoes walk, and cameras move, these objects sense their own motions without need for external references. The following sections describe several example applications of human-controlled motion-sensing devices and the characteristics of their related motions. Figure 5.1 summarizes the characteristic input motion levels for general user-controlled devices. For each application, estimated motion data ranges are given along with experimentally recorded ranges. The experimental motion data was gathered both from video analysis and from a three-axis accelerometer-based inertial measurement unit (IMU) with a range of $\pm 10g$. This IMU used Analog Devices' accelerometer model ADXL05, a type of capacitive pendulous accelerometer to be described later.

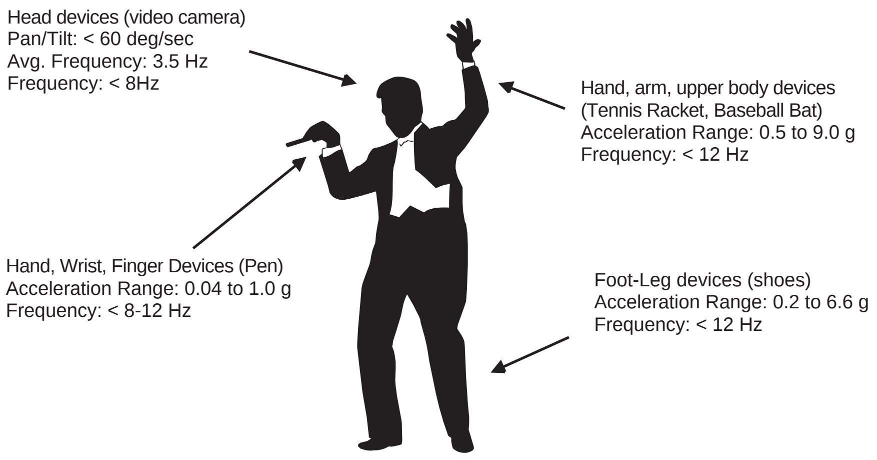

Head devices (video camera)
Pan/Tilt: < 60 deg/sec
Avg. Frequency: 3.5 Hz
Frequency: < 8Hz
Hand, arm, upper body devices
(Tennis Racket, Baseball Bat)
Acceleration Range: 0.5 to 9.0 g
Frequency: < 12 Hz
Hand, Wrist, Finger Devices (Pen)
Acceleration Range: 0.04 to 1.0 g
Frequency: < 8-12 Hz
Foot-Leg devices (shoes)
Acceleration Range: 0.2 to 6.6 g
Frequency: < 12 Hz

Figure 5.1: Characteristics of motion for common human-controlled devices.

5.1.1 Pen

Personal digital assistants and signature verification devices both employ forms of handwriting recognition - each analyzes the path of a pen tip on a writing surface. If a pen is given inertial sensors and onboard computation and memory resources, it can sense its motions while it writes and use that motion data to estimate its time-varying position. By employing a pattern recognition method such as a neural network or hidden Markov model [SMS, 1994] on its time-varying pen tip position, the pen can *know* and remember what it has written. Such a "smart" pen could not only save notes and letters but also send email, solve mathematical problems, check for spelling errors, and carry out other standard computer operations.

An estimated range for pen tip accelerations was found by video taping the pens and papers of several people as they signed their names. Pen tip velocities and radii of curvature of a number of characters were used to calculate the corresponding centripetal accelerations, which ranged from 0.1 g to 1.0 g.

Pen tip accelerations in the 2D writing plane were also recorded using the aforementioned IMU attached to a pen tip. Recorded handwriting accelerations ranged, with uniform distribution, from 0.04 to 0.66 g.

The frequency of motion for handwriting will be estimated as the approximate natural frequency of the wrist and hand, 8 to 12 Hz, and should not exceed 20Hz [?]. Considering the relative size and motion scales, the handwriting characteristic frequency described here will act as the frequency limit for other applications such as foot, leg, and arm controlled devices.

5.1.2 The Digital Baton

An application with similar motion-sensing requirements is the 'Digital Baton' [Marrin, 1996], which was developed at the MIT Media Lab. This device, using an orthogonal accelerometer triad for motion sensing and several pressure sensors for analog finger inputs, allows its user to 'conduct' and control computer music orchestrations simply by moving and gripping it in different ways.

5.1.3 Cameras

In addition to the inertial-optical motion estimating video camera being described in this paper, there are a number of other motivations and applications for cameras to know their motions. Currently, one of the main trends of information technology is towards object-based media and media with higher-level "tags" or meta information. The attachment of sensors to a video camera that allow recognition of video-related values, like motion, zoom, focus, and lighting, will allow automatic tagging of the video and audio information with a meta-track. Such information is useful for editing, playback, and other interactions of media types.

5.1.4 Shoes

Just as most types of vehicles have speedometers and odometers, shoes should also be able to sense and track their motions. The medical and athletic fields have relied on various forms of externally referenced walking rate and distance sensors for some time. Shoes embedded with an inertial sensing system would allow walking-sensing to be carried out unobtrusively and in any setting. An inertial shoe pedometer system would work much like the pen and camera described above; inertial sensors would record shoe motion components and an onboard computer would estimate speed and distance traveled. Given sufficient computational, memory, and sensing resources, a proprioceptive shoe system could not only tell its wearer how far and fast he/she is walking but could also diagnose gait abnormalities or alert the wearer that it's time to replace the shoe soles.

For a benchmark estimate of the shoe accelerations associated with walking, consider an average man's walking speed of 3.5 mph (5.13 fps) or 2 steps/sec [Starner, 1996]. The centripetal acceleration of a shoe traveling 5.13 fps about an average adult's knee of radius 2.17 ft [Woodson, 1981] is 12.1 ft/sec^2 (about 0.4 g).

Experimental values of walking foot accelerations were obtained with the previously mentioned IMU fastened to a shoe near the ball of a foot while walking. Recorded accelerations ranged from 0.19 to 6.57 g, with nearly all the acceleration activity located near the mean of 1.59 g.

Given the estimated walking accelerations, inertial sensors used for shoe motion tracking should have an input range of about ± 10 g's.

5.1.5 Bats and Rackets

The final example application area includes toys and tools that are swung or waved expressively by their users. A baseball bat or tennis racket that senses its motions can tell a player how fast he/she is swinging and if used with other sensors and a microprocessor, could give feedback information about a player's performance.

Using the test IMU, hand accelerations were recorded during athletic arm/hand swinging motions. An acceleration range of 0.49 to 9.02 g was found. Most of the acceleration activity was concentrated near the mean value of 2.2 g.

Baseball bat accelerations for a typical youth swing (bat speed of 40 mph, 58.7 fps) [WB, 1990] are estimated as the centripetal acceleration. These will serve as an upper limit. Assuming a swinging arm length of 2 feet and a distance of about 10 inches from the hands' position to the center of mass of the bat, the bat will experience a maximum acceleration of $(58.7fps)^2/2.8ft = 1230ft/sec^2 = 38g's$! At the same time, the handle of the bat will undergo an acceleration of about $(29.3fps)^2/2ft = 429ft/sec^2 = 13g's$.

Given the motion range estimates for these athletic/expressive hand and arm applications, any inertial sensors measuring the motion of a user's hand or arm needs to have an upper input limit of near 10 to 15 g's. If the motion of an object extending from the user's body (like a baseball bat) is to be sensed, a greater input range (about 50 g) is necessary.

Chapter 6

Conclusions and Future Research

In the course of work, research, and development associated with the IOME Cam project, a number of conclusions were reached and an even larger number of directions for future research were found. These aspects will be discussed in the following sections.

6.1 Motion Estimation: Optical and Inertial

Given that the domain of camera motion estimation is an interesting and useful one, characterizing the technologies and methods of performing such estimation is a useful task. The conventional optical motion estimation techniques, while quite thorough in development, suffer from a number of fundamental shortcomings based on their input type and its imposed constraints on the motion estimation problem. For vision based motion estimators, some hindrances in the estimation are encountered unmodelable changes in lighting and texture, dynamic (movement) scenes, and occlusions. Such error sources are typically transient in nature, meaning useful optical information will reappear again and then fade again. Hence, vision-based motion estimator systems are best suited for longer time period estimation. The feature tracking approach exemplifies this characteristic.

Conversely, inertial motion estimation is weakest in long-term applications. Inertial systems perform best in analyzing relative motion or short, intermediate, intervals. For these reasons, the inertial and optical data types are complimentary for the motion estimation problem. Where a sole inertial system would experience fatal baseline drifts over long periods of time, an optical system's longer-period absolute update information can restore the inertial baseline. At the same time, where vision systems are unable to perform optimally from one given frame to another due to visual misinformation, inertial systems are ideal for "filling in the holes".

The work done for this thesis project makes an advancement towards a fully integrated camera motion estimator in which inertial data accompanies optical data in a joint motion estimator. The two information types each have error-prone tendancies, but the advantage of combining them is that the long-term stability of vision data is able to restore inertial error drift and that the frame-to-frame usefulness of inerital data can correct spurious optical errors.

6.2 Applications and Motivations

Motivating applications for a joint inertial-optical motion estimating camera system are numerous. The introduction of this paper described several, including two dimensional and three dimensional computer modeling of scenes, which have applications in compression and transmission of "movie" data as well as creation of visual and audio special effects. Knowledge of positional information about where the camera has been or is going can allow integration of computer generated media entities with the recorded real media.

