Software architecture for controlling an indoor hovering robot from a remote host

Ambika Asthana
University of Wollongong

SOFTWARE ARCHITECTURE FOR CONTROLLING AN INDOOR HOVERING ROBOT FROM A REMOTE HOST

A thesis submitted in partial fulfilment of the requirements for the aware of the degree

Master of Computer Science – Research

from

University of Wollongong

by

Ambika Asthana

School of Computer Science and Software Engineering

2007

Declaration

I, Ambika Asthana, declare that this thesis, submitted in partial fulfillment of the requirements for the award of Masters of Computer Science (Research), in the Department of Informatics, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

Ambika Asthana
30 August 2007.

Abstract

To achieve stable autonomous control of an indoor flying robot is a challenging proposition in the field of robotics today. Many researchers are inspired by the echolocation of bats and vision of bees and attempt to duplicate this behaviour by using sonar sensors, cameras and onboard microprocessors. This project aims to achieve the same goal but with a different approach. We propose to build a software architecture for controlling a four-rotor helicopter, DraganFlyer, from a host computer. In order to do this, we equipped the DraganFlyer with communication devices, an Inertial Navigation Sensor (INS) and batteries.

The DraganFlyer is a four-rotor helicopter that can hover and move freely in air. Due to the near zero friction and damping at slow velocities it is marginally stable in six degrees of freedom. The aim of the overall research project is to understand the dynamics of the DraganFlyer and hence to achieve hover without drift and trajectory following without wandering. Development of software to achieve this level of control from a remote host poses a significant software design problem.

This thesis focuses on the software design problem, i.e. the design and development of real-time software for measuring the dynamics and for control of the DraganFlyer. The software runs on a host Macintosh and is divided into three main sections. One is the measurement of DraganFlyer motion with an INS. The second is the calculation of the control commands. The third is the control of the DraganFlyer via the radio control handset.

As these sections have different timing requirements, a significant part of the software design and testing time was spent examining how to decompose the system based on timing requirements and constraints. Then we had to determine how to couple these modules together to achieve overall timing goals without data loss.

Two types of experimental results are presented. The first results are to test the software, both the correctness of the calculations and their timeliness. The second are measurements of the open loop response of the DraganFlyer.

Acknowledgements

I would like to thank Phillip McKerrow for his guidance, support, advice and patience while supervising this project. He always believed I could do it!

A warm thank you to my parents and brother for their unfaltering support and encouragement throughout the course of my research.

A special thanks to my friends: Lyris Rodrigues, Darryl Correa, Sheenal Shrivastava and Sangeetha Ramu for always standing by me and supporting me in every way possible.

Also, a big thank you to the Robotics Group for their enthusiasm and ideas!

Contents

1.0 Introduction

1.1 Objectives

Our objective is to develop a software architecture for controlling a hovering robot from a remote host. Approaches to this thesis problem fall into two groups. In one most processing is done in an onboard processor with the host enabling user interaction. In the other most processing is done in the host. In this thesis, we study the design of the latter, within a system with a time constraint of 20msec for real-time feedback. With this approach, the user can have lower-level control of helicopter function than with an onboard system that automates these.

Such a system delivers another advantage; the software architecture is not helicopter-specific. Therefore if at any time this particular helicopter becomes unusable or outdated, the same software can interface with another craft that has an on-board Inertial Navigation Sensor (INS). Whilst, an on-board processor will have memory and processing limitations, a remote host computer will be more powerful and acts as a user interface.

While we see a lot of advantages in this approach, there a few shortcomings that we should address. The set-up of this project will be such that a communication link will be required between the remote host and flying robot. This is explained in detail in Chapters 3 and 4. Radio links between the robot and remote computer may cause interference or timing problems such as time delays. There is no fixed solution to overcome this problem, however a few suggestions might be,

1. Synchronizing the software to adjust to the delays caused
2. Testing with a variety of links to see which gives the best result, i.e. minimum time delay.

These topics are further explained in Chapter 3 and 4.

1.2 Research Methodology

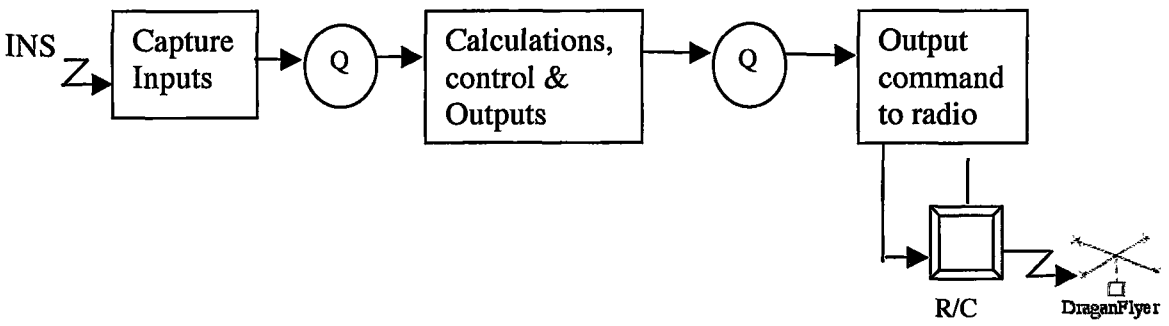

Figure 1.1 Proposed Software Architecture

Figure 1.1 displays the proposed software architecture, which will be developed using LabVIEW 7.0 on a MAC OS X 10.4 operating system. The software shall implement both open-loop and closed-loop control. In order to achieve our goal, we drew-up a set of specifications based on equipment constraints that would guide the design and development of the software. These include:

1. Receive values from an INS every 20msec over a 38KB serial radio link
2. Calculate robot location (xyz and roll, pitch and yaw coordinates) every 20msec
3. Display INS values to user
4. Read user target inputs for throttle, roll, pitch and yaw
5. Calculate output control values using control laws every 20msec
6. Send control values to robot over 19KB serial output to an R/C radio link every 20msec

Chapters 4 and 6 revisit these specifications to design and test several programs. Our aim is to demonstrate that the designed software architecture creates a system that meets the specifications.

1.3 Software Issues

While doing control experiments, some of the issues we anticipated in our software were

1. Timing

    a. Input data arrives from INS every 20msec
    b. Calculates every 20msec
    c. Output transmitted to the DraganFlyer every 20msec, but not synchronized with input

    We can finish calculations within 20msec but we cannot control the timing or synchronization of external hardware. The INS outputs data every 20msec and we try to send output commands at the same rate, but time delays occur often. This implies that we must separate each function based on timing (Figure 1.2). Each process will transfer its data into a queue, the next process will query this queue for data and so on. Hence, data is not lost due to synchronization issues and also processes are not held-up while awaiting values.

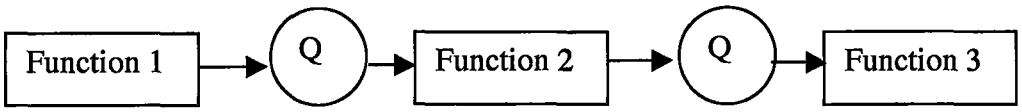

Figure 1.2 Each process transfers data into a queue

2. Performance – each process has to meet certain performance goals so that real-time processing of data is not interrupted. A performance indicator in this scenario could be processing time allocated for every function, a data output set

that is required by the next function, and minimum delay from feedback to response

3. Delays – if delay gets too long from input to output it can cause control instability.

1.4 Literature Review

Milestones in aerial robotics have been achieved in recent years but mainly flying outdoors or limited to lab workbenches. By contrast, aerial robots flying in closed quarters are rarely discussed in the literature.

There is a need to survey or monitor large indoor areas to safe keep stadiums, warehouses, subway tunnels and train stations but this task is human-intensive and time-consuming. Col. John Blitch, director of the September 11th robotic search-and-rescue effort commented at a plenary session that ground-based robots can be used indoors but have "limited mobility and field-of-view" [24].

Indoor

Some indoor flyers are vision based [24] and weigh 10grams with a flying speed of 1.5m/s. Ouchi et al [23] describe a 10-gram microflyer steering autonomously in a textured indoor environment. This has been achieved using fly-inspired sensors and control strategy, which allows it to regulate its own airspeed by means of a small anemometer.

Papers [6,7,9,11] also describe various indoor flying machine designs such as a four-rotor robot, flapping-wing prototype and micro-air vehicles

Control

Researchers have experimented with various control techniques to fly indoor robots. Below we discuss some of them.

- Vision based control – Ostrowski [14] tested the motion of a quadrotor with feedback linearization and back stepping controllers using real-time control and a vision system . Bekey et al [8] displayed the use of a behavioral paradigm to control a craft. This implied that the craft must constantly make sound decisions to maintain its integrity and avoid collisions.

- Image and inertial - Strelow and Singh [10] have described both batch and online algorithms. They concluded that both batch and online estimation can recover accurate motion when both image and inertial measurements are employed.

- Real-time stabilization [13] – Here we saw the first successful real-time control applied to a four-rotor rotorcraft. They proposed a controller based on Lyapunov analysis using a nested saturation algorithm. Their real-time experiments on a

rotorcraft showed that the controller is able to perform autonomously the tasks of taking off, hovering and landing.

- Sensor output feedback [15] - In this work, two types of biologically inspired sensing mechanisms were investigated. The halteres and ocelli. They also developed an attitude control law for the MFI using the output feedback from the halteres and ocelli. Through their work, they have shown that simple schemes (simple sensor architectures and feedback control) can achieve robust global stability.

- Optic flow – In [2] tested an optic flow microsensor for depth perception, which allowed autonomous takeoff and landing and collision avoidance on a closed quadrotor to fly safely and slowly

- Goal-directed - In [12], Fumiya Iida used a synthetic methodology to gain insights into the navigation behavior of bees. Specifically, mechanisms of course stabilization behavior and visually mediated odometer by using a biological model of motion detector for the purpose of long-range goal-directed navigation in 3D environment. The performance tests using a blimp-type flying robot platform showed that the proposed mechanism could be used for goal-directed navigation.

Software Architecture

While most of the above papers have focused on control and gave little regard to the architecture of the software, we see it as a significant component in the successful design of indoor aerial robots.

- Zufferey et al [24] metioned a software for vision-based control. It is a custom-made software that projects an image on the wall and takes an image input, processes it and sends the result to the robot.

- Sastry el al [15] used a software test-bed to simulate the dynamics of the flyer and evaluate control algorithms. They test the performance of their sensor by using the simulator to generate the angular velocities of an insect under hovering conditions similar to those of their flyer. The feedback from the sensor is compared to the simulated dynamics .

Our approach and design are presented in Chapters 4,6 and 7.

1.5 Overview of Thesis

This thesis aims to design a software architecture for the control of an indoor hovering robot from a remote host to implement the structural approach shown in Figure 1.1. Each process is individually identified, designed and tested before continuing to the next stage.

In this chapter, we discussed potential issues, advantages and disadvantages of our methodology. We begin the project by reviewing the model of the DraganFlyer as it was published in, 'Modelling the DraganFlyer four-rotor helicopter', M^{c}Kerrow [1] in

Chapter 2, where torque and control forces are analysed in order to understand the dynamics and kinematics of the robot. Chapter 3 will present a complete functional and technical overview of the Inertial Navigation Sensor (INS), which is a prime tool in our research approach. We then launch into the hardware set-up and software design built for the experiments and calibration of the INS in Chapter 4. In Chapter 5, we record the values for the parameters used in the dynamic and kinematic relationships. The PCBuddy interface and its functionality is described in Chapter 6. The control theory that we will implement and open-loop testing are discussed in Chapter 7. Finally, Chapter 8 concludes this thesis with suggestions of future work and reports on successfully achieved tasks.

2.0 Model of the DraganFlyer[1]

Chapter 2 will discuss the model of the DraganFlyer, its dynamics and kinematics. The structure of this chapter is such that Section 2.1 presents the model of the helicopter. Section 2.2 discusses the forces acting on it. In Section 2.3, we discuss how to manoeuvre the robot so that maximum stability can be achieved while controlling it.

2.1 Model

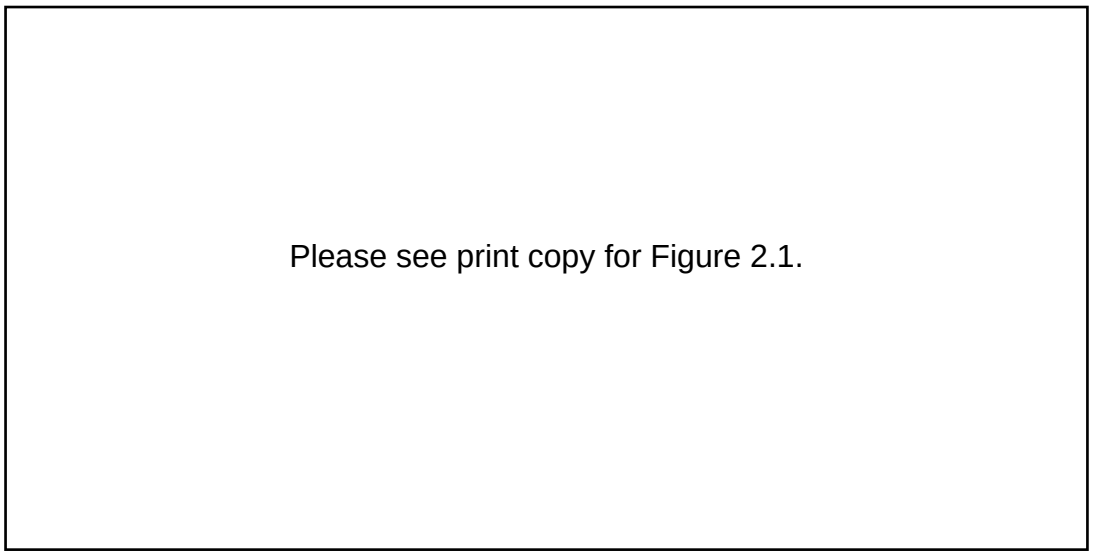

Figure 2.1. Forces acting on the DraganFlyer

The DraganFlyer is modeled as 5 masses. Four masses represent the motors and rotors and are at the vertices of a cross. The fifth mass represents the batteries and electronics and is located below the centre of the robot.
If we consider one of the rotor units, the force balance for each rotor is:

$$f_n = m_n g - f_{mn} \tag{2.1}$$

where:

$n = 0, 1, 2, 3, 4$ are the centre, front, right, rear and left parts
f_n = resultant force vector at rotor n
f_{mn} = force applied by rotor n
m_n = mass of motor n and rotor n

[1] This chapter is direct reference of [1]

2.2 Coordinate Frames

To develop our analysis of the aerodynamics, we use two coordinate frames: a robot frame and a coincident instantaneous frame. The x axis of the robot frame points forward and the z axis points up. The axes of the motors are parallel to the z axis, as shown in Figure 2.1. The robot frame is fixed to the robot and moves with the robot relative to the instantaneous frame. The instantaneous frame is fixed in the world at the current location of the robot frame.

All equations are expressed in the instantaneous frame. When absolute values are required in the world frame they are calculated in the instantaneous frame and transformed to the world frame. Introducing the instantaneous frame results in simpler equations, except that the gravitational vector must be transformed into the instantaneous frame every time it is used.

2.3 Dynamics

A spinning rotor produces a lift force and is subject to a drag force. The lift force is:

$$f_{lift} = \frac{\rho C_l U^2 S}{2} = K_l \omega_{prop}^2 \tag{2.2}$$

where ρ is the fluid density, U is the flow velocity, C_l is the lift coefficient and S = span (b) * chord (c)and the drag force, which is parallel to the direction of blade motion is:

$$f_{drag} = \frac{\rho C_d U^2 S}{2} = K_d \omega_{prop}^2 \tag{2.3}$$

The resultant force applied by the rotor to the robot is:

$$f_{mn} = f_{lift} - f_{drag} = (K_l - K_d)\omega_{prop}^2 \tag{2.4}$$

2.4 Control

The robot is maneuvered by adjusting the force applied by the four rotors. When the forces are equal the robot translates vertically. When they are not, the robot will roll, pitch or yaw before moving in any direction. Therefore we consider the components of the resultant force at the rotors. A force imbalance between rotors 1 and 3 will cause the robot to pitch, and a force imbalance between rotors 2 and 4 will cause it to roll. The components of the resultant force from a rotor due to pitch q and roll f are:

$$f_{nz} = m_n g - f_{mn} * \cos(\theta) * \cos(\phi) \tag{2.5}$$

$$f_{nx} = -f_{mn} \sin\theta \tag{2.6}$$

$$f_{ny} = f_{mn} \sin\phi \tag{2.7}$$

Two components are further modified by yaw y:

$$f_{nx} = f_{nx}\cos\psi - f_{ny}\sin\psi \quad (2.8)$$
$$f_{ny} = f_{ny}\cos\psi + f_{nx}\sin\psi \quad (2.9)$$

We replace the forces produced by each rotor at the end of a leg of the cross by equivalent forces through the centre of gravity (cog) and torques around the cog. The torques due to one rotor are:

$$\tau_{nx} = f_{ny} \times l_0 + f_{nz} \times l_{ny} \quad (2.10)$$
$$\tau_{ny} = -f_{nx} \times l_0 - f_{nz} \times l_{nx} \quad (2.11)$$
$$\tau_{nz} = f_{ny} \times l_{nx} - f_{nx} \times l_{ny} \quad (2.12)$$

Note: all rotors will have one component of length $l_n = 0$. For example, rotor 1 has $l_{ny} = 0$ with a value for l_{nx}, and rotor 2 is the reverse. The total torque in each dimension due to the rotor force is the sum of the torques for the four rotors.

$$\tau_{mx} = \tau_{1x} + \tau_{2x} + \tau_{3x} + \tau_{4x} \quad (2.13)$$
$$\tau_{my} = \tau_{1y} + \tau_{2y} + \tau_{3y} + \tau_{4y} \quad (2.14)$$
$$\tau_{mz} = \tau_{1z} + \tau_{2z} + \tau_{3z} + \tau_{4z} \quad (2.15)$$

Similarly, the force acting though the cog is:

$$f_x = f_{1x} + f_{2x} + f_{3x} + f_{4x} \quad (2.16)$$
$$f_y = f_{1y} + f_{2y} + f_{3y} + f_{4y} \quad (2.17)$$
$$f_z = m_0 g - f_{1z} - f_{2z} - f_{3z} - f_{4z} \quad (2.18)$$

An additional torque occurs around the z-axis due to the torque the motor applies to the rotor to force it to rotate. The front rotor (1) turns in a negative direction around the z - axis forcing air to move in a positive direction along the z-axis. The motor rotates in a positive direction and the gears reverse the direction of rotation of the rotor. So the torque applied by the rotor is negative with respect to the z--axis and the reaction torque from the air opposes it.

It is this torque that is balanced by rotating one pair of rotors in the opposite direction to the other pair. This torque balance between the rotors occurs through the airframe, placing considerable stress on the frame. It is this rotor torque that is used to control yaw when the robot is horizontal. The yaw torque calculated previously results from the robot being at an angle to the horizontal due to roll or pitch action. So the yaw torque equation is modified:

$$\tau_z = \tau_{mz} - \tau_{m1} + \tau_{m2} - \tau_{m3} + \tau_{m4} \quad (2.19)$$

Due to the inertia of the spinning rotors, the DraganFlyer experiences gyroscopic torques when it rolls or pitches. The spin axis of the rotor is parallel to the z-axis (rot z). Roll (rot x) and pitch (rot y) change the direction of the rotor angular momentum. Gyroscopic torque attempts to align the spin axis (z-axis) with the precession axis (x-axis in the case of a roll). The resultant torque is orthogonal to both the spin and precession axes, so a roll around the x-axis coupled with a spin around the z-axis results in gyroscopic torque around the y-axis.

$$\tau_{gy} = I_z \dot{\psi}_z \times \dot{\phi}_x \tag{2.20}$$

$$\tau_{gx} = I_z \dot{\psi}_z \times \dot{\phi}_y \tag{2.21}$$

These torques are subtracted from to the torques from the motors to obtain the total torque:

$$\tau_x = \tau_{mx} + \tau_{gx} \tag{2.22}$$

$$\tau_y = \tau_{my} + \tau_{gy} \tag{2.23}$$

The inertia is the sum of the inertia for the propeller, gears and rotor. For simplicity, we model the propeller as a flat plate:

$$I_{prop} = \frac{m_{prop}\left(b^2 + c^2\right)}{12} \tag{2.24}$$

The inertia of the rotor of the motor, which rotates in the opposite direction to the propeller at a higher speed due to the gear ratio ($g = 9.8066$), is:

$$I_{mot} = \frac{m r_{mot}^2}{2} \tag{2.25}$$

so the total inertia for the motor rotor system is:

$$I_{rotor} = I_{prop} - \frac{I_{mot}}{g} \tag{2.26}$$

Another source of torque is centripetal force, which arises from rotary motion. When a mass is rotating around an axis, the magnitude of the velocity may be constant but the direction of the velocity vector is continuously changing. The cause of this angular acceleration is centripetal force. These centripetal forces are directed toward the centre of rotation. Hence, the centripetal force for yaw is directed toward the z-axis of the robot, while the centripetal force for roll is directed toward the x-axis. As the robot is symmetrical with a rigid airframe, these forces cancel one another out through the airframe, placing stress on the airframe during high-speed rotations.

3.0 Inertial Navigation Sensor[2]

The Inertial Navigation Sensor (EiMU), manufactured by CSIRO, is an instrument for measuring acceleration, rate, magnetometer, temperature, roll, pitch and heading.

Inertial Sensors provide high frequency data with short-term accuracy, so they should be used in conjunction with an external sensor to provide low frequency data with long-term accuracy.

The output of inertial sensors profoundly depends on the environmental factors in which the application finds itself (i.e. the platform vibration and ambient temperature).

The EIMU's micro-controller processes data every 20msec. The EIMU contains two dual-axis accelerometer chips. This means that there are actually four accelerometers measuring the three Cartesian axes (x, y and z). The chips are mounted so that the z-axis accelerations are measured by two accelerometers.

Data is sent from the EIMU in the form of a packet over a serial port. Packet configuration and type are described in detail in Tables 4.3 and Table 4.4.

The INS initially transmitted data to the computer using a USB port. But, now the EIMU has been mounted on the DraganFlyer, Bluetooth serial device transmits information from the DraganFlyer to the host machine.

3.1 Physical Implementations of INS

Originally inertial navigation technology used stable platforms, which are still commonly used, particularly for ships and submarines where highly accurate unaided navigation is required over long periods. At the core of these systems is a platform on which inertial sensors are mounted. This platform is not sensitive to the rotational motion of the vehicle, and uses a number of gimbals arranged to provide at least three degrees of rotational freedom, to minimise the coupling between the vehicle and the platform. The movement of these three gimbals is controlled by motors, which are activated by information provided by gyroscopes.

Three gyroscopes inside the inertial sensor spin in different directions on axles. The axles are placed so that they form 90-degree angles with each other, like three edges of a box meeting at a corner. The axles keep their directions as long as the gyroscopes continue to spin. Each gyroscope is supported by gimbals (movable frames) so that it stays in position as the vehicle rolls, pitches, or yaws. Together, the gyroscopes establish an inertial reference system (stable set of lines). The accelerometers detect changes in the vehicle's motion in reference to the stable lines defined by the gyroscopes

[2] This chapter is a direct reference from Farzad Salim's project report, 'Testing the Embedded Inertial Measurement Unit', June 2004. It is included with minor revision in order to enable the reader to understand the design of the software in Chapter 4.

Many modern INSs are configured in what is described as a 'strapdown' mode; the inertial sensors are mounted directly to the vehicle and hence are not isolated from its angular motion. The signals produced by the inertial sensors are resolved mathematically using angular measurements from rate gyroscopes in a computer before the actual calculation of navigational information. This use of a computer to establish and resolve the inertial data reduces the mechanical complexity of the INS and frequently reduces the cost and size of the system.

In a strapdown system the sensors go through the full effects of vehicle motion. So for these systems a higher bandwidth and dynamic rage is required. This higher dynamic range can affect the stability of scale factor terms, and may also introduce larger non-linearity errors. High bandwidth implies the sensor produces noisier data. On the other hand the gimballed arrangement requires minimal computational processing since the platform is maintained to correspond with the navigation frame. In comparison a strapdown arrangement needs to compute and resolve terms relating to the body, navigation and inertial frames.

3.2 How Inertial Navigation System works

A typical INS consists of three main components. *Gyroscopes* measure the rate of rotation along the given axes. *Accelerometers* measure the changes in speed and direction. These measurements are sent to a *computer* that uses them to calculate the vehicle's velocity, heading, attitude, and finally to derive the vehicle's position.

An INS continually measures changes in a vehicle's speed, attitude and direction, and sends the information to a computer. The computer then calculates the effect of changes on the current model to update the position and direction of the vehicle based on its starting point.

It should also be noted that an accelerometer provides a measure of non-gravitational acceleration, which a vehicle experiences, not the absolute acceleration. This is mainly because the 'proof' mass of the accelerometer and the vehicle is subject to the same gravitational field. Thus in order to determine the acceleration with respect to the chosen reference frame, it is necessary to incorporate a model of the gravitational force which the vehicle may be subjected to during its mission.

Knowledge of the velocity and the position of a vehicle are gathered from the continuous data received from the accelerometers and gyros that are mounted on the host vehicle. Accelerometers are normally parallel to the vehicle's body axes and their output signals may be resolved in the navigation reference frame prior to their integration. Gyros, on the other hand provide a direct measurement of the angular rotation of the vehicle. A further integration process may then deduce the attitude of the vehicle.

In a typical INS, three single-axis gyroscopes or two dual-axis gyroscopes are used to provide the necessary attitude information. A summary of the tasks that an INS must perform in order to accomplish the navigation function is given in below.

- Use gyroscopic sensor to determine the angular motion of a vehicle, from which its attitude relative to a reference frame may be derived.
- Measure specific force using accelerometers.
- Resolve the specific force measurements into the reference frame using the knowledge of attitude derived from the information provided by the gyroscopes
- Have access to a function representing the gravitational field-the gravitational attraction of the Earth in the case of systems operating in the vicinity of the earth.
- Integrate the resolved specific force measurements to obtain estimates of the velocity and position of the vehicle.

3.3 Advantages and Disadvantages of Inertial Navigation

The advantages and disadvantages of INS can be explained by introducing an example from autonomous robot navigation. For the robot to reach its destination, it must (1) move in the right direction and (2) cover the right distance. Without inertial guidance, the vehicle has to rely on odometry, data signals that are transmitted from radio beacons, or other sensors to detect landmarks. Using that information, with methods, such as triangulation or multiplication of the robot speed with the time it has been moving in that direction, deduce the position of the vehicle. However with the use of INS, the robot only needs to consult the information that is produced by using inertial sensors. It can approximate its way despite poor visibility or faulty data from odometers or range sensors, and the absence of landmarks.
The main advantages of using an INS are:

- Self contained
- Non-radiating
- High sample rate
- Relatively cheap
- No extra information from environment required

However, despite these advantages there is one major disadvantage in using inertial sensors – exponential error growth due to the integration of the data relative to the starting position. This means a small error at any stage of position or attitude estimation can lead to a very large drift after a short period of time. Implementing error models to reduce these errors is one of the open areas of research in the field.

There are many errors that cause inaccuracies in the navigation calculations. Some of these are more frequent and need more attention and prevention plans.

The list below is a summarized description of such errors:

- **Bias:** When there is no input, and the signal is not zero; it is usually referred to as the offset value caused by manufacturing imperfections. Bias is also known as "zero offset".
- **Scale factor:** The scale factor is the ratio between a change in the output signal in relation to the changes in input. As most sensors provide an output signal that is directly proportional to input, the scale factor is a single number, defined as the slope of the best straight line that illustrates such a relation (Figure 3.1). The scale factor K is: $K = S/I$ (S = ouput signal and I = true input)
- **Non-linearity Errors:** The scale factor may not be exactly linear but may have second – or higher-order terms. To understand this we usually run a test in which the acceleration or rotation rate is varied one by one, over the full acceleration range, and the sensor output is measured.

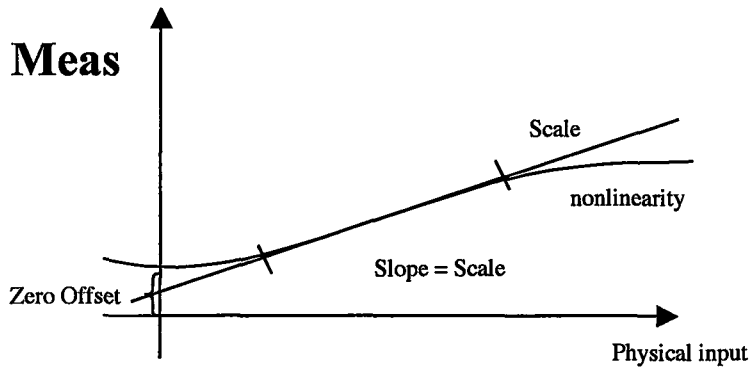

Figure 3.1 Non-linearity of scaling factor

- **Vibration:** Vibration has a negative effect on the performance of an INS. This is mainly due to the low bandwidth of low cost inertial sensors that cause motion rejection in high vibrations. Vibration is usually the combination of angular and linear motions. In low cost inertial sensors, the gyro's bandwidths are less than the accelerometer's, so vibration that results from a small change in attitude will be measured by accelerometers as a component of gravity. This acceleration component will be evaluated as the vehicle motion because the gyros sensitivity was not enough to detect this change that is related to attitude.
- **Temperature Drift:** Even when an inertial sensor is attached to a stable base, its output will wander a bit due to disturbances inside the sensor. Such disturbances come from the sensitivity of sensors (gyro) to heat and temperature that causes the drift in their output.

3.4 Reasons for calibration

Observablity analysis (Park et al., 1995) shows that the states related to the position and heading of a robot that employs dead-reckoning system (INS) is unobservable. That means that dead-reckoning systems have the inevitable characteristics of divergence. Therefore, errors should be deducted before they affect the navigation system.

A typical INS system that is assembled from low-cost solid-state components is almost always constructed in a "strap-down" configuration (Scheding S, Nebot E.M, 1998). This means that gyro and accelerometers are fixed to a common chassis. While this has the advantage of eliminating the moving parts, the strap-down means that substantially more complex software is needed to compute and distinguish true linear acceleration from angular acceleration and body roll or pitch with respect to gravity.

Alignment of the platform to the navigation axes is an initial step in inertial navigation. If the platform is stationary one can assume that the accelerometers will only measure the acceleration due to gravity. Then the attitude of the platform is determined with the accelerometers sensitive to the x and y directions. However the gyro and accelerometers usually drift with time and temperature. This bias must also be estimated by the alignment filter (Barshan, 1994), (Sukkarieh S. & Nebot E.,1999).

