THE UNIVERSITY OF NEW SOUTH WALES
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Control of A 4-Rotor Blade Helicopter

Muhammad Esa Attia & Nawid Jamali

Thesis Report submitted as a requirement for the degree
Bachelor of Engineering (Computer Engineering)

Submitted: November 3, 2004

Supervisor: Dr. M Waleed Kadous
Assessor: Dr. Claude Sammut

**“In theory it works in practice,
but in practice theory never works”**

Abstract

Modern Unmanned Aerial Vehicles (UAV) originated in the early 1970s[1]. They have been primarily of interest for military operations. Research is being done to improve the reliability and decrease the size of such vehicles so they can be used in Search and Rescue operations, surveillance, law enforcement, inspection and aerial mapping.

The DragonI (pronounced "dragon eye") is an unmanned aerial vehicle whose design was spawned out of the existing design of the Draganflyer IV, a 4-rotor blade helicopter created by Zenon Dragan.

This thesis outlines the design and development that went into redesigning the flight control unit situated onboard the Draganflyer. The aim of the thesis is to replace the original flight control unit. The new flight control unit enables access to the internal states of the Draganflyer. This will facilitate the use of the new control unit as a research testbed. Also a new control architecture for the Draganflyer was investigated to improve the stability of the system. A more stable controller was achieved with the system being able to self stabilise and hover autonomously.

[1]"Unmanned Aerial Vehicle," Microsoft Encarta Online Encyclopedia 2004; http://encarta.msn.com

Acknowledgments

There are a number of people whom we would like to thank whose contributions have helped made this project possible. Firstly, we would like to thank our supervisors Waleed Kadous & Claude Sammut who have spent time with us discussing various aspects of the project & providing ideas and alternatives to problems we encountered along the way as well as giving us the opportunity to work on a very rewarding project. We would also like to thank Dave Johnson for his contribution to the project by making time to help us with the design & fabrication aspects of the project as well as allowing us to work in his office for long periods at a time & making use of his equipment.

Other staff members who we would also like to thank include, Dave Snowdon who had helped us during the design phase of the project, looking over our schematics & providing technical support whenever we needed it. Also many thanks to Cameron Stone for giving us a basic run down on how PID's and Kalman filtering works early on in the project as well as teaching us the physics behind the art of Frisbee throwing. Next we would also like to thank Andrew Isaac for various insightful discussion during the course of the project and also for being kind enough to proof read our thesis. We would also like to thank Raymond Sheh for having a keen interest in the progress of our project, as well as providing us with company during the long hours of our work on the project.

Other notable mentions include the people from iCinema for providing us with their force feedback joystick, Phil Preston who kindly lent us his wireless radio module as well as any test equipment and code associated with it and Min Sub Kim for making sure that we are at best of our hygiene by emphasising the importance of daily showers and change of outfit on a regular basis.

We would like to extend our gratitude to Will Parker for sharing with us his expertise in the dynamics of the RC helicopter. We also appreciate the time he spent giving us a crash course in flying an RC helicopter, as well as providing us with a means of practicing these skills on our own with a simulator. Also Jonathan Paxman for lending us a spare battery for the Draganflyer.

Finally, we would also like to thank all staff & students whom have put up with us & the noise we have created in the robotics lab over the past year.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Scope

A remote controlled helicopter called the Draganflyer is the foundation around which this thesis project is based on. The aim of the project is to develop a replacement *Flight Control Unit (FCU)* for the Draganflyer. In it's current state it is just a simple remote control helicopter. While the Draganflyer has been built for recreational purposes, the DragonI will be used as an Unmanned Aerial Vehicle for research. In order to conduct such operations, a new control unit is constructed to replace the current control unit onboard the Draganflyer. The new control unit will be equipped with gyroscopes, accelerometers, a magnetometer and an ultrasonic range finder. These sensory devices give enough information that can be used for autonomous control of the helicopter. Other features incorporated into our system include:

- provide remote access to the internal states of the control unit
- ability to fly autonomously or semi-autonomously
- providing a platform to conduct Artificial Intelligence research such as behavioural cloning and reinforcement learning

This document details the reconstruction of the new Control Unit and experiments done on the DragonI. At the same time it can be used a reference for future work to be conducted.

1.2 System Overview

As mentioned in the project scope, the FCU will be able to do much more than the original control unit onboard the Draganflyer. Figure 1.2 shows an overview of the system.

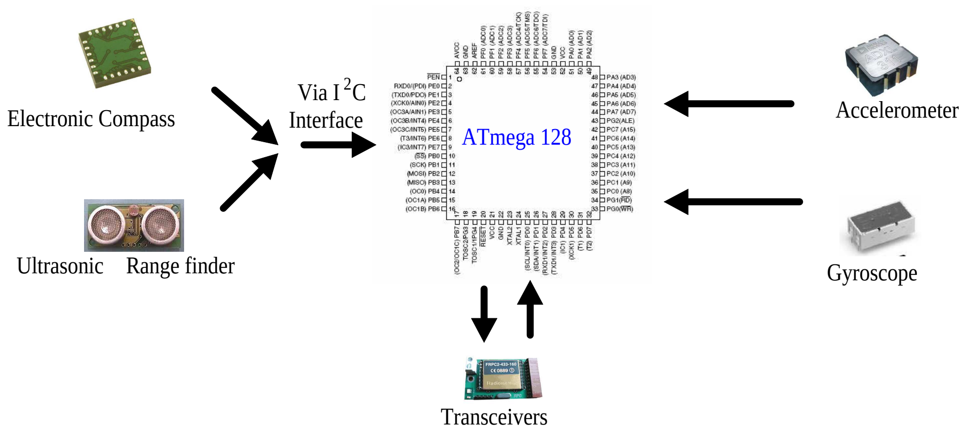


Figure 1.1: main components interact with the microcontroller

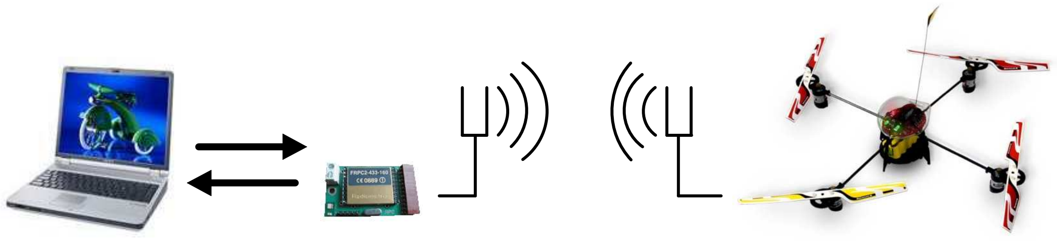


Figure 1.2: Interaction between Computer and DragonI via transceivers

The microcontroller samples the output of the inertial sensors. These signals are then subject to attitude estimation algorithms embedded in the flash memory of the microcontroller. The processed data is then fed into a controller. We have chosen PID controller due to its simplicity. At the same time the data is wirelessly sent to the base station using the *Radiometrix Radio Packet Controllers (RPC)*.

The base station has three main functions. First is to display the sensor data for visual inspection and perhaps can be used for the expert to fly the DragonI based on the sensory information available, so that the system is decoupled from the expert. In order to display the sensor data in a convenient format, a custom Graphical User Interface was developed.

The second function of the base station is to help in development of the PID and hence the control system. During the development stage, the base station was used to send the coefficients of the PID controllers in real time.

Last but not the least, the base station is used for joystick control of the DragonI and hence as a manual override for the autonomous system.

1.3 Outline of issues addressed

During the course of this thesis, certain issues had to be addressed which was critical to the overall success of the project. These issues can be categorised into two separate sections, Hardware & Software. In the subsequent sections, we will give an overview of these issues.

1.3.1 Hardware

From a hardware perspective, the following issues had to be addressed during the course of this thesis

Sensing capabilities

On the original Draganflyer, the circuit board has three gyroscopes on three separate axis (x , y & z) orthogonal to one another, whose function is to measuring the rate of change on each axis. These were about the only sensors built into the original Draganflyer. Therefore it could only measure the rate of change on each axis and had no way of sensing the crafts absolute position. Based on these sensor readings, the Draganflyer only had a means of dampening these changes and nothing more. What we have proposed, is to produce a system with the following onboard sensors:

- **Accelerometers -** These will be used to measure the *absolute tilt* of the craft on the roll and pitch axis.
- **Rate Gyroscopes -** Much like the ones on the Draganflyer these babies will be used to measure the *rate of change* on the roll, pitch and yaw axis.
- **Electronic Compass -** Their role is to measure the craft's *compass heading.*
- **Ultrasonic Range finder -** Their main function is to get the height of the Draganflyer above the ground.

Hardware Sensor Filtering

Although the bulk of the sensor data filtering in this thesis is done in software, sometimes it is more efficient for filtering to be done in hardware especially for low and high frequency noise on the sensor output. These had to be taken into consideration when designing the FCU.

Printed Circuit Board (PCB) development

The Draganflyer system did not make any of the sensor readings available to the user and as such we are not able to make use of the existing circuit board provided by the Draganflyer to do anything useful. Hence there was a need to produce a circuit board which could handle the additional onboard sensors, as well as being able to communicate the data from the sensors back to the user. Furthermore, the new system had to be designed in such a way that the user could interact with the board's microcontroller directly so that we could find out what exactly is happening in the system. This will be done primarily through the USB interface mounted on the DragonI.

Due to the additional sensors/components on the new Flight Control Unit compared to those onboard the original Draganflyer circuit board. We also had to consider the additional power consumption introduced into the system. If the new FCU starts to draw significantly more power than the original Draganflyer FCU, our flight time would be significantly reduced. Also, because the new FCU is meant to be a "drop in" replacement for the old Draganflyer circuit board, we also had to consider the added space the sensors would take up on the new FCU.

Test rig development

The DragonI has the ability to move along all three axis in three dimensional space with a considerable amount of speed. In order to test our system, a testing rig had to be implemented so that the craft could be tested in a safe and controlled manner. In order to do this, we had to restrict the crafts movements within a specified boundary without causing damage to the surrounding areas.

1.3.2 Software

Although hardware will provide the means for us to interact with the onboard sensors and transmit the data from these sensors back to the user, it is the software that does all the intelligent work. Certain software aspects that had to be considered was:

Sensor data gathering

The data streaming into the microcontroller from the sensors onboard the DragonI will come in two forms, analog or digital:

- **Analog sensors -** The data coming into the microcontroller input from the output of the sensors is represented as a voltage relative to the sensory information the sensor is sensing. Hence for the microcontroller to read in this voltage, it needs to quantify the voltage levels into discrete numbers. Onboard the microcontroller, there are facilities for doing this conversion called the *Analog to Digital Converter (ADC).* What we need to do is to write the drivers for the ADC to perform these conversions and use the data from these analog sensors.

- **Digital I^2C sensors -** The I^2C sensors are able to provide digital output data. Therefore there is no need to pass the output of the sensors to the ADC as the information is already digitised. The I^2C sensors operate on an I^2C bus which has its own protocol in terms of communicating with each of the devices connected on the bus. Therefore there is a need to write the drivers necessary to access these I^2C devices.

Control software

The control software implemented into the Flight Control Unit will be responsible for the flight stability of the DragonI. Without the necessary control algorithms built into the system the DragonI would be unflyable. What we have to consider is the control architecture required for this system. Since we are dealing with a real time system, the issues that needed to be addressed include computational complexity and simplicity.

Software Sensor filtering

In a system such as a helicopter, the data coming from the sensors are expected to be very noisy. Due to noise in the system, the data coming from the sensors may not be a true representation of the actual sensory data. Hence there is a need to implement some sort of filtering to obtain a good estimate of sensory data before making use of the sensor data for other purposes such as input into the control software. We must also note that the information that we provide to the control software has to be accurate as our controller would only work as good as the data we give it. Different sensor filtering techniques were investigated during the course of this thesis and appropriate software filtering techniques had to be chosen based on its accuracy as well as complexity for a given sensor.