The entertainment industry has shaped much of IOME Cam and its related work. Special effects creation along with the application of virtual sets are both benefactors and users of the camera motion parameter technology. Another entertainment medium that is quite suitable for inertial video data is the evolving ride movie. The video, motion, and sound-recording IOME Cam makes a natural input to the video, motion, and sound-output theaters being built today.

Another research area, extending from IOME Cam, that is being actively pursued by the author and Media Lab researchers is the concept of MetaMovies[1]

6.3 Inertial Sensing Technologies

The state of inertial sensing has progressed to a point where integration of inertial motion sensors with applications like the IOME Cam are afforable and effective. Performance, price, and size characteristics are allowing cameras to watch where they're going and a host of other devices to become motor-cognizant and motion- responsive.

The technologies associated with inertial sensing are also progressing at enourmous rates[KD, 1996]. In the course of the two years that the IOME Cam and its related projects were ongoing, the state of the art in micromachined silicon accelerometer technology included a decrease in both size and cost by about sixty percent.

6.3.1 The Hummingbird, IOME's IMU

The Hummingbird IMU, originally intended just for the IOME Cam project, through the course of its development, became a general purpose platform for Things That Think-related projects. The sif degree of freedom, temperature sensitive, and user programmable IMU is easy to work with and plug in to other applications, but is lacking in terms of the precision engineering requirements of some IOME Cam related goals. The non-precise component placement and alignment should be allayed in the next version as should the 8-bit A/D data stream (12 or 16 bits would allow for higher motion data precision.)

Also, the Hummingbird's mechanical couplings and interfaces inevitably led to numerical errors in inertial motion estimation. Again, all levels of mechanical / engineering positioning

[1]Introduced in Section 1.2.2, the research project aims to record camera motion and other camera state variables (i.e. zoom and focus) synchronously with the audio and visual action. Camera state data is analyzed to extract higher lever, meta, information relating to the action or content of the movie. This work is being carried on with the Media Lab's Digital Life consortium.

and alignment could be improved upon.

Performance-wise, the IMU and its corresponding modulator and demodulator work only in a post production setting in which speed is not crucial. Ideally, this system should work in real-time (which would allow numerous new application), but is in need of faster and more sophisticated hardware to do so. Modulation and demodulation of the Hummingbird's motion data signal was sufficient for the 90 Hz TDM scheme presented earlier, but could be improved with one of a number of FDM modulation schemes.

6.4 Other Proprioceptive Devices

The IOME Cam is one of a family of proprioceptive devices under development. The notion of devices with a sense of themselves that know how they are moving and can act accordingly is an empowering and technologically exciting one. Projects related to this motion-sensing camera include shoes with inertial pedometers, pens that know what they're writing, golf clubs and baseball bats that assits their users, and a digital conductor's baton that controlls a computer orchestra.

While inertial sensing has been shown to be somewhat of a born-again technology that has found a home in intelligent personal electronics, the development of this IOME Cam project has shown that inertial sensing is a powerful addition to the camera motion parameter estimation problem. Where vision-based estimators once ran into impossible modeling situation, with a lack of visual information, inertial motion data fits in in a corrective manner. The sole-inertial motion estimator also benefits from the long term stability of the optical estimator, showing that the two modes work better together.

Appendix A

Hummingbird - Assembly

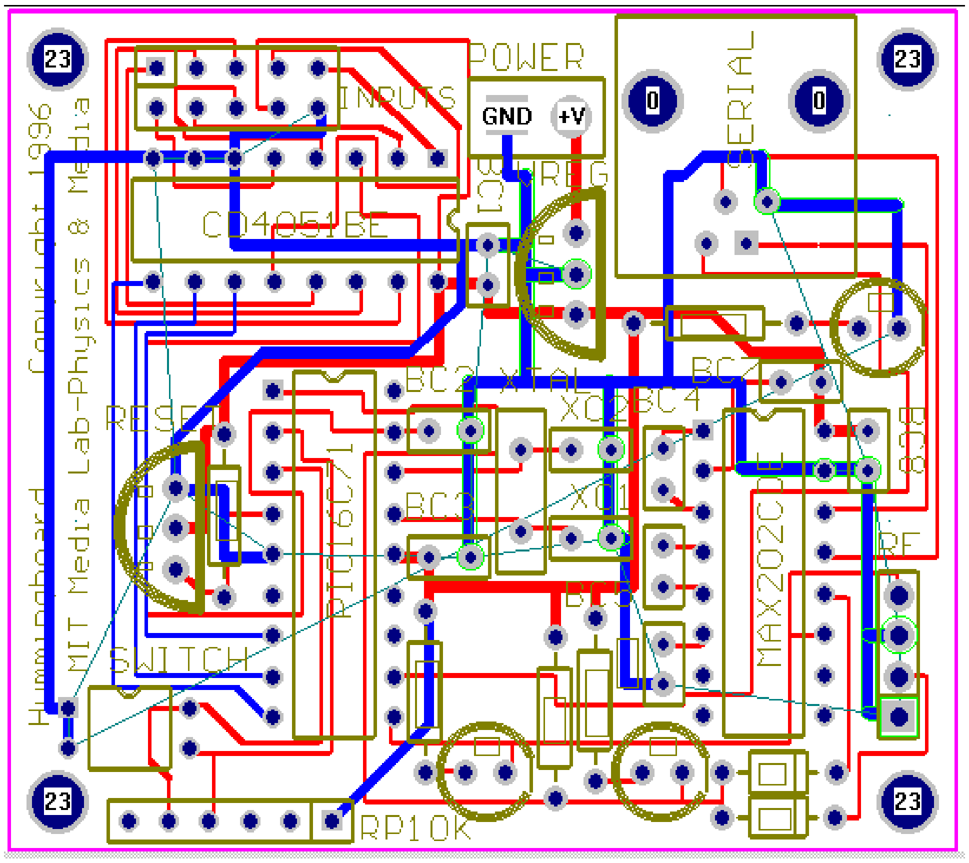


Figure A.1: Hummingboard

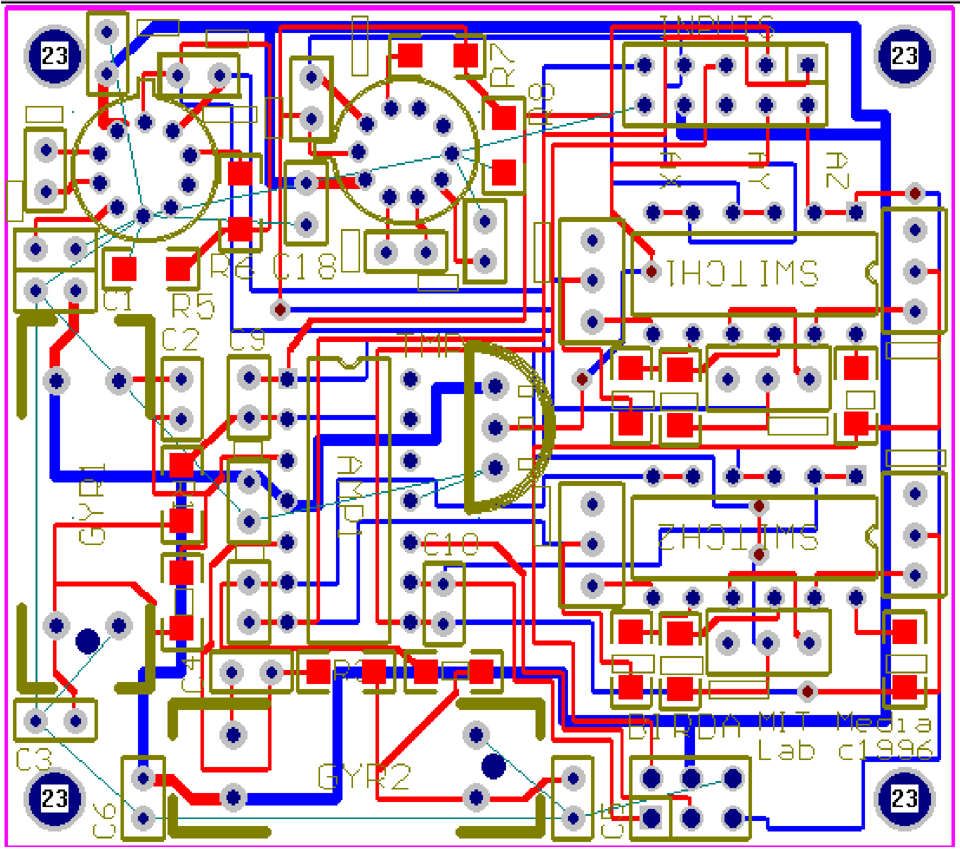


Figure A.2: Birdboard

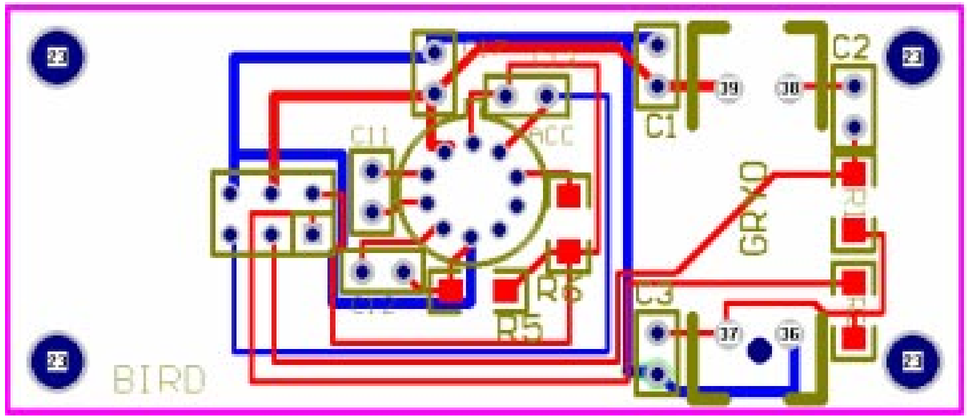


Figure A.3: Birdboard A

A.1 Parts List

```
Parts to order for Hummingbird. All parts were ordered from Digi-Key
Digi-Key Corporation
1-800-digi-key
http://www.digikey.com/
```