3.5 Gyroscope

Gyros are also sensitive to thermal drift. The standard deviation of the noise is also an important specification of most gyros that are used in low-cost inertial sensors. Due to the integration necessary to obtain a heading, a random walk angle will also be added to the output of the gyro. According to Scheding S & Nebot E.M, (1998), this angle is directly proportional to the standard deviation of the noise multiplied by the square root of time.

Non-linearity in gyro measurement is another type of error that results from errors in measured rotations. This is especially prominent when the gyro is rotated and returned to the zero position. Nebot E. (1992) in his experiments with two types of gyros, "BAE" and "Andrews" found, typical accumulated errors for a 90^0 excursion of these two gyros, are 0.8^0 and 1.5^0.

The attitude of the platform can be obtained by integrating the measurements of roll, pitch, and yaw angles from gyroscopes. So the orientation angles must be estimated very accurately as even a small fraction of gravitational acceleration attributed to true linear acceleration results in huge errors.

3.6 Accelerometer

Measurements made by accelerometers are the sum of effects due to linear acceleration, orientation of the platform with respect to the gravity vector and angular acceleration of the platform as it moves around a curve.

In an experiment conducted by Scheding and Nebot (1998), they recorded the output of an accelerometer that was mounted on a vehicle. Figure 3.2 shows the same experiment done on the DraganFlyer, while it is hanging in the air on the end of a tether signal output of that accelerometer with the sensitive axis in the x direction. As this figure illustrates, there are three distinct parts. Section 1, shows acceleration measured while the vehicle is stationary with the rotors stopped. Almost no noise is present in the signal. Then the rotors are turned to give some lift and the vibration is measured by accelerometer. Finally the vehicle is pitched and it starts to move and the accelerometers measure vibration, acceleration due to tilt with respect to gravity, angular and linear acceleration.

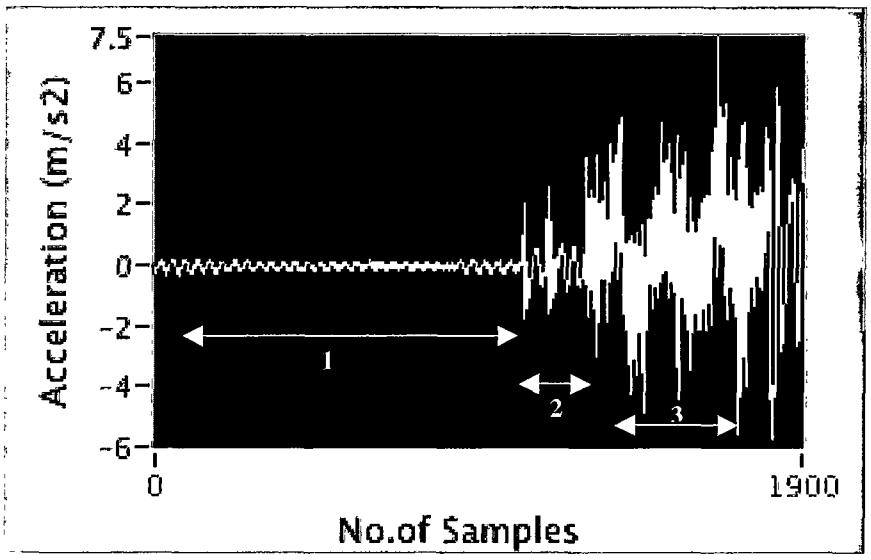

Figure 3.2 Acceleration measures along y axes

Scheding and Nebot (1998) concluded that in order to use the accelerometer to predict position it is necessary to filter out acceleration due to the vibration of the vehicle. Second, which is the most important, due to the small acceleration of the vehicle, the attitude of the platform must be obtained with high precision in order to compensate for the effect of gravitational acceleration.

4.0 INS Measurement Software

In Chapter 3 we described the Inertial Navigation Sensor, which is the main sensor in our experiments. This chapter will discuss the methodology implemented to calibrate sensor readings before INS values are utilized to measure model parameters and for feedback in the control loop. We describe the software that processes physical values from the sensor and a set of experiments with the INS mounted on the DraganFlyer to help carry out successful calibration.

The structure of this chapter is such that Section 4.1 discusses the hardware set-up for these experiments. Section 4.2 presents the range, output modes, types and values from the INS. Section 4.3 is a detailed description of the steps involved in constructing correct and efficient software to complement the real-time hardware technology being used in this project. We then, in Section 4.4, discuss the mathematical formulae applied to extracted data from the INS in order to convert it into parameter values. Calibration experimental design and set-up is described in Section 4.5. Section 4.6 graphically illustrates the results of these experiments and the limitations.

4.1 Hardware Set-up

4.1.1 Hardware

Sets of instruments have been used to complete the communication link between the DraganFlyer and the machine that hosts the software. At the moment the aerial vehicle is flown with a radio controller. The initial model of the DraganFlyer, Figure 4.1, has been modified to accommodate instruments that help to accomplish the task of communication.

Figure 4.1 Model of DraganFlyer without INS and bluetooth device

This communication is important for the software to be able to receive samples from the sensor. Every sample is a data packet that contains acceleration, gyro rate, gyro angle and magnetometer values. Below is a diagram that presents that functional set-up between the DraganFlyer and software.

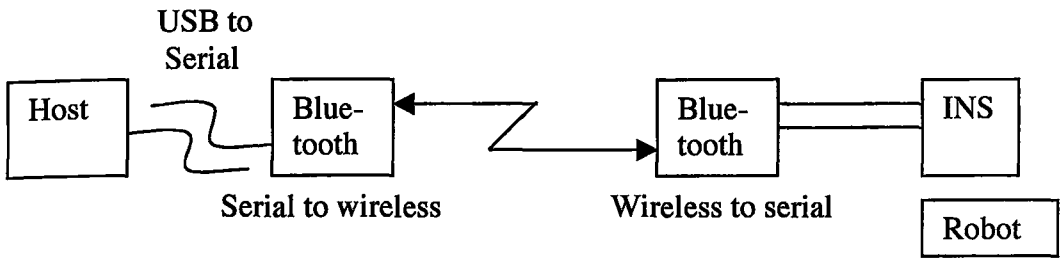

Figure 4.2 Block diagram of physical set-up

The modified DraganFlyer has an INS mounted on it and a Bluetooth device (Figure 4.4) that provides the radio communication link. This device also has an error correction protocol and joins the serial to wireless. The other part of the Bluetooth device is connected to the USB port of the computer through a serial link. This device is bi-directional so it is used to send signals from the INS sensor to the computer through a USB port. The part connected to the computer is not only used to receive signals but also to send signals to the INS. The bluetooth device has a range of 5meters. Often, when the sensor does not respond to automatic commands, a user can manually send commands to the sensor through the software.

Another important item are the batteries being used in the project. There are two types of batteries being used, NICAD (Nickel Cadmium) and Lithium Polymer batteries (Figure 4.3). Both these batteries differ in mass and have advantages/disadvantages. The mass factor is of importance in our project because we are dealing with a flying object and the payload is of much importance, thus since the Lithium battery is lighter (Table 5.2), it helps to reduce the weight on the DraganFlyer and provides space for any future additions to the vehicle.

4.1.2 Advantages of Lithium Batteries

- Last longer
- Lighter in weight

4.1.3 Disadvantages of Lithium Batteries

- Have to be more careful while handling them (charging/re-charging)
- Cannot let the batteries discharge below 3V per cell. There are 3 cells in the Lithium battery we are using.

Figure 4.3 NICAD and Lithium Batteries

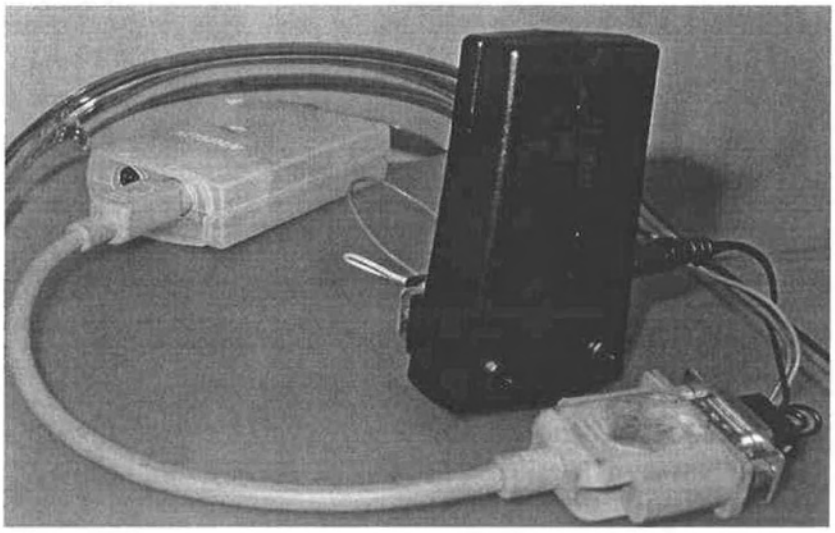

Figure 4.4 Bluetooth device connected to host machine

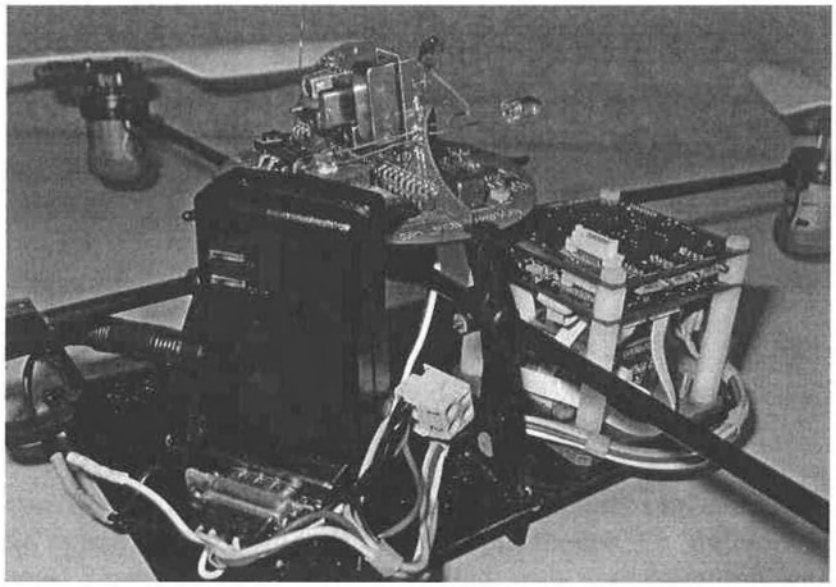

Figure 4.5 DraganFlyer after modifications

4.2 INS Output

The data measured by the INS is raw data within the range of its respective sensor chip, as shown in Table 4.1.

Device	Range
Accelerometer	2G
Rate gyro	100deg/sec
Magnetometer	2gauss

Table 4.1: Sensor Chips in EiMU

Commands to the EiMU are sent in the form of a single byte from a host machine [18]. Some commands have an argument, which takes the form of a second byte. Command packets sent to the EiMU do not have a header or checksum. Table 4.2 shows the packet format of two common commands.

Command	Command argument	Reply	Description
M	<x>	Data Packet	Put EiMU into timed continuous output mode. The value of <x> (0 to 255) is in units of tens of milliseconds
Z	<x>	Z	Put EiMU into ZEROING state – performs a rate bias calculation over interval <x>.

Table 4.2: Serial commands sent to EiMU

4.2.1 Data Output Modes

The EiMU can be set-up (via the RS232 serial interface) to output data in three different ways:

- Polled mode – In this mode the EiMU is normal silent, i.e. no data is output unless it is requested. A poll command will result in a single data packet being sent out by the EiMU
- Continuous Mode – In this mode the EiMU will continuously output data packets, i.e. every 20msec.
- Timed Continuous mode – In this mode the EiMU will continuously output data packets at the requested rate.

In our project we use the RS232 serial interface and continuous mode to output data.
We spent time experimenting with all three output modes, i.e. polled, continuous and timed continuous. Since our application is based on real- time scenarios, we required fast, continuous, calibrated data without manually requesting it every few milliseconds. We

found that it took EiMU too long to respond to a poll. Continuous mode fits this criterion perfectly and outputs data every 20msec.

4.2.2 Data Output types

The EiMU can be set-up (via the RS232 serial interface) to output one of the four types of data:

- Raw – Raw 10-bit sensor data, temperature and time
- Calibrated – 'Calibrated' sensor data, temperature and time. The sensor values have been scaled and calibrated in some way and can be easily converted into engineering units
- Absolute angle – Internally calculated roll, pitch and heading angles are output, in addition to calibrated sensor data, temperature and time.
- Angle only – Internally calculated roll, pitch and heading angles are output along with time.

The calibrated output type best suits our application because we need to further use output INS values in mathematical formulae.

4.2.3 Packet Types and Structure

Data from the EiMU takes the form of a packet starting with a header and ending in a checksum. The EiMU supports non-robust and robust serial data packets:

- Non-robust packets begin with a single byte header (255) and end with a single byte checksum, Table 4.4.
- Robust packets begin with a two byte header (255 followed by 254) and end in a two byte checksum, Table 4.3.

The EiMU can be put into the robust and non-robust serial modes using the 'k' and 'l' commands respectively.

Byte meaning	Byte Number
Header (255)	0
Header (254)	1
Roll rate (MSB)	2
Roll rate (LSB)	3
Pitch rate (MSB)	4
Pitch rate (LSB)	5
Yaw rate (MSB)	6
Yaw rate (LSB)	7
X accel (MSB)	8
X accel (LSB)	9
Y accel (MSB)	10
Y accel (LSB)	11
Z accel (MSB)	12
Z accel (LSB)	13
X mag (MSB)	14
X mag (LSB)	15
Y mag (MSB)	16
Y mag (LSB)	17
Z mag (MSB)	18
Z mag (LSB)	19
Temp (MSB)	20
Temp (LSB)	21
Time (MSB)	22
Time (LSB)	23
Checksum (MSB)	24
Checksum (LSB)	25

Table 4.3: Robust serial data packet formats for both raw and calibrated data

Byte meaning	Byte Number
Header (255)	0
Roll rate (MSB)	1
Roll rate (LSB)	2
Pitch rate (MSB)	3
Pitch rate (LSB)	4
Yaw rate (MSB)	5
Yaw rate (LSB)	6
X accel (MSB)	7
X accel (LSB)	8
Y accel (MSB)	9
Y accel (LSB)	10
Z accel (MSB)	11
Z accel (LSB)	12
X mag (MSB)	13
X mag (LSB)	14
Y mag (MSB)	15
Y mag (LSB)	16
Z mag (MSB)	17
Z mag (LSB)	18
Temp (MSB)	19
Temp (LSB)	20
Time (MSB)	21
Time (LSB)	22
Checksum	23

Table 4.4: Non-robust serial data packet formats for both raw and calibrated data

Non-robust (single byte)
The single byte checksum is calculated as follows:

1. Sum all the bytes in the packet (except the header byte and ofcourse the checksum byte).
2. Divide the result of the above summing by 256.
3. The remainder of this division is the checksum value

Robust (dual byte)
The dual byte checksum is calculated as shown in Algorithm 4.1. Note that the two-byte header is NOT included in the checksum, only the body of the packet is used.

```
unsigned short checksum16(unsigned char      *pkt_body, int pkt_body_len)  {
        unsigned short crc16 = 0, i;

        for(i=0;i<pkt_body_len;i++) {
                crc16 = (unsigned char)(crc16 >> 8)  | (crc16 << 8);
                crc16 ^= pkt_body[i];
                crc16 ^= (unsigned char)(crc16 & 0xff) >> 4;
                crc16 ^= (crc16 << 8) << 4;
                crc16 ^= ((crc16 & 0xff) << 4) << 1;
        }

        return crc16;
}
```

Algorithm 4.1. Dual byte checksum

4.3 Software

The software, 'EiMU Control Software' (ECS), extracts and processes data from the INS and comprises of various complex LabVIEW libraries that integrate to provide a closed loop functionality for calibration. We experimented with eight versions before the most suitable one, i.e. with continuous, calibrated data output was selected.

Below is a description of all developed versions and their different functionality

Version 1.0 – This version is the original software (Figure 4.6) built by Farzad Salim, a research student in Robotics Lab in a previous project. The focus of the following work is to rewrite this software to remove bugs and achieve the desired performance and control. It stimulates a virtual continuous mode by repetitively sending commands for data output in polled mode, thus replicating the continuous mode. You can select Calibrated or raw output. Loop time of sending-receiving data is measured and used in further calculation of distance and acceleration.

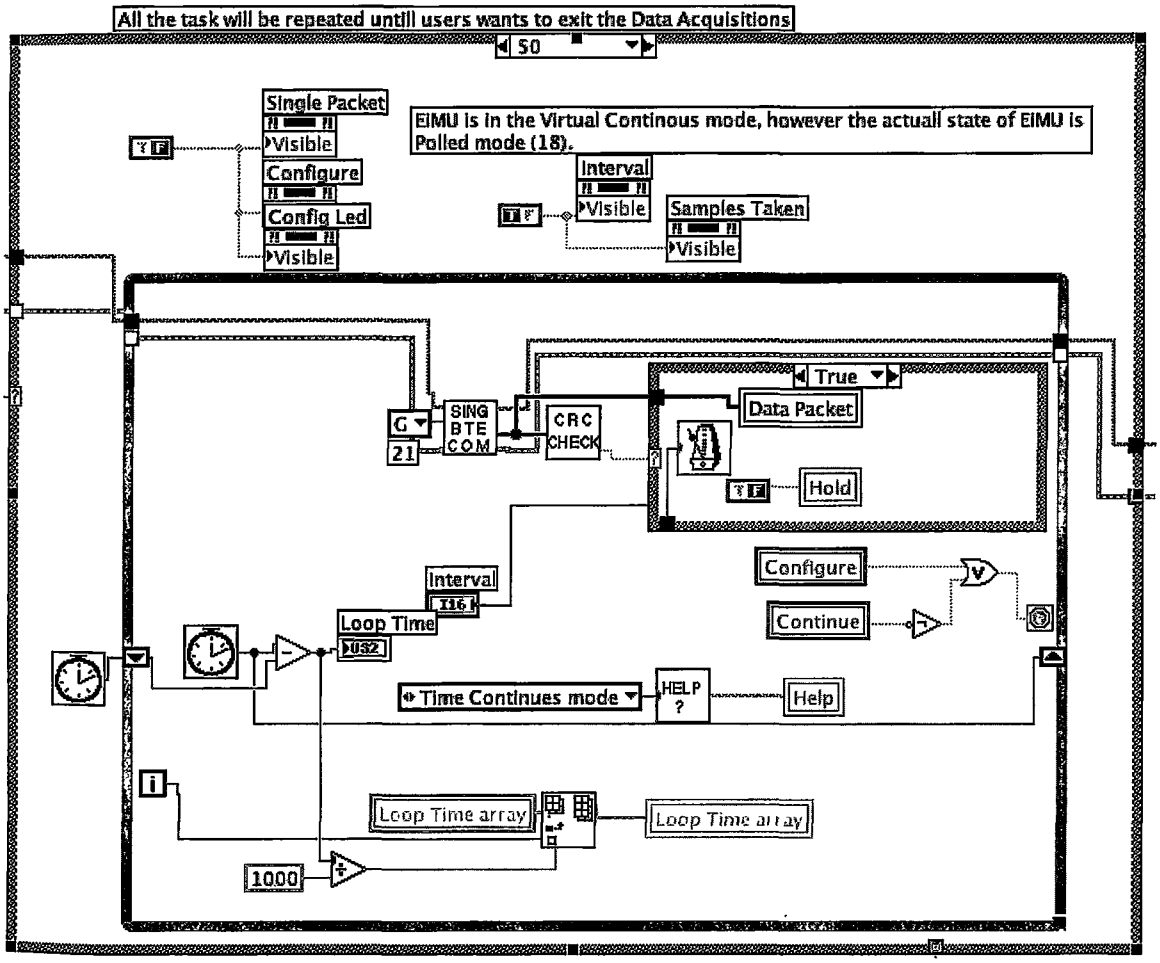

Figure 4.6 LabVIEW code of ECS Version1.0

The block diagram of the ECS vi in Figure 4.2 shows the flow of data from left to right. The outer square is an 'if statement' that selects their mode of operation (virtual continuous). The rounded square is a while loop that executes until an external switch (continue global value) is depressed to stop it. The loop calls two sub vis (SING BTE COM and CR CHECK) to read data packets, which are then written to a global (Data Packets inside inner IF block) for use by calculation vis. The loop time is measured to give a time value between samples for use in calculations.

Version 2.0 – As the time between samples was too short, the original software was then modified (Figure 4.7) to experiment with different output modes. Version 2.0 uses continuous mode with calibrated data output. An array that imitates the functionality of a queue (inside inner IF block) is used to store data. A timer is used to measure the time of the processing loop. This time is then used as sampling time in further calculations. Sampling time is defined as the time taken to generate a new sample by the INS. Therefore it is the time between 2 consecutive samples.

In Continuous mode we send a command (C) once and then use a 'Visa Read' built-in LabVIEW functionality to read data continuously being sent by the INS (inside the EiMU READ.vi).

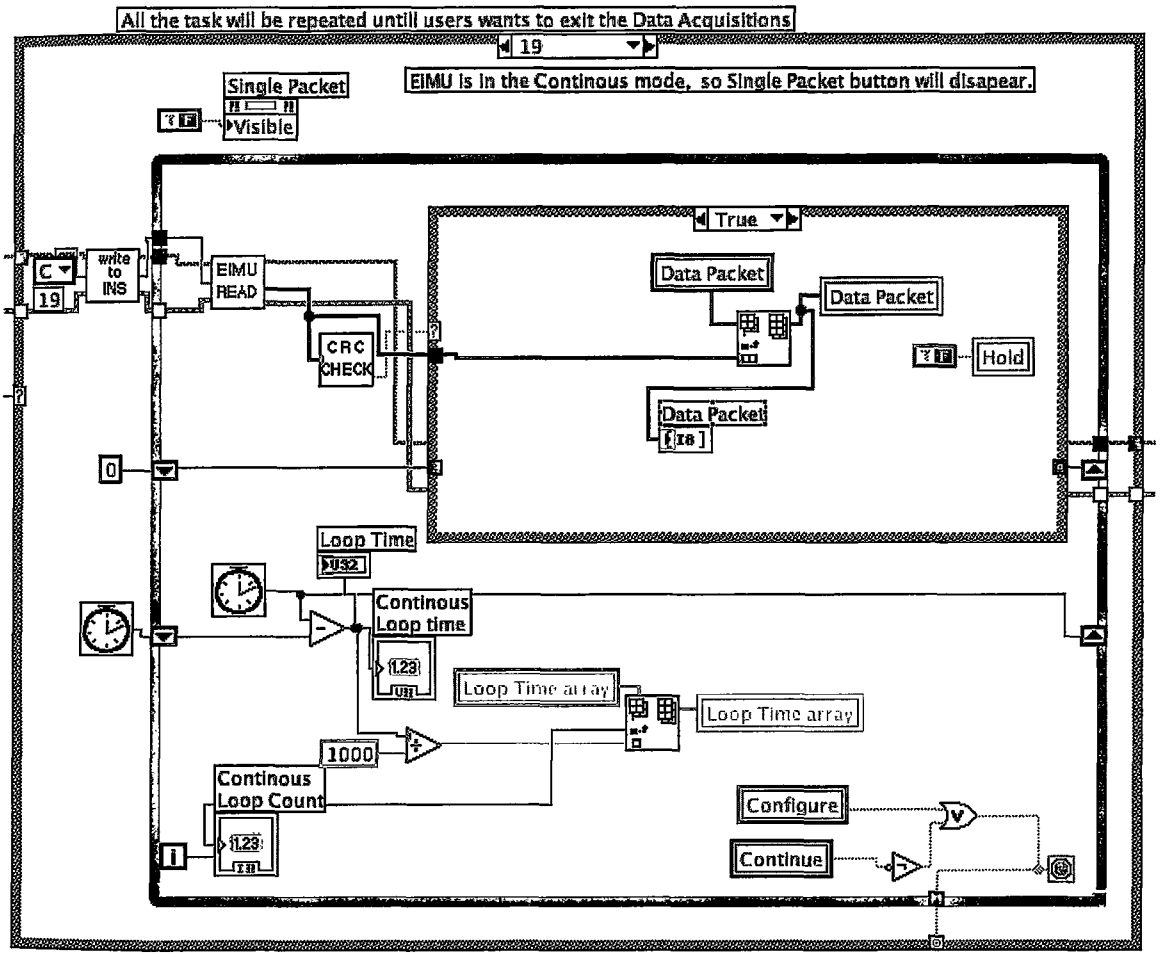

Figure 4.7 ECS Version 2.0 where loop time is used for calculation

Here we found that the time between consecutive samples was still too large. As the EiMU can output a new packet every 20msec, we wanted to improve the software to achieve this data rate. The control of the DraganFlyer will have to be updated at least 50 times to achieve reasonable control.

Version 3.0 – The only difference between Version 3.0 (Figure 4.8) and Version 2.0 is that here we removed all the calculations required after the data has been read from the INS. We did this to discover the reason for time-delay issues and whether it was because the calculations were taking too long or that our artificial queue was not functioning as efficiently as expected.

Another difference is that in this version we did not read the incoming data within the loop. It was read from the self-built VI called, 'SinByteComm'. This VI sends the command and receives INS response to commands. However, doing this caused a logical error because the continuous command was being sent over and over again instead of being sent only once. When this fault was realized, we reverted to Version 2.0 and continued further programming on it.

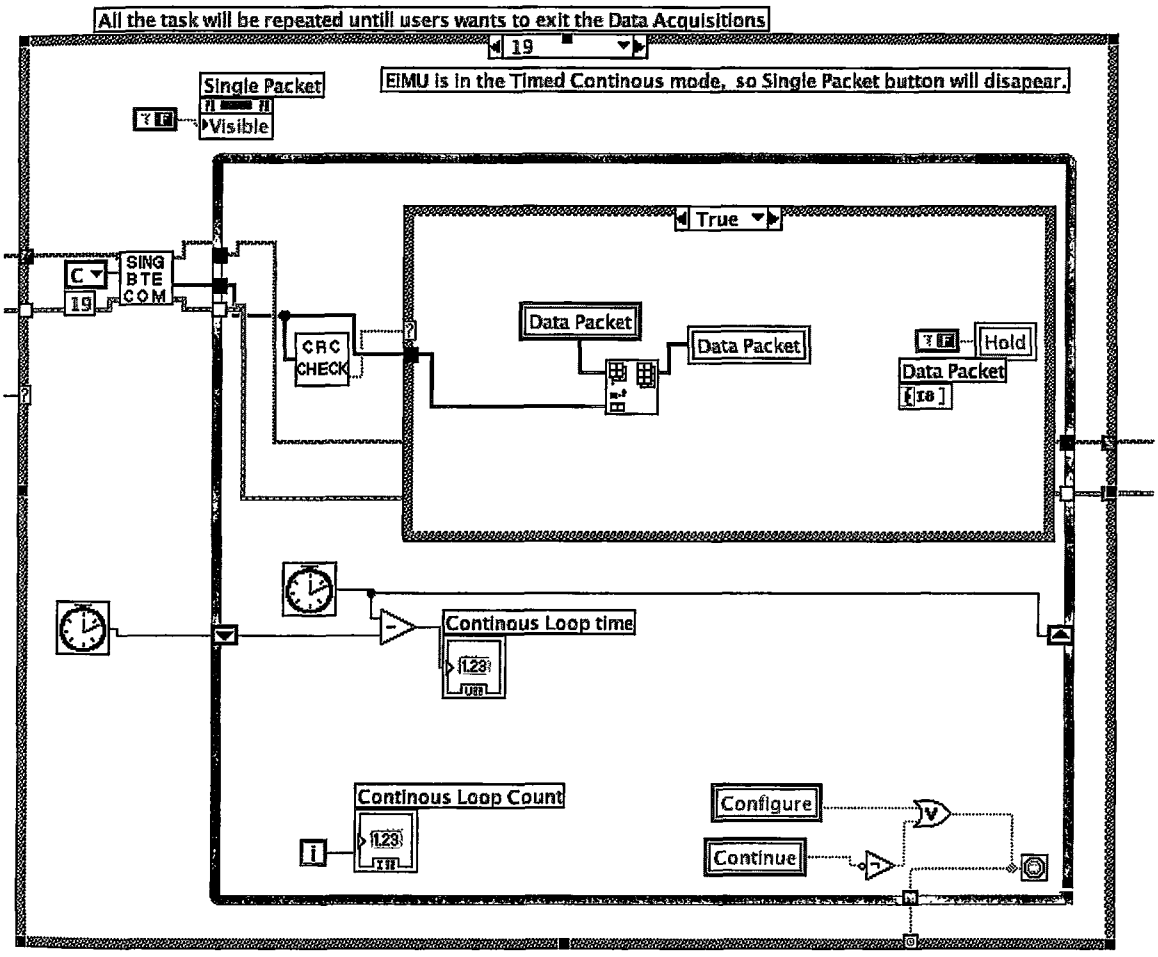

Figure 4.8 ECS Version 3.0

Version 4.0 – In order to experiment with timed continuous mode (Figure 4.9) where we should have been able to set the EiMU output rate as low as 10 more, we simply changed the output mode on Version 2.0. The INS system responded with corrupt data packets and most often we had synchronization issues. After numerous tests and resultant errors, we decided against continuing with this mode.