Wireless Communication Protocol

The DragonI's wireless module is half-duplex (i.e. communication can only take place in one direction at a time). Therefore there is a need to implement a communication protocol to arbitrate the communication taking place between the base station and the DragonI. The purpose of this communication link is to provide a means for the user to access the internal state information of the DragonI and also be able to give commands to the DragonI to perform certain tasks.

Graphical User Interface - GUI

In order for the user to monitor the state of the DragonI at all times, there is a need to display the data transmitted by the DragonI into a useful representation which is not only meaningful but easy to read and understand. This is where the GUI steps in. Its main function would be to output the raw sensor data as well as graph these data over time so that we could take note of any behaviours inherit to the system. The GUI will also provide the user with a means of tuning the DragonI's control software in real-time until the desired behaviour is observed.

1.4 Report Structure

Before we continue any further with the report, we will try to explain the overall structure of the report.

Firstly, would try and give the background associated with this project. This might include stuff such as the background on Unmanned Aerial vehicles as well the flight dynamics of a conventional helicopter as well as those of the Draganflyer. We would also look into the sensing technology incorporated into our system. Finally we will look into the literature review and comment on how these past projects may be of use to us.

Upon doing this, we would then move on to describe the stages involved in the design & implementation of the system. This would start off with a brief system overview of the whole project. From there we will then move on to describing the specifics of the design & implementation of the project. This will include things such as the hardware components , component interfacing & PCB design , software drivers written for the microcontroller (firmware) , as well as those on the base station. We will also discuss the problems associated with implementing the control software and the solutions that we have proposed.

Finally we will evaluate the project based on some experiments conducted on the system. We will start off by describing how the experiment was setup & conducted. From then on we will attempt to display the results from the experiments as well as any conclusions draw from these experiments.

Chapter 2

Background

2.1 UAV basics

Unmanned Aerial Vehicles (UAVs) are remotely piloted or autonomously controlled aircraft. UAVs can be placed into three categories, local, regional, or endurance based on the distance they can cover. UAV's payload includes cameras, communications equipment and other sensors depending on their use.

UAVs have a wide range of applications ranging from civil to military, these may include law enforcement, recovery of aerial maps, search and rescue operations.

2.2 Rotorcraft Flight dynamics

The Draganflyer design was used in this project due to its simple flight dynamics and its stability over conventional helicopter designs. Below is a brief description of the flight dynamics of a standard conventional R/C helicopter & those of the Draganflyer.

2.2.1 Helicopter flight dynamics

A standard helicopter consists of the main rotor placed horizontally at the top of the helicopter and another at the tail, which is mounted vertically. With a standard R/C helicopter design there is considerably more yawing effects due to the main rotors called "Torque" as well as the "P factor". In a conventional helicopter as the main rotor spins one way the motor attached to the rotor tends spin the opposite direction. What we see here is Newton's 3^{rd} law in action which states:

For every action there is an equal and opposite reaction

The “P factor” on the other hand is a non-symmetrical distribution of thrust over the rotating disk formed by the propeller due to varying speeds and angles of attack of the prop blades at various positions. With these two factors (“Torque” & “P-factor”) acting on a helicopter, what happens is that the helicopter tends to spin along its rotor axis in a particular direction (yawing). The tail rotor is used to compensate for these forces or else the helicopter would spin out of control.

2.2.2 Draganflyer flight dynamics

The Draganflyer’s design however is unique in that instead of one rotor that spins at the top of the helicopter there are four smaller ones each 90 degrees next to the other. This design makes it more stable in level flying and does away with the need for a back rotor to stabilise the craft.

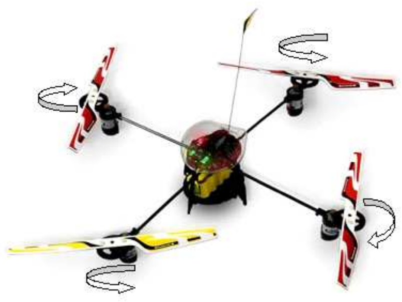


Figure 2.1: The rotating action of the rotor blades

Figure 2.1 shows two sets of rotors. The left & right rotors spin in a clockwise direction while the front & back rotors spin in a counter-clockwise direction. Now because of this, the torque as well as the “P factor” is effectively canceled out, hence providing a much stable flight without the need of a tail rotor. With this design, the craft can maneuver by simply adjusting the speed of the rotors. In the Table 2.2.2 below we can see that by simply changing the speed of one or more of the rotors, the DragonI is able to move in a certain way. Essentially there are four axis of control on the DragonI.

Maneuver	Rotor Action
Roll	vary relative speed of left and right rotors
Pitch	vary relative speed of fore and aft rotors
Collective	vary speed of all rotors simultaneously
Yaw	vary relative speed of clockwise and anti-clockwise rotors

Table 2.1: The DragonI's movement based on rotor speed control

2.3 Control architecture

The definition of the term control theory is as follows: [1]

> "Control theory deals with the behaviour of dynamical systems over time where a dynamical system is a deterministic process in which a function's value changes over time according to a rule that is defined in terms of the function's current value"

In control theory, a controller is a device which can be hardware or software that attempts to control the output of a dynamic system. In order to do this the controller will attempt to manipulate the inputs of the system to achieve the desired behaviour at the output of the system.

In the DragonI, control theory will play a fundamental aspect of the system as it will be primarily used as a means of achieving stable flight. Without the appropriate control architecture in place the DragonI would be deemed unflyable. The problem with designing a control architecture consists of choosing a proper controller considering the system dynamics, which is to be controlled, and the desired performance specifications. During the course of this thesis we mainly concentrated on one type of controller, the Proportional Integral Differential (PID) Controller.

2.4 Behavioural Cloning

Flying an RC helicopter such as Draganflyer is a complex task. A human expert will find it difficult to articulate the control, but can demonstrate. This is because most of the actions of a human controller are sub-cognitive. This is called the tacit skills of the operator. Behavioural cloning is a machine learning method which tries to capture these tacit skills along with the conscious skills by observing the expert and replicating their behaviour.

[1]definition obtained from the www.thefreedictionary.com

There are several approaches to behavioural cloning. The first approach is to train the clone on the entire flight path, from take off to hovering to landing. This approach is not very flexible as the system will not be able to cater for a variety of maneuvers. The second approach is to divide the task into subtasks such as hover, forward etc. [6] The second approach gives more flexibility to the system as complex flight paths can be constructed from a combination of these primitive maneuver.

There are two methods that are in use for machine learning namely symbolic and connectionist. Examples of symbolic method are decision trees and the regression trees and neural networks for connectionist methods. Each has their pros and cons.

Neural networks can give fast predictions, but their training is computationally expensive and they tend to forget what was learned before i.e. "new data causes it to forget some of what it learnt on old data"[2]

Decision trees are robust to outlier effects and are easily readable by humans as they are just bunch of if else statements. But the decision trees work with discrete output values. This is a bit of problem in our case as all of the control outputs we have are continuous. Sammut and Bain [3] used a pre-processor to overcome the problem of continuous values by breaking them into sub-ranges that could be given discrete labels. Another alternative would be to use regression trees.

The process of behavioural cloning involves logging control actions taken by an expert given the state of the environment i.e. the sensory data. This log is used as the input of a learning algorithm. For the DragonI to be used as a research testbed for such experiments, it is important that it provides facilities that logs such information.

There are certain issues that have to be dealt with when logging the actions of the expert while building the knowledge bank. First and most important issue is that the experts have extra sensory information such as being able to "detect wind and pre-compensate for dynamic effects" [6] which is not available to our system. To overcome this problem a simulator can be built which takes the sensory data from the Draganflyer and represents the state graphically to the expert. Then the expert can fly the Draganflyer remotely without using his/her extra sensory information to control.

The second issue is the Lag between the state that triggered the action and the action. Human reflexes can vary considerably, but they take at least one second. To overcome this the, state of the system is logged periodically, and the actions of the expert are matched with a state in the past which may have triggered the action[3]

[2]http://www-2.cs.cmu.edu/ schneide/tute5/node28.html; accessed on 10th of May 2004

2.5 Sensing Technology

The DragonI has several onboard sensors which measure the various aspects of the DragonI's attitude such as roll,pitch,yaw & height. The sensors used onboard the DragonI include:

- tilt sensors
- tilt rate sensors
- altitude sensors
- heading sensors

The following sections will explain the various sensing technology which have been used within the DragonI system.

2.5.1 Tilt sensors

To measure the absolute tilt of the system, we have used an accelerometer. The device we have chosen is the ADXL311 made by analog devices. These accelerometer are able to output an analog voltage which is proportional to the acceleration. The way these accelerometers work is as follows.

The sensor is a surface-micromachined polysilicon structure built on top of the silicon wafer. Figure 2.2 shows the polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against acceleration forces. Deflection of the structure is measured using a differential capacitor that consists of independent fixed plates and central plates attached to the moving mass. The fixed plates are driven by 180 degrees out of phase square waves. Acceleration will deflect the beam and unbalance the differential capacitor, resulting in an output square wave whose amplitude is proportional to acceleration. Phase sensitive demodulation techniques are then used to rectify the signal and determine the direction of the acceleration.[3]

2.5.2 Tilt rate sensors

Finding the absolute tilt of the DragonI alone using accelerometer is not enough. What we would need is some sort of sensor which is able to detect the rate of change along the roll , pitch & yaw axis. By having these sensors we can determine the rate of change within our system.

[3]Description above obtained from the datasheet of the ADXL311

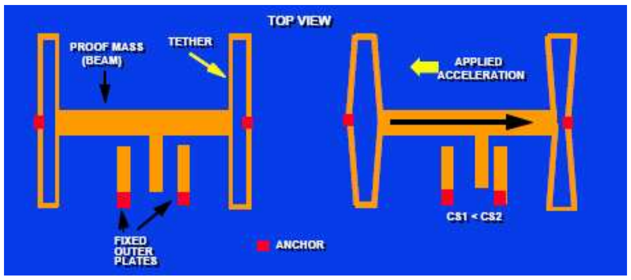


Figure 2.2: Diagram showing accelerometer's operation

The sensors which do this job are called rate gyroscopes. Piezoelectric rate gyroscopes were chosen for our system. The way a piezoelectric gyroscope work is as follows:

A vibrating element, when rotated, is subjected to Coriolis effect that causes secondary vibration orthogonal to the original vibrating direction (see figure 2.3 [4]) .By sensing the secondary vibration, the rate of turn can be detected.

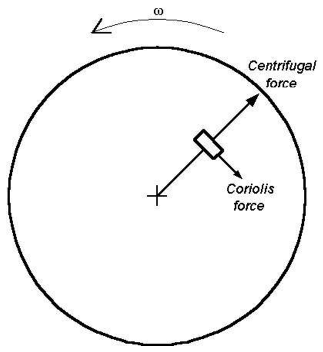


Figure 2.3: The Coriolis effect

2.5.3 Altitude sensors

In order to determine the exact height of the DragonI above the ground. An altitude sensor is required. A conventional altitude sensors normally works by reading the barometric pressure of the atmosphere & determining the height of an object from these readings. This system however is not very accurate for small changes in height and as such is not suitable for our project. Instead we have used an ultrasonic range finder to determine the exact height off the ground. The ultrasonic range finder has two modules, the transducer & the receiver. Sonar

[4]Diagram obtained from The Penguin Dictionary of Physics, 1977

sensors send out a sound wave and measure how long it takes to return. Time is the only unit of measurement.

2.5.4 Bearing sensors

One of the first things we had noticed when trying to fly the original Draganflyer was the fact that it had a natural tendency to yaw quite considerably. One way of stopping this behaviour is to make the craft have some kind of compass bearing toward which it points itself. The way we have solved the problem was to use an electronic compass to give us a compass heading. The way our electronic compass is as follows.

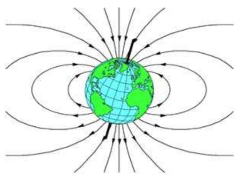


Figure 2.4: The Earths magnetic field

There are two magnetometers measuring the two horizontal vector components of the earth's magnetic field (see figure 2.4). These sensors are placed orthogonally to each other. Called the X and Y-axis sensors, each sensor on an electronic compass measures the magnetic field in its sensitive axis and the arc tangent $\frac{Y}{X}$ provides the heading of the compass with respect to the X-axis. A two-axis compass can remain accurate as long as the sensors remain horizontal, or orthogonal to the gravitational (downward) vector. In moving platform applications, two axis compasses are mechanically "gimbaled" to remain flat and accurate.[5]

2.6 Review of Literature

This section looks into what have already been done in this field of work. There have been several attempts by different people who have come up with a UAV based projects. Although most people have adopted the conventional helicopter design for their work on UAVs; there are some groups which have adopted the Draganflyer model (4-rotor blade design) as the basis of their work.