Part Number	Description	Price	Manufacturer
Gyro Circuit			
3299W-104-ND	3/8” multiturn 100 K pot	56.40/200	Bourns
P4908-ND	0.047uF cer cap	7.83/200	Panasonic
P5292-ND	4.7uF elec- trolytic cap	5.85/200	Panasonic
Accelerometer Circuit			
3299W-105-ND	3/8” multiturn 1M pot	56.40/50	Bourns
56KEBK-ND	56K 1/8W res.	4.70/200	n/a
270KEBK-ND	270K 1/8W res.	4.70/200	n/a
PCC223BCT-ND	22nF smt cap	20.00/100	Panasonic
PCC333BCT-ND	33nF smt cap	20.00/100	Panasonic
General Bird Board			
A5506-ND	SPST 6-crqt switch	15.13/10	AMP
MAX475CPD-ND	10MHz qd opamp	154.50/25	n/a
3M1110-ND	2mm 10 pin dual row header	10.40/10	n/a
3M1008-ND	2mm 8 pin dual row header	10.40/10	n/a
3M1108-ND	2mm 8 pin dual row header	12.90/10	n/a
General Hummingboard			
MAX202CPE-ND	rs232 transceiver	29.40/10	n/a
CT2082-ND	2-cqt SPST dip switch	8.90/10	n/a
H2031-ND	2mm 10 pin vert. pcb sock	10.70/10	n/a
WM4700-ND	2 pin right angle header	1.13/10	n/a
102CR-ND	blue LED	7.83/10	CREE
LT1078-ND	red LED	1.38/10	LITEON
LT1080-ND	green LED	1.38/10	LITEON
3M1010-ND	2mm 10 pin dual row socket	12.90/10	n/a

Table A.1: Parts, Prices, and Manufacturer Information for Hummingbird.

Appendix B

Hummingbird - Programming

B.1 PIC microcontroller code

```
; ---------------------------------------------------------------------
; hum_au_1 verpcam driver code - send 6Kbaud audio signal
;               including 7 chanels of dataw
;
; General Purpose Inertial Controller PIC Controlling Code
; Chris Verplaetse
; MIT Media Lab
;
; May-Oct 1996
; ---------------------------------------------------------------------

; -- Device Configuration

        DEVICE  PIC16C71,HS_OSC,WDT_OFF,PWRT_OFF,PROTECT_OFF
        id      'ADC1'  ; Device ID

; -- Port Assignments

serial_in       EQU     RA.2    ;trigger pin - listens to PC
;serial_in       EQU     RB.0    ;trigger pin - listens to PC
MUXA            EQU     RB.1    ;PIC pin7 --> mux select A
MUXB            EQU     RB.2    ;PIC pin8 --> mux select B
MUXC            EQU     RB.3    ;PIC pin9 --> mux select C
TX              EQU     RB.4    ;serial data out
DEBUG1          EQU     RB.5    ;debugging pin
DEBUG2          EQU     RB.6    ;debugging pin

; -- Constants

;BIT_K           EQU     24      ;24 for 19200
;HALF_BIT_K      EQU     12
;BIT_K           EQU     50       ;50 if for 9600
;HALF_BIT_K      EQU     25
BIT_K           EQU     83      ;83 for 6000
```

```
HALF_BIT_K      EQU     42
;BIT_K           EQU     102      ;102 if for 4800
;HALF_BIT_K      EQU     51
;BIT_K           EQU     206      ;206 if for 2400
;HALF_BIT_K      EQU     103
SENSOR_TOT      EQU     6       ;there are *7* sensors 0 through 6

; -- File Register Initialization

org 0Ch
counter1 ds 1
counter2 ds 1
delay_ctr       ds      1
xmt_byte        ds      1
bit_ctr         ds      1
rbit_ctr ds 1
rcv_byte ds 1
rcv_trig ds 1
sindex ds 1
sindexo ds 1
bytenum ds 1

; -- Initializations


; -- ADC Setup

;AD_clk =       0               ;PIC oscillator period X 2 (<=1MHz)
;AD_clk =       64              ;PIC oscillator period X 8 (<=4MHz)
AD_clk  =       128             ;PIC oscillator period X 32 (<=16MHz)
;AD_clk =       192             ;Independent RC Oscillator, 2-6 us

AD_ch  =       0                ;ADC Channel 0 (Ain0, pin 17)
;AD_ch  =       8               ;ADC Channel 1 (Ain1, pin 18)
;AD_ch  =       16              ;ADC Channel 2 (Ain0, pin 17)
;AD_ch  =       24              ;ADC Channel 3 (Ain0, pin 17)

AD_ctl  =       AD_clk | AD_ch

;AD_ref =       0               ;RA.0-RA.3 Analog, Vdd reference
;AD_ref =       1               ;RA.0-RA.2 Analog, RA.3 reference
AD_ref =       2               ;RA.0/1 Analog, RA.2/3 Digital, Vdd reference
;AD_ref =       3               ;RA.0-RA.3 Digital Input, Vdd reference

; Set starting point in program ROM to zero. Jump past interrupt vector to beginning of program. (This
; program doesn't use interrupts, so this is really not needed, but it will be easier to add interrupts later if required.)

org 0
jmp start
org 5

start           clrb rp0
mov     intcon,#0       ;Turn interrupts off.
                mov     adcon0,#10000000b       ;Set AD clock and channel.
setb rp0 ;Enable register page 1.
                mov     TRISA,#255      ;set porta to input.
                mov     TRISB,#0        ;set portb to outputs.
                mov     adcon1,#AD_ref  ;Set usable pins, Vref.
clrb rp0 ;Back to register page 0.
                setb    adon            ;Apply power to ADC.
                clrb    DEBUG1
                clrb    DEBUG2
                clrb    TX
clr rcv_byte
clr rcv_trig
clr xmt_byte
mov sindex,#0 ;initialize sensor index
mov bytenum,#65

:goesync nop
                mov     xmt_byte,#00000000b    ;sync byte
                mov     bit_ctr,#4      ;8 data bits in a byte
:xmitsync       rr      xmt_byte        ;move bit 0 of xmt_byte (from adcsample)
                movb    TX,c            ;transmit the bit
                call    period          ;delay
                djnz    bit_ctr,:xmitsync   ;Not 8 bits yet ? Send next bit

:goe            nop
call muxsele ;select mux ch0
call littledally ;give mux time to settle after address
call adcsample
                setb    c               ;start bit
                call    xmt
                mov     bit_ctr,#8      ;8 data bits in a byte
:xmit           rr      xmt_byte        ;moves lsb into c
                call    xmt
                djnz    bit_ctr,:xmit   ;Not 8 bits yet ? Send next bit
                clrb    c
                call    xmt
                inc     sindex          ;increment sindex
                cjbe    sindex,#SENSOR_TOT,:goe ;go to next sensor
mov sindex,#0
                jmp     :goesync  ;start over

xmt             movb    TX,c
                call    halfperiod
                movb    TX,/c
                call    halfperiod
                ret

halfperiod      mov     delay_ctr,#42
:delay_loop     nop
                djnz    delay_ctr,:delay_loop
                ret

period          mov     delay_ctr,#83
:delay_loop     nop
                djnz    delay_ctr,:delay_loop
                ret

adcsample       setb    go_done         ;Start conversion
:not_done       jnb     go_done,:gotit  ;Poll for 0 (done)
                jmp :not_done
:gotit mov     xmt_byte,adres  ;Move ADC result into xmt_byte
;                mov     xmt_byte,#01000001b    ;debug
clrb ADIF ;not sure if this helps ?
                ret

littledally mov delay_ctr,#50 ;this should be about
:little_loop nop
djnz delay_ctr,:little_loop
ret

muxsele         mov     sindexo,sindex  ;copy sindex to sinexo for manipulation
                mov     c,#0            ;dummy bit to insert into c
                rr      sindexo         ;bit 0 of sindex goes to c
                movb    MUXA,c          ;assign bit 0 of sindex to MUXC
                rr      sindexo         ;bit 1 of sindex goes to c
```

```
movb    MUXB,c          ;assign bit 0 of sindex to MUXB
rr      sindexo         ;bit 2 of sindex goes to c
movb    MUXC,c          ;assign bit 0 of sindex to MUXA
ret

nop
nop
nop
```