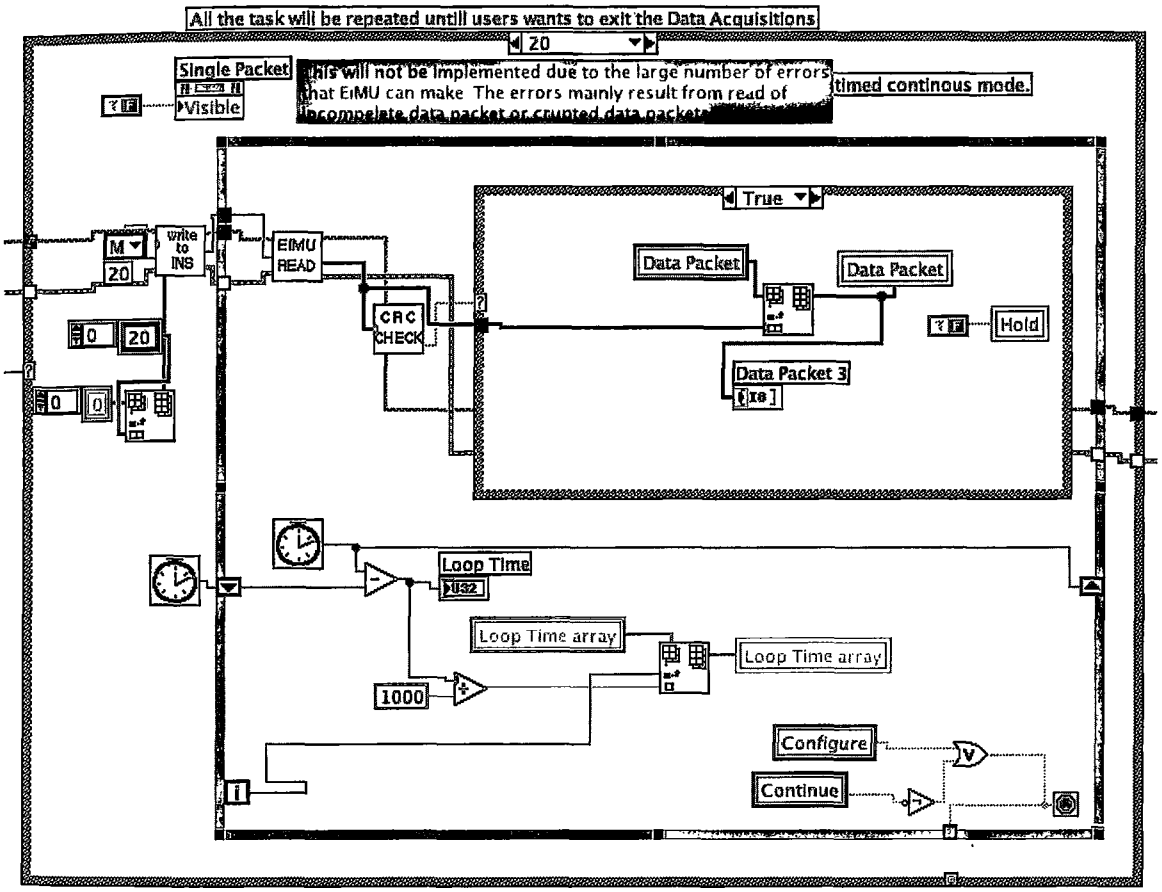

Figure 4.9 ECS Version 4.0

Version 5.0 – Here we replaced the array-based queue with a built-in LabVIEW queue in continuous mode (Figure 4.10). Using a queue instead of an array reduces processor memory usage. As with Version 2.0, here we use sampling time from the INS data instead of calculated loop time.

This version produced best results so far. It seemed to be able to synchronize with the INS at the 50Hz rate and read data without corruption. However, it was taking too long indicating a performance problem in the calculations.

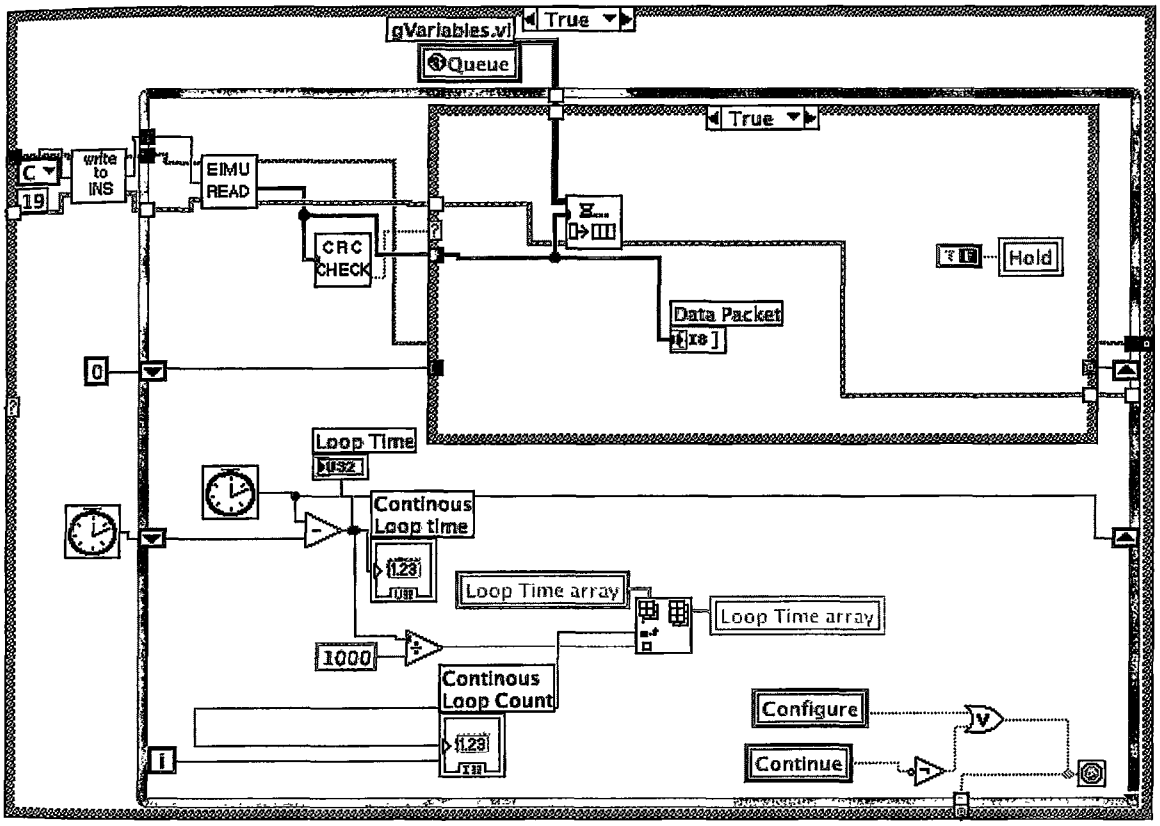

Figure 4.10 FillQ.vi code of ECS Version 5.0.

The idea of the queue is to separate hard real-time data acquisition from later soft real-time processing as illustrated in Figure 4.11.

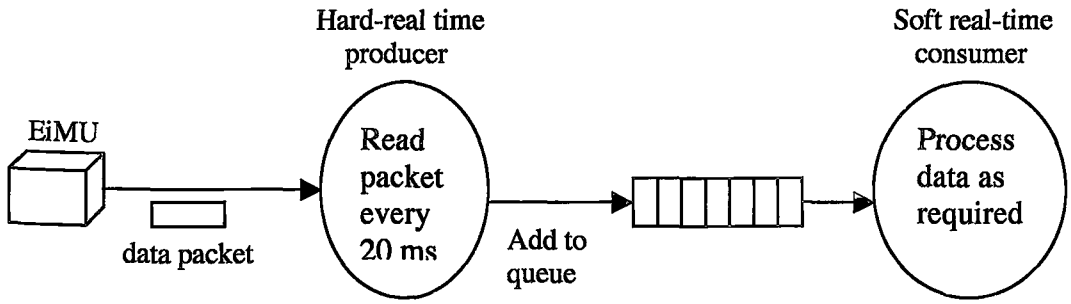

Figure 4.11 Use of producer/consumer queue to separate hard and soft real-time concerns

In Figure 4.12, we can see that a vi at the top left called 'FillQ'. This vi runs in parallel to the ECS vi and consumes the data from the queue. To separate hard and soft real-time concerns we put all data processing into this vi.

Version 6.0 - Similar to Version 5.0, except here we cumulated all processing that can be seen in Figure 4.10 into a separate module called 'FillQ'. It was observed that this did not reduce any processing time. Sampling time was not used here.

Version 7.0 – After moving the data processing into a separate vi that read from the queue, we put timers in every loop in both vis so that we could discover which processor loop is taking more time than it should.

In this version, we removed all the data calculations from the vi that reads from the queue (FillQ) so that all it did was read the data from the queue and display it. This enabled us to separately measure the time to read the packets from the timer to calculate the data.

Without the data processing the system was able to read packets from the EiMU, write them to the queue, read them from the queue and display them every 20 ms with time to spare. So we proved that the software could handle the hard real-time data acquisition and that excessive time was being consumed in the data processing.

Now that we had timing values, we could look at synchronization issues. The EiMU is sending 25 byte packets at 33Kbaud, taking a time of 7.6msec per packet. However, we found that the software was taking less than 1msec (finest measurement time) to read the packet and put it on the queue. This is because the Macintosh buffers the input characters. When a read is done, if the correct number of characters are not available Mac OSX puts the requesting thread into wait, freeing CPU time for other threads. So that the LabVIEW read.vi does not wait for the buffer to fill, we set the loop to run every 20msec (using a metronome vi). It waits for the buffer full and is synchronized after that. Thus, the ECS vi is synchronized to the end of the packet transmission and uses less than 1msec of CPU time, leaving the other 19msec for other threads and processing including FillQ.vi

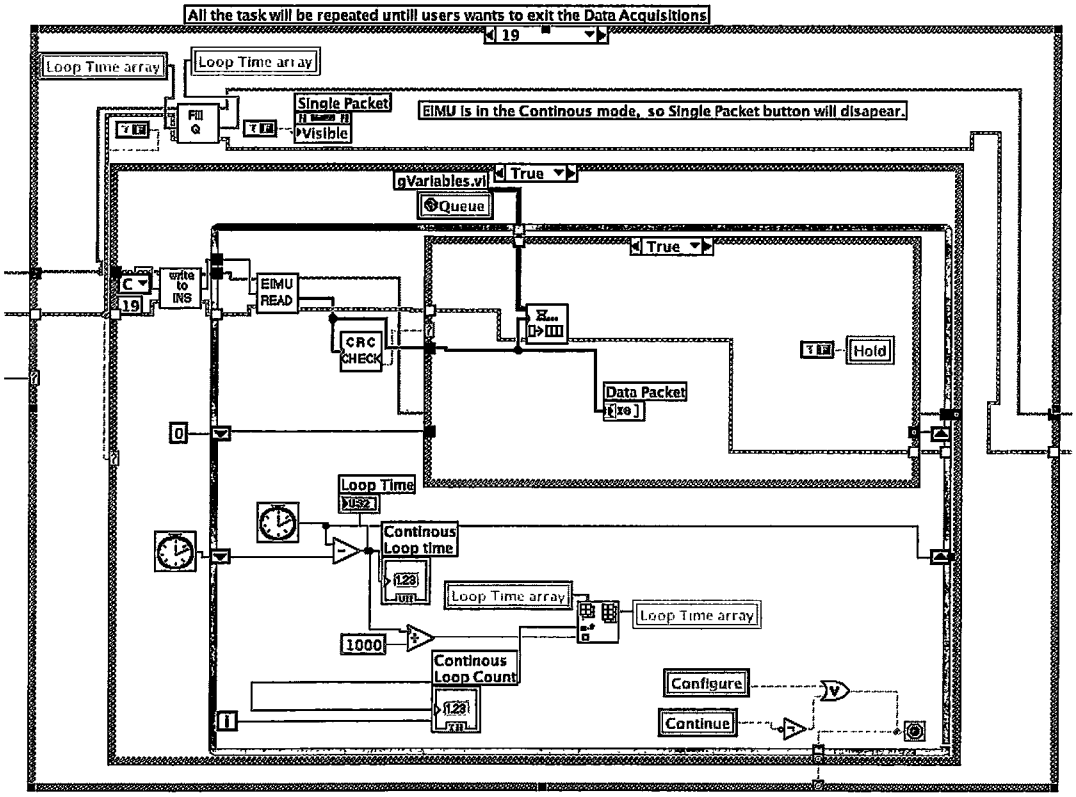

Figure 4.12 ECS Version 7.0.

Version 8.0 – This is the same as Version 7.0, except that we have added data calculations here. We discovered that there were 3 LabVIEW built-in integral functions that were running in a calculation loop for each sensor value. This decreased the read processing loop, caused synchronization errors and ensued in the data packet queue being overloaded. These functions were deleted once discovered and replaced by self-built ones.

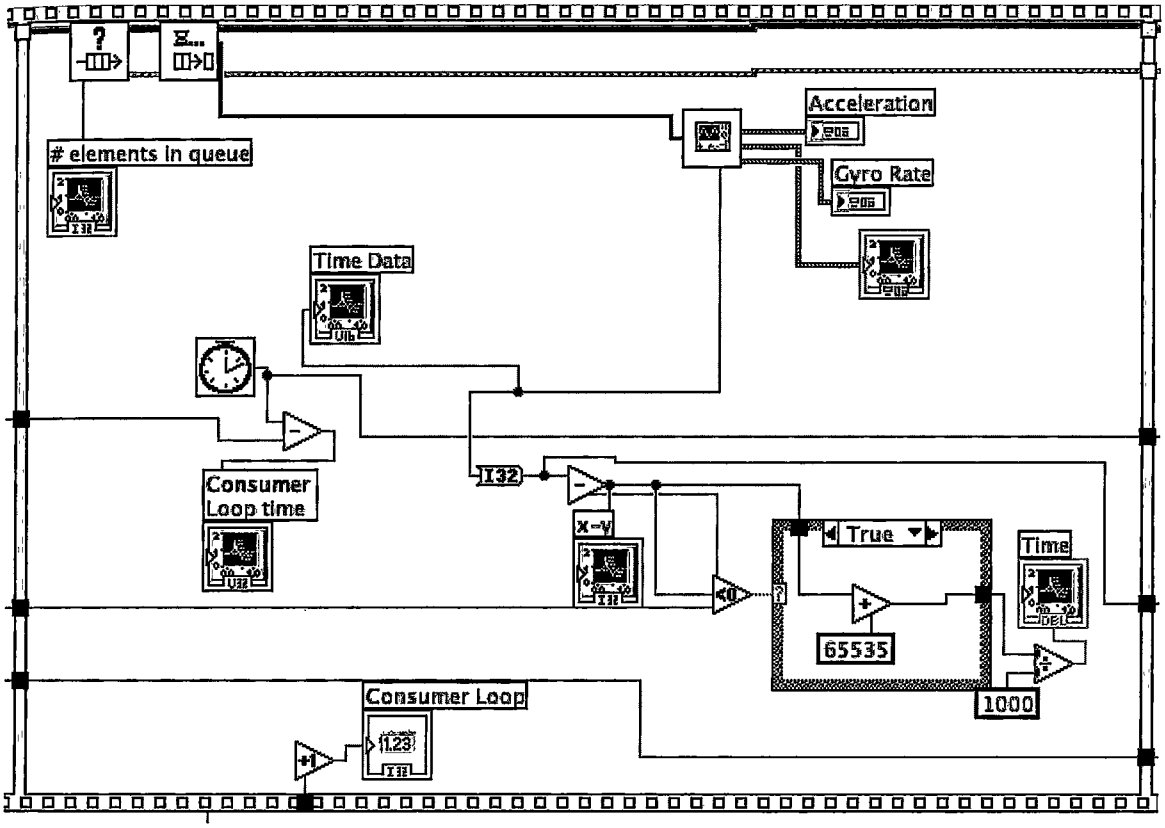

Figure 4.13 ECS Version 8.0.

In conclusion, Version 5.0 was chosen as the final software for experimentation. Using continuous calibrated mode, our sampling time averaged to 20msec. This meant we had 20msec to read a data packet, store it in a queue, and convert it to engineering units and store in a text file. With the help of a simultaneous processing loop to convert INS data to standard units and queues, all the above listed tasks were accomplished within 20msec. We chose to use sampling time instead of processing loop time in differential equations so that we get accurate results from gyroscope and accelerometer values. Converting INS values to Standard Units, i.e. G and deg/sec takes less than 1msec.

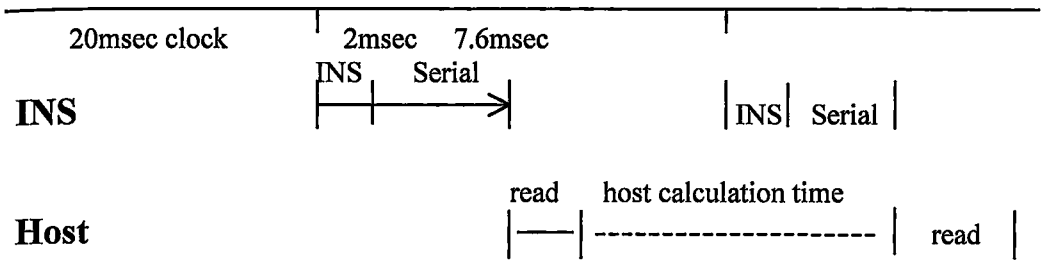

Figure 4.14 Timing delay is measuring, calculating and reading INS values

Finally, the data read from the queue for processing is delayed by 10.6msec (Figure 4.14 shows 2msecec for reading + 7.6msec for serial transmission + 1msec for capture) relative to the EiMU reading it. When we come to feedback control, these delays together with any delays caused by integration (Section 4.4) will impact on the controllability of the craft.

4.4 Data Extraction

Each sensor data value is represented in the data packet as a 16-bit signed short. These values are read from the sensor using a 'Visa Read' vi. They are then stored as signed 8-bits in a 1D array in the same order as that of the robust data packet (Table 4.3). This data packet array is sent to 'RawDisplay.vi' where each piece of data is converted into useful engineering units using the equations in Section 4.5.

Within 'RawDisplay.vi', first gyroscope and accelerometer measurements for each axis are extracted from the data packet array so that they can be converted into Standard Units. Before this is done, we need to represent them as signed 16-bit integers. We use 'Join.vi', which combines the MSB and LSB to produce a 16-bit unsigned integer. Then using a built-in LabVIEW function called 'To Word Integer', we convert the 16-bit unsigned integer to a 16-bit signed integer, which can now be used in a formula. Figure 4.15 illustrates the functionality of 'RawDisplay.vi'. The software written by a previous student, failed to do this, and resulted in strange errors in the data until the bug was discovered. This bug caused many weeks delay.

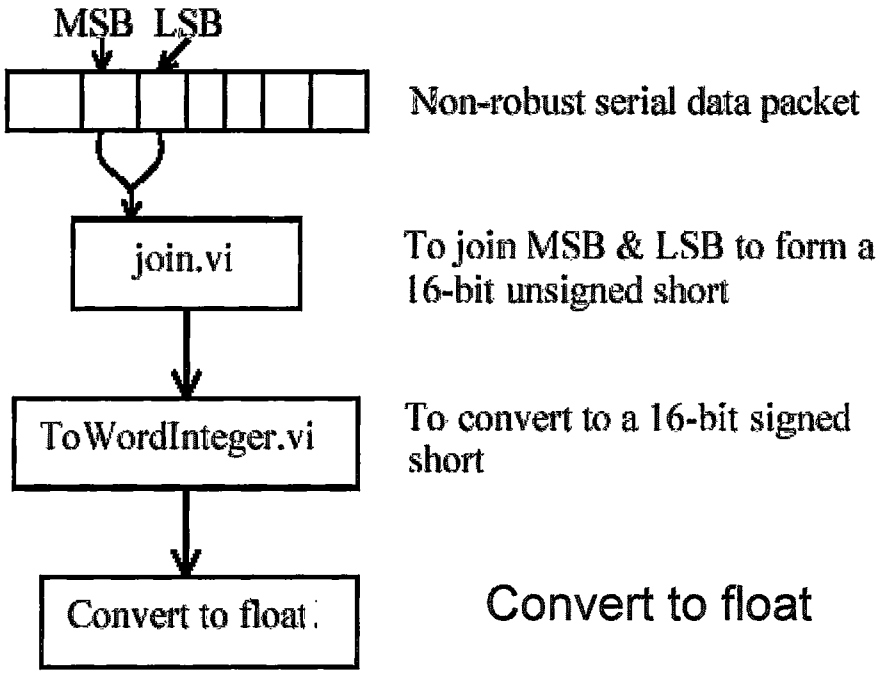

Figure 4.15. Block diagram representing flow of data within LabVIEW Module

4.5 Data Conversion

Each sensor data value is represented in the data packet as a 16-bit signed short (the MSB- most significant byte and LSB – least significant byte) [18]. Each piece of data must be converted into useful engineering units using equations (4.1), (4.2) where *data* is the above explained float value:

Accelerometer

To convert the data value to a G value:

$$accel = R * data / 1024 \quad (4.1)$$

where R is the range of the accelerometer, i.e. R is 2

Rate gyro

To convert the data value to a degrees/second value:

$$rate = S * R * data / 512 \quad (4.2)$$

where R is the range of the rate gyro and S is the scale factor; R is 100 and S is 1.5

4.5.1 Zeroing process

The rate gyro chips used in the EiMU are subject to changes in their rate bias due to temperature changes [18]. This means that a rate gyro will read non-zero even when stationary. To reduce this problem it is normal to 'zero out' this bias. Zeroing may be performed using the 'Z' command. After the EiMU receives this command, it goes into ZEROING state and averages the readings of each rate gyro for the specified time of the readings. The EiMU then automatically reverts back to NORMAL state. These new zero offsets are NOT automatically saved to EEPROM.

We found that zeroing the gyros was effective but there was no command to zero the accelerometers. Even after a zeroing, there is no guarantee that the rate bias will not change. Changes in temperature will change it and vibrations of certain frequencies may also change it. The EiMU should be vibration isolated as much as possible.

After analysing the data collected from random tests, where the DraganFlyer was kept still for approximately 100-200 samples, we confirmed that there is a zero offset in the accelerometer and gyro rate readings. An offset can be either a positive value or a negative value. It is important to subtract this offset from the readings so that when calculations are done from these values, we get as accurate results as possible.

As Figure 4.16 demonstrates, we take the average of the first 20 readings on every axis and subtract this value from each sample to remove the zero error.

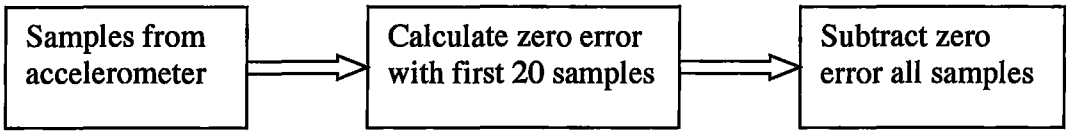

Figure 4.16 calculating a remaining zero offset from accelerometer readings

4.5.2 Integration

Below is the set of finite difference equations that is applied to calculate velocity from acceleration and position from velocity. This calculates both linear and angular values.

$$a_t = (v_t - v_{t-1}) \div \delta t \quad (4.3)$$

where, a_t and v_t are current acceleration and velocity respectively, v_{t-1} is velocity earlier and t is the time. δt is the time taken by the INS system between two consecutive measurements, also known as the Sampling time.

$$v_t = v_{t-1} + a_t * \delta t \quad (4.4)$$

where, v_t and a_t are current velocity and acceleration respectively, v_{t-1} is velocity earlier and δt is the sampling time

$$d_t = d_{t-1} + v_t * \delta t \quad (4.5)$$

where, d_t and v_t are current distance and velocity respectively, d_{t-1} is velocity earlier and δt is the sampling time

Values for velocity and position are calculated by integrating values from the previous time step. Thus, the velocity is the velocity for the time period just past. Similarly position. Thus, the calculations do not introduce a delay in the feedback, apart from calculation time (Figure 4.14). Although by physics, we know that position lags velocity and velocity lags acceleration and the process of integration introduces a 90° phase lag in each case.

Equations 4.4 and 4.5 are both linear equations, so a small error in acceleration (zero error) will result in Equation 4.4 integrating out linearly with respect to time. However, a double integration from acceleration to distance where Equation 4.4 and 4.5 are combined results in a non-linear output in response to a zero error. For this reason, angle values calculated from gyroscopic velocity are not as prone to drift and noise as distance values calculated from accelerometer outputs are.

When calculating values from accelerometer values, we take the values stored in text files and read them into a 2D array. Values for feedback control are calculated directly from the values read from the queue. The accelerometer values are still stored in the file for later analysis. From here onwards, they are processed in a loop till the array reads empty. The values measured by the Accelerometer are in G units and need to be converted to m/s^2. Figure 4.17 illustrates this conversion in LabVIEW.

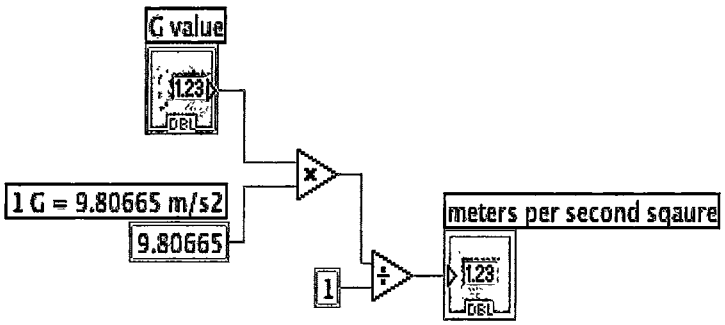

Figure 4.17. Shows conversion of linear acceleration values in unit G to m/s^2

Figure 4.18 displays how these acceleration measurements are integrated to produce velocity profiles and then velocity values are integrated to give distance graphs as shown in Figure 4.19 using equation (4.4) and (4.5) respectively.

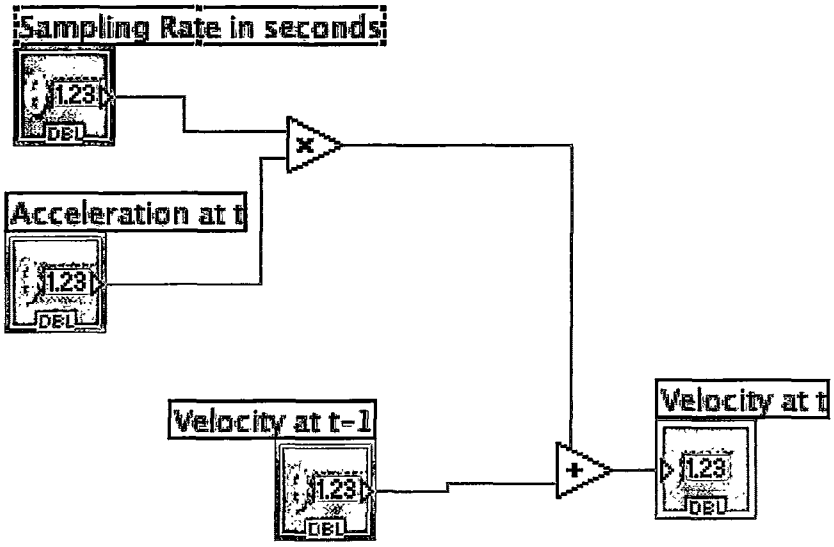

Figure 4.18. Integral of acceleration using self-built module, 'Integral.vi' to produce velocity

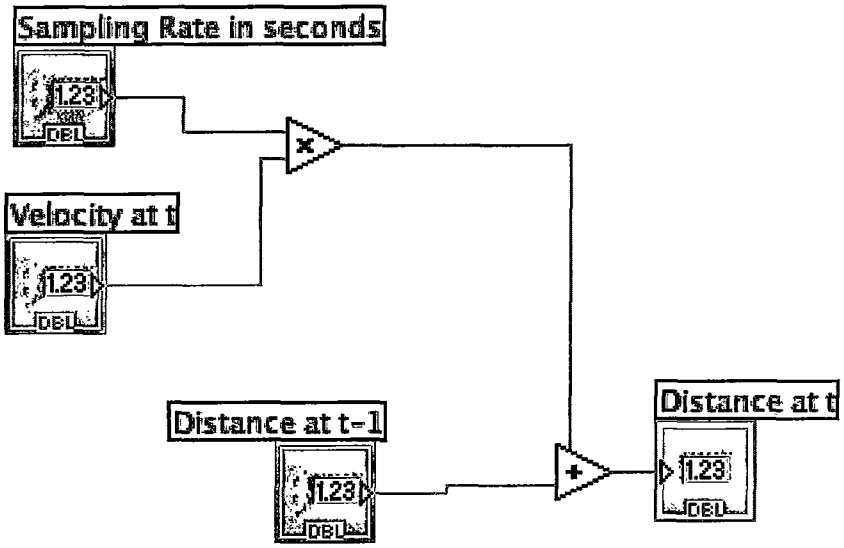

Figure 4.19. Integral of velocity using self-built module, 'Integral.vi' to produce distance

Calculated values are displayed in graphs. The same process is applied when we derive rotation angle from gyro rates measured by gyroscopes in units of deg/sec. Gyro rate is integrated by applying Equation (4.5) and differentiated by Equation (4.3) to produce angular acceleration. Figure 4.20 illustrates 'Derivate.vi' that implements Equation (4.3). We are not using a complementary filter while calculating angular displacement (Figure 4.21) since the INS has an in-built complementary filter.

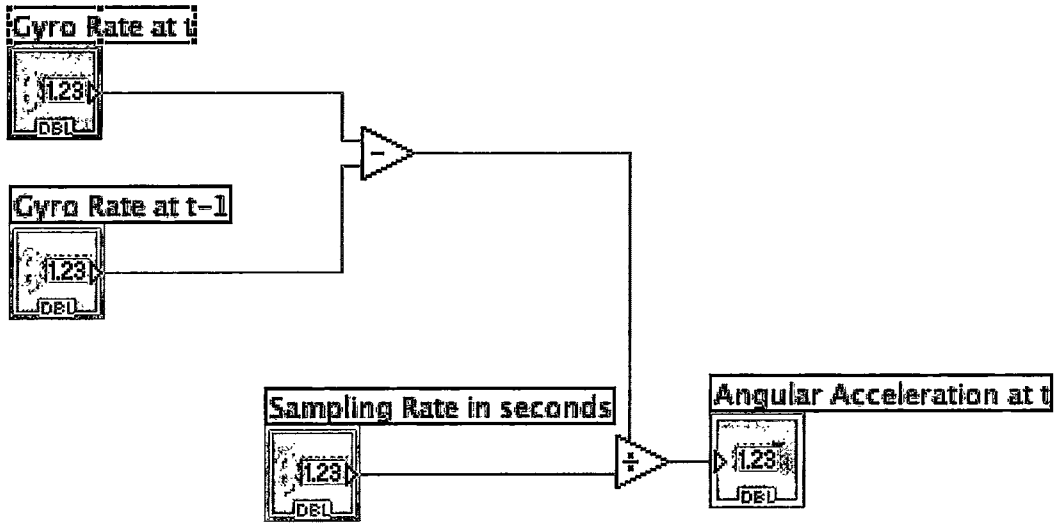

Figure 4.20. Derivative of angular velocity to produce angular acceleration

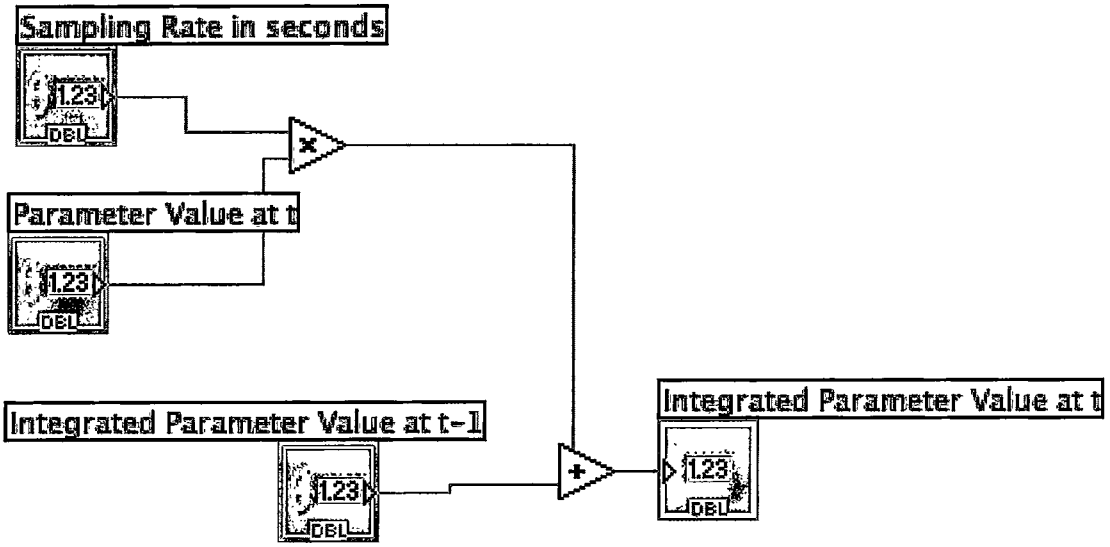

Figure 4.21. Integral of angular velocity to produce angular displacement

4.5.3 Sampling Time

As can be seen in Figure 4.22 between samples 0-636, sampling time is normally about 20msec and occasionally 40msec due to the computer performing some other function. This signifies that we have 20msec to process received data from the INS and empty the data packet queue before the next packet.