[5]see Honeywell website for more details http://www.ssec.honeywell.com/magnetic/landnav.html

2.6.1 Semi-Autonomous Control of a Draganflyer

Researchers of Pennsylvania State University's Rotorcraft Centre are currently doing research similar to that which is currently being undertaken in this thesis. What they have proposed to do is to create a semi-autonomous UAV based on the Draganflyer design. At the present moment, their goal is to add sensors on their Draganflyer and based on these sensory inputs, making the Draganflyer much easier to fly autonomously. Their ultimate goal is to modify the FCU within their Draganflyer unit such that it would improve handling qualities of the Draganflyer so that anyone can fly the craft with very limited training & the ability to provide way point navigation with the onboard *Global Positioning System (GPS)* they plan on installing.[5]

2.6.2 Autonomous control of a Draganflyer based on control theory

Researchers from the Université de Technologie de Compiégne (France) have also done work in the field of control theory by using the Draganflyer as a basis of their research. What they ultimately set out to do was to take-off, hover and land autonomously over certain point. The big drawback to this setup is that it was a tethered system and cannot really be classified as a UAV as the sensors are off board. [10]

2.6.3 Phaeton Project

The Phaeton Project involved students retrofitting the Draganflyer X-Pro (a larger version of the Draganflyer IV) by changing not only the flight control unit but also the motors , blades & mountings to increase the crafts efficiency. Their objective was to demonstrate an autonomous control system for an indoor flying vehicle by having it take-off , hover , translate to a given location , return to point of origin and land.[9]

2.6.4 The BXflyer

Not all UAV projects are done in Research Labs or universities, Bruce J. Weimer who published his work in the Seattle Robotics Society Newsletter detailing his experience in creating an autonomous 4 bladed helicopter based on the Draganflyer design during his spare time. Although he only managed to fly straight up for 10 seconds, decelerate for 2 seconds and then turn-off to land/crash on the ground. There are some important concepts, in terms of design, explained within the article. The fact that he was able to use a combination a accelerometer/gyroscopic data & the use of PID control would be of interest to this thesis project. [4]

2.6.5 Autopilot Project,Sourceforge

The Autopilot Sourceforge project's ultimate aim is to create an autonomous R/C helicopter. At the present moment the system isn't totally autonomous yet, but from the tests conducted, it has been proven that the inertial measurement unit (IMU) installed within the helicopter and the use of Kalman filtering to extract data from these sensors have worked well. There are quite a number of projects out there similar to the auto pilot project (commercial & non commercial) who have adopted the conventional helicopter design to base their UAV projects on. [6]

2.6.6 MARVIN - Multi-purpose Aerial Robot Vehicle with Intelligent Navigation

MARVIN is an autonomously flying robot based on a model helicopter. It is designed for search operations in unknown environments, with possible threats such as fires, water fountains, and smoke.

The overall system consists of the helicopter with an on-board computer and a second computer serving as a ground station. While flight control is done on-board, mission planning, human user interaction, and digital image procession take place on ground. Sensors for autonomous operation include carrier phase Differential Global Positioning System (DGPS) equipment, accelerometers, magnetometers, gyroscopes, flame sensors and ultrasonic transducers. Image acquisition is done through a digital photo camera.

MARVIN used two off-board computers for its control instructions. One PC runs the mission control software and acts as a user interface to the overall system, while the other run the vision processing software.

MARVIN's control has two components to it. First is the flight route planning which is handled by the off board Personal Computer (PC). The second is the flight control or the auto pilot which run on the on-board computer.

The Flight Route Planning module has as its primary responsibility of generating collision free paths that can be followed by the robot.

The Flight Control module has as it main responsibilities the stabilisation of the the helicopter and also to follow the linear path segments determined by mission control.

The control system uses combination of simulations and in-flight experiments to generate control parameters[7].

[6]http://autopilot.sourceforge.net

2.6.7 Autonomous inverted helicopter flight via reinforcement learning

At Stanford University, a Bergen industrial twin helicopter is used to implement control using reinforcement learning for autonomous flight. Their work involved modifying the the existing helicopter, and instrumenting it with Inertial Measurement Unit (IMU) that has accelerometers and rate-gyroscopes, magnetic compass and Global Positioning System (GPS).

They also have a PC104 flight computer onboard which reads the sensory data and processes them using Kalman filters to obtain attitude estimation. The communication between the onboard system and the base station is achieved via 802.11b wireless.

A human pilot was used to model the dynamics of the helicopter. Actions of the human pilot along with the state of the helicopter were logged, which comprised of tne helicopter's position (x, y, z), orientation (roll ϕ, pitch θ, yaw ω), velocity ($\dot{x}$, $\dot{y}$, $\dot{z}$) and angular velocities ($\dot{\phi}$, $\dot{\theta}$, $\dot{\omega}$). The interesting part is the reduction of state from 12-dimensions to only 3-dimensions. A reduction from 12 to 8 dimensions was achieved the methods described in [2], and the rest was made just by observing that some states such as roll and pitch can be obtained from roll and pitch rate by simply applying numerical integration.

After obtaining the model of the helicopter, a graphical simulator was built to verify and test the model. This simulator was also used for the testing of the controller. Their approach to finding a controller for the system was to use reinforcement learning rather than Proportional Integral and Differential controllers. By using the reinforcement learning, their team was able to successfully implement a controller for inverted flight of a helicopter[1]

Chapter 3

System design and implementation

3.1 Hardware Components

3.1.1 Requirements

Redesigning the Flight Control Unit required careful consideration of certain requirements and design constraints. These include:

- **Low power consumption:** The power consumption of new FCU should not be significantly higher than that of the original one. It should be able to utilise the current battery and provide a reasonable flight time.
- **Total Mass and size:** The new FCU is to be a drop in replacement for the original FCU. The original system has a payload of about 150 grams. And as such, we are constrained by the size and mass of the unit.
- **Self Contained:** The new FCU should be able to estimate it attitude without any external help, and hence perform as an autonomous system.
- **Low Cost:** The new FCU should be cost effective, so that production of such a unit is economically viable.

3.1.2 Component Selection

With the system constraints in mind, research was conducted into the types of sensors and other electronic parts that were at our disposal in the market. Since the components are being used on a very noisy (vibrating) platform, it is worth to note that during the selection of components a careful consideration was made in regards to the noise tolerance of the components, where cost was not an issue. The following types of sensors that were considered to be incorporated into the system.

Rate Gyroscopes

Gyroscopes are very useful in many applications. Their primary function is to determine the speed or rate of angular change. To choose the right rate gyroscope sensor, some features, such as power consumption, weight, dimension, etc., must be taken into consideration. For this thesis, two gyroscopes models were considered

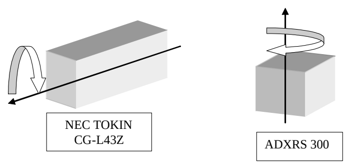


Figure 3.1: Axis at which rate of change measured

CG-L43Z

The CG-L43Z is a single axis, miniature angular rate sensor (ceramic gyro) made up of a single piezoelectric ceramic column printed on electrodes. A voltage between 0-5 volts proportional to the rate of angular change is then output. The CG-L43Z measures the rate of angular change along its x-axis. This is desirable as we then only need two circuit boards mounted orthogonally to each other to measure rate of angular change on all 3-axis. It was this factor and the fact that it was easy to solder this particular component which led to it being chosen for this project.

ADXRS 300

The ADXRS 300 is also a miniature angular rate sensor. In terms of size, it is much smaller than the CG-L43Z. But the nasty thing about this packages is that it uses a “Scale Ball Grid Array” *Integrated Circuit (IC)* configuration, which is rather difficult to solder. It also happens to measure rate of change along its y-axis, which would require three circuit boards to be mounted orthogonally to one another. Due to these two factors the ADXRS was not chosen as a suitable component for the project as it it hard to physically interface.

Accelerometers

The ADXL311 is a low power, dual-axis accelerometer built on a single monolithic IC using the iMEMS process (MEMS: Micro-Electro-Mechanical System). It can measure both dynamic acceleration, such as vibration, and static acceleration, such as gravity, with a full-scale range of 2g (where g = force of gravity), producing outputs as analog voltages proportional to acceleration. For this thesis the main toll of ADXL311 will be to measure the tilt orientation of the craft - in which the accelerometer uses the force of gravity as an input vector to determine its orientation in space - and due to this, the range of interest is 1g. The ADXL311 has provisions for limiting its output bandwidth, which is achieved by adding (low-pass) filtering capacitors at the X_{OUT} and Y_{OUT} pins . The accelerometer bandwidth selected will ultimately determine the measurement resolution (smallest detectable acceleration) via noise reduction . The equation for the 3dB bandwidth is:

$$F_{-3dB} = \frac{1}{[2\pi.32k\Omega.C_{(x,y)}]} \tag{3.1}$$

From the equation above we can see by selecting a capacitor value of $0.1uF$, the frequency at which the accelerometer will is 50Hz. By varying the value of the capacitor we can change the rate at which we can sample the accelerometer, but as the sample rate is increased the noise to signal ratio also starts to increase.

Electronic Compass

The Honeywell HMC6352 is a 2-axis digital electronic compass. By using a two-axis *magneto-resistive(MR)* magnetic field sensor design with the required analog and digital support circuits, the HMC6352 is able to compute its heading. One nice feature of this IC is its calibration routine. Magnetic errors can be introduced if operated near strong magnetic sources such as motors or transformers in test equipment. These magnetic errors can typically be reduced or eliminated by performing the calibration routine. The main reason why the electronic compass was seen as necessary was because the DragonI's flight dynamics is such that it tends to yaw once it gets off the ground. By using a compass we are able to keep track of the crafts heading and correct its flight path if significant yawing takes place. Also, by having this device we can effectively tell the craft where its heading should be as it can use the earths magnetic field as a reference to its current heading and its proposed heading. The Microcontroller is able to interact with the compass via the I^2C bus interface.

Range Finder

For this thesis, several range-finding alternatives were looked into. The primary role of the range-finder is to act as an altimeter. A conventional altimeter can't be used in this system as they are not sensitive to small variations in height as they are based on reading atmospheric pressure. Hence the best alternative was to use some kind of a range-finder. The following range-finders were investigated.

Sharp IR Range Finder

The Sharp GP2D02 and GP2D12 sensors are IR range finding sensors. Their biggest advantage is that they are very accurate. A limitation is that their range is limited to approx 80 cm. Hence this makes them only appropriate for close-in work. Another characteristic of the Sharp sensors, that can be either an advantage or disadvantage depending on their use, is that the detection cone of the device is very narrow. Also, these devices are relatively low power compared to ultrasonic range finding devices.

Polaroid Sonar Range Finder

The Polaroid Sonar is very good at detecting obstacles with a very big range, but is demanding to interface. Plus, it is large and expensive. The transducer is so big that it is the predominant feature on small robots. While the range of the sensor at the high end can be very handy, most robots don't need information on what is out 5 meter from the robot. Due to the weight/size factor & the fact it also consumes a lot of power. The Polaroid sensor was seen as an unsuitable choice for the project.

Ultrasonic Range Finder

The SRF08 is an ultrasonic range finder whose primary role is to determine the DragonI's height above the ground. The SRF08 works by transmitting a sonic pulse, after which it then waits for an echo from objects the pulse encounters. By looking at the difference in time, from the time of transmission to the time at which the echo is received. The SRF08 is then able to determine its distance from an object. In the case of the DragonI, the SRF08 will be positioned in such a way that it would always be facing the ground. This effectively enables us to measure the height of the DragonI above the ground.