Appendix C

Demodulator MATLAB code

```
function [retval] = process(filename,startnum,len)
% (<filename>,<start-data-point>,<length>)
%
% writeoutfile('verpcam4.snd',2455,600)

endnum = startnum + len; %end data-point of this sequence

fidr = fopen(filename,'r'); % filehandle open

x = fread(fidr,(endnum),'short'); % x = all data points
y = x(startnum:endnum); % y = our data points of interest
% ------ begin condition the signal (rp)
n = 10;
z = conv([1:n]*0+1,y)/n;
nice = y-z((n/2):(len+n/2)); % nice is nicer than y (closer to bits)
figure(1);
plot(nice(1:len),'b');
title('nice');
legend('nice');
%retval = nice;
% ------- end conditioning the signal
status = fclose(fidr); % close filehandle

use = nice;
nicemax = max(nice); % nice maxvalue
nicemin = min(nice); % nice minvalue
nicemean = mean(nice); % nice meanvalue
nicemedian = median(nice); % nice medianvalue
nicestd = std(nice); % nice standard deviation

if abs(nicemin) > abs(nicemax)
nicemax = abs(nicemin); % nice maxvalue (confirmed)
end

nicenorm = nice./nicemax; % normalized nice values (between 0 and 1)

for i = 1:len
if nicenorm(i) > 0.0
nicebits(i) = 1; % nice --> nicebits (either ON or OFF)
```

```
elseif nicenorm(i) <= 0.0;
nicebits(i) = 0;
end
end
nicebitsplot = nicebits .* .2; % nicebitsplot (easier to see / for plotting)

% -------------- plot nicebitsplot -------------
figure(2)
plot(nicenorm, 'g');
hold on;
plot(nicebits .* .2,'r');
hold off;
title('nicebitsplot');
legend('nicenorm','nicebits');

nicederiv(1:len) = zeros(size(1:len)); % nicederiv(x) is the derivative of
for i = 1:len % nice // the time rate of change
  if i > 1 & i < len-1
nicederiv(i) = (nicenorm(i+1)-nicenorm(i-1))/2.0;
  end
end

badcount = 1; % count of bad areas
badsize = 0; % duration of bad area
flagstart = 0; % flag first point
flagend = 0; % flag last point
toggle = 0;

% here, we are finding all the BAD data points (ie. "non bit"s)
% these data points COULD account for wrongful bit presence - ie. if
% the "nice" is hovering around zero flipping often there will be excess
% bits for each zero-skipping
for i = 1:len
if (abs(nicenorm(i)) < 0.4) & (abs(nicederiv(i)) < 0.3) % NON-bit
% nicenorm < 0.4 - bits are gonna be either high or low,
% not zero. nicederiv > 0.3 when quick transitions are happening
% ie. when bits are happening.
stat(i) = -0.2; % these are not bits
if i > 2 % just so we're not bugging with the start
if flagstart == 0
flagstart = i;
flagstartarray(badcount) = flagstart; % bad bits (starting points)
toggle = 1;
elseif toggle == 1
badsize = badsize + 1;
end
end
else % this IS a bit
stat(i) = 0.2; % these are bits
if i > 2 % don't waste time with the first couple data points
if toggle == 1 % if we are in the middle of a NON-bit
toggle = 0; % Non-bit is done
flagend = i-1; % end of non bit
flagsizearray(badcount) = flagend - flagstart;
flagendarray(badcount) = flagend; % array of non bits
badcount = badcount + 1;
flagstart = 0; % reset this
end
end
end
end


% ----------- plot nicederiv and nicebitsplot ----------
```

```
figure(3);
%plot(nicenorm,'r');
%hold on;
plot(nicederiv,'g');
hold on;
plot(nicebitsplot,'b');
plot(stat,'c');
hold off;
title('nicebits');
legend('nicederiv','nicebitsplot');

% now we'll determine which of the alleged non-bit areas are indeed
% true non-bit areas and which are just 'week' bits
count = 1;
for i = 1:length(flagsizearray)
if flagsizearray(i) > 10 % if the non-bit zone is longer than
% 10 data points we'll believe it's not a bit
nobitsstart(count) = flagstartarray(i);
nobitsend(count) = flagendarray(i);
count = count + 1;
end
end

nobitsstart
nobitsend

lennba = length(nobitsstart); %length of nobitarray
count = 1;
for i = 1:lennba-1
wordsize(count) = nobitsstart(i+1) - nobitsend(i);
count = count + 1;
end

wordsize

% the avg wordlength is 563 points
% (563 points / 6 bytes) (1 byte / 10 bits) = 9.3833 points / bit
bitlensequence = [4 5 5 4 5 5];
bitlenseqctr = 1;

do = 1;
ictr = 1;
halfbitctr = 1;
nobitsstart(length(nobitsstart)+1) = nobitsend(length(nobitsstart)) + 563
lengthnobitsend = length(nobitsend) %debugging echo print
while (ictr <= length(nobitsend)),
j = nobitsend(ictr);
go = 0;
while ((j <= nobitsstart(ictr+1)) & j<=length(nicebits)),%do until end of this word/frame
   if go == 0
if nicebits(j) == 1 % false start
j = j+1;
elseif nicebits(j) == 0
go = 1;
end
% 'go equals 0'
   elseif go == 1
thehalfbits(halfbitctr) = nicebits(j);
      halfbitctr = halfbitctr + 1;
j = j + bitlensequence(bitlenseqctr);
if j == 7
j = 1;
end
% 'go equals 1'
```

```
   end
end
      ictr = ictr + 1;
end

lengththehalfbits = length(thehalfbits)



bitctr = 1;
for i = 2:2:length(thehalfbits)
thebits(bitctr) = thehalfbits(i);
bitctr = bitctr + 1;
end

lengththebits = length(thebits)
fidw = fopen('bits.out','W');
for wi = 1:10:length(thebits)
% wi
fprintf(fidw,'%d %d %d %d %d %d %d %d %d %d\n',thebits(wi:wi+9));
end
status = fclose(fidw);

% thebits

%flagsizearray
%flagstartarray
%flagendarray
%nicebits
%wordsize
```

Appendix D

Kalman filter MATLAB code

```
function [retval] = setkal
%

DTR = pi / 180; % degrees to radians conversion

dt = 1/90;
dtdata = 1/180;

[realrate,realangle] = testdata(dtdata,DTR);

figure(1);
plot(realrate,'b');
hold on;
plot(realangle,'r');
hold off;
legend('realrate','realangle');
title('actual state');

% x: state vector x = [angle rate]'
% y: observations y = [gyro]'

% A: system update matrix x(k+1) = A x(k) + eta  <-- noise
A = eye(2);
A(1,2) = dt;

eta = [0.45*dt 0.45]'; % state noise / fluctuations
% eta = [0 .11]';

% B: observation matrix y(k) = B x(k) + epsilon  <-- noise

B = [0 1];

epsilon = [0.45];

% Q: error covariance matrix for random forcing function
Q = zeros(2);
Q(1,1) = eta(1).^2;
Q(2,2) = eta(2).^2;
```

```
% R: error covariance matrix for measurements
R = zeros(1);
R(1,1) = epsilon(1).^2;

A
eta
B
epsilon
Q
R

% initial estimate for state x(t|t-1)  and  error P(t|t-1)
k = 1;
x_est = [0 0]';
P_est = Q;

for k = 1:90
  % estimate the new observable y(k|k-1) = B x(k|k-1)
  y_est = B * x_est;

  % measure a new value for the observable
  dataindex = lookupindex(k,dt,dtdata);
  y_meas = realrate(dataindex);

  % compute the kalman gain matrix
  K = P_est * B' * (B * P_est * B' + R)^(-1);
% note: B could be variable based on measurements

  % estimate the new state
  x_estolde = x_est;
  x_est = x_estolde + K * (y_meas - y_est);

  % update the error matrix
  P_olde = P_est;
  P = (1 - K*B) * P_olde;

  %predict the new state
  x_pred = A * x_est;

  %predict the new error
  P_pred = A * P * A' + Q;

  % update varnames for next iteration
  x_est = x_pred;
  P_est = P_pred;

kalgainang(k) = K(1);
kalgainrate(k) = K(2);
angpred(k) = x_pred(1);
ratepred(k) = x_pred(2);

end

figure(2)
plot(ratepred,'b')
hold on;
plot(angpred,'r')
hold off
legend('predicted rate','predicted angle')
title('estimates')

figure(3)
plot(ratepred-realrate(1:2:180),'b')
hold on
```

```
plot(angpred-realangle(1:2:180),'r')
hold off
legend('rate error','angle error')
title('errors')

figure(4)
plot(kalgainrate,'b')
hold on;
plot(kalgainang,'r')
hold off
legend('kal gain matrix rate','kal gain matrix angle')
title('kalman gain matrix')

function[dataindex] = lookupindex(k,dt,dtdata)
% find appropriate mock-observable index
ourtime = k .* dt;
dataindex = round(ourtime / dtdata);


function[realrate,realangle] = testdata(dtdata,DTR)
% function to generate test data

realanglelast = 0;
dtdata = 1/180;

for i = 1:180 % t = i/180
realrate(i) = (sin((i-1) * DTR))^2;
realangle(i) = realanglelast + realrate(i).*dtdata;
realanglelast = realangle(i);
end
```

Appendix E

Hummingbird Demo Code

The following three files are source code from a Microsoft Visual C/C++ demo application for the Hummingbird. They create an OpenGL "avatar" of the hummingbird which mimics the motions of the Hummingbird in the real world. The three files are named glwin.c, incon.c, and incon95.c. They can be accessed and downloaded via the World Wide Web from the Hummingbird home page http://physics.www.media.mit.edu/projects/hummingbird.