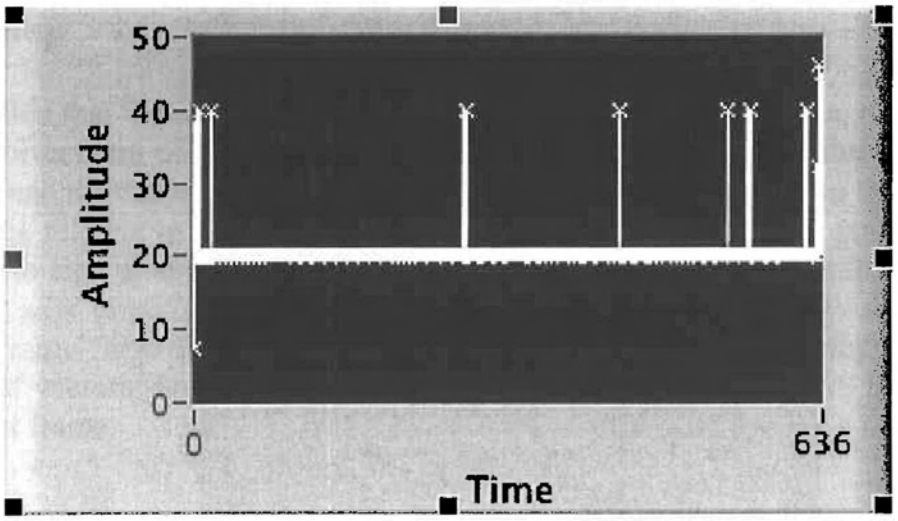

Figure 4.22 Recording of sampling time

The value used in the calculations was the INS measured value in Figure 4.22. Using the time measured by the INS, rather than the expected time, removes the errors from the integration that can be caused by the computer not responding at exactly every 20msec, or missing the occasional packet due to other processing. One of the problems with using a desk top machine as a host is that you don't know what other processes the operating system is supporting, e.g checking disk buffer cache and writing buffers to disk, and responding to network enquiries.

4.6 Testing

Having verified that the software meets the required performance criteria, the next step is to test the correctness of the data values. This process tests both the correctness of the calculations and the calibration of the INS. In addition, it is used to obtain the direction of the INS frame relative to the world frame. When the INS was mounted on the robot, the criteria was to find a suitable place, near the central axis, roughly parallel to the cross bars, so the axis direction are only partially known (Figure 4.23). World and robot coordinate frame form a right-handed set of vectors, but the INS frame forms a left-handed set of vectors. So the x value has to be inverted to move the INS measurements into the robot frame.

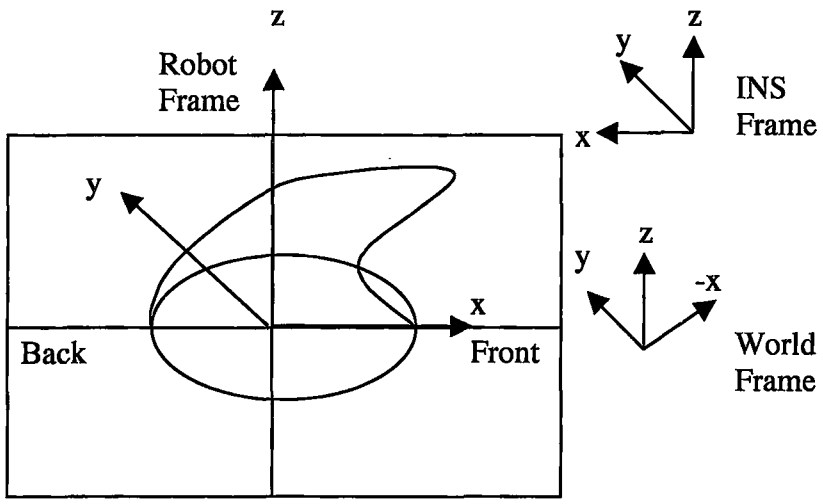

Figure 4.23 Relationship between INS axes and world reference frame

Experiments are done inside the Robotics Laboratory under normal environmental conditions such as room temperature of 20°C is maintained, no external forces such as wind or thrust affect the test, except the forces intentionally applied. Tests were needed to calibrate accelerometer and gyroscope values on all three planes, *x, y, z* and on positive and negative directions of each plane. We measure the distance, with a ruler, in all three planes to calibrate accelerometer and use a protractor to measure the angle moved and co-ordinate that to the angle calculated by the software.

4.6.1 Commercial Test Units

There are various commercial test units such as, the EVO Series by Wuilfert that can be used for calibration purposes. Wuilfert, a leading provider of rate tables and motion simulators, has introduced the new EVO Series to provide testing and calibration solutions that evolve as your needs change. These low cost motion tables are designed for loads with small to medium mass and to provide benefits that are currently unavailable. Advantages of the EVO series include advanced software capabilities; and the ability to upgrade system performance as produce design, testing and calibration requirements evolve. The rate table system consists of a rate table together with a modular control unit

and a master PC. An optional temperature chamber with integrated control capabilities is also available. EVO series rate tables have been optimised for MEMS sensors, components, and inertial systems up to a payload of 22lbs. The EVO series offers rapid "out of the box" implementation resulting from the intuitive ProAxe software and modular design.

EVO series compact rate tables are designed for maximum utility. The single axis EVO-10 rate table (Figure 4.24) can be operated in both a vertical or horizontal orientation for testing the angular position, velocity (rate), and acceleration sensitivity of payloads. An optional pedestal is also available.

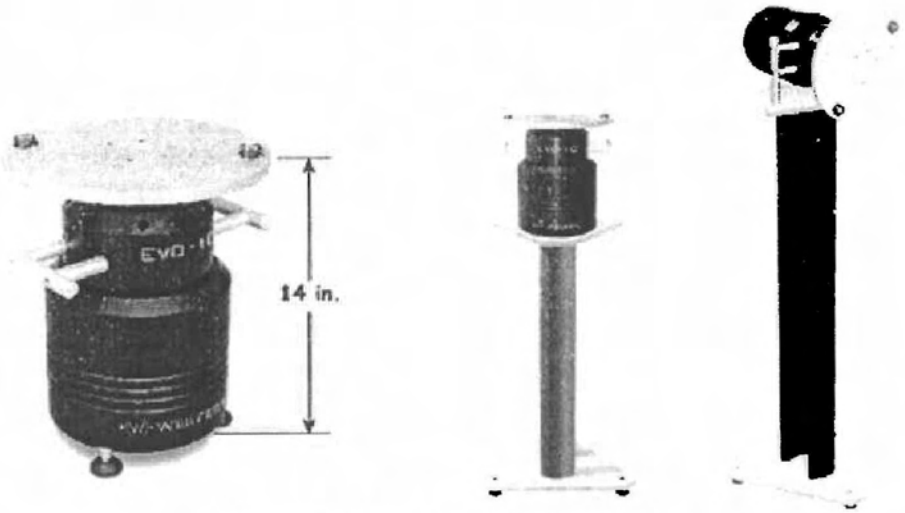

Figure 4.24. Single-axis EVO-10 rate table

The compact design allows for an efficient use of floor space while providing full functionality including a response frequency of 75Hz. The EVO-10 rate table weighs less than 66 lbs, and can tests loads up to 22lbs at speeds up to 3000deg/sec and accelerations up to 200deg/s^2, while providing a 34 (optional 46) channel slip ring for full data access. Motion is achieved through the use of direct drive brushless torque motors providing a distinct advantage over brush-type motors or belt transmissions, including maintenance-free operation and high acceleration rates. The EVO-10 single-axis able has a positional accuracy of ±20 arc-sec with a rate resolution of 0.001 deg/sec. In addition, rate accuracy and rate stability of ±0.005% can be achieved.

The EVO Series is designed with the goal of maximizing the user's return on investment by allowing the product to evolve as requirements change. It can be upgraded with hardware and software options that extend capability and functionality.

In addition to the EVO-10, the EVO-20 provides dual-axis testing. The EVO-20 (Figure 4.25) provides unlimited two-axis motion, and is a direct upgrade to the EVO-10 series. The two-axis yoke design is comprised of the EVO-10 providing the inner-axis.

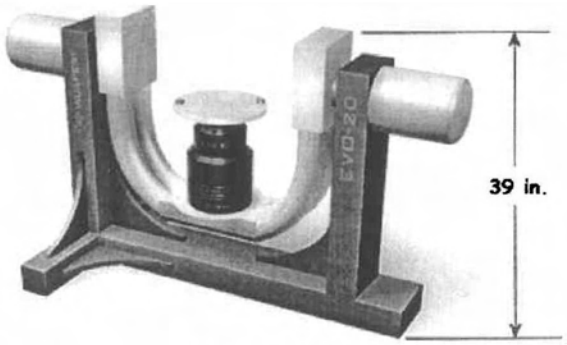

Figure 4.25 EVO-20

An optional temperature chamber can be incorporated to verify a payload's performance as a function of temperature. The temperature chamber can be added to an existing system for testing a single-axis in either a vertical or horizontal orientation (Figure 4.26) within an allowable temperature range of –40C to +80C (optional –55C to +125C).

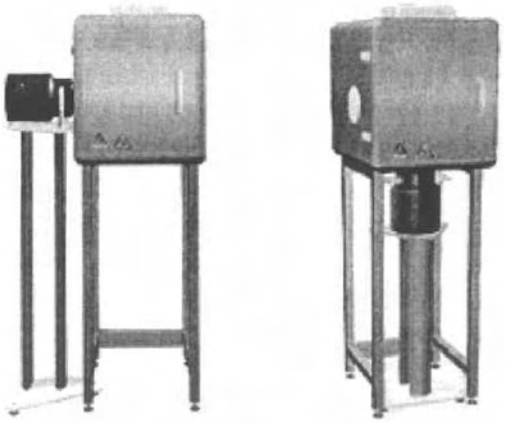

Figure 4.26 Optional temperature chamber

The same temperature chamber can also be added to an existing two-axis configuration, thereby extending the performance of an existing system with minimal incremental investment (Figure 4.27).

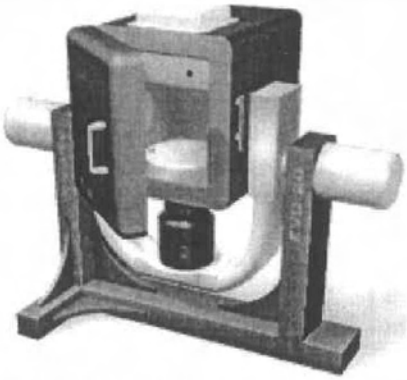

Figure 4.27 Temperature chamber on a two-axis configuration

However, we conducted calibration using a ruler and tripod. The results produced were accurate enough.

4.6.2 Test Design

When the DraganFlyer is controlled using 4 controls to command six degrees of freedom, coupling of forces occur and we can see significant movement in more degrees than desired. This effect was not optimal for calibration purposes. The tests are designed in such a way that the result would portray the effect of a command on each individual plane, i.e., x,y or z for angular rotation and linear displacement. Thus decoupling the forces.

Table 4.5 describes each test plan, its limitations and the expected result of each test

Sensor Type	Test Name	Experiment Description	Limitations	Equipment	Expected Test Result
Accelerometer	*x*-axis positive distance	DraganFlyer is moved 30cm horizontally on a smooth, flat surface.	Accuracy of ruler measurement	This displacement is measured by placing a 100cm long ruler parallel to its movement	The slope on the graph should display a displacement of 30cm.
	x-axis negative distance	DraganFlyer is moved 39cm horizontally on a smooth, flat surface in the opposite direction.	Accuracy of ruler measurement	This displacement is measured by placing a 100cm long ruler parallel to its movement	The slope on the graph should display a displacement of -39cm.

Sensor Type	Test Name	Experiment Description	Limitations	Equipment	Expected Test Result
Accelerometer	*y-axis* positive distance	DraganFlyer is moved 14cm sideways on a smooth, flat surface.	Accuracy of ruler measurement	This displacement is measured by placing a 100cm long ruler parallel to its movement	The slope on the graph should display a displacement of 14cm.
	y-axis negative distance	DraganFlyer is moved 20cm sideways on a smooth, flat surface in the opposite direction.	Accuracy of ruler measurement	This displacement is measured by placing a 100cm long ruler parallel to its movement	The slope on the graph should display a displacement of -20cm.
	z-axis positive distance	DraganFlyer is placed on a tripod and moved 40cm vertically upwards in the air.	Accuracy of ruler measurement	This displacement is measured by placing a 100cm long ruler parallel to its movement	The slope on the graph should display a displacement of 40cm.
	z-axis negative distance	DraganFlyer is placed on a tripod and moved -40cm vertically downwards in the air.	Accuracy of ruler measurement	This displacement is measured by placing a 100cm long ruler parallel to its movement	The slope on the graph should display a displacement of -40cm.
Gyroscope	*x*-axis positive angle	DraganFlyer is rotated 90° towards around the x-axis.	Since the DraganFlyer is placed on a tripod, there is no strap-on to hold it firmly in position. This may cause movement in other axes.	A measurement dial on the tripod is used to measure an accurate 90° rotation.	The slope on the graph should display angle moved as 90°.

Sensor Type	Test Name	Experiment Description	Limitations	Equipment	Expected Test Result
Gyroscope	*x-axis* negative angle	DraganFlyer is rotated -90° around the x-axis.	Since the DraganFlyer is placed on a tripod, there is no strap-on to hold it firmly in position. This may cause movement in other axes.	A measurement dial on the tripod is used to measure an accurate 90° rotation.	The slope on the graph should display angle moved as -90°.
	y-axis positive angle	DraganFlyer is rotated 90° around the y-axis.	Since the DraganFlyer is placed on a tripod, there is no strap-on to hold it firmly in position. This may cause movement in other axes.	A measurement dial on the tripod is used to measure an accurate 90° rotation.	The slope on the graph should display angle moved as 90°.
	y-axis negative angle	DraganFlyer is rotated -90° around the y-axis.	Since the DraganFlyer is placed on a tripod, there is no strap-on to hold it firmly in position. This may cause movement in other axes.	A measurement dial on the tripod is used to measure an accurate 90° rotation.	The slope on the graph should display angle moved as -90°.

Sensor Type	Test Name	Experiment Description	Limitations	Equipment	Expected Test Result
Gyroscope	*z-axis* positive angle	DraganFlyer is rotated 90degrees clockwise on a smooth, flat surface	None	Using a protractor, a circle is drawn on a flat, smooth surface. After the DraganFlyer has been moved, the protractor is placed on the circle to measure the angle moved	The slope on the graph should display angle moved as 90°.
	z-axis negative angle	DraganFlyer is rotated 90degrees anti-clockwise on a smooth, flat surface	None	Using a protractor, a circle is drawn on a flat, smooth surface. After the DraganFlyer has been moved, the protractor is placed on the circle to measure the angle moved	The slope on the graph should display angle moved as -90°.

Table 4.5. Description of each test plan, its limitations and expected result

4.6.3 Test Result

Below are the test results. Some of the results are not exactly as expected but the reasons have been discussed.

Accelerometer
***x-axis* positive Distance**

In this experiment, we physically moved the DraganFlyer along a desk in the positive x direction a distance of 30cm. Then we measured the displacement with a ruler to be 31.5cm. The measurement of the same displacement with the INS system is (0.40m – 0.085m) = 0.315m = 31.5cm (Figure 4.29).

Referring to Figure 4.29. we see positive x acceleration, as measured by the INS, in the top graph and as expected it resembles a sine-wave where peak acceleration is approximately 5.5m/s2. Acceleration is integrated to produce the velocity profile in the second graph and as expected, from the acceleration graph the velocity increases smoothly to a maximum and then decreases as the DraganFlyer is brought to a halt.

Velocity is further integrated to generate the distance graph at the bottom of Figure.4.29, which displays smooth motion.

However, it can be noticed after the acceleration that the distance graph is still rising at 0.01cm every 4 samples due to drift in the accelerometer. This drift causes a velocity error of 0.01m/s when the DraganFlyer is at rest after motion, which is barely visible in the velocity graph. No error is visible in the acceleration graph. Also, the average of samples 350-370 before the motion is 0 and the average of samples 410-440 after motion is 0 indicating no change in zero error. The distance graph illustrates that it takes a very small error to cause a double integration to drift.

Similar tests were done for x-axis negative distance and y -axis and z -axis distance. These are included in Appendix A. The one issue of note is the noise on the y -axis accelerometer when it is near zero (Figure 4.28). This noise causes a velocity error of 0.01m/sec (Appendix A.2). Figure 4.28 shows the complete 324 samples that were taken for this particular experiment. The graph displayed a significant change in zero before and after motion. This offset was also noticed in the acceleration graph of Appendix A.2, therefore we can say that the accelerometer measurement of y-axis always shows a significant zero error.

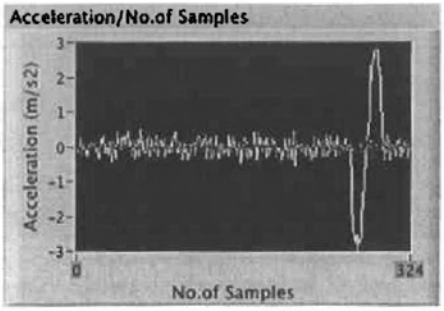

Figure 4.28 Shows the zero offset before and after motion

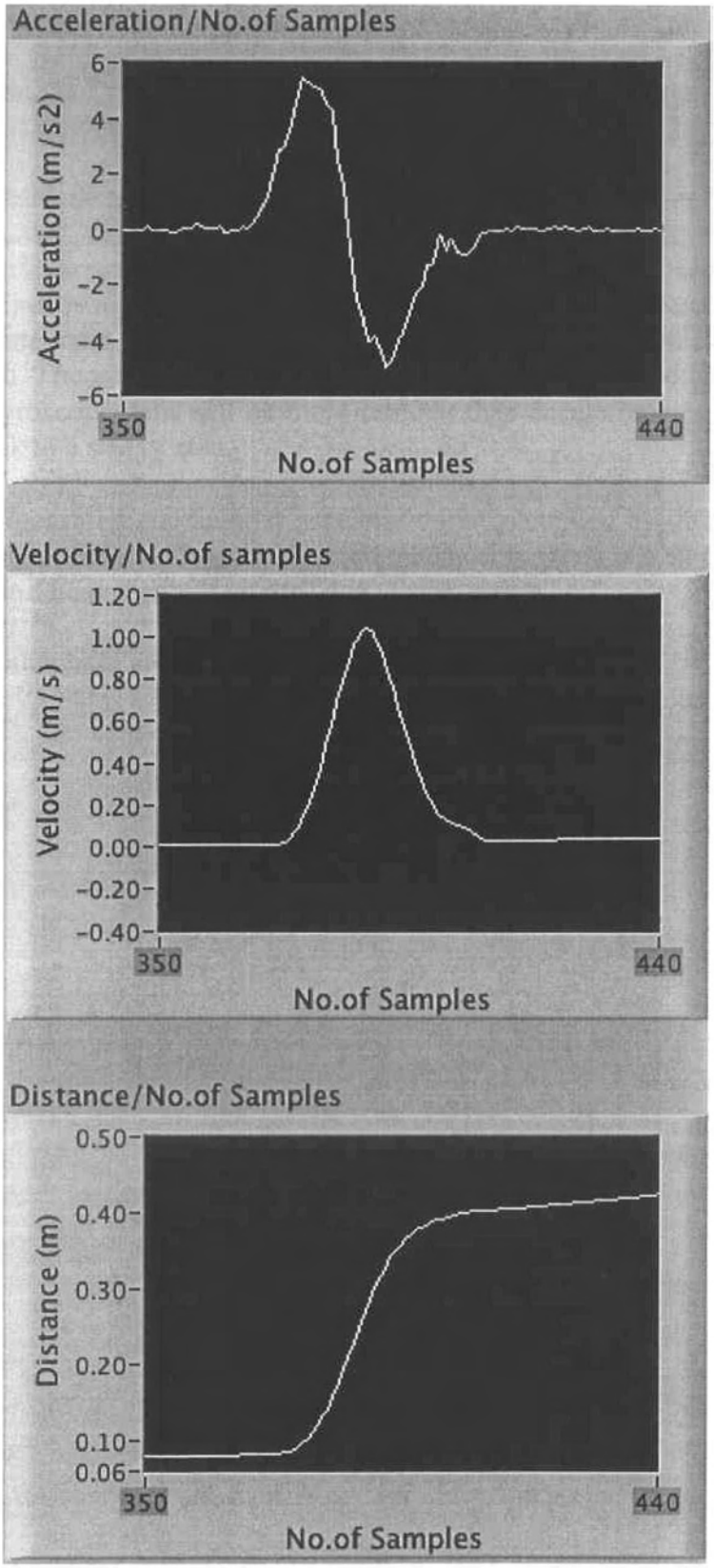

Figure 4.29 Shows displacement of 31.5cm on x-axis

x-axis positive Angle

In this experiment the DraganFlyer is rotated 90 in the positive x angle. The scale on the tripod measures it as 90 and INS system measures it as (83 – (-4)) = 87.

Figure 4.30 contains three graphs, i.e. angular acceleration, angular rate and angle. Since this experiment was carried out using a tripod and a palm, many vibrations were measured by the INS system. Angular rate increases smoothly but decreases with jerks before settling into zero. Given the vibrations in the rate, derived acceleration is not smooth either. Integrating angular rate produces the angle rotated, which is displayed in the bottom graph. There is no drift in this gyroscope graph before and after motion. This indicates that gyroscope data will be more reliable than accelerometer data. The angle graph settles back to a steady zero.

Angular rate is measured and angular acceleration is calculated by differentiating rate, which is why the acceleration is so noisy. Here, the position data is a single integration of the measured rate data and shows no drift due to zero errors.

Test results for other axes are included in Appendix A and B.

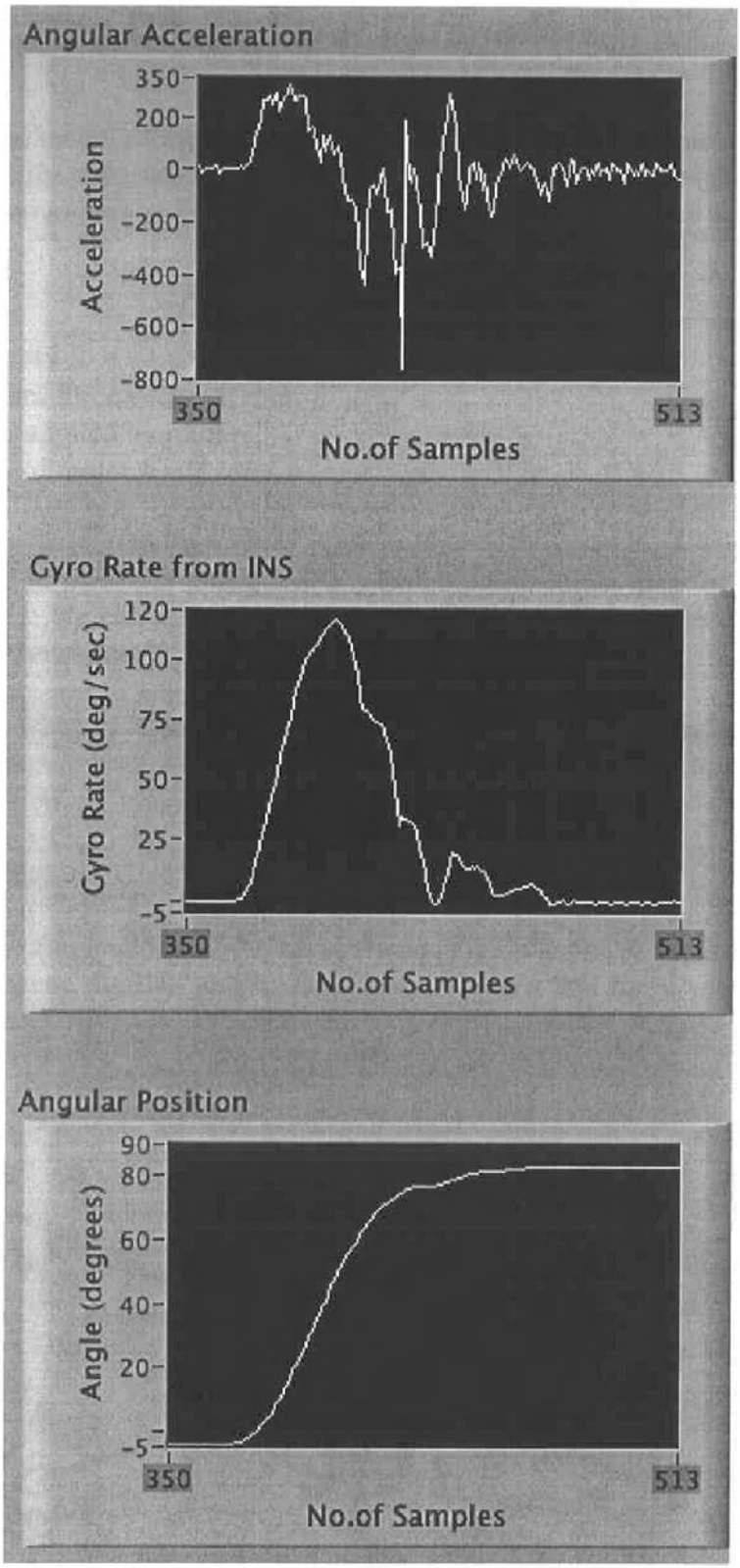

Figure 4.30 Shows rotation of 78deg on x-axis

5.0 Measuring DraganFlyer Parameters

We discussed the model of the DraganFlyer and its dynamics in Chapter 2. In this chapter we will discuss the measurable parameters in the model of the DraganFlyer. We will identify those parameters and describe how they can be used to control the DraganFlyer.

$$f_n = m_n g - f_{mn} \tag{5.1}$$

n where $n = 1$ to 4
f_n = resultant force vector at rotor n
f_{mn} = force applied by rotor n
m_n = mass of motor 1 and rotor n

We can now fill in these variables. Using a weighing machine, mass of motor and rotor can be measured. We can measure the force applied at rotor n because we send it as a command from the computer.

As discussed in Chapter 2, forces and torques can be calculated from linear and angular accelerations. Experiments to measure their dynamic parameters are reported in Chapter 7.

5.1 Kinematics

Figure 5.1 shows the location of the robot frame (Figure 2.1) and Figure 4.22 shows the relationship between the INS and world frames. We saw that the x value from the INS has to be inverted.

Please see print copy for Figure 5.1

Figure 5.1 Forces acting on the DraganFlyer

We measured all the dimensions that are used in calculations, such as length and width of blades, Table 5.1.

Item	Measurement (cm)
Rods l_1, l_2, l_3, l_4	16cm
Blades l_{b1}, l_{b2}, l_{b3}, l_{b4}	18.5cm
Width of blades w_{b1}, w_{b2}, w_{b3}, w_{b4}	3cm

Table 5.1 measurements of length and width of blades

Item	Mass (g)
DraganFlyer with batteries, with cover	703
Sinker at the end of tether	47
NICAD Battery 1	187
NICAD Battery 2	192
Lithium Battery	112

Table 5.2 Weight of items on-board DraganFlyer

We can calculate the inertias values by placing values from Table 5.1 and 5.2 into Equation 2.26. The mass of the DraganFlyer may vary depending on the batteries being used and any sensor payload, thus it must always be weighed before experimentation (Table 5.2). Also due to the mounting of certain devices on the aerial vehicle, it should be assured that the mass of these devices is equally distributed over the flyer. Figure 5.2 shows how the devices are positioned on the base of the DraganFlyer to balance it equally.

An INS, Bluetooth and battery are mounted on it. Note the position of the battery is in such a manner so that it can balance the weight of the INS and Bluetooth.