It was decided that the SRF08 Ultrasonic Range-finder would the the most appropriate choice for this particular project due to its light weight, big ranging capabilities & relatively low power consumption relative to those of the Polaroid sonar range-finders.

Transceivers

The choice of Radiometrix Radio Packet Controller (RPC) transceivers was based on availability and suitability of the transceivers for the project. The RPC is a self-contained plug-on radio port that requires only a simple antenna, 5V supply and a byte-wide I/O port on a host microcontroller. The module provides all the RF circuits and processor intensive low-level packet formatting and packet recovery functions & checking required to inter-connect a number of microcontrollers in a radio network. It is very easily mountable on any of the bi-directional ports of the ATmega128 microcontroller. The RPC has a reliable 30m in-building range and 120m open ground range, deemed to be an adequate range for this thesis.

MOSFETS & MOSFET Drivers

In order to drive the motors on the DragonI, MOSFETS have to be used. In order to control the speed of the motors a microcontroller would send a pulse width modulated signal (PWM) to the MOSFETS. The reason for this is because the microcontroller is not able to source enough current to drive the motors directly. Hence the MOSFET is used to supply the required current to the motors with the microcontroller varying the amount of current provided to the motors by changing the width of the PWM signals it gives to the MOSFETS. The smaller the PWM signal, the less current the MOSFETS give out to the motors & vice versa. The Gate drivers allow the microcontroller to switch the MOSFETS at a much faster rate. Why is this important ? In order to vary the speed of motors you must switch the motors on & off . This makes the MOSFETS more effective in terms of controlling the motors as they are able to switch faster .

3.2 Component Interfacing and PCB Design

During the design of the circuit board several factors had to be taken into consideration before the final design was decided upon & fabricated. The reason why these issues need to be thought through is because once the boards have been fabricated it is rather difficult to make any changes to the printed circuit board.

All up there are three boards which needs to be fabricated. Two of these boards (vertical board & horizontal board) join up to make the Flight Control Unit on the DragonI & the third would serve as an intermediary between the RPC module and the PC. This board will be called the RPC Board.

3.2.1 Design considerations

Board size/dimension

One of the main constraints that had to be addressed in the new FCU was its size/dimensions. Although the new FCU is feature packed compared to the original unit, it has to be a “drop-in” replacement for the original unit.Not only that, certain components had to placed in a specific manner so that the connectors can easily connect to the FCU without much difficulty.

Mass

One other constraint imposed upon the DragonI is weight. The design of the DragonI chassis,rotors and motor characteristic is such that it should be able to lift approx 4 ounces or 113.4 grams of payload effortlessly. If the payload weight is increased performance will degenerate.

Power Consumption

Power consumption is another major issue that had to be addressed when designing the FCU. The motors themselves are able to draw approximately 4 amps at 100% duty cycle. Therefore the motors will have the most dominant effect on the battery life. It would be best if the current consumption is minimised so that there is no additional cost in terms of power with the new FCU placed in the dragonI. It would be ideal if the system is able to provide approximately 15 min of flight time on a single 11.2 V (2100 MAH) Lithium Polymer Battery provided with the original Draganflyer.

3.2.2 Flight Control Board

The flight control unit is situated onboard the DragonI. It is made up of several subsystems, which interface to the microcontroller. Each of these subsystems perform a specific task crucial to the overall performance of the DragonI. Below is an explanation of each of these subsystems

Gyroscope Interface

The gyroscope calculates the rate of angular change within the system on a single axis. The sensor basically sends an analog voltage at its output which corresponds to a specific rate of angular change. The problem is that the output signal amplitude is small and as such the resolution received by reading the signal through the analog to digital converters (ADC) is poor. Therefore the signal has to be amplified through an op-amp circuit. Not only that there are high and low frequency noise present in the output of the signal which needs to be filtered out before it is being read into the ADC. Therefore a high and low pass filter needs to be placed on the output signal to get better results from the analog sensor.

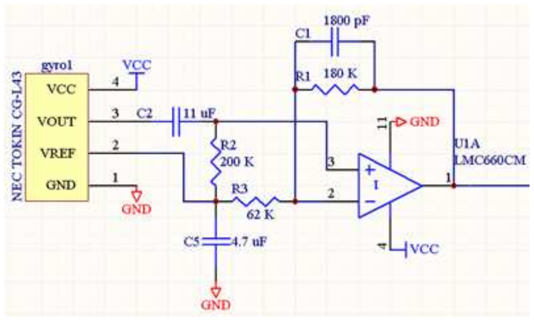


Figure 3.2: Circuits placed on the output of the gyro to reduce noise level of the signal & improve signal resolution

Accelerometer Interface

The accelerometer calculates the absolute tilt along two axis. The sensor just like the gyroscope mentioned above, outputs an analog voltage relative to the tilt of the sensor. The accelerometer unlike the gyroscope has significantly less external components. All up there are four major external components interfaced to the accelerometer (see figure 3.3):

- Bandwidth selecting capacitors - These capacitors are placed on the sensor output of the x-axis and y-axis.

- Bias resistor - If no resistor is present, the ADXL311 may appear to work but will suffer degraded noise performance. The value of the resistor used is not critical. Any value from $50K\Omega$ to $2M\Omega$ can be used.

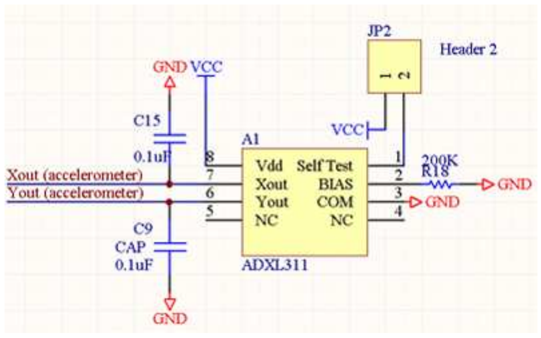


Figure 3.3: Circuit diagram accelerometer

Regulated Power Supply

The linear regulator is the basic building block of nearly every power supply used in electronics. Linear regulators are so easy to use that it is virtually foolproof with very little external components, plus it is inexpensive. In the DragonI circuit, two linear regulators are used. The 9 Volt regulator is used to provide power to the camera while the 5 Volt regulator provides power to most of the components on the circuit board. Since the linear regulators are only able to provide approx 1-1.5 amps, they are not used to power up "power hungry" components such as the gate driver, FETS, & motors. (see figure 3.4)

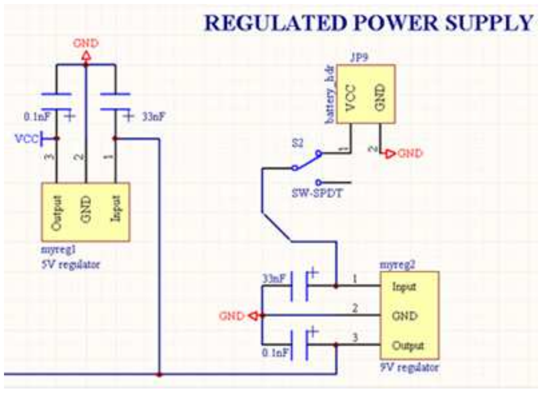


Figure 3.4: Circuit diagram of Regulated Power Supply

Electric Motor Interface

The DragonI has four rotor blades, which have to be controlled by the microcontroller. Figure 3.5 shows the microcontroller connected to the motors via gate drivers and MOSFETs. In order for the microcontroller to vary the speed of the motors it need to source a lot of current. The microcontroller itself is not able to provide the amount of current needed to run the motors, hence the need for MOSFETs. The gate drivers are there to improve the switching capability of the MOSFETs.

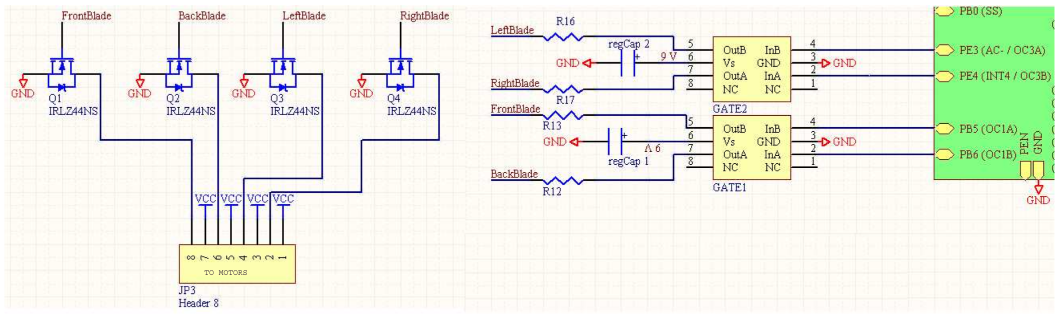


Figure 3.5: Circuit diagram of gate drivers & FETS

Electronic Compass & Ultrasonic Range Finder Interface

The electronic compass & ultrasonic range finder has the most simple interface to the microcontroller. Since both these devices work on the I^2C protocol, interfacing them would be easy. Each I^2C device has four lines going into them

- Power
- Ground
- SDA - Serial Data Line
- SCL - Serial Clock Line

Also Each device connected to the bus is software addressable by a unique address and simple master/ slave relationships exist at all times. Therefore the electronic compass as well as the ultrasonic range finder can both use the same address and clock lines as they would only respond to their own unique address. Since the I^2C Protocol is active low (i.e the SDA/SCL lines need to be asserted low when in use), pull-up resistors are placed between V_{cc} & the I^2C lines (see

Figure 3.6). It is important to note down what the default address is for each device that you wish to put on the I^2C bus. The reason being, if the user was the transmit a command to a particular device address which is common to two or more devices. These devices would respond simultaneously causing contention on the I^2C bus. When this happens you are more likely to see smoke coming out than anything else.

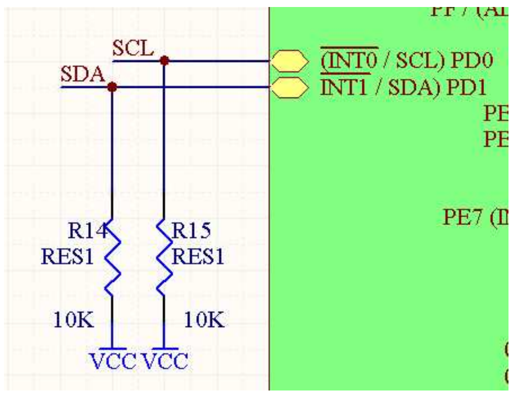


Figure 3.6: Circuit diagram of the I^2C interface

USB UART Interface

There are two USB UART Interfaces in this project. One is situated within the DragonI Board & the other is on the RPC Board. The USB uart interface on the DragonI is primary used for debugging purposes while the on the RPC Board, its main purpose is to serve as a link between the RPC & the computer. At the core of the USB Interface is the FTDI USB-Serial converter chip. It's primary function is to parse the data between the UART pins of of the microcontroller and the PC. The FTDI chip also allows the USB Interface to appear as a standard serial port to the computer as well as synchronised the data stream coming to and fro from the PC to the microcontroller. (see figure 3.7)

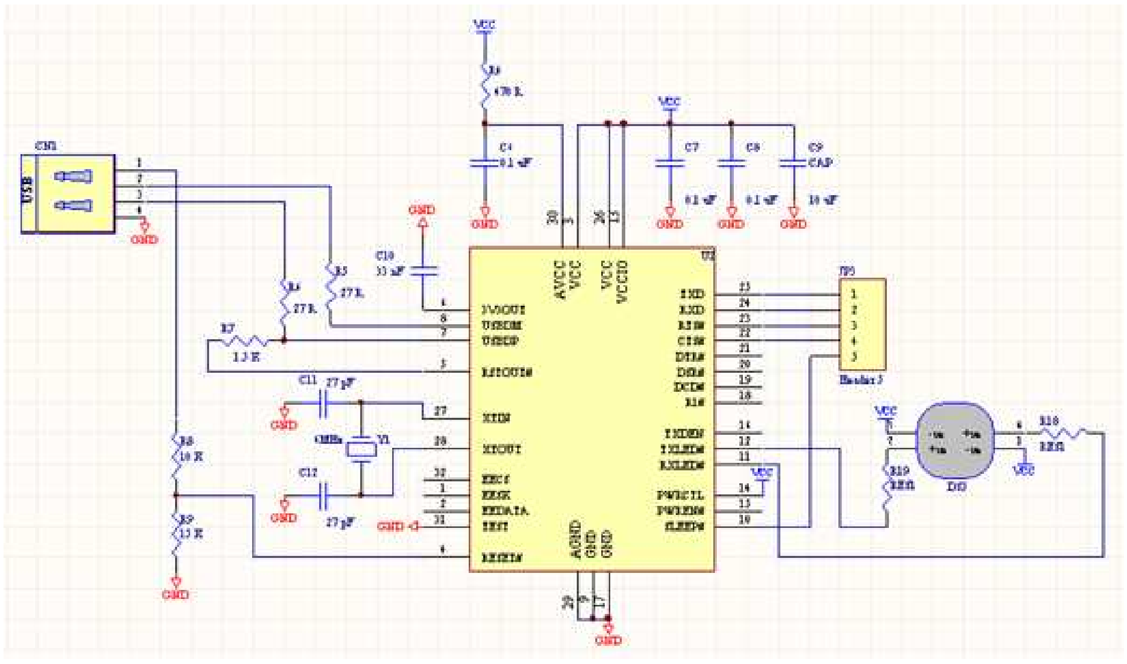


Figure 3.7: Circuit Diagram of the USB interface

3.2.3 RPC Board

As mentioned in section 3.2.2 above. The RPC Board will serve as a primary link between the RPC module and the PC. The board itself is pretty simple. Its specifications include:

- An ATmega 32 microcontroller
- An external crystal clock (7.3728 MHz) for the microcontroller
- Reset button
- UART interface consisting of a USB socket and the FTDI USB-Serial Converter chip
- An EEPROM chip
- Switching Regulator chip
- UART synchronisation clock.