E.1 glwin.c

```
// incon95.c           "inertial controller dot c"
// Chris Verplaetse, MIT Media Lab
//
// original OpenGL code from Ed Boyden's "Practical
// Physicist's Guide to OpenGL". Thanks Ed !

#include <windows.h>
#include <windowsx.h>

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#include <GL\gl.h>
#include <GL\glu.h>
#include <GL\glaux.h>

#include "GLWin.h"

LRESULT CALLBACK WndProc (HWND, UINT, UINT, LONG);
void CreateNewGLWindow(HWND hwnd);
HWND CreateGLWindow(int glxpos, int glypos, int glxsize, int glysize, HINSTANCE hInstance);
HWND GLwnd;
HANDLE hInst;
float radius=7.0,yaw=0.0,pitch=70.0,roll=70.0;
void polarView(GLfloat radius, GLfloat yaw, GLfloat pitch, GLfloat roll);
```

```
float Function(Vector x);
void Plot();
LRESULT CALLBACK GLWndProc (HWND hwnd, UINT message, UINT wParam, LONG
lParam);


#if defined(SECOND_WINMAIN)
int WINAPI WinMain(HANDLE hInstance, HANDLE hPrevInst, LPSTR lpszCmdParam,
 int nCmdShow)
{
static char sAppClassName[]="GLWin";
HWND hwnd;
MSG msg;
WNDCLASSEX wndclass;

  wndclass.style         = CS_HREDRAW | CS_VREDRAW;
  wndclass.cbSize  = sizeof(WNDCLASSEX);
  wndclass.lpfnWndProc   = WndProc ;
  wndclass.cbClsExtra    = 0 ;
  wndclass.cbWndExtra    = 0 ;
  wndclass.hInstance     = hInstance ;
  wndclass.hIcon         = LoadIcon (NULL, IDI_WINLOGO);
  wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
  //wndclass.hbrBackground = GetStockObject (BLACK_BRUSH) ;
  wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName  = NULL;
  wndclass.lpszClassName = sAppClassName;

  if (!RegisterClassEx(&wndclass)) return 0;

       hwnd = CreateWindow (sAppClassName,// window class name
    "GLWin",// window caption
    WS_OVERLAPPEDWINDOW,// window style
    CW_USEDEFAULT,            // initial x position
    CW_USEDEFAULT,                 // initial y position
    CW_USEDEFAULT,                // initial x size
    CW_USEDEFAULT,                 // initial y size
    HWND_DESKTOP,// parent window handle
    NULL,// window menu handle -- NULL to use class menu
    hInstance,// program instance handle
    NULL) ; // creation parameters
 hInst=hInstance;

     ShowWindow (hwnd, nCmdShow);
 UpdateWindow (hwnd);

 while (GetMessage (&msg, NULL, 0, 0))
     {
  TranslateMessage (&msg) ;
DispatchMessage (&msg) ;}
     return msg.wParam;
}
#endif

void CreateNewGLWindow(HWND hwnd)
{
int width=100, length=120;
HDC hdc;
    GLfloat maxObjectSize, aspect;
    GLdouble near_plane, far_plane;
    static GLfloat ambientProperties[] = {0.95, 0.95, 0.95, 1.0};
    static GLfloat diffuseProperties[] = {0.8, 0.8, 0.8, 1.0};
    static GLfloat specularProperties[] = {1.0, 1.0, 1.0, 1.0};
static GLfloat light_position0[] = {1.0, 1.0, 1.0, 1.0};
static GLfloat  pinkAmbient[] = {.95, .65, .65, 1.0};

    near_plane = 1.0;
    far_plane = 200.0;
    maxObjectSize = 3.0;
    radius = near_plane + maxObjectSize/2.0;

    pitch = 70.0;
    roll = -40.0;

// this was CreateGLWindow
GLwnd=CreateGLWindow(250, 250, 2*width,2*length, hInst);
hdc = GetDC(GLwnd);
//Enter custom GL code here
//Initialization

    glClearColor( 1.0, 1.0, 1.0, 1.0 );
    glClearDepth( 1.0 );
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
glLightfv( GL_LIGHT0, GL_AMBIENT, ambientProperties);
glLightfv( GL_LIGHT0, GL_DIFFUSE, diffuseProperties);
glLightfv( GL_LIGHT0, GL_SPECULAR, specularProperties);
glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, 1.0);
glLightfv(GL_LIGHT0, GL_POSITION, light_position0);
glEnable( GL_LIGHT0 );
    glMatrixMode( GL_PROJECTION );
glLoadIdentity();
    aspect = (GLfloat) width/(GLfloat) length;
    gluPerspective( 45.0, aspect, near_plane, far_plane);
    glMatrixMode( GL_MODELVIEW);
glLoadIdentity();
}

HWND CreateGLWindow(int glxpos, int glypos, int glxsize, int glysize,
HINSTANCE hInstance)
{
       //Creates GLWindow of desired size, and yields
       char  GLName[]="GLName";
       HWND hGLwnd;
       HDC hdc;
       HGLRC hglrc;
       WNDCLASS GLwndclass;
   extern LRESULT CALLBACK WndProc (HWND, UINT, UINT, LONG);
       PIXELFORMATDESCRIPTOR pfd = {
sizeof(PIXELFORMATDESCRIPTOR),   // size of this pfd
1,                     // version number
PFD_DRAW_TO_WINDOW |   // support window
PFD_SUPPORT_OPENGL |   // support OpenGL
PFD_DOUBLEBUFFER,      // double buffered
PFD_TYPE_RGBA,         // RGBA type
24,                    // 24-bit color depth
0, 0, 0, 0, 0, 0,      // color bits ignored
0,                     // no alpha buffer
0,                     // shift bit ignored
0,                     // no accumulation buffer
0, 0, 0, 0,            // accum bits ignored
32,                    // 32-bit z-buffer
0,                     // no stencil buffer
0,                     // no auxiliary buffer
PFD_MAIN_PLANE,        // main layer
0,                     // reserved
0, 0, 0                // layer masks ignored
};
```

```
    int iPixelFormat;

      GLwndclass.style    = CS_HREDRAW | CS_VREDRAW;
  GLwndclass.lpfnWndProc   = WndProc;
  GLwndclass.cbClsExtra    = 0 ;
  GLwndclass.cbWndExtra    = 0 ;
  GLwndclass.hInstance     = hInstance;
  GLwndclass.hIcon         = LoadIcon (NULL, IDI_WINLOGO);
  GLwndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
  GLwndclass.hbrBackground = GetStockObject (BLACK_BRUSH) ;
  GLwndclass.lpszMenuName  = "";
  GLwndclass.lpszClassName = GLName ;

     RegisterClass(&GLwndclass);

 hGLwnd = CreateWindow (GLName,
 "",
 WS_OVERLAPPEDWINDOW | WS_VISIBLE | WS_CLIPSIBLINGS,
 glxpos,
 glypos,
 glxsize,
 glysize,
 HWND_DESKTOP,
 NULL,
 hInstance,
 NULL);

hdc=GetDC(hGLwnd);
iPixelFormat=ChoosePixelFormat(hdc, &pfd);
SetPixelFormat(hdc, iPixelFormat, &pfd);
hglrc = wglCreateContext (hdc);
wglMakeCurrent (hdc, hglrc);
return hGLwnd;
}

void polarView(GLfloat radius, GLfloat yaw, GLfloat pitch,
       GLfloat roll)
{
    glTranslated(0.0, 0.0, -radius);
    glRotated( -yaw, 0.0, 1.0, 0.0 );
    glRotated( -pitch, 1.0, 0.0, 0.0);
    glRotated( -roll, 0.0, 0.0, 1.0);
}

float Function(Vector x)
{
return  x[2]-x[0]+x[1];
}


void Pointer()
{
static GLfloat blueAmbient[] = {0.0, 0.0, 1.0, 1.0},
redAmbient[] = {1.0, 0.0, 0.0, 1.0},
greenAmbient[] = {0.0, 1.0, 0.0, 1.0};

// polarView(radius,yaw,pitch,roll);
auxWireCube(2.0);
glPushAttrib(GL_LIGHTING_BIT);
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, blueAmbient);
auxSolidTorus(0.25, 1.5);
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, redAmbient);
auxSolidCone(.75, 2.0);
// auxWireTeapot(.75);
// PlotFunctionGL(Function,-1.0,1.0,-1.0,1.0,-1.0,1.0,50,50);
glPopAttrib();
}

void Camera()
{
static GLfloat blue[] = {0.0, 0.0, 1.0, 1.0},
red[] = {1.0, 0.0, 0.0, 1.0},
black[] = {0.0, 0.0, 0.0, 1.0},
green[] = {0.0, 1.0, 0.0, 1.0},
beige[] = {0.85, 0.85, 0.80, 1.0},
darkblue[] = {0.0, 0.0, 0.40, 1.0},
darkgray[] = {0.25, 0.25, 0.25, 1.0};

glPushAttrib(GL_LIGHTING_BIT);
glRotated (90.0, 0.0, 0.0, 1.0);
//glRotated (4.0, 20.0, 20.0, 1.0);


glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT, black);
//auxWireBox (2.0, 4.0, 2.0);
auxSolidBox (2.0, 4.0, 2.0);

glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT, darkgray);
glTranslated (0.0, -3.0, -0.35);
//auxWireCylinder (0.5, 1.0);
auxSolidCylinder (0.5, 1.0);
glTranslated (0.0, 1.0, 0.0);

glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT, blue);
glTranslated (0.5, 0.0, 0.45);
auxWireSphere (0.1);

glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT, red);
glTranslated (0.0, 0.0, -0.15);
auxWireSphere (0.1);

glPopAttrib();
}

void globe_view();

void Plot()
{
HDC hdc;

hdc = GetDC(GLwnd);

glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glPushMatrix();
globe_view();
Camera ();
glPopMatrix();
SwapBuffers(hdc);
}
```