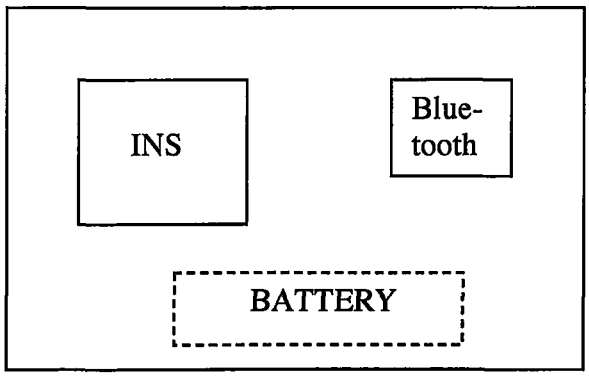

Figure 5.2 Layout of devices on DraganFlyer

5.2 Experimental Set-up

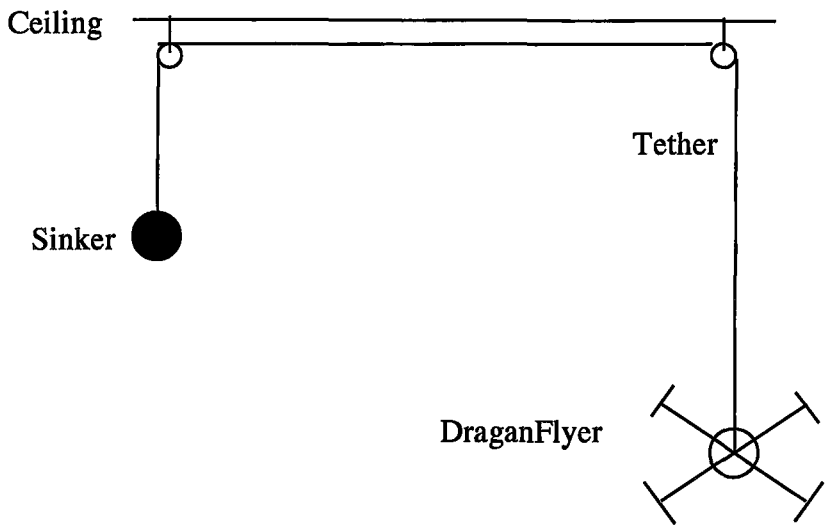

Figure 5.3 Experimental Set-up in Lab for testing

In our earlier experiments we found that we couldn't fly the DraganFlyer without crashing it. In fact, crashed caused damage to the electronics that resulted in us not being able to perform one set of experiments. In order to protect the expensive INS unit we hung the DraganFlyer from a tether, with a sinker to pull the tether out of the way as it rose up in the air (Figure 5.3). This placed constraints on its motion that stopped it crashing, but limited he distance over which we could conduct experiments.

5.3 Tether

During free fall gravity, the mass at the end of the tether is $M_T = M_D + M_S$
where, M_D is the mass of the DraganFlyer and M_S is the mass of the sinker. During DraganFlyer's vertical acceleration, $M_T = M_D - M_S$

5.4 RPM and Air Velocity

We have used real data to find the relationship between the RPM and Air velocity (m/s). Figure 5.4 shows the data and graph.

Air Velocity	RPM
0	0
0.8	390
1.3	690
1.3	900
1.9	1140
3.1	1620
2.7	2010
3	2280

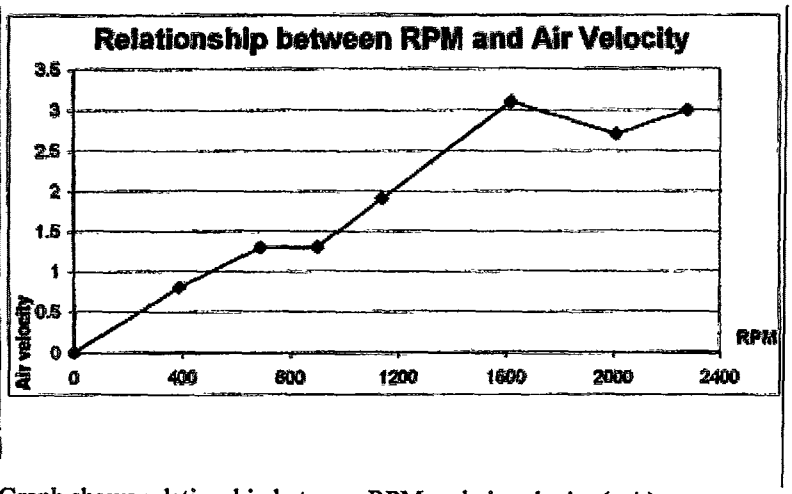

Figure 5.4 Graph shows relationship between RPM and air velocity (m/s)

We provided the above data into software called, Mathlab 6.2. It generated the following graph with a smooth curve.

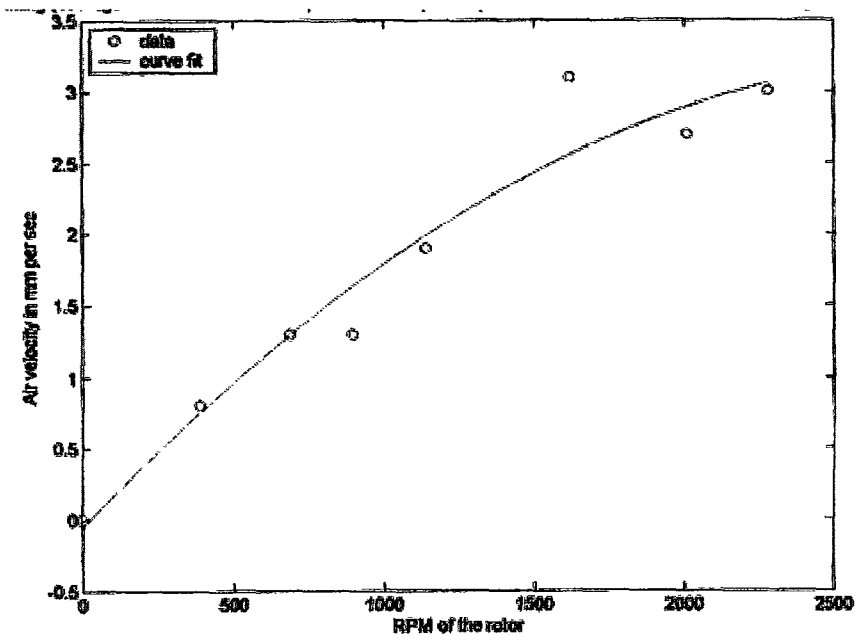

Figure 5.5 Curve used to find the equation of the rotor

Since the curve in Figure 5.5 looks reasonable, we are going to use Figure 5.5 to find the equation of the rotor. However the fifth point (1600, 3.2) would appear to be incorrect. While the curve in Figure 5.5 is non-linear, a straight line (Equation 5.1) can be fitted to all points except the fifth (1600 RPM).

$$airvelocity = 0.3 + 0.0016 \times RPM \qquad (5.1)$$

6.0 Motion Command Software

In this chapter we will discuss communication between the DraganFlyer and host machine so that commands can be sent to DraganFlyer. This involves completing the hardware set-up and developing the output software.

6.1 Bi-directional Hardware set-up

In Figure 4.2 we showed the bi-directional communication between the host computer and the INS to set-up the INS and read feedback data. In Figure 6.1 we show the unidirectional connection between the host and the DraganFlyer send flight control commands to the DraganFlyer.

The primary hardware device in use here is called PCBuddy. It is a serial port to RC buddy box interface, that allows full control of RC equipped vehicles via a user's own PC program which they can write in any programming language.

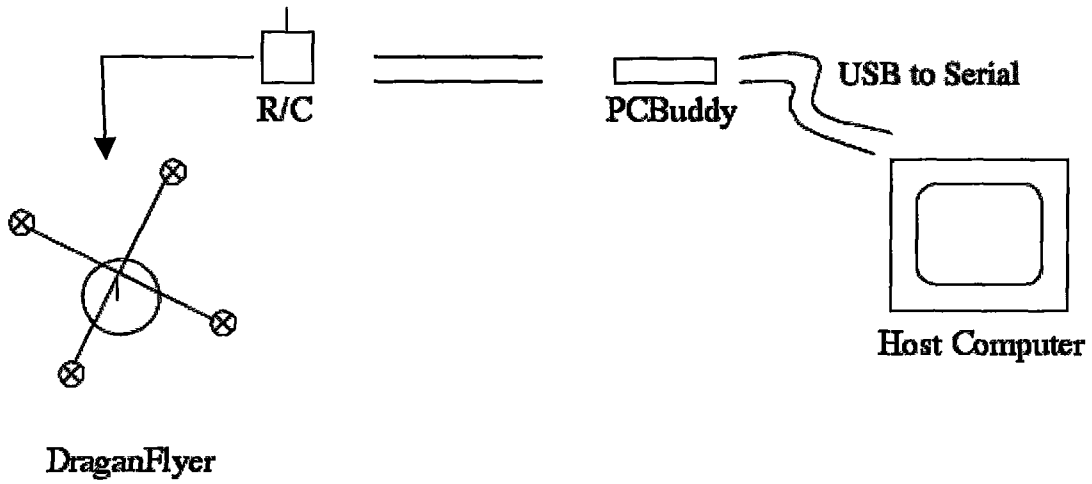

Figure 6.1. Block diagram of hardware set-up for unidirectional communication from the host computer to the DraganFlyer

Figure 6.1 demonstrates the hardware set-up between the radio control of the DraganFlyer and the computer, which is hosting software to control commands being transmitted. Values are sent from the USB port of the computer to PCBuddy where they are converted to a pulse train and then sent via serial cable to the trainer input of the radio control interface. To initiate this arrangement, we need to press a 'Trainer' button on the radio control, which then transfers control over to the software program.

6.2 PCBuddy

The PCBuddy Interface is designed so that it can be used with Futaba Radio Control equipment to enable remote robotic vehicles to be controlled using a computer program. At the transmitting end the equipment configuration is a Macintosh with the PCBuddy Interface plugged into a USB to serial interface, linked to an R/C transmitter via its training socket, or 'buddy-box interface'. At the receiving end an R/C receiver is connected to four electronic motor speed controllers, one for each motor on the DraganFlyer

6.2.1 Servo Control Frames

Figure 6.2 shows a typical servo control waveform. A typical R/C servo will provide, at its output arm, a mechanical movement of approximately ±45 degrees about a nominal centre position (Figure 6.2). The precise position within this range is determined by the width of the input pulse sent to it by the receiver and the range over which the input pulse varies is typically 1msec to 2msec. These limit values provide the extremes of movement and a pulse width of 1.5msec provides the nominal centre position. In order to assure best performance the servo requires that the control pulses are sent periodically and, although servos are usually fairly tolerant of the exact rate, around 50 times per second, or every 20msec, is normal (every 8msec is too fast and ever 30msec is too slow).

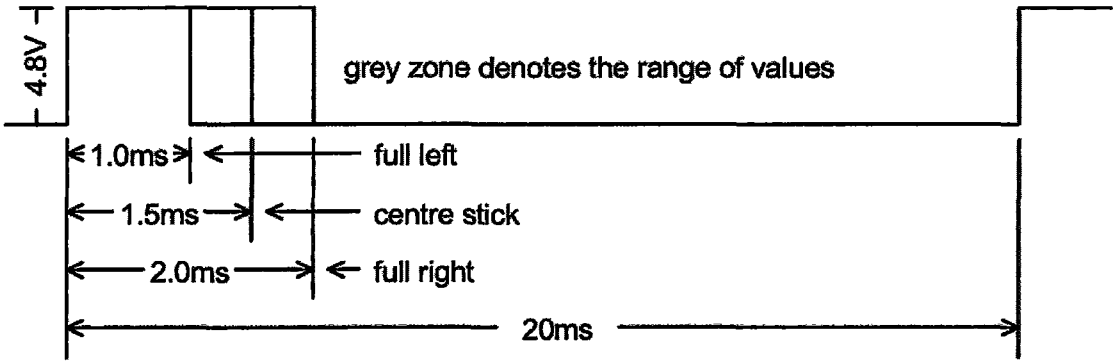

Figure 6.2 Servo control waveform

6.2.2 Multi-Channel Frame

In order to control many servos using a single radio link the servo control frames of multiple servos are combined to form a Multi- Channel frame as shown in Figure 6.3. This is the basic signal sent by the transmitter to the receiver, which then separates out the individual servo control frames and distributes them to each servo. After the final channel has been transmitted and before the start of the next frame there is a period containing no information, known as the sync time. The receiver uses this to establish the frame boundaries thus ensuring that the channel information is distributed correctly to each servo.

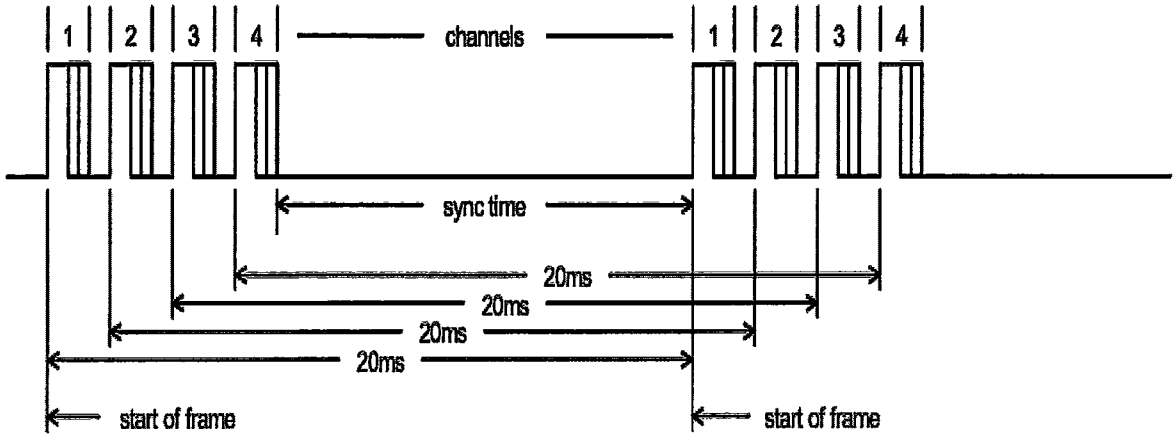

Figure 6.3. Multiple servos combine to form a multi-channel frame

6.2.3 Downloading Servo Position Data

The PCBuddy Interface also makes use of the sync time. Having sent the final channel terminating pulse to the transmitter it signals to the host computer that it is ready to receive a channel information update, if one is available. If the host has a new data set it transmits it and the new channel data received is presented to the transmitter in the next frame. Figure 6.4, shows an oscilloscope trace of a single 8-channel frame including a data update. If no update is ready then the servo control pulse information from the previous frame is repeated.

The upper trace, denoted R1 in Figure 6.4, shows the PCBuddy Interface output signal and depicts a full set of 8 channels. The cursor bars bracket Channel 8 with the dotted cursor aligned with the start of the final channel- terminating pulse.

The middle trace, denoted 2 in Figure 6.4, shows the CTS control signal sent to the PC by the PCBuddy Interface indicating that it is ready to receive new data.

The lower trace, denoted 1 in Figure 6.4, shows the activity on the RS232 transmit line from the PC with the individual data bytes clearly identifiable. The PC must respond within 1.2msec else the PCBuddy removes the CTS signal and ignores the data on the RS323 line.

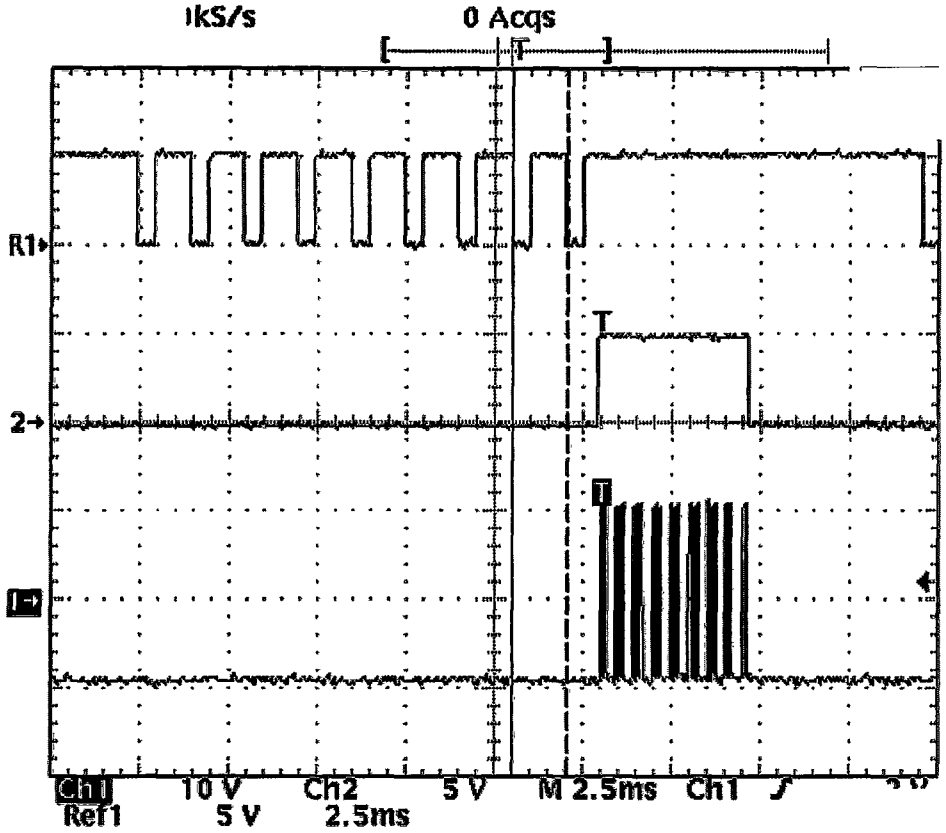

Figure 6.4. Single 8-channel frame

6.2.4 Modulating the Transmitted Carrier

The signal accepted by the transmitter, via its training socket, from the PCBuddy Interface is used to modulate the transmitted carrier. Depending on the design of a particular transmitter this signal may be of the form shown in Figure 6.3 or, possibly, an inverted version. The PCBuddy Interface may be factory configured to provide either form thus ensuring its use with the widest range of transmitters.

6.3 Software

As with the Inertial Navigation Sensor, PCBuddy requires an interface to be controlled as well. RC-Electronics provides a small PC application in its package however; we applied the above logic to create our application in LabVIEW. Several versions were created to overcome a synchronization problem that was being incurred during transmission of signals from the host computer to PCBuddy.

The PCBuddy signals to the host that it is ready for data by asserting clear to send (CTS). If the host takes longer than 1.2msec to respond, the PCBuddy ignores the response. The programme in the PC micro in PCBuddy cannot read serial data while it is generating the R/C pulse train and it does not have a hardware buffer.

We found that the best response that we could achieve with a Macintosh host, a USB to serial converter and a LabVIEW vi to read and respond to a change in CTS was 1.5msec. This appears to be due to USB protocol timing. So we have written a vi that polls the CTS signal to detect a change. When the change is detected it sends the character stream to the PCBuddy at 19.2Kbaud and then waits for 18msec (Figure 6.5).

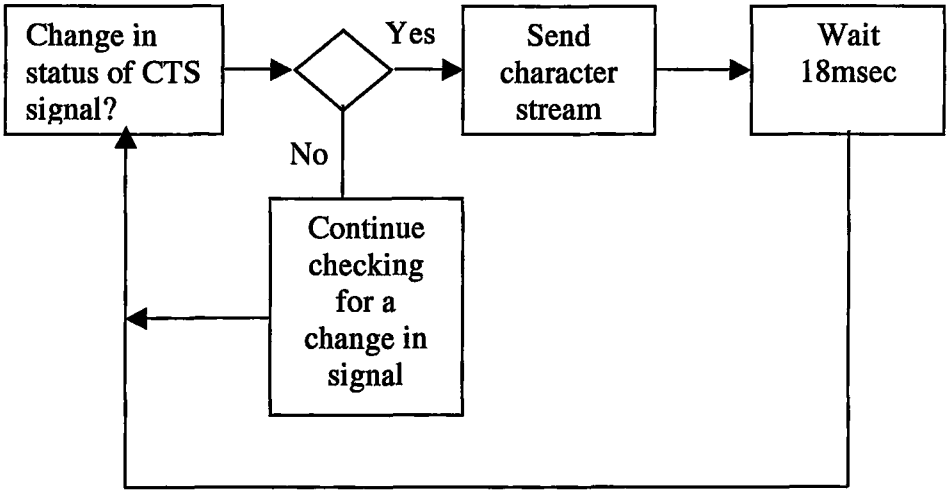

Figure 6.5. Block diagram representing output process of PCBuddy.vi

There are three parallel processes that need to be integrated so that each one's inputs can be controlled in real-time. The PCBuddy control panel allows real-time control (Figure 6.6) over command inputs transmitted to DraganFlyer. Data acquisition panel (Figure 6.7) presents data feedback in measurable units on graphs. The third process calculates velocities and positions for the INS acceleration and rate measurements.

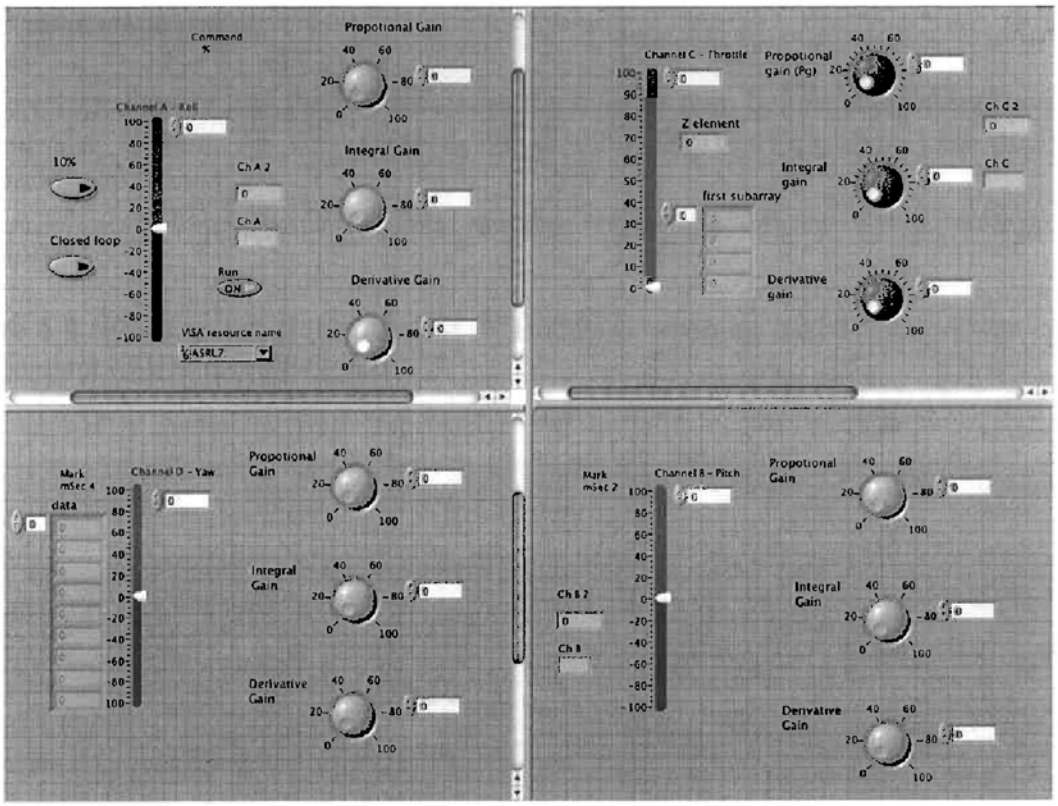

Figure 6.6. PCBuddy control panel

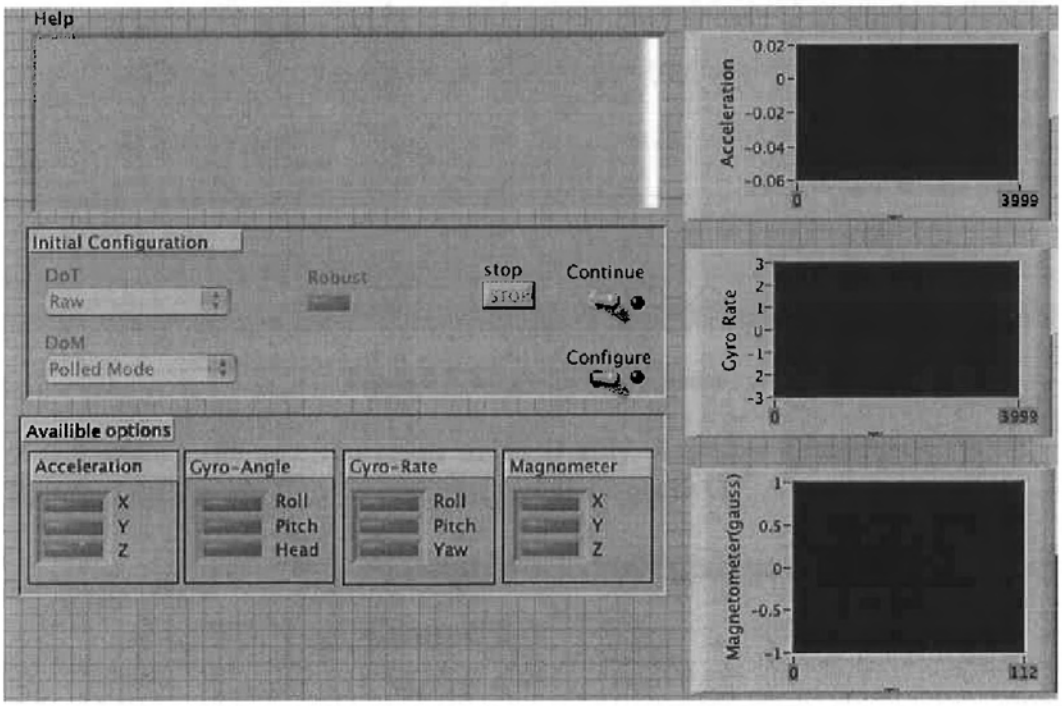

Figure 6.7. Data acquisition panel

6.3.1 Software stages

We developed and tested the PCBuddy interface on its own before integrating it with the rest of the system. In this section we discuss the configuration and design of 'PCBuddy.vi'.

Version 12

Figure 6.8 shows us the front panel of PCBuddy.vi when an initial version was created independent of 'DataAcquisiton.vi'.

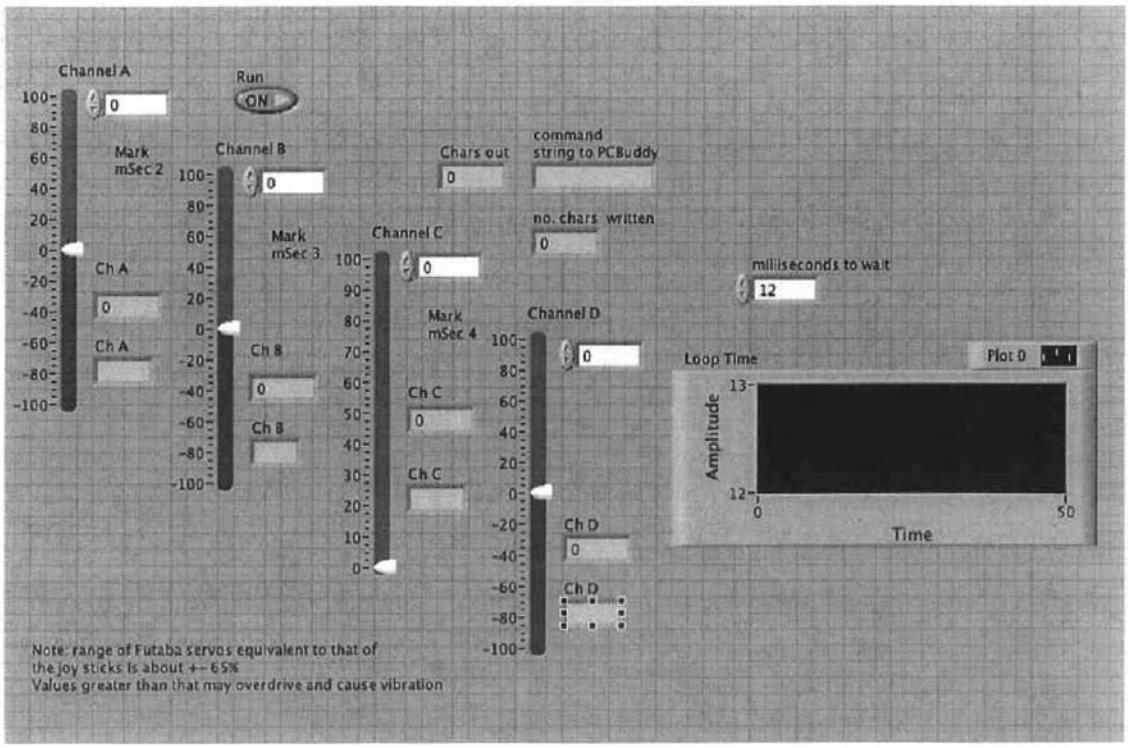

Figure 6.8. Single 8-channel frame

Figure 6.9 shows the configuration of PCBuddy.vi where Channel A, B, C and D correspond to roll, pitch, throttle and yaw inputs respectively.

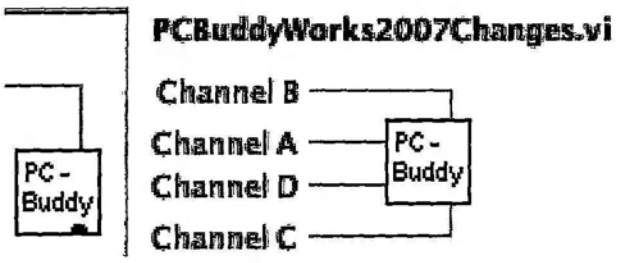

Figure 6.9. PCBuddy.vi

At this stage of development, we monitored PCBuddy.vi's performance on its front panel. Figure 6.8 shows a graph which displays the loop time for the CTS signal-polling loop. The y-axis on this graph displays a constant time value unless a CTS signal has been detected. When this occurs, the loop takes 1msec – 2msec longer than usual to execute and then returns to its normal rate. Another indicator was the DraganFlyer's response to the commands sent through this interface.

In a later stage of development (Section 7.2.1), we will use 'DataAcquisiton.vi' from ECS version 8.0 and modify it, such that both, PCBuddy.vi and DataAcquisiton.vi can run simultaneously in real-time with minimal time delays.

7.0 Control Software

In this chapter we will identify the control theory (Section 7.1) we have chosen and then describe our software (Section 7.2). We then test both, open-loop and closed-loop control in the laboratory with the DraganFlyer. The results are analysed to understand the performance of the system.

7.1 Control Theory

7.1.1 Control loop basics

The purpose of a control loop is to automate what an intelligent operator with a gauge and a control knob would do [19,20,21,22]. A control loop consists of three parts:

- Measurement by a sensor connected to the process (or the "plant"),
- Decision in a controller element,
- Action through an output device ("actuator") such as a control valve.

As the controller reads a sensor, it subtracts this measurement from the target value to determine the error. It then uses the error to calculate a correction to the process's input variable (the "action") so that this correction will remove the error from the process's output measurement.

A controller can be used to control any measurable variable, which can be affected by manipulating some other process variable. For example, it can be used to control temperature, pressure, flow rate, chemical composition, speed, or other variables.