The RPC module has four control lines (RXA , RXR , TXA , TXR), four data lines , one power pins , two ground pin and a reset line. The four data & control lines can be easily interfaced to any of the available ports of the RPC board (in our case we have chosen PORT C - see figure 3.8). The RXR pin on the RPC module is also connected to an external interrupt pin on the microcontroller which runs an interrupt routine whenever an incoming data packet is received via the RPC module. The RPC draws its power from the USB port and hence does not require a battery.

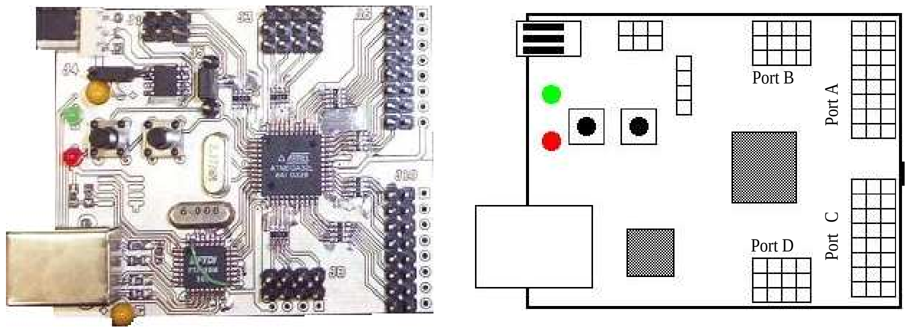


Figure 3.8: The RPC Board - AVR 32 Development Board

3.2.4 Soldering Method

The components of the system were soldered on the fabricated boards a module at a time and tested for proper functionality. This approach made sure the end product was functioning properly.

The Microcontroller being the heart of the system was one of the first modules to be soldered. It's functionality was tested by running some sample programs. After successful testing of the microcontroller the subsequent modules were soldered. The functionality of the each module was tested with the help of the firmware that had already been developed.

3.3 Embedded components & Firmware

3.3.1 Microcontroller

At the heart of the FCU is the microcontroller. Every major component within the FCU is coupled to it. The microcontroller gathers input from various sensors/components, processes this input into a set of instructions and uses its output facilities to accomplish something useful.

The ATmega128 is an 8-bit microcontroller running single cycle instructions and possesses a well-defined I/O structure that limits the need for external components. Furthermore, the ATmega128 is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits [1], and as such make it a viable choice for DragonI's FCU. A quick perusal of its data sheet confirmed that it possessed all of the essential functions required for this thesis:

- 8-channel 10-bit Analog-to-Digital Converter (ADC): for sampling the accelerometer & gyroscope sensors
- 6 Pulse Width Modulated (PWM) Channels: for controlling the 4 motors driving the rotors on the craft
- 53 programmable I/O lines: for data transfer of the RPC transceivers and miscellaneous inputs and outputs
- I^2C, UART & SPI support
- max clock speed of 16 MHz
- an operating range of up to 5.5V
- external interrupts

I^2C **or Two Wired Interface - TWI**

I^2C was designed by Philips Semiconductors in early 1980s, it is a shorthand for Inter Integrated Circuit communication. It is a simple, low-bandwidth, short-distance protocol. The bus can operate at speeds up to 400Kbps. I^2C is easy to use and link multiple devices together since it has a built-in addressing scheme. I^2C is a true multi-master bus including collision detection and arbitration to prevent data corruption. Two of the embedded modules, namely the Ultrasonic

[1] www.avrfreaks.net

Range Finder and the Electronic compass use the I^2C bus to communicate to the Flight Control Unit.
ATmega128 provides full hardware support for I^2C bus, making the job of writing the I^2C bus driver easy. Before the bus driver was written, a study of the bus specification was undertaken to ensure the devices connected to the Flight Control Unit function properly.

Topology

I^{2}C operates over two wires as shown in the Figure 3.9. The two signals are serial data (SDA) and serial clock (SCL). Using these two signals it is possible to support serial transmission of data.

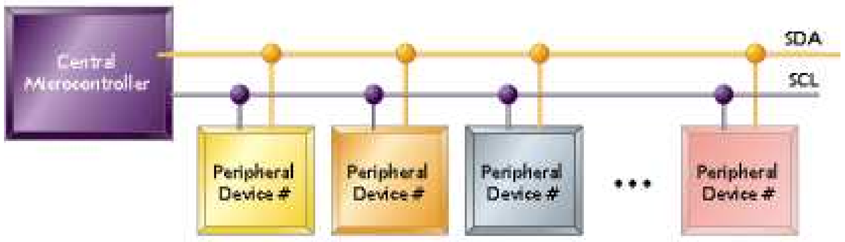


Figure 3.9: I^2C topology

I^{2}C slave devices come with a predefined device address. The master transmits the device address of the intended slave at the beginning of every transaction. Each slave is responsible for monitoring the bus and responding only to its own address. In case of the Electronic Compass and the Ultrasonic Range finder, both devices always behave as slave hence the topology of the DragonI system involved a single master (i.e. FCU) and multiple slaves. Each of the devices have been assigned a unique address and would respond to their own address.

Communication

Master begins the communication by issuing the start sequence followed by a unique 7-bit slave device address, with *Most Significant Bit (MSB first)*. The eighth bit after the start is $read/\overline{write}$. This bit specifies whether the slave is now to receive or to transmit data. The slave sends an ACK bit to acknowledge the receipt of the address. Then the transmitter as indicated by the $read/\overline{write}$ bit transmits a byte of data starting with the MSB. At the end of the byte, the receiver issues an ACK bit to acknowledge the receipt of data (see Figure 3.10). In a write transaction, when the master is done transmitting all of the data bytes it wants to

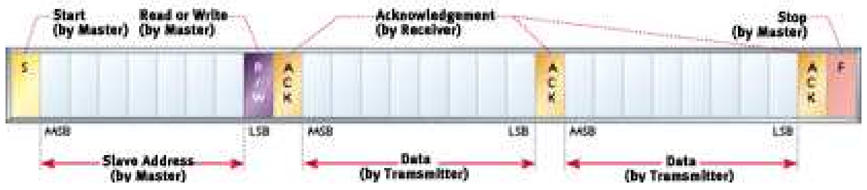


Figure 3.10: I^2C data format

send, it monitors the last ACK and then issues the stop condition. In a read transaction, the master does not acknowledge the final byte it receives. This tells the slave that its transmission is done. The master then issues the stop condition.

Since the FCU will always behave as a master and the Ultrasonic Range Finder and Electronic Compass will behave as slaves, only the parts of the protocol that were related to master were written . Since both of the devices operated at $100Kbps$, the bus was set to operate at this baud rate in the initialisation sequence (See Appendix M: i2c.c & i2c.h).

It is worth to notice that while the I^2C driver is written using polling, a time out mechanism has been put in place to make sure the failure of a certain device to respond does not obstruct the execution of the rest of the system.

Analog to Digital Converter - ADC

The accelerometer and the gyroscope sensor data are sampled using 10-bit ADC channels provided on ATmega128. In order to access the ADC outputs, driver program (See Appendix M:adc.c & adc.h) was written to initialise the ADC and functions were written to return the sampled data. Perhaps the most salient selection to note is that the AVcc pin (connected to Vcc) as the maximum voltage reference, by setting the ADC Multiplexer Selection Register(ADMUX0) appropriately.

One of the other important functionality of the the ADC is to monitor the voltage on the regulator. The RPC transceivers being very sensitive to voltage levels, a drop of about $0.5V$ may cause the transceivers to malfunction and as a result the system may go out of control.

Pulse Width Modulation - PWM

Pulse width modulation (PWM) is a technique for controlling analog circuits with a microcontroller's digital output. PWM is a way of digitally encoding analog signal levels, by modulating the duty cycle of a square wave. Timer Counter 1 & 3 are used to switch the motors of DragonI using PWM. Two important factors had to be considered when choosing PWM type:

1 PWM frequency should be high to allow fast switching of the motors

2 PWM resolution should be high to enable smooth control of the DragonI

Essentially, as the timer/counter increments from zero to its maximum value ($2^{10} = 1024$ for 10-bit), its value is constantly compared to the user-defined value. For inverted PWM signal, a low voltage level (logical 0) is output for the time period when the user-defined value is higher than time/counter's current value (see Figure 3.11).

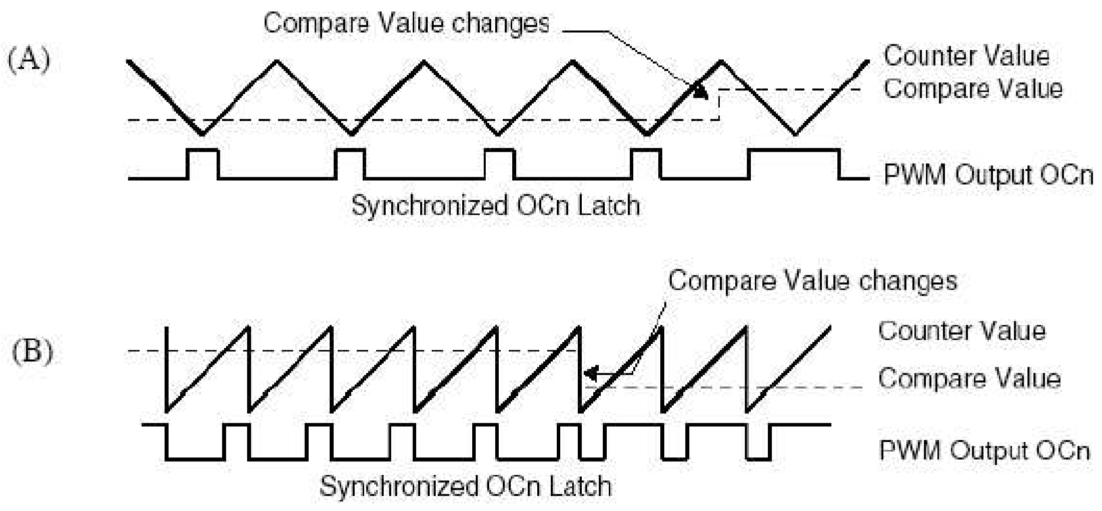


Figure 3.11: PWM output (A) Up/down mode (B) Overflow mode (Fast PWM, inverted)

Looking at Figure 3.11, clearly Fast PWM will be the choice as it satisfied the first requirement giving the highest switching frequency. Fast PWM has the option of being anything between 2-bit & 16-bit. The resolution of the PWM and the frequency of PWM are inversely proportional to each other and hence maximising both simultaneously is impossible. It was realised that a 10-bit resolution would be more than enough to give a smooth control of the DragonI. Therefore a 10-bit Fast PWM was selected to switch the motors. The frequency at which the motors are switched is given by:

$$F_{pwm} = \frac{f_{clk}}{N \cdot (1 + TOP)} = \frac{8M}{1 \cdot (1 + 2^{10})} = 7.8KHz$$

Where $N = Prescaler\ Value$ & $TOP = 2^{Resolution}$

This frequency was considered to be sufficient for switching the motors. Now the task of producing these PWM signals on the AVR present itself, and after reading through the relevant sections of the ATmega128 data sheets, the driver program for initialisation and formation of PWM were written (See Appendix M: pwm.c & pwm.h).