E.2 incon.c

```
// incon.c          "inertial controller dot c"
// Chris Verplaetse, MIT Media Lab
//


#include <windows.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <mmsystem.h>

//#include "fish95.h"
//#include "status3d.h"

/**************************************************************************
 *
 * Takes a window and sets up a timer proc to go with a given amount
 * of time indicated by Speed (in Milliseconds) to indicate when the
 * next timer is supposed to get fired off.  The way Windows works is
 * that if this routine is called with the same window handler and the
 * same id number (in our case always hard coded), the timer
 * will be reset to the new interval
 *
 *
 **************************************************************************/

UINT     TimerID;
#define MAX_LOOP 4000
#define NUM_SENSORS 6

char *hCommDev;
unsigned char inbuf[50];


void ShowError(int err, char* name) {
MessageBox(NULL, name, "Error!", MB_ICONSTOP);
}

// Sets up the fish. Causes bugs when used directly, though
int INCOM_SET(HWND hwnd) {
int err, is_err;
// char * commstring;
DCB dcb;
COMMTIMEOUTS CommTimeOuts;

DWORD     N_Bytes_Written;
BOOL    WErr;
OVERLAPPED WriteOs;

is_err = 0;

hCommDev = CreateFile("COM1", GENERIC_READ | GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, 0, NULL);

// set baud rate etc
err = BuildCommDCB("COM1: baud=9600 parity=N data=8 stop=1", &dcb);
if (err < 0) {
is_err += err;
ShowError(err, "BuildCommDCB");
```

```
}


is_err=0;

dcb.fOutxCtsFlow=FALSE;
    dcb.fOutxDsrFlow=FALSE;
    dcb.fDtrControl=DTR_CONTROL_DISABLE;
    dcb.fDsrSensitivity=FALSE;
dcb.fNull=FALSE;
    err = SetCommState(hCommDev,&dcb);
    if (err < 0) {
        is_err += err;
        ShowError(err, "SetCommState");
        }
// Set IO buffers
SetupComm(hCommDev, 4096, 4096);
// Set Timeouts
CommTimeOuts.ReadIntervalTimeout = 100;
    CommTimeOuts.ReadTotalTimeoutMultiplier = 10;
    CommTimeOuts.ReadTotalTimeoutConstant = 10;
    CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
    CommTimeOuts.WriteTotalTimeoutConstant = 5000;
SetCommTimeouts(hCommDev, &CommTimeOuts);

TimerID = SetTimer(hwnd, 1, 5, NULL);
if (!TimerID) {
MessageBox(NULL, "I couldn't create the fish timer!", "I'm sorry, but...", MB_ICONSTOP);
exit(-1);
}

    WriteOs.hEvent=(NULL,TRUE,FALSE,NULL);
WErr=WriteFile(hCommDev, "R", 1,&N_Bytes_Written, &WriteOs);
if (WErr==FALSE) {
err=GetLastError();
ShowError(err,"Serial Port Write Error");
return -1;
}

if (is_err<0) {
     MessageBox(NULL, "Serial port errors.", "Notice!", MB_ICONSTOP);
     exit(-1);
}
}


int READ_UPDATE(void) {
int     err, num_in, loop_count;
    int     done;
    int     tries;
DWORD     N_Bytes_Read, N_Bytes_Written;
BOOL    WErr, RErr;
OVERLAPPED ReadOs, WriteOs;

ReadOs.hEvent=(NULL,TRUE,FALSE,NULL);
WriteOs.hEvent=(NULL,TRUE,FALSE,NULL);

    tries = 0; done = 0;
while ((!done) && (tries < 10)){
num_in = 0;
loop_count = 0;

// Here we're going to twiddle until the sync byte comes in (FF).
do {
```

```
RErr=ReadFile(hCommDev, inbuf, 1, &N_Bytes_Read, &ReadOs);
} while (inbuf[0] != 0xff);

while ((num_in < NUM_SENSORS) && (loop_count < MAX_LOOP)) {
loop_count++;
RErr=ReadFile(hCommDev, inbuf+num_in,NUM_SENSORS,&N_Bytes_Read, &ReadOs);
num_in+=N_Bytes_Read;
     if (RErr==FALSE) {
err=GetLastError();
ShowError(err,"ReadFile Error");
return -1;
}
}
if (num_in >= NUM_SENSORS) done = 1;

tries++;
WErr=WriteFile(hCommDev, "R", 1,&N_Bytes_Written, &WriteOs);
      if (WErr==FALSE) {
err=GetLastError();
ShowError(err,"WriteFile Error");
return -1;
}
}
if (!done) {
      ShowError(err, "Serial port read timed out");
      return -1;
    }
return(N_Bytes_Read);
}



/*
 * gets the current fish levels
 * Should have already issued a write command

int read_fish(void)
{
    int     err, num_in, loop_count;
    int     done;
    int     tries;
DWORD     N_Bytes_Read, N_Bytes_Written;
BOOL    WErr, RErr;
OVERLAPPED ReadOs, WriteOs;

ReadOs.hEvent=(NULL,TRUE,FALSE,NULL);
WriteOs.hEvent=(NULL,TRUE,FALSE,NULL);

    tries = 0; done = 0;
while ((!done) && (tries < 10)){
num_in = 0;
loop_count = 0;
while ((num_in < NUM_SENSORS) && (loop_count < MAX_LOOP)) {
loop_count++;
RErr=ReadFile(hCommDev, inbuf+num_in,NUM_CHARS,&N_Bytes_Read, &ReadOs);
num_in+=N_Bytes_Read;
     if (RErr==FALSE) {
err=GetLastError();
ShowError(err,"ReadFile Error");
return -1;
}
}
if (num_in >= NUM_CHARS) done = 1;
```

```
tries++;
WErr=WriteFile(hCommDev, "R", 1,&N_Bytes_Written, &WriteOs);
       if (WErr==FALSE) {
    err=GetLastError();
            ShowError(err,"WriteFile Error");
            return -1;
}
    }
    if (!done) {
      ShowError(err, "Serial port read timed out");
      return -1;
    }
return(N_Bytes_Read);
}

*/
```