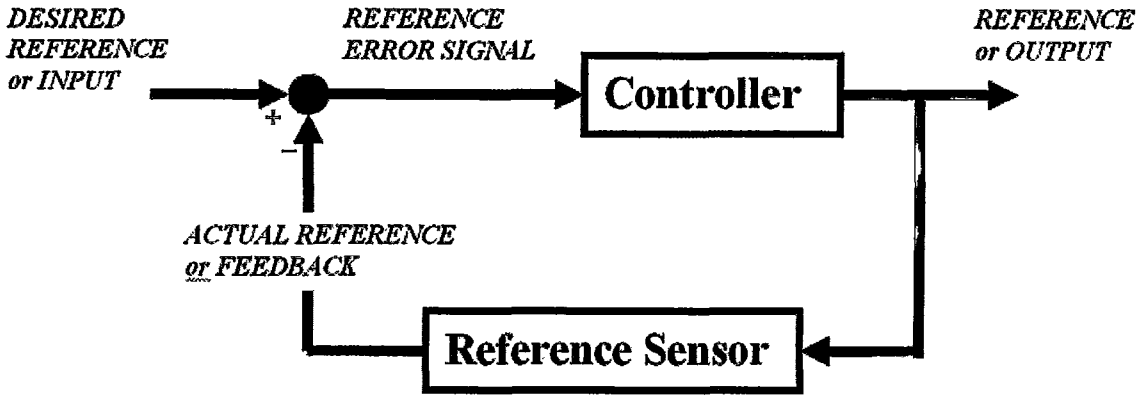

Figure 7.1 Generic model of a controller

A proportional-integral-derivative controller (PID controller) is a generic control loop feedback mechanism widely used in industrial control systems. A PID controller attempts to correct the error between a measured process variable and a desired target value by calculating and then outputting a corrective action that can adjust the process accordingly, based upon three parameters.

The PID controller calculation (algorithm) involves three separate parameters; the Proportional, the Integral and Derivative values. The proportional value determines the reaction to the current error, the integral determines the reaction based on recent errors and the derivative determines the reaction based on the rate by which the error has been changing. The weighted sum of these three actions is outputted to a control element such as the position of a control valve or power into a heating element.

7.1.2 PID controller theory

The PID control scheme is named after its three correcting terms, whose sum constitutes the output. The three terms are

Proportional term
The proportional term responds to a change in the process variable proportional to the current measured error value. The proportional response can be adjusted by multiplying the error by a constant K_p, called the proportional gain or proportional sensitivity. The gain is also frequently expressed as a percentage of the proportional band. The proportional term is written as:

$$m = K_p e = \frac{100}{PB} e \tag{7.1}$$

where:

m = output signal
K_p = proportional gain
e = error equal to (setpoint value - process variable)
PB = proportional band

A high gain results in a large response to a small error, a more sensitive system (Also called a narrow proportional band). Note that by setting the proportional gain too high, the system can become unstable,. In contrast, a small gain results in a small response to a large error, and less sensitive system (Also called a wide proportional band), which is undesirable as the control action may be insufficient to respond to system disturbances.
Finally pure proportional control will never theoretically settle at its target value, but will rather approach the target with a steady state error that is a function of the proportional gain, this is known as a steady state error.

Integral term
The contribution from the integral term is proportional to the past and current values and duration of the error signal. The integral term algorithm calculates the accumulated proportional offset over time that should have been corrected previously (finding the offset's *integral*). While this will force the signal to approach the setpoint quicker than a proportional controller alone and eliminate steady state error, it may also contribute to system instability, as the controller will always be responding to past values. This instability causes the process to overshoot the setpoint since the integral value will continue to be added to the output value, even after the process variable has reached the desired setpoint.

The responsiveness of the integral function can be calibrated to the specific process by adjusting the constant T_i, called the integral time.

The equation is written as:

$$m = \frac{1}{T_i} \int_0^t e(\tau)\, d\tau \tag{7.2}$$

where:

m = output signal
T_i: integral time
e = error equal to (setpoint value - process variable)

Although mathematically the integral starts at $t = 0$, it is often the case that the integral will be windowed, to prevent errors no longer considered to be relevant to the system from being accounted for. In that case, the lower limit of the integral is changed to $t_{twindow}$, where *twindow* is a specified constant.

Derivative term

The derivative term provides a braking action to the controller response as the process variable approaches the setpoint. To accomplish this the process error is predicted at a time in the future *Td*, calculated by analyzing the slope of error vs. time (i.e. the rate of change of error, which is its first derivative with respect to time) and adding the anticipated proportional term to the current correction.

Derivative control is used to reduce the magnitude of the overshoot produced by the integral component, but the controller will be a bit slower to reach the setpoint initially. As differentiation of a signal amplifies the noise levels, this mode of control is highly sensitive to noise in the error term, and can cause a noisy controlled process to become unstable.

By adjusting the constant, *Td*, the derivative time, the braking action sensitivity is controlled.

The derivative term is written as:

$$m = T_d \frac{de}{dt} \tag{7.3}$$

where:

m = output signal
T_d = derivative time
e = error equal to (setpoint value - process variable)

7.1.3 PID algorithm implementation

Parallel / "non-interacting" form
The PID algorithm can be implemented in several ways. The easiest form to introduce is the parallel or "non-interacting" form, where the P, I and D elements are given the same error input in parallel. The output of the controller (i.e. the input to the process) is given by

$$\text{Output(t)} = P_{\text{contrib}} + I_{\text{contrib}} + D_{\text{contrib}} \tag{7.4}$$

where P_{contrib}, I_{contrib}, and D_{contrib} are the feedback contributions from the PID controller, defined below:

$$P_{\text{contrib}} = K_p\, e(t) \tag{7.5}$$

$$I_{\text{contrib}} = \frac{1}{T_i}\int_0^t e(\tau)\, d\tau \tag{7.6}$$

$$D_{\text{contrib}} = T_d \frac{de}{dt} \tag{7.7}$$

where:

$e(t) = setpoint - measurement(t)$ is the error signal
τ is the time in the past contributing to the integral response
K_p, T_i, T_d are constants that are used to tune the PID control loop:

- K_p: Proportional Gain - Larger K_p typically means faster response since the larger the error, the larger the feedback to compensate.
- T_i: Integral Time - Smaller T_i implies steady state errors are eliminated quicker. The trade-off is larger overshoot: any negative error integrated during transient response must be integrated away by positive error before we reach steady state.
- T_d: Derivative Time - Larger T_d decreases overshoot, but slows down transient response.

In the ideal parallel form, the standard parameters T_i and T_d are replaced with (K_i and K_d).

Series / interacting form
Another representation of the PID controller is the series, or "interacting" form. This form essentially consists of a PD and PI controller in series, and it made early (analog) controllers easier to build.

7.1.4 Limitations

The PID controller algorithm itself has some limitations. Further practical application issues can arise from instrumentation connected to the controller.

One common problem is "integral windup". It might take too long for the output value to ramp up to the necessary value when the loop first starts up. There are many schemes for

solving this issue. For example sometimes this can be compensated for with a more aggressive differential term. Sometimes the loop has to be "preloaded" with a starting output. Another option is to disable the integral function until the measured variable has entered the proportional band. A more common method is to prevent the integral term accumulating above or below certain pre-determined bounds, or by limiting the time period that the integral is considered over.

Another problem with the differential term is that small amounts of noise can cause large amounts of change in the output. Sometimes it is helpful to filter the measurements, with a running average, or a low-pass filter. However, low-pass filtering and derivative control cancel each other out, so reducing noise by instrumentation means is a much better choice. Alternatively, the differential band can be turned off in most systems with little loss of control. This is equivalent to using the PID controller as a *PI* controller.

The proportional and differential terms can also produce undesirable results in systems subjected to instantaneous "step" inputs (such as when a computer changes the setpoint).

Digital implementations of a PID algorithm may have limitations owing to the sampling rate of the data, and the limits of internal calculation and precision. For example, very old programmable logic controller (PLC) systems may have used only 12 or 16 bits to represent internal variables. Additionally, some software implementations do not correctly handle internal overflow or extreme values, or may arbitrarily limit the values for the adjustable gain parameters.

Another problem faced with PID controllers is that they are linear. Thus performance of PID controllers in non-linear systems (such as HVAC systems) is variable. Often PID controllers are enhanced through methods such as scheduling or fuzzy logic.

7.2 Software

The Figure 7.2 displays how all the processes in the system use queues to communicate arrays of feedback values in real-time. We experimented with closed-loop control as well as open loop control. During closed-loop control, a global queue is used to access feedback in the Data acquisition panel so that it can be used to by our PID control law to generate the next command value, which will reduce the error between the reference value and feedback value.

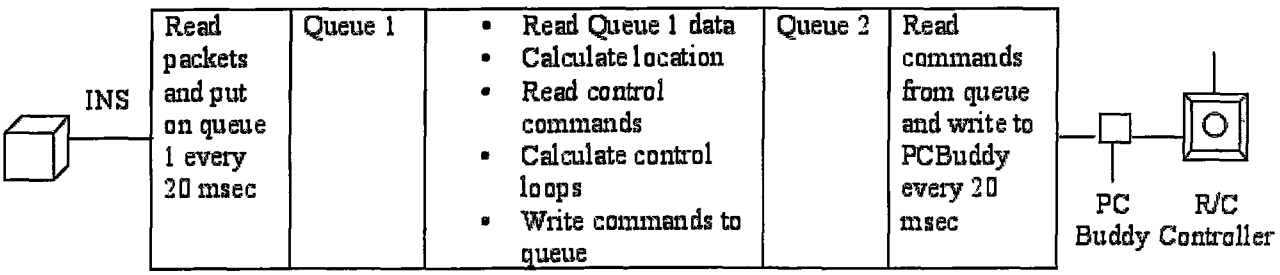

Figure 7.2 Architecture of Software

7.2.1 Software stages

Version 13

We modify 'DataAcquitions.vi' such that 'PID.vi' is called and executed before data from INS queue is read.

The 'PID.vi' is introduced in Version 13 and its configuration, as shown in Figure 7.3, is such that values for proportional gain, integral gain, derivative gain, command (position or angle) feedback and reference are taken as inputs, are converted into finite difference form (Equations 7.8, 7.9, 7.10) so that only an incremental calculation is done each time slip. These are implemented in the PID.vi, shown in Figure 7.4, so these inputs and *output t* is sent to PCBuddy.vi as Channel A, B, C or D.

$$P_{t^{contrib}} = K_p e_t \tag{7.8}$$

$$D_{t^{contrib}} = K_d d(e_t - e_{t-1}) \tag{7.9}$$

$$I_{t^{contrib}} = I_{t-1contrib} + K_i e_t \tag{7.10}$$

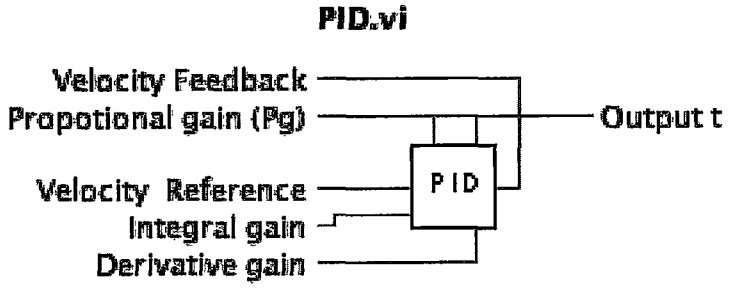

Figure 7.3. PID.vi

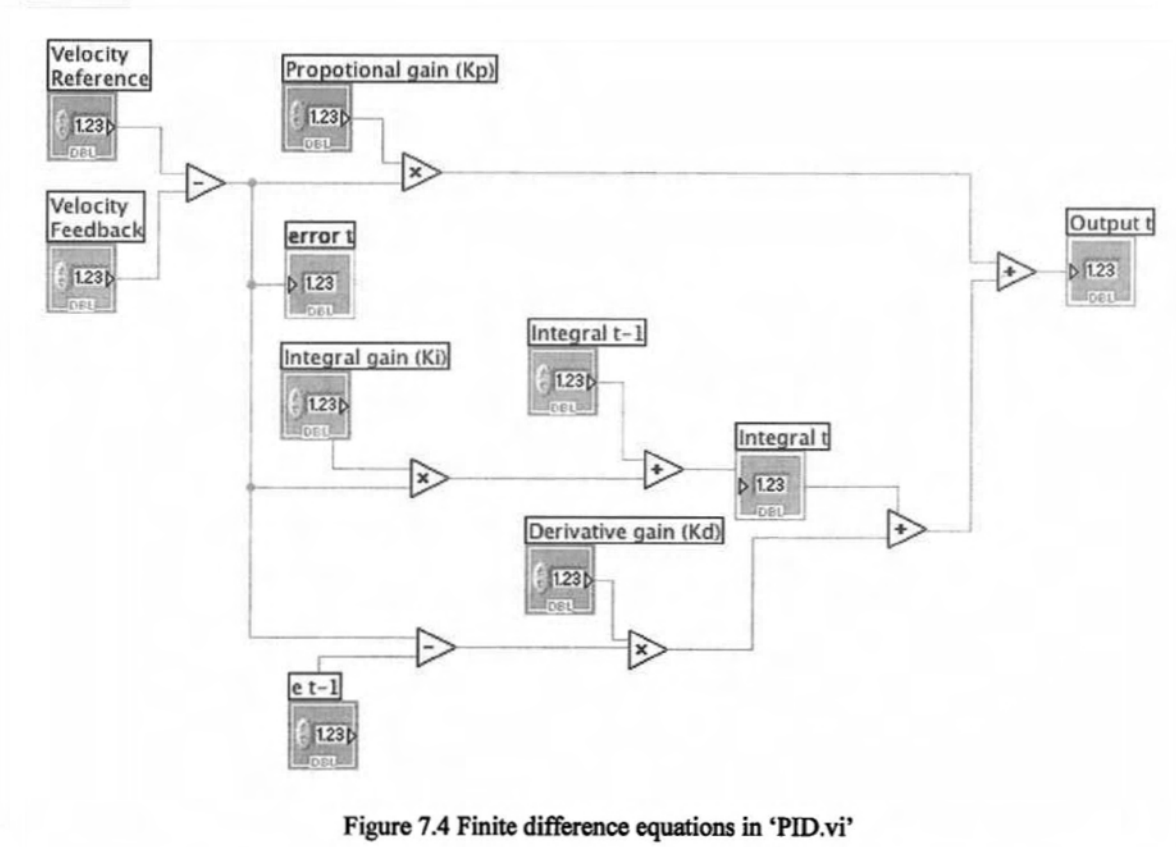

Figure 7.4 Finite difference equations in 'PID.vi'

Figure 7.5 demonstrates how both, PCBuddy.vi and PID.vi are executed in DataAcquisiton.vi. Though PID.vi takes command feedback as an input, closed-loop control had not been implemented in this version, therefore PID.vi could not produce accurate results.

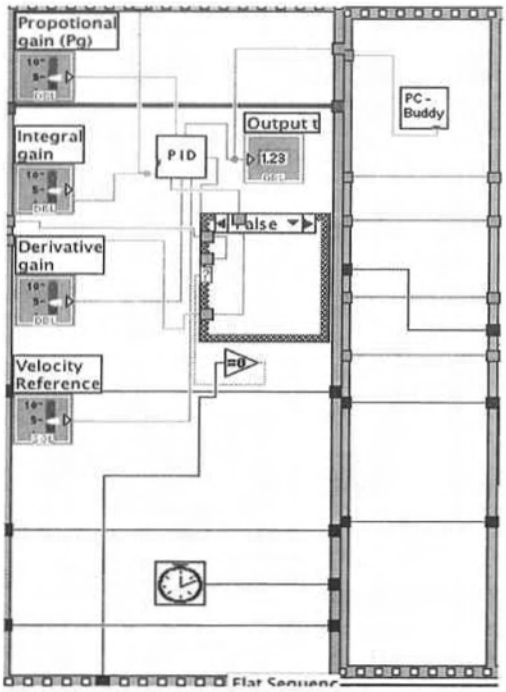

Figure 7.5. Execution loop within 'DataAcquisiton.vi'

Version 14

In version 13, we could simply wire feedback to PCBuddy.vi but since we needed to execute this process in parallel to DataAcquisition.vi and yet independent of it, we opted to use global arrays to transport data between two processes. Figure 7.6 shows a global array transferring values from DataAcquisition.vi to PID.vi through PCBuddy.vi.

Also, 'PID.vi' is now executed inside 'PCBuddy.vi' whenever a command needs to be transmitted to the radio control

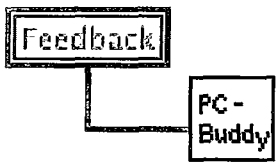

Figure 7.6 global array, 'Feedback', carried values from DataAcquisiton.vi to PCBuddy.vi, through to PID.vi

Version 15

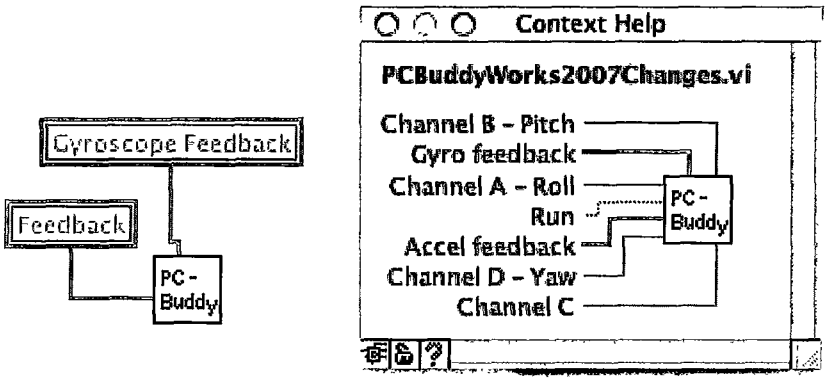

Figure 7.7 PCBuddy.vi configuration

Version 15 is the same as version 14 apart from two modifications. They are:

- We have allocated a global array connector for angle feedback from 'DataAcquisiton.vi'. (Figure 7.7)
- We added a switch that allows us to give a 10% impulse to the command reference after PID calculations.

Version 16

In this version we provided the option to send commands to the radio control via a closed-loop control or a open loop control. An open loop control will simply take the given reference and transmit it. A closed-loop control will subtract the reference from the feedback and the error is sent as the next reference. We can send a 10% impulse with either type of control.

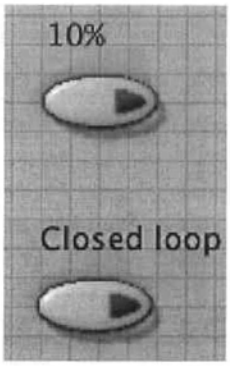

Figure 7.8 Closed-loop and 10% switch control on PCBuddy.vi interface

Version 17

In this version global arrays are replaced with FIFO (First-In-First-Out) queues. The benefit of this is:

- Queues buffer data thus they are capable of handling huge chunks of information in a real-time application that does constant large processing
- They are simpler to queue, de-queue and maintain

Figure 7.9 demonstrates that because all data packets are now transferred between processes using queues, 'PCBuddy.vi' does not require parameters when it is called. This reduces run-time memory load and allows for faster execution.

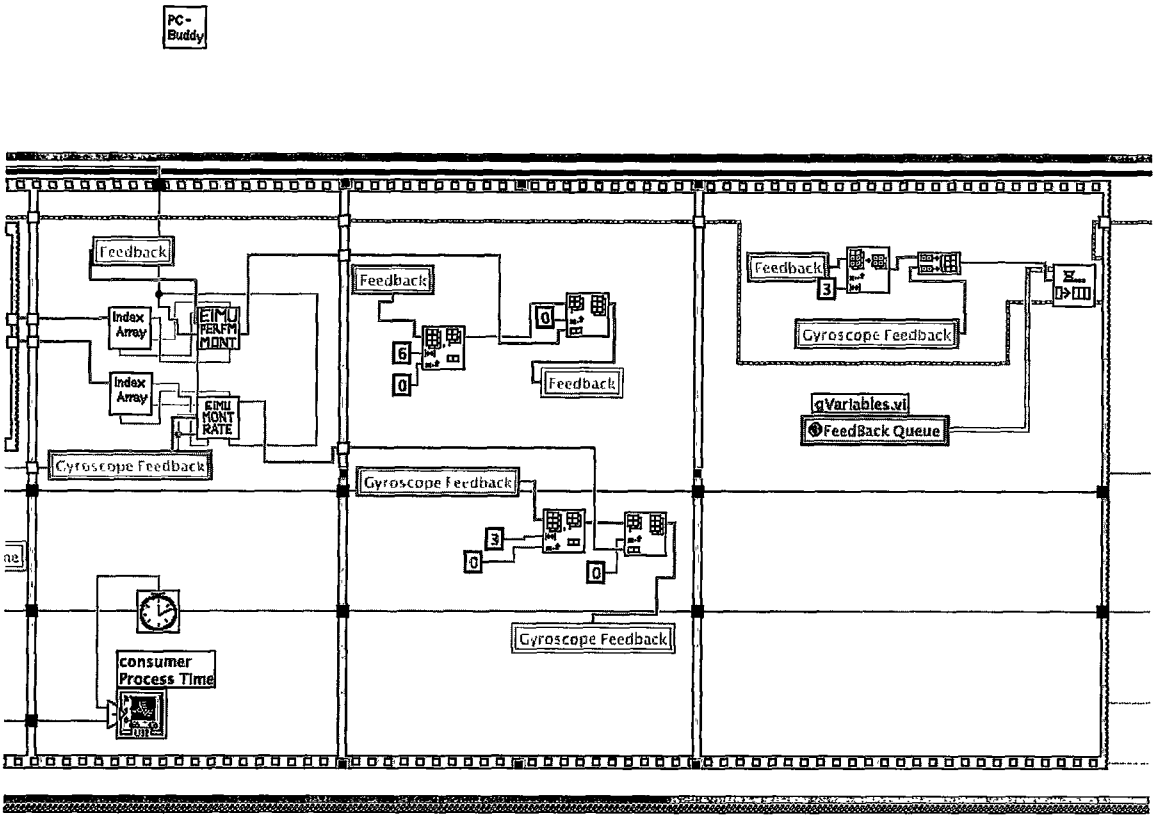

Figure 7.9 PCBuddy executed from DataAcquisiton.vi without any input parameters

7.3 System Advantages/Disadvantages

A difficult aspect of this system is that we have to use 4 controls on the application to control six degrees of freedom. This is a difficult challenge to overcome in order to achieve autonomous control

7.4 Open-Loop Control

7.4.1 Test Design

These experiments are designed to check the response of DraganFlyer to directives sent by host machine. Also, the results of these experiments will demonstrate whether any changes need to be made to the implemented control law. If not, then we ideal results would show that DraganFlyer responds to host machine without much time delay and INS sends feedback to ECS. This feedback should be processed within 20msec and the next command reference should be produced and transmitted by PCBuddy.

Test Type	Test Name	Experiment Description	Limitations	Equipment	Expected Test Result
Open loop testing	Roll	Send throttle command from radio control and switch over to interface, give a roll value with proportional gain	Due to an electronic fault in the DraganFlyer, there are occasional jerks during the experiment. This put noise into motion of robot.	This experiment is carried out by hanging the DraganFlyer on a tether in the laboratory. This is done to cause minimal damage while flying and achieve a certain amount of control over it.	Gyroscope graph on Data Acquisition panel should display real-time feedback beginning with a negative x-axis swing. Response time of DraganFlyer to commands sent from PCBuddy control interface should not be delayed.
	Pitch	Send throttle command from radio control and switch over to interface, give a throttle and pitch value with proportional gain	Due to an electronic fault in the DraganFlyer, there are occasional jerks during the experiment. This put noise into motion of robot.	This experiment is carried out by hanging the DraganFlyer on a tether in the laboratory. This is done to cause minimal damage while flying and achieve a certain amount of control over it.	Gyroscope graph on Data Acquisition panel should display real-time feedback beginning with a negative y-axis swing. Response time of DraganFlyer to commands sent from PCBuddy control interface should not be delayed.

Test Type	Test Name	Experiment Description	Limitations	Equipment	Expected Test Result
	Yaw	Send throttle command from radio control and switch over to interface, give a yaw value with proportional gain	Due to an electronic fault in the DraganFlyer, there are occasional jerks during the experiment. This put noise into motion of robot.	This experiment is carried out by hanging the DraganFlyer on a tether in the laboratory. This is done to cause minimal damage while flying and achieve a certain amount of control over it.	Gyroscope graph on Data Acquisition panel should display real-time feedback beginning with a negative z-axis swing. Response time of DraganFlyer to commands sent from PCBuddy control interface should not be delayed.

Table 7.1. Description of each test plan, its limitations and expected result

7.4.2 Test Result

Impulse during roll

In this experiment, we hung the DraganFlyer on a tether, attached to the roof of the laboratory. The PCBuddy interface, Figure 6.6, is set up with a proportional gain of 1.5 and initial roll reference of -20. Using the radio control, we gave DraganFlyer initial throttle and then switched over to the PCBuddy interface and sent a roll input as per the defined reference. The gyro rate graph displayed a wave of motion in the x-axis beginning with a negative roll, Figure 7.11. From sample 252-288 the DraganFlyer is rotating in the negative x-axis. From sample 288 onwards, the robot swings back due to the roll command being reduced to zero and the force applied by the tether.

After the impact of the roll reference, the tether causes the robot to swing in a pendulum like motion as described in Figure 7.10.

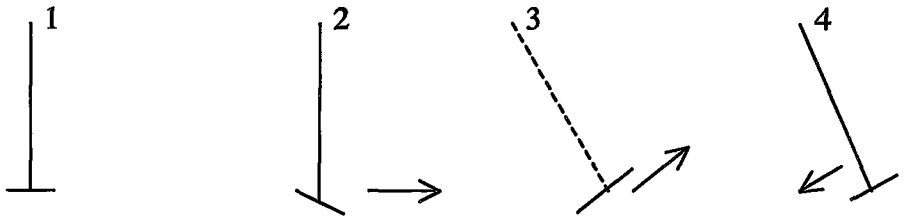

Figure 7.10 shows us the steps of DraganFlyer's response to roll a command

where,

1) is the robot's initial position before any commands are sent
2) is its start-off position when throttle is given
3) is its motion in response to a roll, pitch or yaw command
4) is when the robot's swings back due to the tether

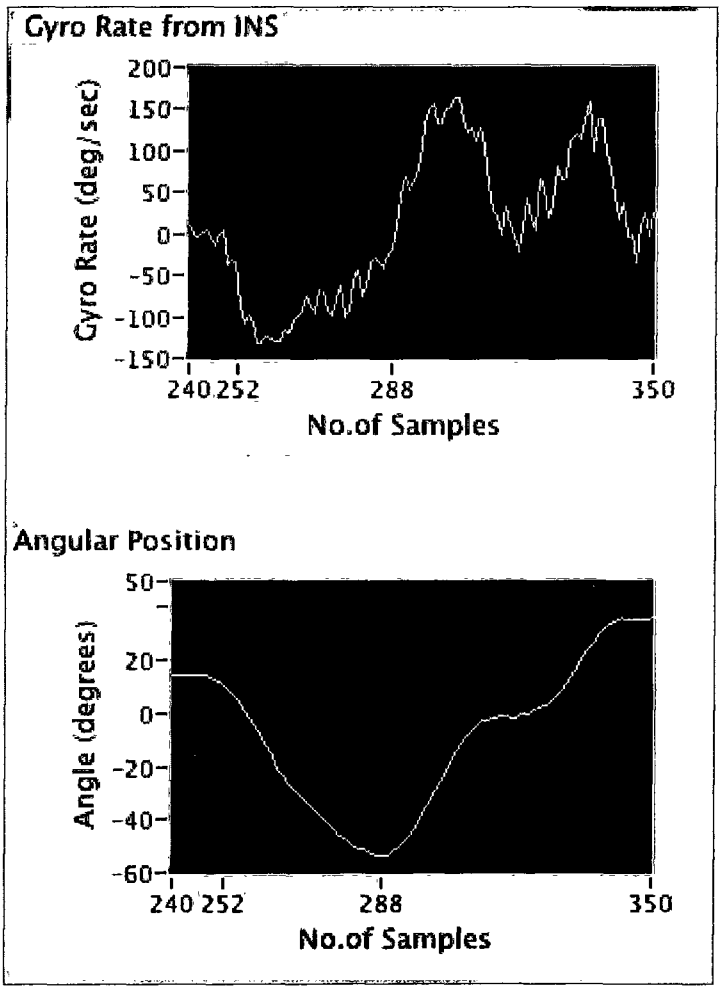

Figure 7.11 Shows response of DraganFlyer to roll command from Samples 252 - 288

Impulse during pitch

In this experiment, we hung the DraganFlyer on a tether, attached to the roof of the laboratory. The PCBuddy interface, Figure 6.6, is set up with a proportional gain of 1.5 and initial pitch reference of -20. Using the radio control, we gave DraganFlyer initial throttle and then switched over to the PCBuddy interface and sent a pitch input as per the defined reference. The gyro rate graph displayed a wave of motion in the y-axis beginning with a negative pitch, Figure 7.12.

From sample 273-300 the DraganFlyer is pitching forwards in the negative x-axis in response to the command sent from PCBuddy. Sample 300 onwards, the robot swings backwards in reverse motion due to the pitch and the force applied by the tether.

Here also, the tether causes the DraganFlyer to swing backwards as described in Figure 7.10.