It is worth noting that inverted output PWM was used. The choice was made in the light of the fact that one end of motor is connected to battery and the other to connected to the PWM output of ATmega128 via Gate Drivers. Naturally we would want value of zero to give us minimum motor speed while maximum PWM value to give us maximum value. Therefore inverted output was deemed to be most appropriate.

Timer Modules

Interrupt capabilities of Timer/Counter0 is used to emulate a system clock. When initialising Timer/Counter0, clear timer on compare match mode was selected (see Appendix M: timer.c & timer.h). In this mode the counter counters up until there is a match between the counter value and the user defined value. At this point an interrupt is generated and the timer is set to zero. The timer was specifically set to generate an interrupt every $1ms$. The values for initialisation were calculated using the following formula:

$$period_{timer} = \frac{OCR \cdot N}{f_{clk}} = \frac{250 \cdot 32}{8M} = 1ms$$

Where N = prescaler $\in \{1, 8, 32, 64, 128, 256, 1024\}$, OCR = User defined value

Note: In calculation of OCR, a random value of N was chosen and then the equation was algebraically solved for OCR.

3.3.2 Electronic Compass Interface

The Electronic Compass communicates via I^2C bus as a slave device. It uses a layered protocol with the interface protocol defined by the I^2C bus specification, and the lower command protocol defined by Honeywell(see Table 3.1 for full list of commands). The command format is:

$(Command\ ASCII\ Byte)\ (Argument\ Binary\ MS\ Byte)\ (Argument\ Binary\ LS\ Byte)$

In order for the Flight Control unit to be able to communicate with the device, a driver had to be written to implement Honeywells command protocol(See Appendix M: hmc3265.c & hmc2365.h). The most important initialisation choice was to define the operational mode. The Electronic Compass has three modes:

Command Byte ASCII (hex)	Argument 1 Byte (Binary)	Argument 2 Byte (Binary)	Response 1 Byte (Binary)	Response 2 Byte (Binary)	Description
w (77)	EEPROM Address	Data			Write to EEPROM
r (72)	EEPROM Address		Data		Read from EEPROM
G (47)	RAM Address	Data			Write to RAM Register
g (67)	RAM		Data		Read from RAM Register
S (53)					Enter Sleep Mode (Sleep)
W (57)					Exit Sleep Mode (Wakeup)
O (4F)					Update Bridge Offsets (S/R Now)
C (43)					Enter User Calibration Mode
E (45)					Exit User Calibration Mode
L (4C)					Save Op Mode to EEPROM
A (41)			MSB Data	LSB Data	Get Data. Compensate and Calculate New Heading

Table 3.1: HMC Interface Commands/Responses

- **Standby Mode:** In this mode, every time a heading is requested new measurement of sensors is computed and returned. This mode is useful to get the latest data but requires $6ms$ to calculate the new heading. Being used in a real time application this delay was not acceptable and hence this mode of operation was discarded.

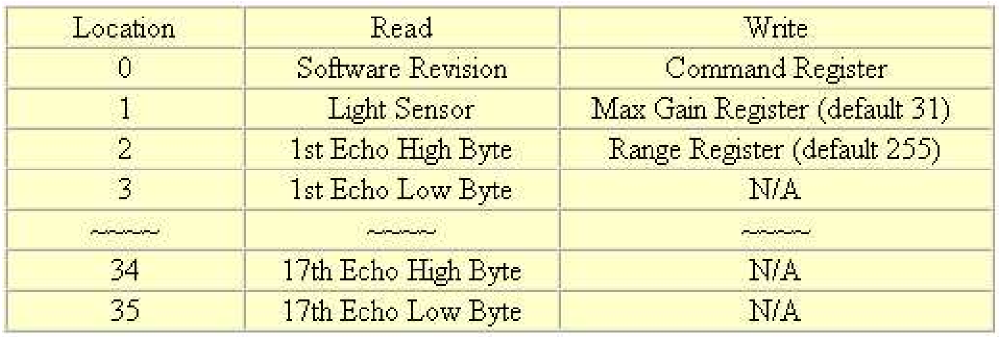
- **Query Mode:** In this mode, every time a heading is requested it is immediately returned. After each heading request the heading is automatically updated and ready for use. The trade off in this mode is the previous query latency for the advantage of an immediate read of data. This would mean the system will have a slow response time to changes in heading and hence it was deemed to be unsuitable.

- **Continuous Mode:** In this mode the system performs continuous sensor measurements and data computations at a frequency of choice {1Hz, 5Hz, 10Hz, or 20 Hz}. This mode appealed the most to our system as it will not be putting any latency in to the system and yet we will be getting a fast response. Of course this will come at the cost of extra power consumption, compared to other electronics involved the extra power needed for this operation was negligible. Since our system is refreshing at $100Hz$, the obvious choice of frequency was $20Hz$.

Other initialisation consideration worth mentioning is the periodic set/rest function which when enabled performs a re-alignment of the magnetic sensors in case of sensor perming, operating temperature shift. It also has an inbuilt moving averages filter whose window can be set to any value from 0 to 16. With the electronic compass sitting amidst such densely populated circuit and four motors, a lot of noise is expected hence the window was set to 16.

3.3.3 Ultrasonic Range Finder Interface

The Ultrasonic Range Finder(URF) communicates to the Flight Control Unit via I^2C sharing the bus with Electronic Compass. To the programmer it just appears as a set of 36 register containing the ranging information (see Figure 3.12).



Location	Read	Write
0	Software Revision	Command Register
1	Light Sensor	Max Gain Register (default 31)
2	1st Echo High Byte	Range Register (default 255)
3	1st Echo Low Byte	N/A
~~	~~~~~	~~~~~
34	17th Echo High Byte	N/A
35	17th Echo Low Byte	N/A

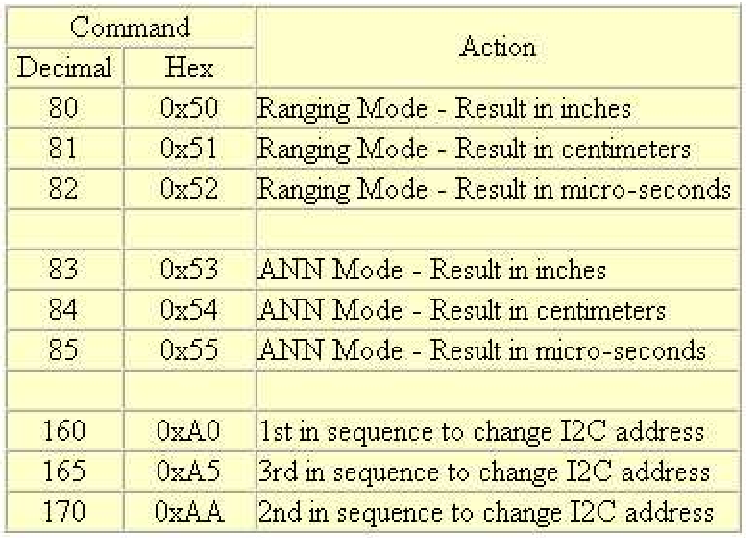
Figure 3.12: Ultrasonic Range Finder Registers

The URF executes a ranging calculation when a write operation is performed on register 0, also known as Command Register. The unit of reading returned depends on the value that was written to the Command Register. URF can return a value in cm, inches & microseconds(see Figure 3.13). The URF registers are updated after 65ms, these registers are not accessible in while the ranging is in progress.

A driver program was developed to communicate with the device over the I^2C bus(see Appendix M:srf08.c & srf08.h). It is worth to notice that the time delay for the ranging is not incorporated into the driver program, this is because the URF does not have any facilities to inform the user when the ranging is completed. This made the driver flexible, the user can use polling or can use a timer to read the ranging information after 65ms. DragonI being a real time application the latter was the obvious choice.

3.3.4 Transceiver Interface

The wireless communication between the DragonI and the base station was accomplished using the Radiometrix Radio Packet Controllers(RPC). The RPC is interfaced to the the Flight



Command		Action
Decimal	Hex	
80	0x50	Ranging Mode - Result in inches
81	0x51	Ranging Mode - Result in centimeters
82	0x52	Ranging Mode - Result in micro-seconds
83	0x53	ANN Mode - Result in inches
84	0x54	ANN Mode - Result in centimeters
85	0x55	ANN Mode - Result in micro-seconds
160	0xA0	1st in sequence to change I2C address
165	0xA5	3rd in sequence to change I2C address
170	0xAA	2nd in sequence to change I2C address

Figure 3.13: Ultrasonic Range Finder Commands

Control Unit using bidirectional ports of the the microcontroller.

To facilitate communication between the RPC and the microcontroller we were required to create a driver program (See Appendix M: transceiver.c & transceiver.h). When developing the drivers for the transceivers, the use of interrupts was deliberately avoided. This made the drivers very simple and efficient, while the use of interrupts was left for the caller of the driver routines. The user at their choice can make use of the interrupts, when it is necessary. This approach was adopted in the development of the communication protocols.

3.3.5 RPC Board

The RPC Board was developed as a facade between the PC and the Radio Packet Controllers. The microcontroller (ATmega32) provides all of the essential functions that are required of it:

- 33 programmable I/O lines: For data transfer of the RPC transceivers
- USART support: For USB transfers to PC
- An operating range of up to 5.5V
- External interrupts: Essential for efficient operation of RPC

USART

The communication link between the PC, and the RPC Board was achieved by utilising the USART module of ATmega32. The USART is interfaced with an intermediate device to convert the serial data to USB. A driver was written for this task which can be found in Appendix M(uart.c).

The most important initialisation choices were to define the timing characteristics of baud rate, character length, parity and number of stop bits, which must be in agreement with that of the computer program reading the data supplied to the USB. A baud rate of 56.7Kbps was selected which is higher than the 40Kbps maximum throughput of the RPC. Character length was set to 8-bit with no parity bit and a single stop bit.

Timer

Once again the interrupt facilities of the timer was used to emulate system clock. The driver that had been written for the FCU was ported to work on ATmega32.

Transceiver

The code for the transceiver from the FCU was also ported to work on ATmega32.

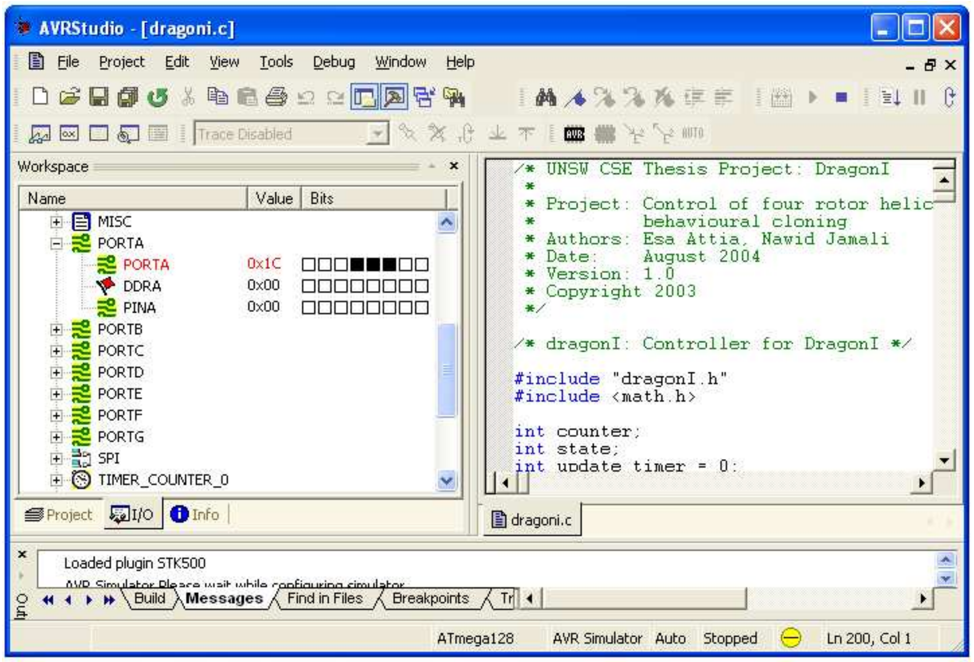
3.3.6 Firmware Development Method

Once the embedded components were decided upon, it was considered that the design and the development of the firmware were to be conducted in parallel. This would make sure that we obtain optimal throughput, by using all of the human resources available to us. This section describes the approaches taken to achieve this goal.