E.3 incon95.c

```
// incon95.c           "inertial controller dot c"
// Chris Verplaetse, MIT Media Lab
//
//               contributions by Rehmi Post and
//                                Ed Boyden III

 /*
This will be a first attempt at establishing serial communications
with the inertial controller (IMU)  "dof box"
----Chris Verplaetse July 14

This code now contains a simple [orientation] motion model and an openGL
graphics representation of the DOF box
----Chris Verplaetse Rehmi Post July 17
*/

#include<windows.h>
#include<stdio.h>
#include<string.h>
#include<time.h>
#include<commctrl.h>
#include<math.h>
#include "GLWin.h"

#define NUM_SENSORS 6
#define PI 3.1415926

LRESULT CALLBACK WindowFunc(HWND, UINT, WPARAM, LPARAM);

int INCOM_SET(HWND);
int READ_UPDATE(void);
char szWinName[] = "WinClock"; // name of window class //
char str[260] = ""; // holds output string //
unsigned char inbuf[50];
int X = 1, Y = 10; // screen location //
int wawa = 0;
float xpos, ypos, zpos; // translational dof coordinates
double xang=0.0, yang=0.0, zang=0.0; // rotational dof coordinates
float xangrate, yangrate, zangrate;
```

```
double gxav, gyav, gzav;
double axav, ayav, azav;
float accx=0.0, accy=0.0, accz=0.0;
float gyrx, gyry, gyrz;
double axdif, aydif, azdif, gxdif, gydif, gzdif; //
double action; // use this for "zuvting" (zero velocity updates)

int initializing_motion_est = 20;

int calibrating_x = 0;
int calibrating_y = 0;
int calibrating_z = 0;
float x_scale = 1.0;
float y_scale = 1.0;
float z_scale = 1.0;
float x_temp, y_temp, z_temp;

double alpha, beta, gamma;
double ix=-1.0, iy=0.0, iz=0.0; // the i^ unit vectroid
double kx=0.0, ky=0.0, kz=1.0; // the k^ unit vectron
double l1, l2, l3, m1, m2, m3, n1, n2, n3;
double xtemp, ytemp, ztemp;
double gmag;

int calibrating_ax = 0;
int calibrating_ay = 0;
int calibrating_az = 0;
double axmin = 200.0, axmax = 10.0,
aymin = 200.0, aymax = 10.0,
azmin = 200.0, azmax = 10.0;
double axzg = 0.0, ayzg = 0.0, azzg = 0.0; // zero-g values of accelerometers
double axsens = 1.0, aysens = 1.0, azsens = 1.0;
double inorm, jnorm, knorm;

float bufav[8];
float num_samples = 10.0;

int WINAPI WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst,
   LPSTR lpszArgs, int nWinMode) {
HWND hwnd;
HWND hwnd2;
MSG msg;
WNDCLASS wcl;


// define a window class //
wcl.hInstance = hThisInst; // handle to this instance //
wcl.lpszClassName = szWinName; // window class name //
wcl.lpfnWndProc = WindowFunc; //window funtion //
wcl.style = 0; // default style //

wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION); // icon style //
wcl.hCursor = LoadCursor(NULL, IDC_ARROW); // cursor style //
wcl.lpszMenuName = NULL; // no menu //

wcl.cbClsExtra = 0;
wcl.cbWndExtra = 0;

// make the window background white //
wcl.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);

// register the window class //
if (!RegisterClass (&wcl)) return 0;
```

```
// now that a window class has been registered, a window may //
// be created //
hwnd = CreateWindow(
szWinName, //name of window class //
"DOF Box - Development Window",// title //
WS_OVERLAPPEDWINDOW, // window style - normal //
CW_USEDEFAULT, // x coorinate - let Windows decide //
CW_USEDEFAULT, // y coorinate - let Windows decide //
//CW_USEDEFAULT, // width - let Windows decide //
//CW_USEDEFAULT, // height- let Windows decide //
1000,
1000,
HWND_DESKTOP, // no parent window //
NULL, // no menu //
hThisInst, // handle of this instance of the program //
NULL // no additional arguments //
);

// attempt to set up communications with serial port //
INCOM_SET(hwnd);
gxav=115.0, gyav=115.0, gzav=122.0;
axav= 82.9, ayav=102.2, azav=143.0;
//MessageBox(hwnd, "incom_set done", "INCOM_SET done", MB_OK);

// display the window //
ShowWindow(hwnd, nWinMode);
UpdateWindow(hwnd);

UpdateWindow(hwnd);

#ifdef WAWAWA
// this will be our text output window
hwnd2 = CreateWindow(
szWinName, //name of window class //
"DOF Box - Developement Window",// title //
WS_OVERLAPPEDWINDOW, // window style - normal //
CW_USEDEFAULT, // x coorinate - let Windows decide //
CW_USEDEFAULT, // y coorinate - let Windows decide //
//CW_USEDEFAULT, // width - let Windows decide //
//CW_USEDEFAULT, // height- let Windows decide //
1000,
1000,
HWND_DESKTOP, // no parent window //
NULL, // no menu //
hThisInst, // handle of this instance of the program //
NULL // no additional arguments //
);

ShowWindow(hwnd2, nWinMode);
UpdateWindow(hwnd2);
//
#endif WAWAWA




// start a timer - interupt once per millisecond //
SetTimer(hwnd, 1, 1, NULL);

// create the message loop //

while(GetMessage(&msg, NULL, 0, 0)) {
```

```
TranslateMessage(&msg); // allow use of keyboard //
DispatchMessage(&msg); // return control to windows //
}

KillTimer(hwnd, 1); // stop the timer //

return msg.wParam;
}

int calibrating_status = 1;

#ifdef WAWA

void calibrate_accelerometers () {

if (calibrating_status == 1) {
if (accx < axmin) axmin = accx;
if (accy < aymin) aymin = accy;
if (accz < azmin) azmin = accz;

if (accx > axmax) axmax = accx;
if (accy > aymax) aymax = accy;
if (accz > azmax) azmax = accz;

calibrating_status = 1;
}
}

#endif


void motion_est (unsigned char *inbuf)
{
//static float gxav=115, gyav=116, gzav=121;
static float w_new, w_last;
double axlast, aylast, azlast;

axlast = accx;
aylast = accy;
azlast = accz;

accx = bufav[0] / num_samples;
gyry = bufav[1] / num_samples;
accy = bufav[2] / num_samples;
gyrz = bufav[3] / num_samples;
accz = bufav[4] / num_samples;
gyrx = bufav[5] / num_samples;

if (initializing_motion_est > 0) {
w_new = .10;
w_last = .90;
initializing_motion_est--;
} else {
w_new = 0.001;
w_last = 0.999;
}
gxav = w_new*gyrx + w_last*gxav;
gyav = w_new*gyry + w_last*gyav;
gzav = w_new*gyrz + w_last*gzav;
axav = w_new*axav + w_last*axav;
ayav = w_new*ayav + w_last*ayav;
azav = w_new*azav + w_last*azav;

axdif = accx - axlast;
```

```
aydif = accy - aylast;
azdif = accz - azlast;
gxdif = gyrx - gxav;
gydif = gyry - gyav;
gzdif = gyrz - gzav;


action = sqrt(axdif*axdif + aydif*aydif + azdif*azdif + gxdif*gxdif + gydif*gydif + gzdif*gzdif);

//xangrate = (gyrx - gxav) / .09; // x angle rate in deg/sec
//yangrate = (gyry - gyav) / .09; // y angle rate in deg/sec
//zangrate = (gyrz - gzav) / .09; // z angle rate in deg/sec

xangrate = gxdif / .09; // x angle rate in deg/sec
yangrate = gydif / .09; // y angle rate in deg/sec
zangrate = gzdif / .09; // z angle rate in deg/sec

xang = xang + (xangrate * 0.055) * x_scale;
yang = yang + (yangrate * 0.055) * y_scale;
zang = zang + (zangrate * 0.055) * z_scale;

alpha = (xangrate * 0.055) * x_scale * PI / 180;
beta = (yangrate * 0.055) * y_scale * PI / 180;
gamma = (zangrate * 0.055) * z_scale * PI / 180;

l1 = cos(sqrt(beta*beta+gamma*gamma));
l2 = -sin(gamma);
l3 = sin(beta);
m1 = sin(gamma);
m2 = cos(sqrt(alpha*alpha+gamma*gamma));
m3 = -sin(alpha);
n1 = -sin(beta);
n2 = sin(alpha);
n3 = cos(sqrt(alpha*alpha+beta*beta));

xtemp = ix;
ytemp = iy;
ztemp = iz;

ix = l1*xtemp + m1*ytemp + n1*ztemp;
iy = l2*xtemp + m2*ytemp + n2*ztemp;
iz = l3*xtemp + m3*ytemp + n3*ztemp;

xtemp = kx;
ytemp = ky;
ztemp = kz;

kx = l1*xtemp + m1*ytemp + n1*ztemp;
ky = l2*xtemp + m2*ytemp + n2*ztemp;
kz = l3*xtemp + m3*ytemp + n3*ztemp;

gmag = sqrt(ix*ix+iy*iy+iz*iz);

inorm = (accx - axzg) / axsens;
jnorm = (accy - ayzg) / aysens;
knorm = (accz - azzg) / azsens;

if (calibrating_ax == 1) {
if (accx < axmin) axmin = accx;
if (accx > axmax) axmax = accx;
if (accy < aymin) aymin = accy;
if (accy > aymax) aymax = accy;
if (accz < azmin) azmin = accz;
if (accz > azmax) azmax = accz;
```

```
}
else if (calibrating_ax == 0) {
axzg = axmin + (axmax-axmin)/2;
ayzg = aymin + (aymax-aymin)/2;
azzg = azmin + (azmax-azmin)/2;

axsens = (axmax - axmin) / 2;
aysens = (aymax - aymin) / 2;
azsens = (azmax - azmin) / 2;
}

}

void globe_view() {
double magnitude;

magnitude = sqrt(ix*ix+iy*iy+iz*iz);
ix = ix / magnitude;
iy = iy / magnitude;
iz = iz / magnitude;

gluLookAt(ix*8.0, iy*8.0, iz*8.0,// eye
0.0, 0.0, 0.0,// center
kx, ky, kz); // up
}

extern float roll, pitch, yaw;
FILE *outfile;
FILE *infile;

void CreateNewGLWindow(HWND);
void Plot();

// this funtion is called by Windows 95 and is passed //
// messages from the message queue //
LRESULT CALLBACK WindowFunc(HWND hwnd, UINT message,
WPARAM wParam, LPARAM lParam) {
HDC hdc;
PAINTSTRUCT paintstruct;
//struct tm *newtime;
//time_t t;
UINT accxlevel, gyrylevel, accylevel, gyrzlevel, acczlevel, gyrxlevel;
char wawas[260];
static int err; // testing

switch(message) {
case WM_CHAR : // process keystroke //
X = Y = 10; // display characters here //
sprintf(str, "%c", (char)wParam); // stringsize character //
switch (wParam) {
case ',':
num_samples --;
if (num_samples < 3)
num_samples = 3;
break;
case '.':
num_samples ++;
if (num_samples > 40)
num_samples = 3;
break;
case 'a':
initializing_motion_est = 100;
break;
case 'r' : // load saved scale values
```

```
infile =  fopen("scales","r");
if (infile == NULL)
MessageBox(hwnd, "The file scales was not opened", "Bad Deal", MB_OK);
else {
fscanf(infile,"%f %f %f", &x_scale, &y_scale, &z_scale);
}
fclose(infile);
break;
case 's' : // save current scale values
outfile = fopen("scales","w+");
if (outfile == NULL)
MessageBox(hwnd, "The file scales was not opened", "Bad Deal", MB_OK);
else {
fprintf(outfile,"%f %f %f", x_scale, y_scale, z_scale);
}
fclose(outfile);
break;
case 'z':
xang = yang = zang = 0;
ix = -1.0;
kz = 1.0;
iy = iz = kx = ky = 0.0;
break;
case '1':
if (calibrating_x) {
x_scale = 90.0 / (xang - x_temp);
calibrating_x = 0;
} else {
x_temp = xang;
x_scale = 1.0;
calibrating_x = 1;
}
break;
case '!': // Gotta finish dealing with this accelerometer - zero g - zufting stuff
if (calibrating_ax) { calibrating_ax = 0; }
else { calibrating_ax = 1; }
break;
case '@' :
calibrating_ax = 2;
break;
case '2':
if (calibrating_y) {
y_scale = 90.0 / (yang - y_temp);
calibrating_y = 0;
} else {
y_temp = yang;
y_scale = 1.0;
calibrating_y = 1;
}
break;
case '3':
if (calibrating_z) {
z_scale = 90.0 / (zang - z_temp);
calibrating_z = 0;
} else {
z_temp = zang;
z_scale = 1.0;
calibrating_z = 1;
}
break;
case 'q':
PostQuitMessage(0);
break;
default:
```

```
InvalidateRect(hwnd, NULL, 1); // paint the screen //
break;
}
break;
case WM_KEYDOWN:
switch (wParam) {
case VK_F1:
CreateNewGLWindow(hwnd);
Plot();
break;
case VK_LEFT:
pitch-=1;
Plot();
break;
}
break;
case WM_PAINT : // process a repaint request //
hdc = BeginPaint(hwnd, &paintstruct); // get dc //
TextOut(hdc, X, Y, str, strlen(str)); // output string //
EndPaint(hwnd, &paintstruct); // release dc //
break;
case WM_TIMER : // timer went off //
{
int i, j;

for (i = 0; i< 6; i++)
bufav[i] = 0.;
for (i=0; i<(int)num_samples; i++) {
err = READ_UPDATE();
for (j = 0; j < 6; j++)
bufav[j] = bufav[j] + inbuf[j];
}
}
inbuf[NUM_SENSORS] = 0;
motion_est(inbuf);


/*
sprintf(wawas, "raw (%6.2f, %6.2f, %6.2f)  avg (%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f)
dif(%6.2f %6.2f %6.2f..%6.2f %6.2f %6.2f) angle(%6.2f, %6.2f, %6.2f)
globe(%6.2lf %6.2lf %6.2lf) %5.2f  action: %6.2f   ",
//gyrx, gyry, gyrz,
accx, accy, accz,
axav, ayav, azav, gxav, gyav, gzav,
axdif, aydif, azdif, gxdif, gydif, gzdif,
xang, yang, zang,
ix, iy, iz, gmag, action);

/*
sprintf(wawas, "  amin(%6.2f, %6.2f, %6.2f) amax(%6.2f, %6.2f, %6.2f)
azg(%6.2f, %6.2f, %6.2f) norm(%6.2f, %6.2f, %6.2f)  mag(%6.2f)          ",
axmin, aymin, azmin,
axmax, aymax, azmax,
axzg, ayzg, azzg,
inorm, jnorm, knorm,
sqrt(inorm*inorm+jnorm*jnorm+knorm*knorm));
*/
sprintf(wawas, "    i(%6.2f %6.2f %6.2f)  k(%6.2f %6.2f %6.2f)    ",
ix, iy, iz,
kx, ky, kz);
strcpy(str, wawas);
InvalidateRect(hwnd, NULL, 0); // update screen //

Plot ();
```

```
accxlevel = 200 - accx;
gyrxlevel = 350 - gyrx;
accylevel = 500 - accy;
gyrylevel = 650 - gyry;
acczlevel = 800 - accz;
gyrzlevel = 950 - gyrz;

wawa++;
hdc = GetDC(hwnd); // get device context //

SetPixel(hdc, wawa, accxlevel, RGB(255,0,0));
SetPixel(hdc, wawa, gyrylevel, RGB(255,100,0));
SetPixel(hdc, wawa, accylevel, RGB(255,0,255));
SetPixel(hdc, wawa, gyrzlevel, RGB(0,100,255));
SetPixel(hdc, wawa, acczlevel, RGB(0,0,255));
SetPixel(hdc, wawa, gyrxlevel, RGB(0,0,0));

ReleaseDC(hwnd, hdc); // release device context //
if (wawa >= 1000) {
wawa = 1; // loop the X graph
InvalidateRect(hwnd, NULL, 0); // update screen //
}
break;
case WM_RBUTTONDOWN : // process right mouse button //
CreateNewGLWindow(hwnd);
Plot();
break;
case WM_LBUTTONDOWN : // process left mouse button //
strcpy(str, "Left Button is Down.");
X = LOWORD(lParam); // set X to current mouse position //
Y = HIWORD(lParam); // set Y to current mouse position //
InvalidateRect(hwnd, NULL, 1); // paint the screen //
break;
case WM_DESTROY : // terminate the program //
PostQuitMessage(0);
break;
default :
// let Windows 95 process any messages not specified //
// in the previous switch statement //
return DefWindowProc(hwnd, message, wParam, lParam);
}
return 0;
}
```