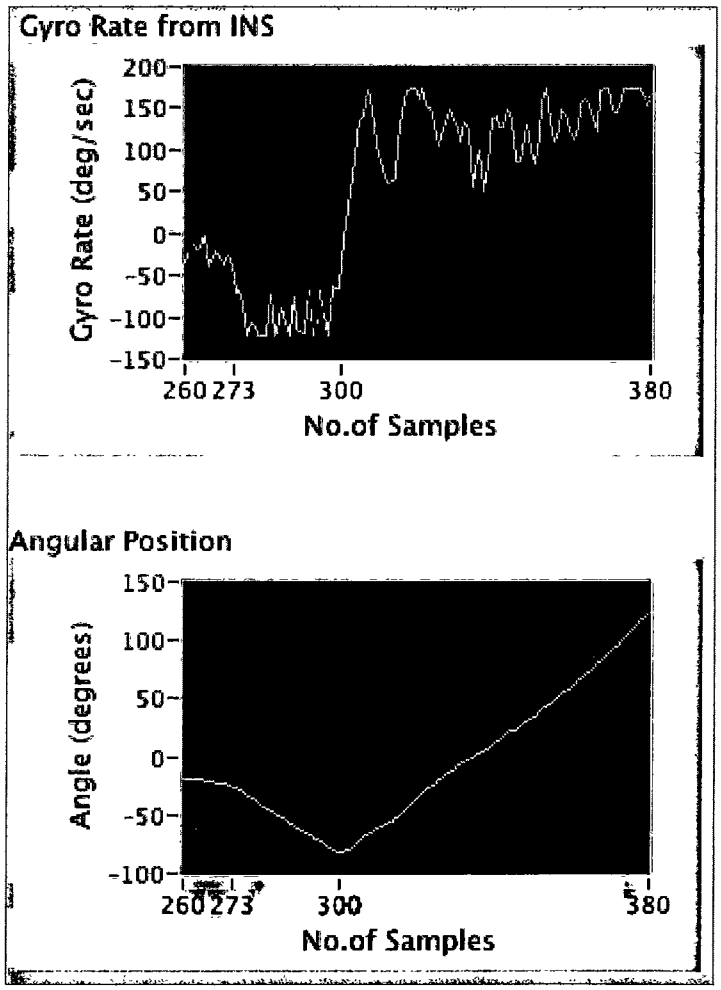

Figure 7.12 Shows response of DraganFlyer to pitch command from Samples 273 - 300

Impulse during yaw

In this experiment, we hung the DraganFlyer on a tether, attached to the roof of the laboratory. The PCBuddy interface, Figure 6.6, is set up with a proportional gain of 1.5 and initial yaw reference of -20. Using the radio control, we gave DraganFlyer initial throttle and then switched over to the PCBuddy interface and sent a yaw input as per the defined reference. The gyro rate graph displayed a wave of motion in the z-axis beginning with a negative yaw, Figure 7.13.

The angular position graph illustrates that the DraganFlyer was already in yaw when the command was sent. From sample 503-584 the DraganFlyer is rotating around the z-axis in response to the yaw command sent from PCBuddy. At sample 584, the yaw command is tuned to zero. During yaw the DraganFlyer was in circular motion and when the yaw command was suddenly stopped, it swung in reverse motion due to the force applied by the tether.

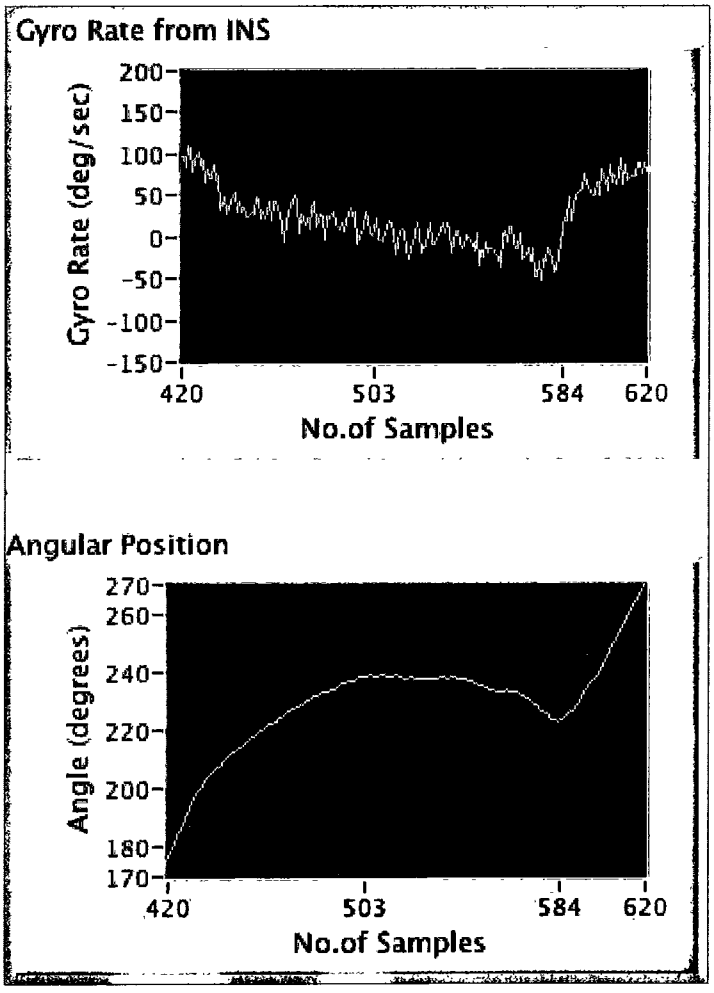

Figure 7.13 Shows response of DraganFlyer to roll command

7.5 Closed-Loop Control

7.5.1 Software Design

Closed loop was implemented as the last stage of the software, thus it exists only in ECS Version 17. Figure 6.6 shows the interface that allows a user to enter a proportional, derivative and integral gain for throttle, roll, pitch and yaw. The software sends commands to DraganFlyer via closed-loop control, the reference value entered by the user is subtracted from the feedback reference and the resultant error is sent to 'PID.vi' as feedback. The 'PID.vi' then calculates the next velocity reference and this output is sent to radio control via PCBuddy.

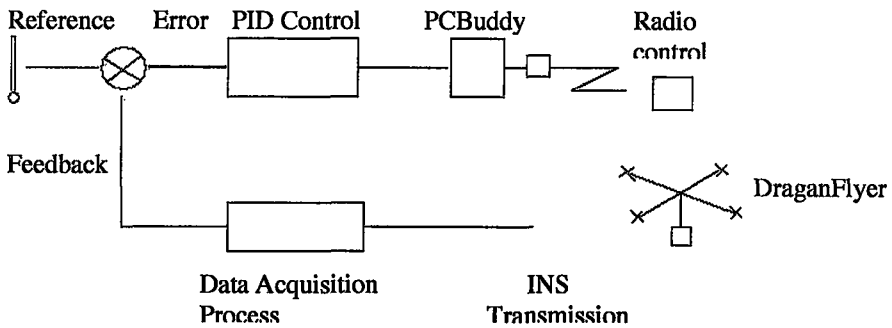

Figure 7.14 Shows controller, plant and processor.

7.5.2 Test Design

Test Type	Test Name	Experiment Description	Limitations	Equipment	Expected Test Result
Closed loop testing	Roll	Set PID values for test. Turn on the radio control and DraganFlyer. Switch over to host computer, give a throttle value to start the propellers. Maintain this throttle and then input a roll reference	Due to an electronic fault in the DraganFlyer, there are occasional jerks during the experiment. This put noise into motion of robot.	This experiment is carried out by hanging the DraganFlyer on a tether in the laboratory. This is done to cause minimal damage while flying and achieve a certain amount of control over	Gyroscope graph on Data Acquisition panel should display real-time feedback with a wave-like graph on the x-axis. Response time of DraganFlyer to commands sent from PCBuddy control interface

		with proportional gain. Feedback from INS is sent to PID controller so that it can produce next command input. Re-run with different PID values to improve.		it.	should not be delayed.
	Pitch	Set PID values for test. Turn on the radio control and DraganFlyer. Switch over to host computer, give a throttle value to start the propellers. Maintain this throttle and then input a roll reference with proportional gain. Feedback from INS is sent to PID controller so that it can produce next command input. Re-run with different PID values to improve.	Due to some electronic fault in the DraganFlyer, there are occasional jerks during the experiment. This put noise into motion of robot.	This experiment is carried out by hanging the DraganFlyer on a tether in the laboratory. This is done to cause minimal damage while flying and achieve a certain amount of control over it.	Gyroscope graph on Data Acquisition panel should display real-time feedback with a wave-like graph on the y-axis. Response time of DraganFlyer to commands sent from PCBuddy control interface should not be delayed.

Test Type	Test Name	Experiment Description	Limitations	Equipment	Expected Test Result
	Yaw	Set PID values for test. Turn on the radio control and DraganFlyer. Switch over to host computer, give a throttle value to start the propellers. Then input a yaw reference with proportional gain. Feedback from INS is sent to PID controller so that it can produce next command input. Re-run with different PID values to improve.	Due to some electronic fault in the DraganFlyer, there are occasional jerks during the experiment. This put noise into motion of robot.	This experiment is carried out by hanging the DraganFlyer on a tether in the laboratory. This is done to cause minimal damage while flying and achieve a certain amount of control over it.	Gyroscope graph on Data Acquisition panel should display real-time feedback with a wave-like graph on the z-axis. Response time of DraganFlyer to commands sent from PCBuddy control interface should not be delayed.

Table 7.2 Description of each test plan, its limitations and expected result

7.5.3 Experiments

The experiments described in Section 7.5.2 could not be executed due to a rotor control fault. A crash during the previous experiments caused an intermittent fault that could not be resolved in time for testing.

We made various attempts to solve the problem.

1. Checked electronics as far as we could without a circuit diagram but could not determine cause

2. Bought from USA new control electronics (Figure 7.15) but they failed when we tried to use them. They had an added feature - light sensors. We don't know the cause of incompatibility but it may indicate a motor fault. Due to the lack of time, could not pursue the problem any further in this thesis.

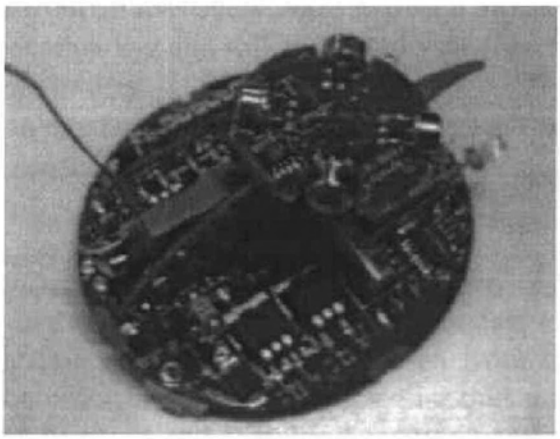

Figure 7.15 New electronics with light sensors

Since a solution was not found, we concluded that it would be best to perhaps replace the existing DraganFlyer model for further experimentation. Again, this was not a feasible solution in the given timeframe for this thesis.

After analysing the DraganFlyer's behaviour and response during testing, we arrived at a few possible causes of intermittent rotor control fault. They are as listed below

1. It could be a malfunction in the radio interface but this is hard to find or prove

2. Low battery voltage has often been a cause of low response times.

3. We also suspected a failed component or dry joint in electronics and hence bought new electronics.

4. Intermittent low motor resistance or motor sticking (unlikely because only closed loop or controller is via gyro – motor voltage open loop).

8.0 Conclusion

Here we summarize the results of this project and suggest future work.

8.1 Summary

In this project we have successfully developed a software architecture that allows a user to control a hovering robot from a remote computer using open-loop control.

We began by understanding the problem and identifying the hardware framework required to establish a bi-directional communication link that will meet the performance criteria of the project (see Chapter 2 and 3). Once the electrical set-up was complete and functional, we moved our focus to identifying the specification of the software. As we had described in Chapter 1, our primary specification was,

1. Receiving values from an INS every 20msec over a 38KB serial radio link
2. Calculate robot location xyz and roll, pitch and yaw coordinates
3. Display INS values to user
4. Read user target inputs for throttle, roll, pitch and yaw
5. Calculate output control values using control laws
6. Send control values to robot over R/C radio link every 20msec

After analysing our requirements, we realized that we needed a data flow model that handles the flow of data in the form of queues from one process to another. This was very important because each data set had to be transferred from one function to another simultaneously and in the order that it was processed in. thus a First-In-First-Out queue system was implemented to input and output all readings and measurements from the INS.

Requirements 1-3 were met in the first phase of development, i.e. Chapter 4. Requirements 4-6 were met in the second phase, i.e. Chapter 7. Implementing such a data flow model in software was an iterative process, as shown in each of these chapters, and many versions were created and tested before the best-suited ECS (EiMU Control Software) was selected.

Chapter 7 discusses PID control theory and why it would be the most effective solution for closed loop control of the helicopter. The test results in Section 7.4 prove that our control design was quite efficient. These open-loop tests were conducted smoothly using the interface.

Effectively, this thesis has contributed to the field of robotics by proving that a remote host is capable of measuring the motion of an indoor hovering helicopter and enabling remote piloting.

8.2 Future work

These are suggestions for future work with the DraganFlyer. One is to build a framework for complete autonomous control of the craft using a remote host. This may be done using the software libraries that we have developed in this project since they are generic.

A second is to develop AI control for high-level commands, once autonomous flight is achieved.

9.0 References

[1] Phillip M^{c}Kerrow, Modelling the DraganFlyer four-rotor helicopter, Proceedings of 2004 IEEE International Conference on Robotics & Automation, April 2004

[2] W. E. Green, P. Y. Oh, Keith Sevcik and G. Barrows, Autonomous Landing for Indoor Flying Robots Using Optic Flow, Proceedings of IMECE'03, 2003 ASME International Mechanical Engineering, November 15–21, 2003

[3] Nicoud, J.D., Zufferey, J.C., "Toward Indoor Flying Robots", IEEE, International Conference on Robots and Systems, Lausanne, pp. 787- 792, October 2002.

[4] S. Puntuntan and M. Parnichkun, Control of Heading Direction and Floating Height of a Flying Robot, Industrial Technology, 2002, IEEE ICIT '02, Page(s): 690 - 693 vol.2

[5] Samir Bouabdallah, Pierpaolo Murrieri and Roland Siegwart, Design and Control of an Indoor Micro Quadrotor, Proceedings of the 2004 IEEE International Conference on Robotics and Automation, New Orleans, L.A, April 2004

[6] P.Pounds, R. Mahony, P.Hynes & J.Roberts, Design of a Four-Rotor Robot, Proceedings 2002 Australian Conference on Robotics and Automation, 27-29 November 2002

[7] J. M. Grasmeyer and M. T. Keennon, Development of the Black Widow Micro Air Vehicle, 39th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, Jan. 2001.

[8] Andrew H. Fagg, M. Anthony Lewis, J. F. Montgomery, George A. Bekey, The USC Autonomous Flying Vehicle: An Experiment In Real-Time Behavior- Based Control, Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent, Robots and Systems, July 26-30 1993, pp. 1173-80

[9] Paul Y. Oh, William E. Green and Geoffrey Barrows, Closed Quarter Aerial Robot Prototype to Fly In and Around Buildings, Int. Conference on Computer, Communication and Control Technologies, Vol.5, pp. 302-307, July 2003

[10] D. Strelow and S. Singh, Online Motion Estimation from Image and Inertial Measurements, In Proceedings INERVIS: Workshop on Integration of Vision and Inertial Sensors, Portugal, June 2003

[11] S. Doncieux, J. B. Mouret, L. Muratet, J. A. Meyer, The ROBUR project: towards an autonomous flapping-wing animat, Proceedings of the Journées MicroDrones, 2004

[12] Fumiya Iida, Biologically Inspired Visual Odometer for Navigation of a Flying Robot, Robotics and Autonomous Systems, 30 September 2003, vol. 44, no.3, pp. 201-208(8)

[13] Pedro Castillo, Alejandro Dzul, and Rogelio Lozano, Real-Time Stabilization and Tracking of a Four-Rotor Mini Rotorcraft, IEEE Transactions On Control Systems Technology, VOL.12, NO. 4, July 2004

[14] Erdinc Altug and James P. Ostrowski, Control of a Quadrotor Helicopter Using Visual Feedback, Proceedings of the 2002 IEEE, International Conference on Robotics & Automation, Washington, DC . May 2002

[15] L. Schenato, W.C. Wu and S.S. Sastry, Attitude Control for a Micromechanical Flying insect via sensor Output Feedback, IEEE Transactions on Robotics and Automation, February 2004

[16] A. Ollero, J. Ferruz, F. Caballero, S. Hurtado and L. Merino, Motion compensation and object detection for autonomous helicopter visual navigation in the COMETS system

[17] Bouabdallah, S., Noth, A. and Siegwart, R., PID vs LQ Control Techniques Applied to an Indoor Micro Quadrotor. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Sendai, Japan, 2004.

[18] J. Roberts, P. Corke and L.Overs, EiMU – User Manual Version 0.4 (firmware version 00.11), CSIRO Manufacturing and Infrastructure Technology, June 2003

[19] Liptak, Bela (1995), Instrument Engineers' Handbook: Process Control. Radnor, Pennsylvania, Chilton Book Company, 20-29.

[20] Van, Doren, Vance J., "Loop Tuning Fundamentals". Control Engineering, July 01, 2003

[21] Sellers, David. An Overview of Proportional plus Integral plus Derivative Control and Suggestions for Its Successful Application and Implementation
5 May 2007.

[22 Graham, Ron, PID Control Tuning, 5 May 2007.

[23] Takaya, T. Kawamura, H. Minagawa, Y. Yamamoto, M. Ouchi, A Motion, Control in Three Dimensional Round System of Blimp Robot, SICE-ICASE, 2006. International Joint Conference , Oct. 2006, Page(s): 1291-1294

[24] Floreano, D., Nicoud, J.D., Beyeler, A., Klaptocz, A., Zufferey, J.C., A 10-gram Microflyer for Vision-based Indoor Navigation , IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'2006), Beijing, China, 9-15 October (2006), p. 3267-3272, 2006

10.0 Appendices

10.1 Appendix A – Accelerometer Test results

A.1 *x*-axis negative Distance

In this experiment, we physically moved the DraganFlyer along a desk in the negative x direction a distance of 30cm. Then we measured the displacement with a ruler to be 39cm. This difference in estimated displacement and actual displacement could account for as human error, since a person manually pushes the DraganFlyer. The measurement of the same displacement with the INS system is (-0.37m – (-0.02m)) = -0.39m = 39cm.

In Figure 4.30 we can see negative x acceleration, as measured by the INS, in the top graph. It is similar to a sine wave with peak acceleration at approximately –9m/s2. Acceleration is integrated to produce the velocity graph. Expectantly, the velocity increases smoothly and then decreases as the DraganFlyer is brought to a stop. The distance graph at the bottom of Figure.4.30 is a result of doubly integrating acceleration. It shows smooth motion in the negative direction, as expected. In comparison to the positive distance graph at the bottom of Figure.4.28 this graph has minimum drift once the DraganFlyer is at rest, i.e. 0.01cm every 15 samples.

This drift causes a velocity error of 0.01m/s when the DraganFlyer is at rest after motion, though before motion there is no drift detected and velocity measures 0m/s. No error is visible in the acceleration graph but the average of samples after motion is 0 indicating no change in zero.

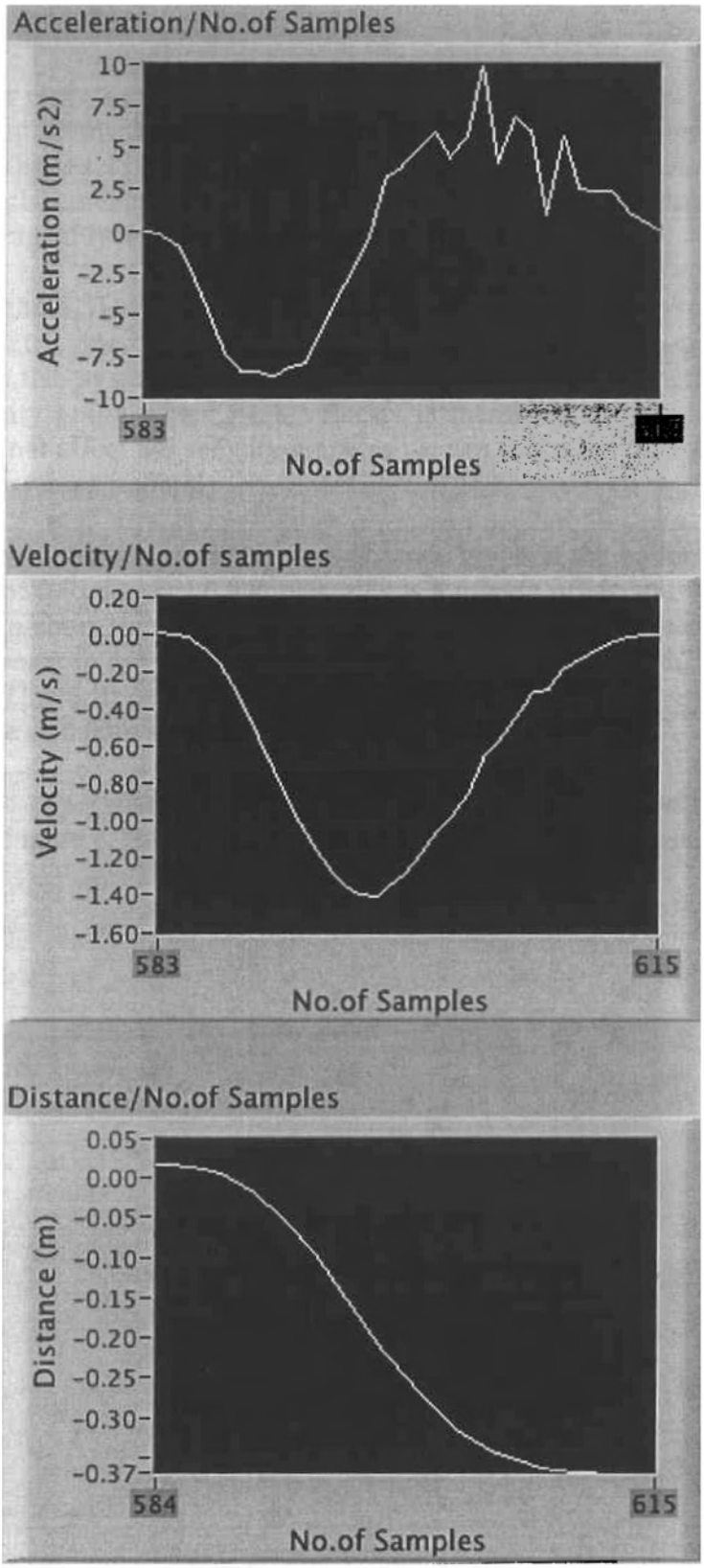

Figure 4.30 Shows displacement of –39cm on x-axis

A.2 *y*-axis positive Distance

The DraganFlyer is physically moved in positive *y* direction a distance of 30cm. We then measured the displacement with a ruler to be 42cm. However, measurement of the same displacement with the INS system is (0.06m – (-0.37m)) = 0.42m = 42cm. The difference in estimated displacement and actual displacement can be accounted for by extra thrust applied to the DraganFlyer while physically pushing it.

Referring to Figure 4.31 we can see positive *y* acceleration in the top graph. It is not as smooth as expected due to jerks concurred by the INS during motion. Deceleration measured by the INS system also is not smooth. Acceleration is integrated to produce a pulse-like velocity graph where peak velocity is measured, as 1.36m/s. Jerks on the acceleration did not affect the velocity graph as we can see in the second graph of Figure 4.31. Velocity increases smoothly

Velocity is further integrated to generate a distance graph at the bottom of Figure .4.31. There is a drift of 0.01m every 6 samples, this is a minute offset and will not affect the control loop in a short time, but if left to integrate it will affect it over a long time. Immensely, however in order to enhance real-time communication, this offset will have to be averaged. This drift causes velocity to continue rising, after DraganFlyer has come to a stop, at a rate or 0.03m/s every 4 samples.

In comparison to experiments in the positive *x* direction, drift in the bottom graphs of Figure 4.31 and Figure 4.31 are similar and INS measurement is consistent throughout.

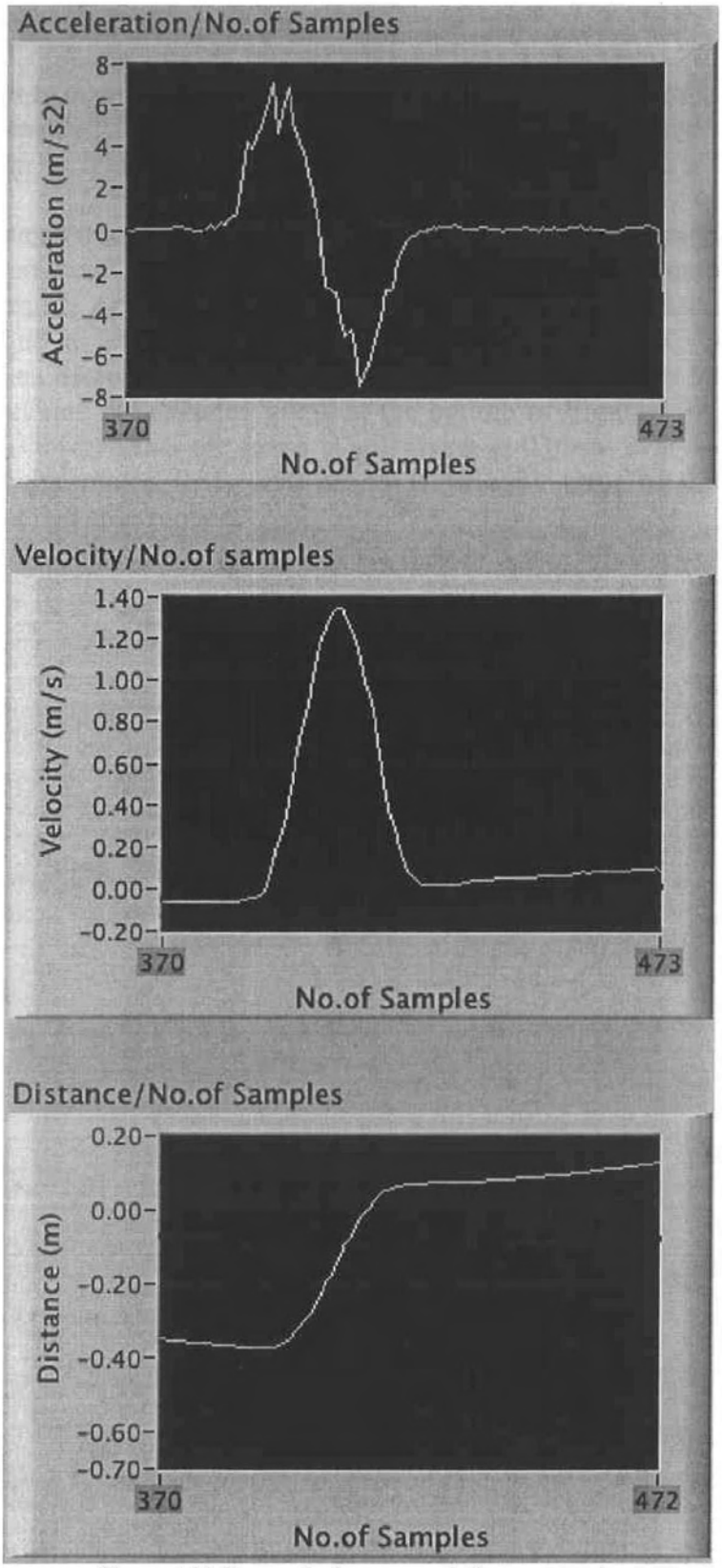

Figure 4.31 Shows displacement of 12cm on y-axis

A.3 *y*-axis negative Distance

The DraganFlyer is manually pushed in the negative *y* direction a distance of 30cm. This displacement measured 21cm with a ruler and the INS system measured it as (-0.24m – (-0.03m)) = -0.21m = -21cm.

Figure 4.32 contains three graphs of which the top graph illustrates acceleration in the *y*-axis. Ins measurement system produces a graph similar to a sine-wave with peak acceleration –3m/s2. Acceleration is integrated to produce the velocity profile in the second graph and as expected, from the acceleration graph the velocity increases smoothly and then decreases as the DraganFlyer is brought to a halt. Velocity is further integrated to generate the distance graph at the bottom of Figure.4.32, which displays smooth motion. Observe that the graph is still rising at 0.01cm every 4 samples due to drift in the accelerometer. This drift causes a velocity error of 0.01m/s when the DraganFlyer is at rest after motion.

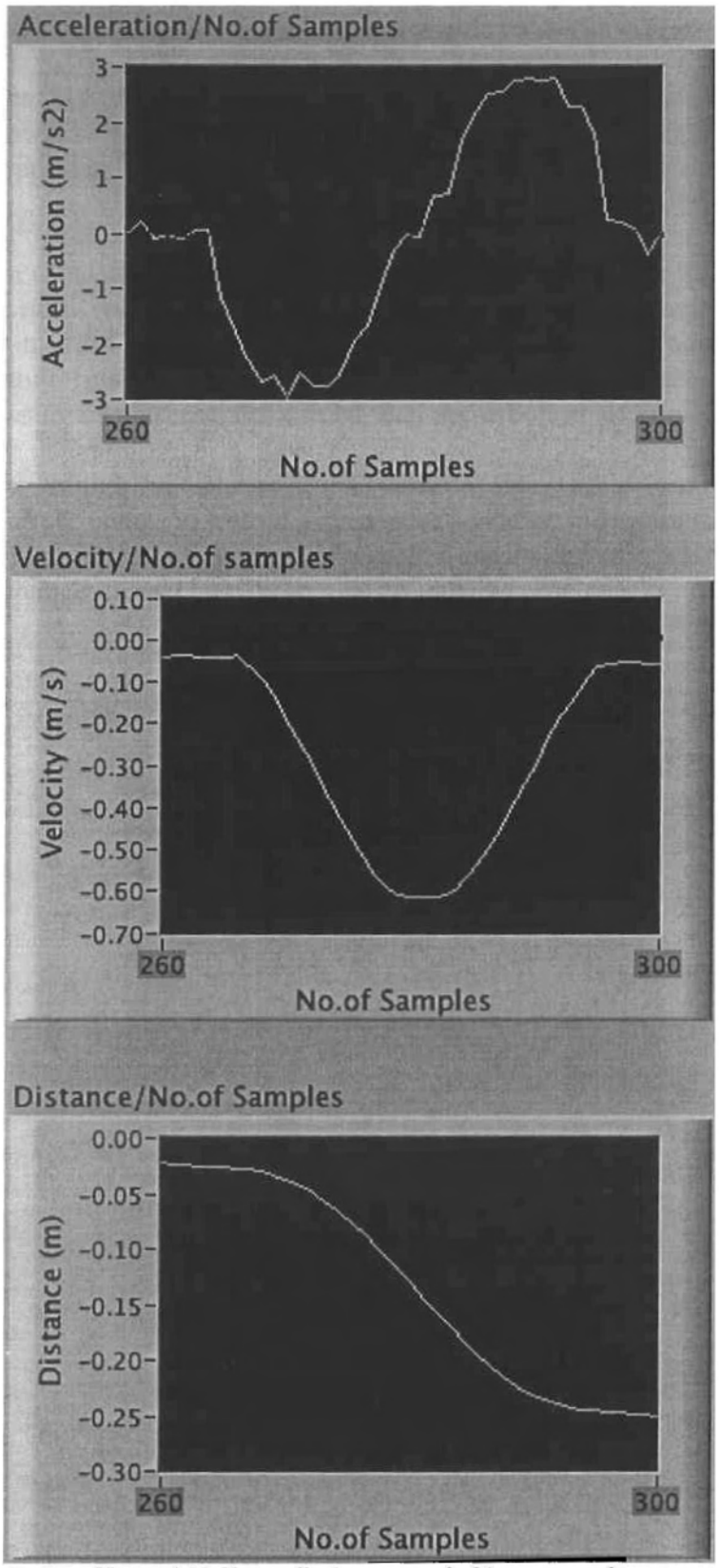

Figure 4.32 Shows displacement of –21cm on y-axis

A.4 *z*-axis positive Distance

In this experiment we place the DraganFlyer at a height of 60cm from the ground and then bring it down vertically in positive *z* direction. With a ruler this displacement measured 61cm and the INS measurement system also measured it as (0.74m – 0.13m) = 0.61m = 61cm.