AVR Studio 4

To achieve development of firmware in parallel to design of the system, there was a need for a development kit. Lack of any such suitable kit during the design phase led us to exploring the option of using a simulator.

AVR Studio is an Integrated Development Environment (IDE) for writing and debugging AVR applications, Figure 3.14 shows a snap shot of the IDE. Amongst other features, AVR Studio provides a chip simulator, and although it does not provide all of the powerful functionalities of the ATmega128, such as the ADC, it facilitated the testing of most of our code modules. This software not only proved to be useful during the stages where we lacked a functioning board, but also helped us to debug sections of code written in latter stages of the thesis.


AVRStudio - [dragoni.c]
File Project Edit View Tools Debug Window Help
Trace Disabled
Workspace
Project I/O Info
dragoni.c
Build Messages Find in Files Breakpoints

Name	Value	Bits
MISC		
PORTA		
PORTA	0x1C	□□□■■■□□
DDRA	0x00	□□□□□□□□
PINA	0x00	□□□□□□□□
PORTB		
PORTC		
PORTD		
PORTE		
PORTF		
PORTG		
SPI		
TIMER_COUNTER_0		

```
/* UNSW CSE Thesis Project: DragonI
 *
 * Project: Control of four rotor helic
 *          behavioural cloning
 * Authors: Esa Attia, Nawid Jamali
 * Date:    August 2004
 * Version: 1.0
 * Copyright 2003
 */

/* dragonI: Controller for DragonI */

#include "dragonI.h"
#include <math.h>

int counter;
int state;
int update_timer = 0;
```

```
Loaded plugin STK500
AVR Simulator Please wait while configuring simulator
```

```
ATmega128   AVR Simulator   Auto   Stopped   Ln 200, Col 1
```

Figure 3.14: Snapshot of AVR Studio 4

The simulator gives a graphical representation of the ATmega128 ports together with its internal registers. The values of a port can be changed by manual manipulation any time during the simulation. To run a simulation, the .hex or .coff file that is generated by avr-gcc is loaded into the simulator. For further information on how to use AVR Studio4 please refer to AVR Studio documentation found in Appendix P.

3.4 Wireless Communication

3.4.1 Wireless Transceivers

The Radio Packet Controller (RPC) is a self-contained plug-on radio port that requires only a simple antenna, 5V supply and a byte-wide I/O port on a host microcontroller. The module provides all the RF circuits and processor intensive low-level packet formatting and packet recovery functions required to inter-connect a number of microcontrollers in a radio network. It is very easily mountable on any of the bi-directional ports of the ATmega32 & ATmega128 microcontrollers used in this project. The RPC has a reliable 30m in-building range and 120m open ground range, deemed to be an adequate range for this project. It also provides an on-board collision avoidance (Listen before Transmit), system which could be used when coming up with a transmission protocol later on.

FUNCTIONAL DESCRIPTION

A data packet of 1 to 27 bytes downloaded by a host microcontroller into the RPC's packet buffer is transmitted by the RPCs transceiver and will "appear" in the receive buffer of all the RPCs within radio range. The data packet received by the RPCs transceiver is decoded, stored in a packet buffer and the host microcontroller signaled that a valid packet is waiting to be uploaded. On receipt of a packet downloaded by the Host, the RPC will append to the packet:

- Preamble,
- start byte
- and an error check code.

The packet is then coded for security and mark-space balance and transmitted through the BiM Transceiver as a 40kbit/s synchronous stream. When not in transmit mode, the RPC continuously searches the radio noise for valid preamble. On detection of preamble, the RPC synchronizes to the in-coming data stream, decodes the data and validates the check sum. The Host is then signaled that a valid packet is waiting to be unloaded.

Transmission Protocol Procedure

The sequence for a byte transfer from the host (microcontroller) to the RPC is asynchronous and proceeds as follows:

1. HOST asserts TX Request line low to initiate transfer.
2. Wait for RPC to pull TX Accept low (i.e. request is accepted). The data lines must not be set to output until TX Accept goes low.
3. Set data lines to output and place LS nibble on the data lines.
4. Negate TX Request (set to 1) to tell RPC that data is present.
5. Wait for RPC to negate TX Accept (i.e. data has been accepted).

Receive Protocol Procedure

The sequence for a byte transfer from the RPC to the host(microcontroller) is asynchronous and proceeds as follows:

1. RPC will assert RX Request line low to initiate transfer.
2. Host pulls RX Accept low (i.e. request is accepted by the host).
3. RPC will turn on its bus drivers, place LS nibble onto data lines and negate RX Request.
4. Host reads the data and negates RX Accept (i.e. data has been accepted).

3.4.2 Wireless Implementation

In order for the user to interact with the DragonI a wireless communication link had to be established between the DragonI and the base station. This communication link was achieved with the help of the Radio Packet Controller. As described in Section 3.4.1 the RPC's just act as a medium to transfer data, a byte at a time. The RPC can only transfer data in one direction at a time and hence are half duplex systems. All of the network related protocols are left for the user to implement. Two different methods were used to achieve this communication link, for reasons that will be discussed below.

Parallel Port

Having at our disposal a library [2] that implemented the low level interactions to communicate with the RPCs via the parallel port, made it our first choice. Not only would this save us time in terms of code development, it would also make the interface to the RPC module less complex.

[2]Courtesy of Phil Preston

Utilising the existing libraries communication protocols were devised to establish a communication link between the base station and the RPCs. Even the simplest of the communication protocols (i.e sending a byte and listening for an echo) failed to establish a reliable link. This alerted us that there is some intrinsic problems with the system and a thorough testing of the RPCs were conducted.

During the system testing, indeterministic behaviours were observed. These included

- **Loss of Data Packets:** The data packets sent to the RPC connected to the parallel port never arrived. After conducting some tests it was found that the RPC connected to the parallel port needed a longer period to change from TX to RX (i.e. time between transmitting data and getting ready to listen to incoming data). The RPC connected to the FCU did not show any such behaviours. It was established that the hardware it self was intact by simply exchanging the units and running the same tests. Nevertheless this explained the loss of packets observed with our protocols and hence ruled out any fundamental flaws in the devised protocols.

- **Losing power during reset sequence:** The strangest behaviour was that the RPC would loose power during/after the reset sequence.

- **Loss of power during transmission:** Sometimes the module connected to the parallel port would lose power.

Apart from the problems mentioned above, the parallel port communication had another major drawback. The user software does not have any access to the system interrupts, only kernel modules can access the interrupts. To retrieve data from the RPCs, polling had to be used. The immediate implications of this were

1. The communication rate would be compromised, which may not have been sufficient for real time systems such as the one we are building.

2. Even more serious implication was due to the way the RPCs operate. In case of any valid packet arriving at the receive buffer of the RPC, the module gets locked until this data packet is uploaded. This may occur at any time even in between transmission of packets. To deal with this problem a check on the receive buffer had to be conducted before sending every byte of data. This would not only add to the overhead of the program and hence slow down the communication rate, it was also not going to work 100% of time, just because the atomicity of the operation can never be guaranteed. At any time between the checking and sending data to RPC a new packet can arrive.

Even if the hardware problems were investigated, pinpointed and fixed. The solution would never be anywhere near the system requirements. The overhead from the complex software that would deal with problems mention above did not appeal to our system. There was no time to explore new radio modules, just waiting for the component to arrive would have been enough to delay the work beyond what could have been accommodated, let alone the fact that the new hardware would also need testing and development of drivers. A decision was made to explore other alternative methods of communicating with the RPC, which is described in the following section.

Universal Serial Bus - USB

It was already established (from tests conducted above) that the behaviour of RPC was deterministic when it was interfaced to the microcontroller. After having quick look at the resources available to us, it was recognised instead of developing a new board with USB interface from scratch, it would be fastest to make to make use of the CSE development board designs available for the ATmega32 microcontrollers. The designs were studied to determine the suitability of the boards for the intended use. A quick look at the schematics confirmed the critical requirements of the system were met. These include:

- Availability of a port to interface the RPC
- Availability of at least one external interrupt pin (very important as discussed above)
- Power requirements needed for proper operation of the RPC

A development board, along with the required components was acquired for soldering [3]. Soldering was commenced and after testing the development board (called *RPC board* hereof), drivers for the RPC board were written (see Section 3.3.5 for more details). With new board none of the hardware issues outlined above were observed and with the comfort of access to the interrupts, the resultant communication protocols were made more robust, efficient and the overheads were minimised. Please refer to Section 3.4 for the details of communication protocols implemented on the system.

3.4.3 Transmission Protocol

In order for two way communication to take place, a handshaking protocol is needed to arbitrate communication between two RPC modules. For the project there will be two way communication

[3]Courtesy of David Johnson

between the PC and the DragonI unit. Both of which are equipped with a RPC transceiver unit attached to a host microcontroller. The protocol is divided into two parts:

- The PC side &
- The DragonI side

One of the main reasons why a protocol was needed for this system is because the system would essentially lock itself into a particular state where one side is continuously sending while the other side is continuously receiving when ideally we would want half duplex like communication continuously. Because the RPC modules would block upon receiving data in its buffer & until this data is read and the buffer is cleared, the RPC module would block itself from sending any data. Hence what was observed is that whichever RPC module starts receiving data first it would potentially sit there waiting forever until no data is being received, after which it can then start transmitting data. But since data is continuously being sent the receiving end never gets a chance to transmit data back. The second reason why the handshaking protocol was needed is because there is a chance the data being received may be out of sync due to one side holding the data back as it is receiving data and hence needs to block. When it gets the chance to send data the data may already be considered “old”. The protocol used in this system is the Master-Slave protocol due to its simplicity.

Master-Slave Protocol

It was decided that the Master-Slave Protocol was to be implemented to arbitrate the communication between the PC and the DragonI. The PC side of the system is the Master. It initiates the data transfer by requesting to send data to the DragonI unit. The sequence for a transfer session from the PC to the DragonI RPC is asynchronous and proceeds as follows:

* The PC side sends a Control Byte which has a value of 2 followed by another packet signaling the start of transmission (the TXR Packet). The Control byte is a preamble which lets the receiving RPC know that it is expecting two bytes of incoming data including the control byte itself. After sending the two bytes of data the PC then waits for a reply. If there is no reply after a period of time, it will timeout and set its state to idle and repeat the transmission process again until a reply is received.

- The RPC on the DragonI sees the incoming control byte & checks to see how many packets of data are coming in. It then goes on to read the TXR byte which signals the start of data transmission cycle. After successfully reading in the TXR byte an ACK is sent back to the PC. The RPC then waits for the next incoming packet.

* Once the PC has received the ACK from the DragonI, it then proceeds to transmit data along with its control byte to the DragonI. After transmitting data, the PC then waits for data to be sent to it from the DragonI (i.e accelerometer , gyroscope , ultrasonic data etc). If no data is received after a certain timeout period. The PC will reset its state and restart the transmission cycle.

- The DragonI receives the control byte which tells it how many packets are incoming. It then starts reading in the data coming in from the PC.