Bibliography

[AP, 1995] Ali Azarbayejani and Alex P. Pentland, *Recursive Estimation of Motion, Structure, and Focal Length*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 17, No. 6, June 1995.

[BB, 1982] Dana H. Ballard and Christopher M. Brown, *Computer Vision*, Prentice-Hall, 1982

[BKE, 1994] N. Barbour, K. Kumar, J.M. Elwell Jr., *Emerging Low(er) Cost Inertial Sensors*, presented at 22nd Joint Service Data Exchange for GN&C, 1994.

[Becker, 1996] Shawn Becker, *Vision-assisted modeling for model-based video representations*, Ph.D. Thesis, Massachusetts Institute of Technology, 1996.

[Boas, 1983] Mary L. Boas, *Mathematical Methods in the Physical Sciences*, John Wiley & Sons, New York, 1982.

[Bove, 1989] V. Michael Bove, Jr., *Synthetic Movies Derived from Multi-Dimensional Image Sensors*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, April 1989.

[BH1992] Robert Brown, Patrick Hwang, *Introduction to Random Signals ans Applied Kalman Filtering*, John Wiley & Sons, Inc. New York, 1992.

[Davenport] Application ideas related to camera-operator-motion-recognition are based on discussions with or ideas of Glorianna Davenport at the MIT Media Laboratory.

[KD, 1996] Personal communiations with Joe Kung and Jim Dosher, both of Analog Devices Inc.

[Elwell, 1991] John Elwell, The Charles Stark Draper Laboratory, Inc. *Progress on Micromechanical Inertial Instruments.* Published by the American Institute of Aeronautics and Astronautics. 1991

[Ferren] Virtual set-related applications are based on discussions with Bran Ferren at Walt Disney Imagineering in Glendale, CA., 1996.

[FB, 91] Richard S. Figliola, Donald E. Beasley. *Theory and Design for Mechanical Measurements.* Wiley. New York. 1991.

[FD, 1994] E. Foxlin and N. Durlach, "An Inertial Head Orientation Tracker With Automatic Drift Compensation Doe Use with HMDs," *Proceedings from VRST '94, Virtual Reality Software and Technology*, Singapore (August 23-26), 1994).

[Gershenfeld, 1995] Neil Gershenfeld, *The Nature of Mathematical Modeling*, to be published , Cambridge University Press, 1997.

[Horn, 1986] Berthold Klaus Paul Horn, *Robot Vision*, The MIT Press, 1986

[Kalman, 1960] R.E. Kalman, Research Institute for Advanced Study, *A New Approach to Linear Filtering and Prediction Problems*, Journal of Basic Engineering, Transactions of the ASME, March 1960.

[Kuo, 1991] Benjamin C. Kuo. *Automatic Control Systems.* Prentice Hall. Englewood Cliffs, New Jersey. 1991.

[HH, 1994] Paul Horowitz and Winfield Hill, *The Art of Electronics*, Cambridge University Press, 1994.

[Lawrence, 1993] Anthony Lawrence, *Modern Inertial Technology.*, Springer-Verlag, New York. 1993.

[Mackenzie, 1990] Donald Mackenzie, *Inventing Accuracy: A Historical Sociology of Nuclear Missile Guidance*, MIT Press, Cambridge, MA, 1990.

[Marrin, 1996] Teresa Marrin, *Toward an Understanding of Musical Gesture: Mapping Expressive Intention with the Digital Baton*, Masters Degree Thesis, Massachusetts Institute of Technology, 1996.

[Massey, 1996] Michael Massey and Walter Bender, *Salient Stills: Process and Practice*, IBM Systems Journal, VOL 35, NOS 3&4, 1996.

[Nakamura, 1990] Takeshi Nakamura, *Vibration Gyroscope Employs Piezoelectric Vibrator*, JEE, September 1990.

[Pinhanez, 1995] Claudio Pinhanez and Aaron Bobick, *Intelligent Studios: Using Computer Vision to Control TV Cameras*, IJCAI'95 Workshop on Entertainment and AI/Alife, April 1995.

[Schwarz, 1976] J.A. Schwarz, "Micro-Navigator (MICRON)," *AGARD Conference Proceedings No. 176 on Medium Accuracy Low Cost Navigation*, AGARD (1976), pp. 6/1-14.

[Sony] Sony Pro-Betacam SP Camcorder, UVW-100K/100PK Operating Instructions Manual.

[SMS, 1994] T. Starner, J. Makhoul, R. Schwartz, and G. Chou, "On-line Cursive Handwriting Recognition Methods," *IEEE Conference on Acoustics, Speech, and Signal Processing*, Adelaide, Australia (April 1994), Vol. V, pp. 125-128.

[Starner, 1996] T. Starner, "Human Powered Wearable Computing," *IBM Systems Journal*, Vol. 35, Nos. 34, 618-629, 1996.

[Szeliski, 1993] Richard Szeliski and Sing Bing Kang, *Recovering 3D Shape and Motion from Image Streams using Non-Linear Least Squares*, DEC Cambridge Research Laboratory, Technical Report Series, March 1993.

[Teodosio, 1992] Laura Teodosio, *Salient Stills*, Master's Thesis, Massachusetts Institute of Technology, June 1992.

[Verplaetse, 1996] Christopher Verplaetse, *Inertial Proprioceptive Devices: Self Motion-Sensing Toys and Tools*, IBM Systems Journal, VOL 35, NOS 3&4, 1996.

[WB, 1990] R.G. Watts and T.A. Bahill, *Keep Your Eye On the Ball: The Science and Folklore of Baseball*, W.H. Freeman and Company, New York, 1990.

[Woodson, 1981] Wesley E. Woodson, *Human Factors Design Handbook.* McGraw-Hill, 1981.