The results from this experiment are similar to those above. Figure 4.33 contains three graphs; acceleration, velocity and distance. The acceleration graph shows smooth acceleration up to 13m/s2 but deceleration is not consistent. Velocity increases and decreases smoothly. Integrating acceleration twice produces a distance graph displaying constantly increasing displacement as can be seen at the bottom of Figure 4.33.

As with the other graphs, the distance graph continues rising at 0.01cm every 12 samples after the DraganFlyer comes to a stop. This causes a change in zero in the velocity graph though it is not significant. An average of samples before and after displacement shows that there is no change in zero.

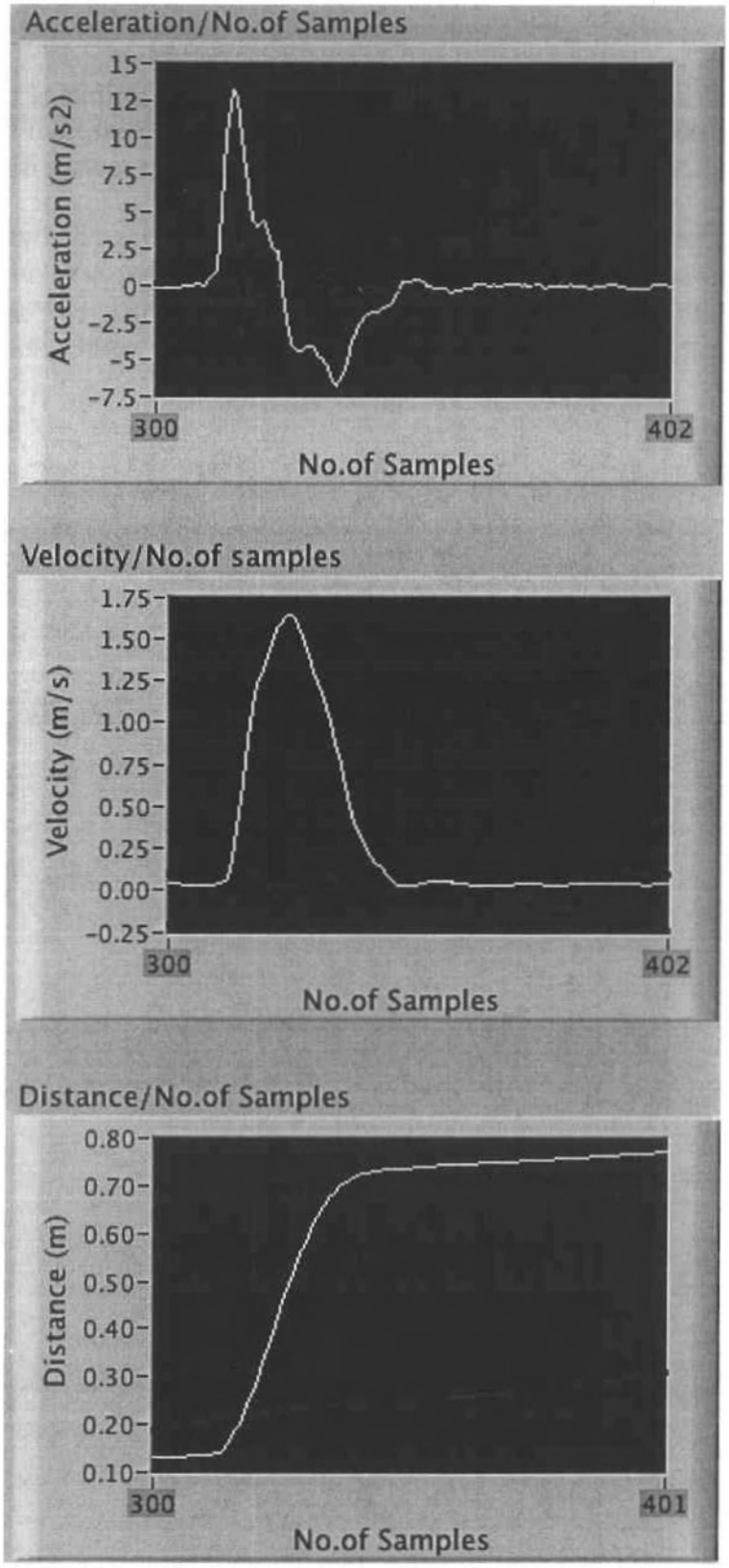

Figure 4.33 Shows displacement of 29cm on z-axis

A.5 *z*-axis negative Distance

This experiment is similar to the above *z*-axis distance. We place the DraganFlyer on a flat surface and then raise it 60cm in height. With a rule this displacement measured 62cm and the INS system measured (-0.65m – (-0.03m)) = 0.62m = 0.62cm

Figure 4.34 shows the acceleration, velocity and distance graphs, which increase and decrease smoothly. As with other accelerometer measurements, there is drift on the z negative-axis as well. However an average of samples 0-340 shows that there is no significant change in zero.

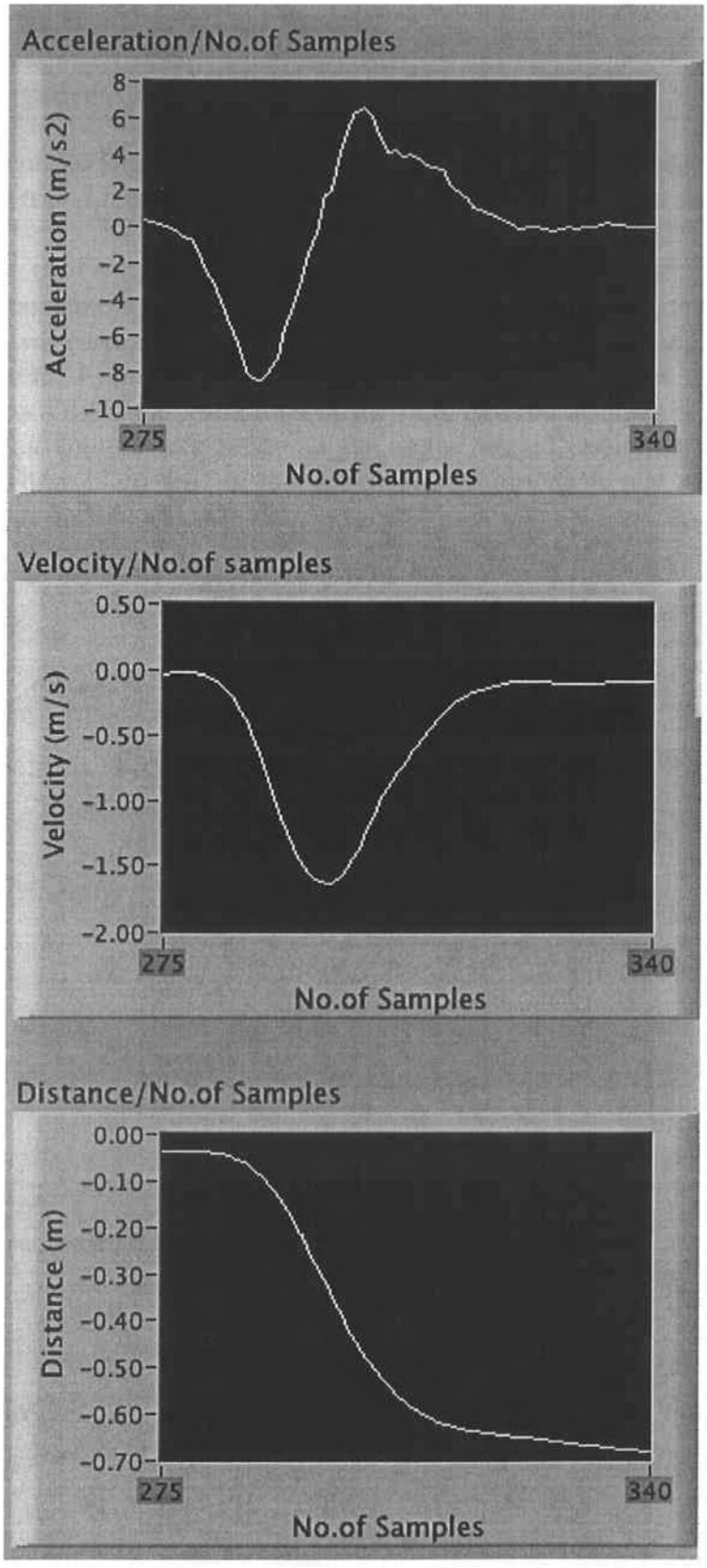

Figure 4.34 Shows displacement of –47cm on z-axis

10.2 Appendix B – Gyro Test Results

B.1 *x*-axis negative Angle

In this experiment the DraganFlyer is rotated 90 in the negative x angle using a tripod. The scale on the tripod measures it as 90° and INS system measures it as (-89° – (- 4°)) = -85°.

Figure 4.35 contains three graphs, i.e. angular acceleration, angular rate and angle. Since this experiment was carried out sing a tripod and a palm, many vibrations were measured by the INS system. Angular rate increases smoothly but decreases with jerks before settling into zero. Given the vibrations in the rate, derived acceleration is not smooth either. Integrating angular rate produces the angle rotated, which is displayed in the bottom graph. There is no drift in this gyroscope graph before and after motion. The angle graph settles back to a steady zero.

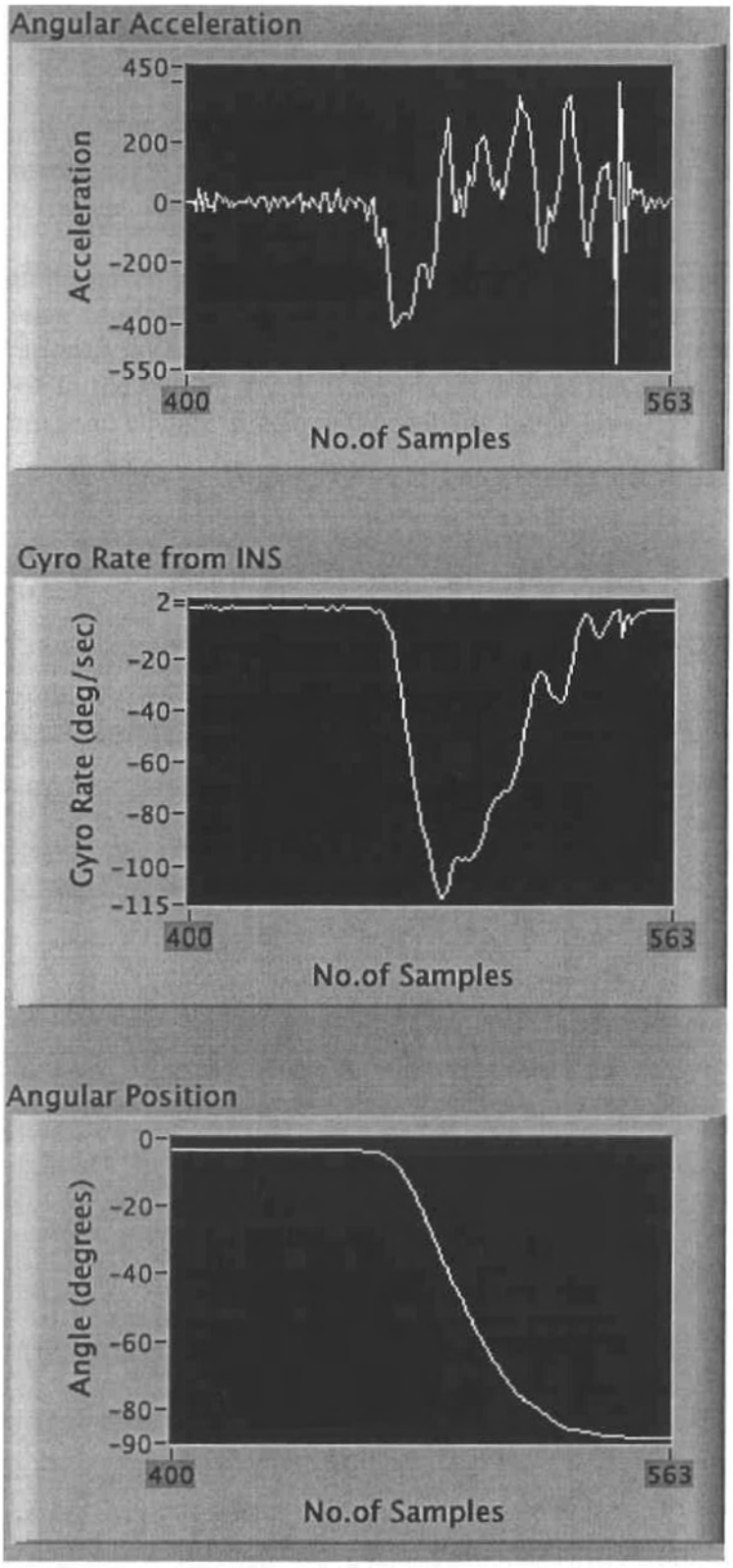

Figure 4.35 Shows rotation of –71deg on x-axis

B.2 *y*-axis positive Angle

In this experiment the DraganFlyer is rotated 90 in the positive y angle. Using a protractor, the rotated angle measured 87° and INS system measures it as (90° - 3°) = 87°.

Figure 4.36 contains three graphs, i.e. angular acceleration, angular rate and angle. Angular rate increases and decreases smoothly. We then derive acceleration from angular rate. The acceleration is not smooth. Integrating angular rate produces the angle rotated, which is displayed in the bottom graph. There is no drift in the angle graph before and after motion. There is no change in zero in the velocity graph also.

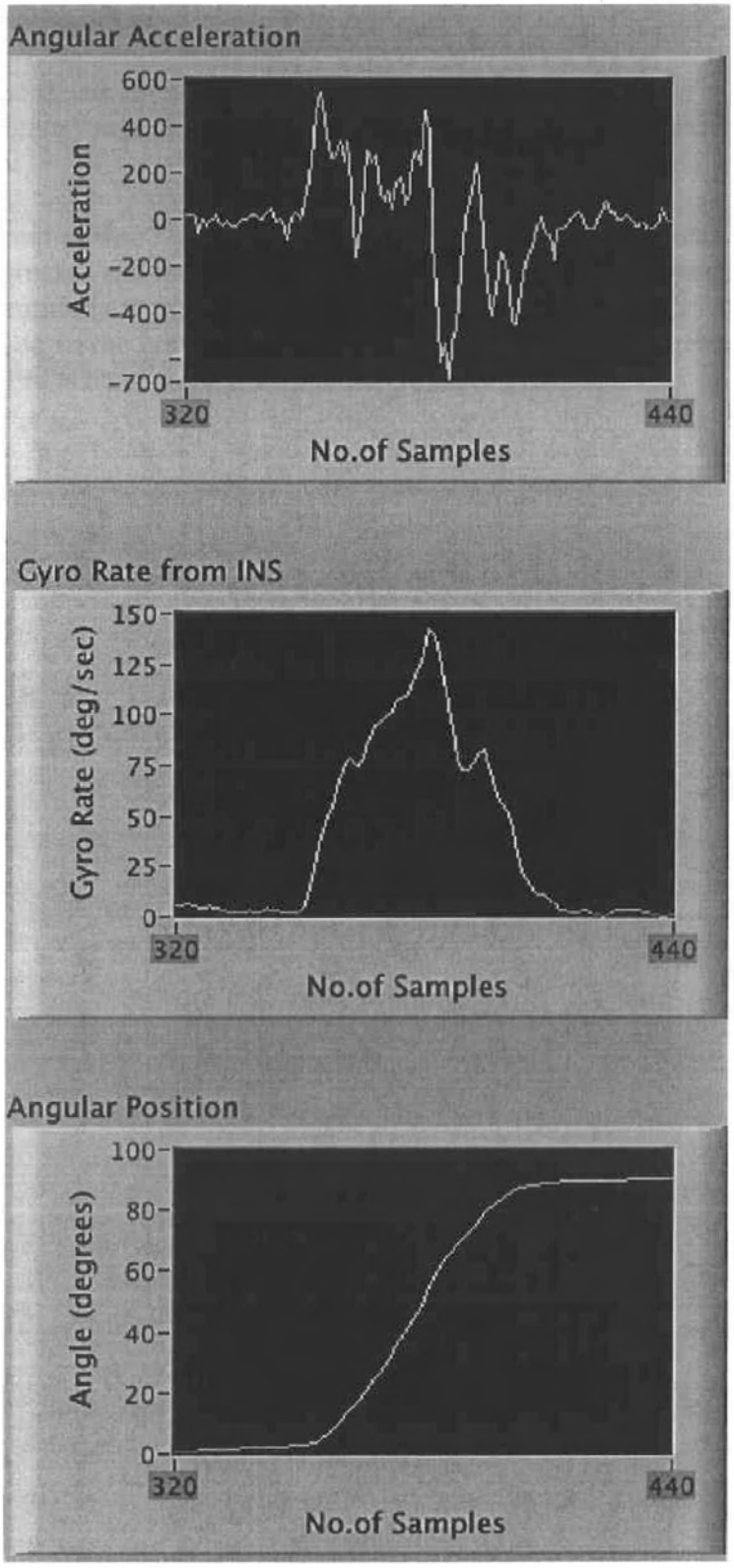

Figure 4.36 Shows rotation of 87deg on y-axis

B.3 *y*-axis negative Angle

In this experiment we rotate the DraganFlyer 90 in the negative y angle. Using a protractor, the rotated angle measured 84° and the INS system measured it as (–84° – 0°) = 84°.

Figure 4.37 contains three graphs, i.e. angular acceleration, angular rate and angle. Angular rate decreases and increases smoothly. We then derive acceleration from angular rate. The acceleration is not smooth. Integrating angular rate produces the angle rotated, which is displayed in the bottom graph. There is no drift in the angle graph before and after motion. There is no change in zero in the velocity graph also.

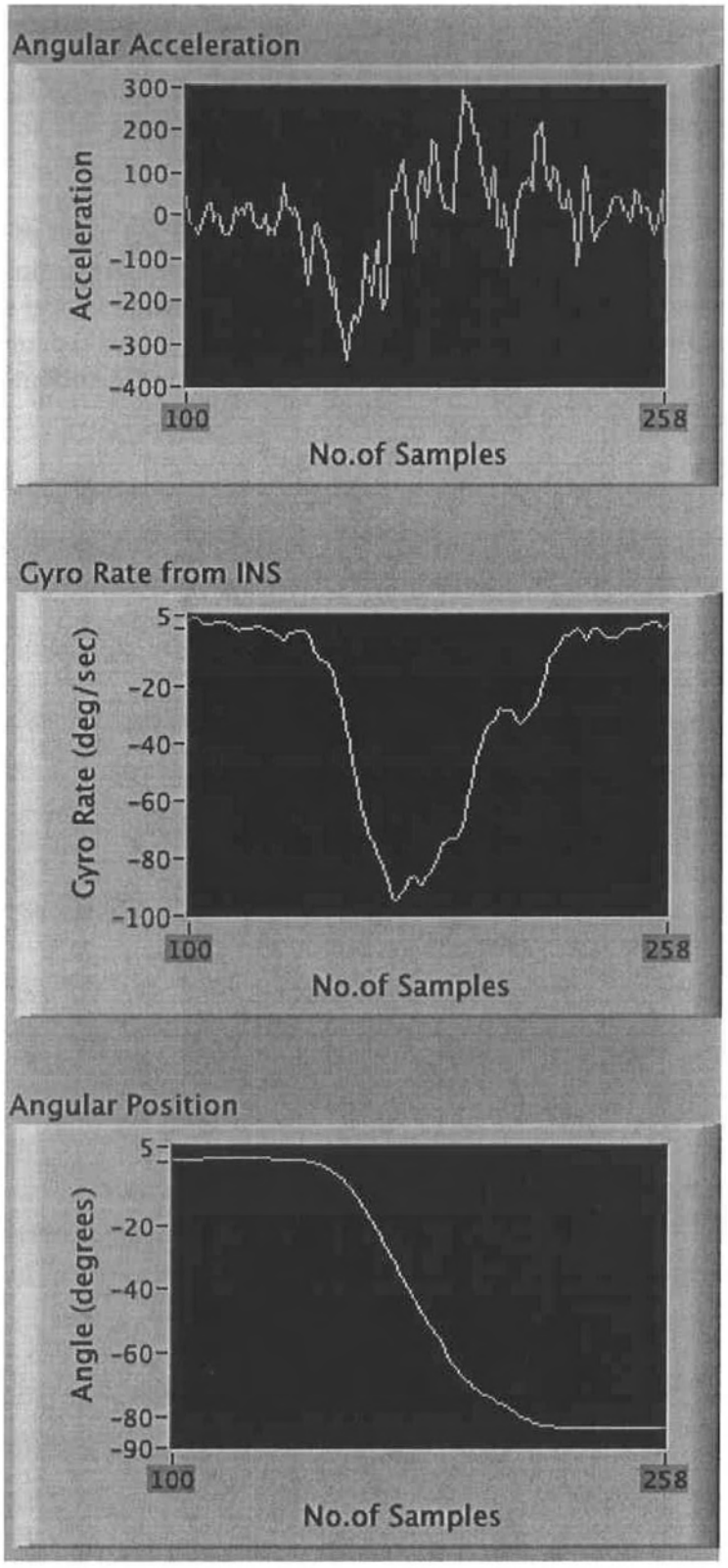

Figure 4.37 Shows rotation of –84deg on y-axis

B.4 *z*-axis positive Angle

In this experiment we rotate the DraganFlyer 90° in the positive z angle on a flat surface. Using a protractor, the rotated angle measured 91.5° and the INS system measured it as (89° – 0°) = 89°.

Figure 4.38 shows three graphs, i.e. angular acceleration, angular rate and angle. Angular rate decreases and increases smoothly. We then derive acceleration from angular rate. The acceleration is a lot smoother than other graphs. Integrating angular rate produces the angle rotated, which is displayed in the bottom graph. There is no drift in the angle graph before and after motion. There is no change in zero in the velocity graph also.

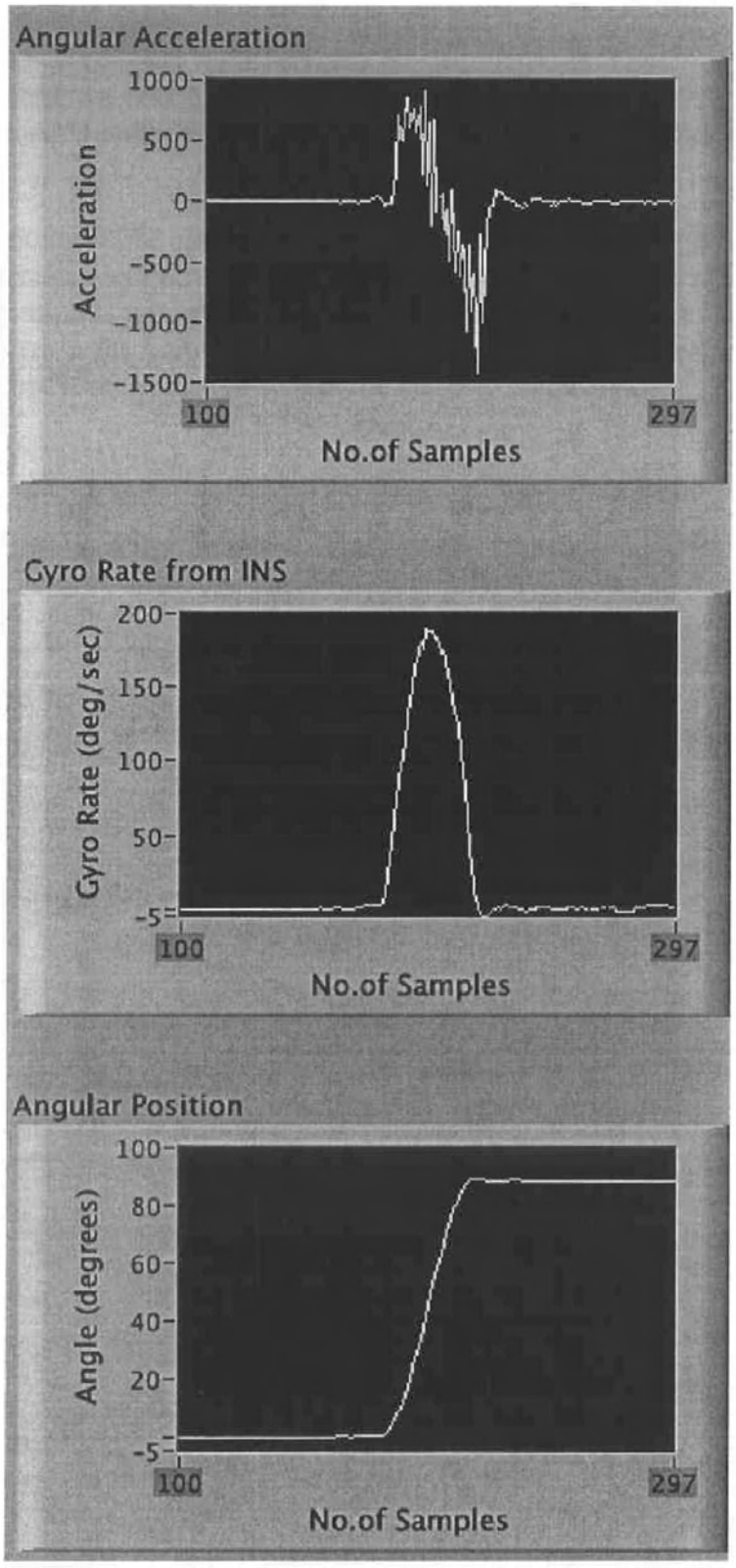

Figure 4.38 Shows rotation of 89deg on z-axis

B.5 *z*-axis negative Angle

In this experiment we rotate the DraganFlyer 90 in the negative z angle. Using a protractor, the rotated angle measured –86° and the INS system measured it as (–84° – 2°) = 86°.

Figure 4.39 contains three graphs, i.e. angular acceleration, angular rate and angle. Angular rate decreases and increases smoothly. We then derive acceleration from angular rate. The acceleration is not smooth. Integrating angular rate produces the angle rotated, which is displayed in the bottom graph. There is no drift in the angle graph before and after motion. There is no change in zero in the velocity graph also.

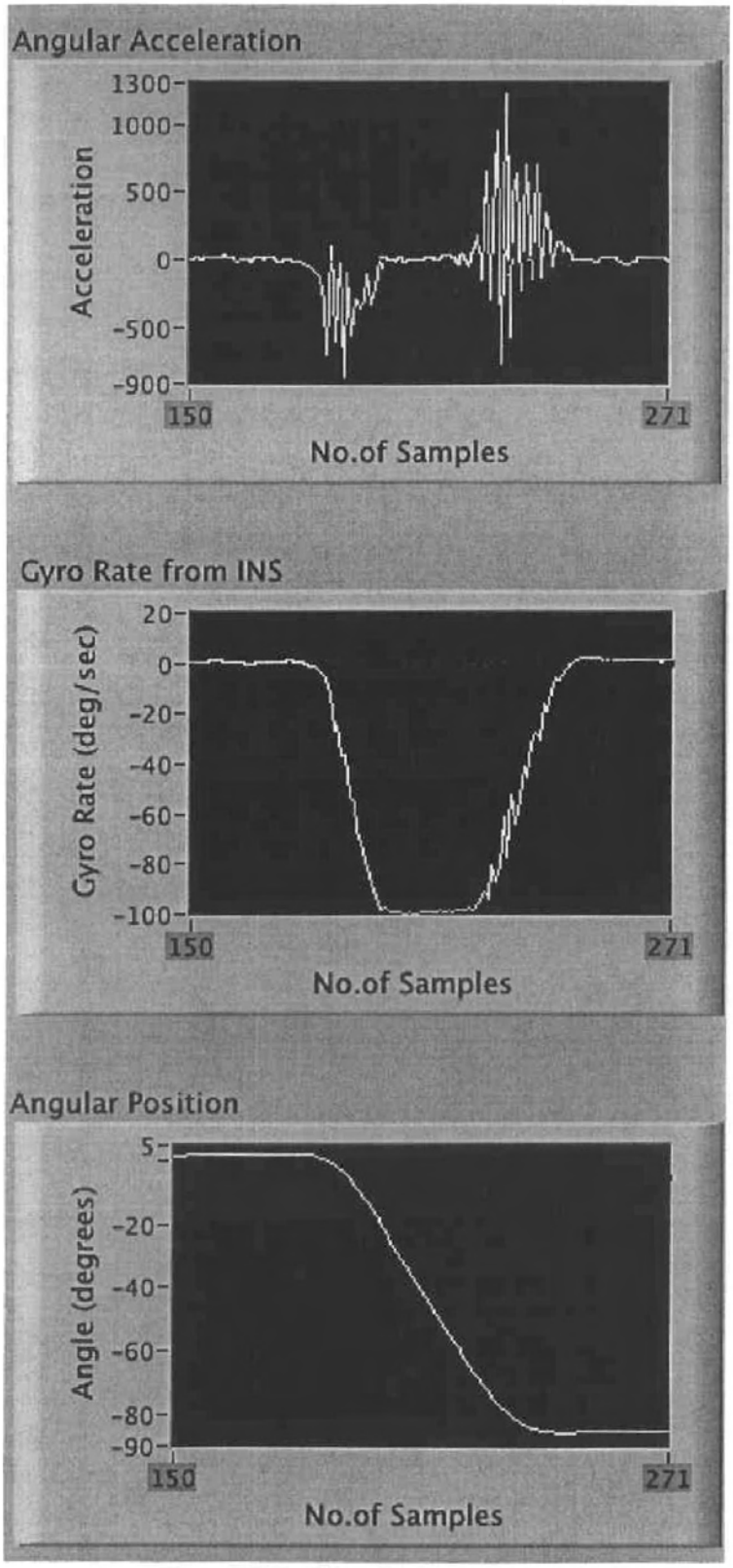

Figure 4.39 Shows rotation of –86deg on z-axis