- after receiving all the data from the PC. The DragonI then begins to transmit data back to the PC. After doing this it then set its state to IDLE and the whole transmission process is repeated again.

Protocol Implementation

In order to implement the Master-Slave transmission protocol mentioned above (see section 3.4.3), software needs to be implemented on both sides of of the transmission medium (PC & DragonI side). Since the PC will be the Master in this Protocol. Its code will be slightly more complex than those on the DragonI side as it needs to issue the commands and respond to the slaves reply at every stage of the transmission protocol.

The RPC board has has 4 transmission states:

- IDLE State
- TXR State
- TXA State
- RXA State

Since the Protocol is interrupt based, the behaviour of the RPC board would be different depending on weather or not an interrupt has been triggered. When the RPC Board is *not* interrupted (i.e the interrupt line is high)it means that it is either in the process of *sending data* to the user or *waiting for data* to be received from the DragonI (see figure 3.16).

On the other hand when the RPC board is interrupted (i.e the interrupt line is asserted low) it means that the RPC has received a data packet in its buffer which has to be cleared

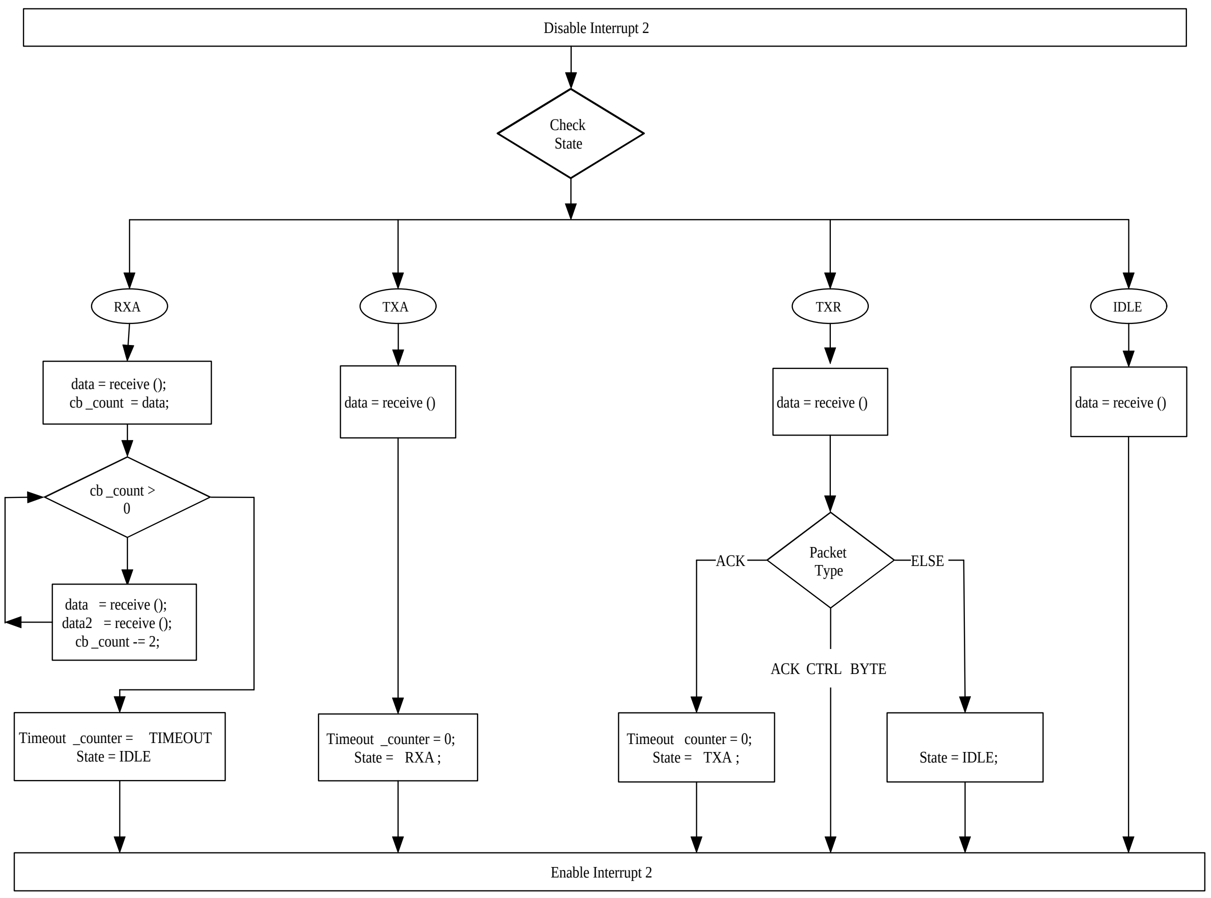

Disable Interrupt 2
Check State
RXA
TXA
TXR
IDLE
data = receive (); cb _count = data;
data = receive ()
data = receive ()
data = receive ()
cb _count > 0
data = receive (); data2 = receive (); cb _count -= 2;
Packet Type
ACK
ELSE
ACK CTRL BYTE
Timeout _counter = TIMEOUT State = IDLE
Timeout _counter = 0; State = RXA ;
Timeout counter = 0; State = TXA ;
State = IDLE;
Enable Interrupt 2

Figure 3.15: RPC Board receive protocol

(see section 3.4.1 for detailed description of RPC operation). Hence the RX line on the RPC module gets asserted low. Since the RX line on the RPC is connected to the external interrupt pin on the microcontroller. This would cause our program to interrupt & execute a different set of instructions. When in an interrupted state, the RPC board is in the process of *receiving data* from the DragonI.(see figure 3.15)

The DragonI on the other hand will act as the slave in this transmission protocol. It will respond to the master's request (RPC board) by either transmitting back an acknowledgment or data to the RPC board. The DragonI has five transmission state:

- IDLE State
- TXR State
- TXA State
- RXA State

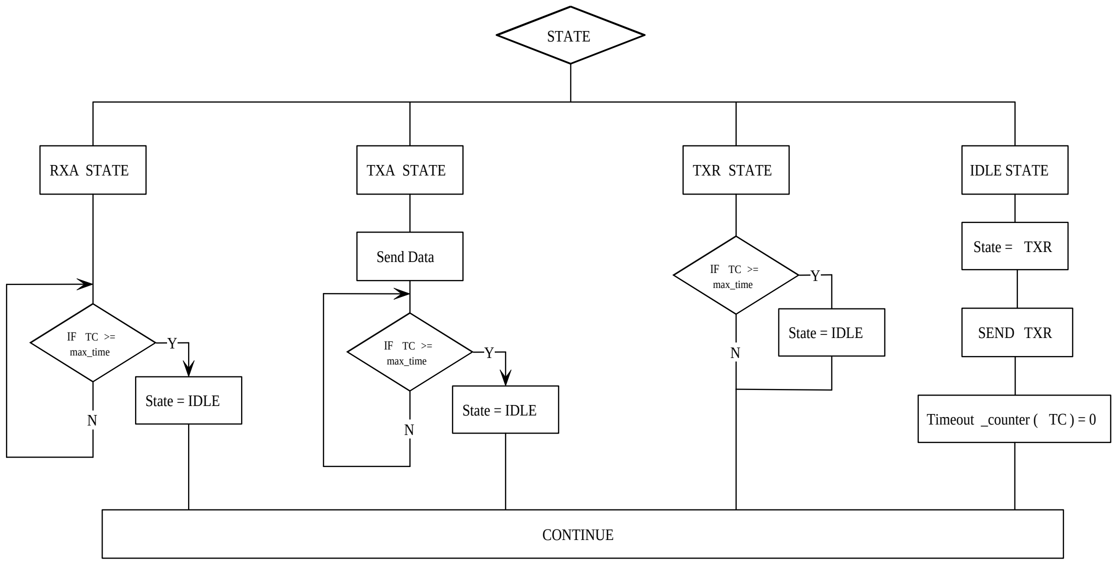


Figure 3.16: RPC Board send protocol

- SEND State

Whenever the RPC module on the DragonI detects an incoming packet, the DragonI will execute its interrupt routine and depending on the current transmission state of the DragonI, it will execute the appropriate transmission protocol response. (see figure 3.17 for details)

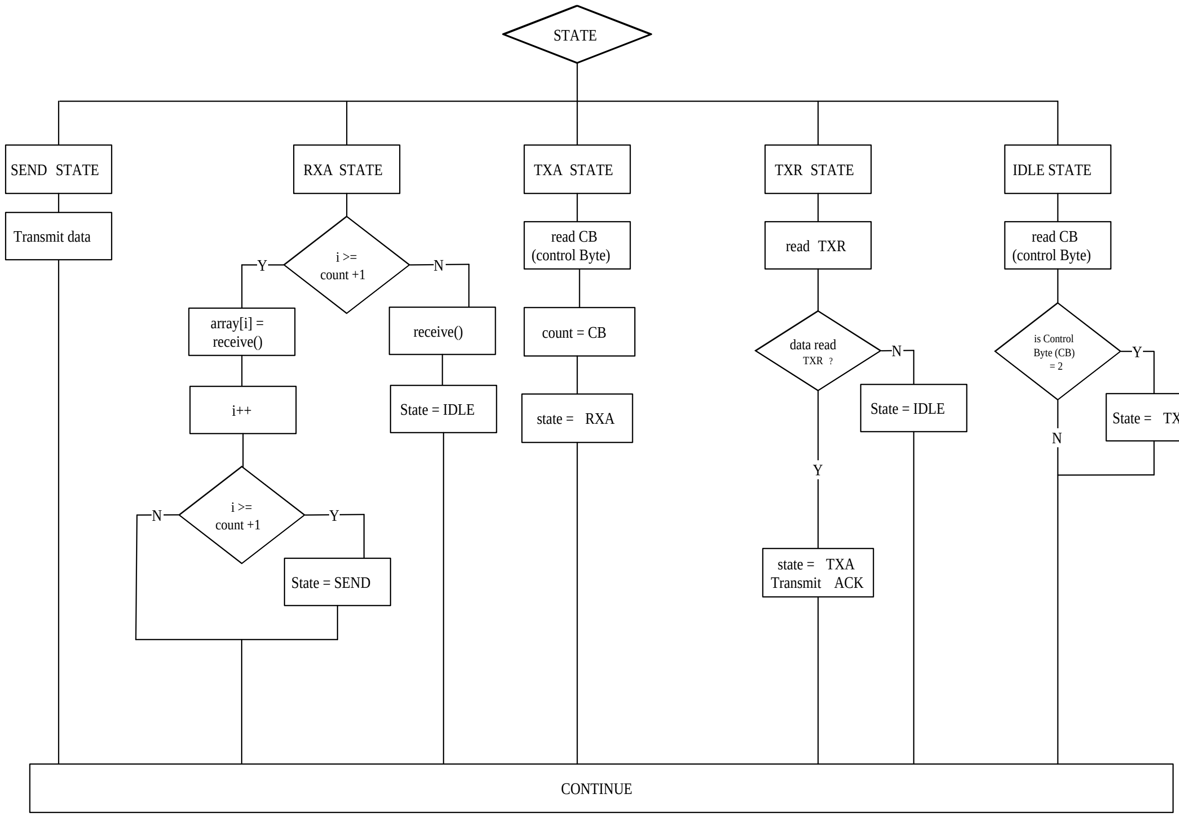


Figure 3.17: The transmission protocol onboard the DragonI

3.5 Graphical User Interface

It was decided that the user interface would be coded in C and C++. One of the main reasons why C was used, was that the libraries for the joystick (libjsw.h) and serial port communications (serial.h) which we have written ourselves, were in C. C++ was used for the rest of the application mainly due to the ease of coding inherent with an object oriented language, especially for GUI applications. It was decided that the main interface to the DragonI would operate under a Linux platform. The reason for this was mainly due to the fact that Linux is more friendly as far a coding is concerned.

The GUI was designed using QT version 3.1.1 which readily comes with the Red Hat 9 package. Because QT has support for both C and C++ it was the ideal choice for the development of the Graphical User Interface. One of the great features of QT was the fact that it comes as a toolkit with QT designer - a GUI development application, which makes coding the GUI a simple task as objects can simply be dragged and dropped into their appropriate locations with very little coding involved.

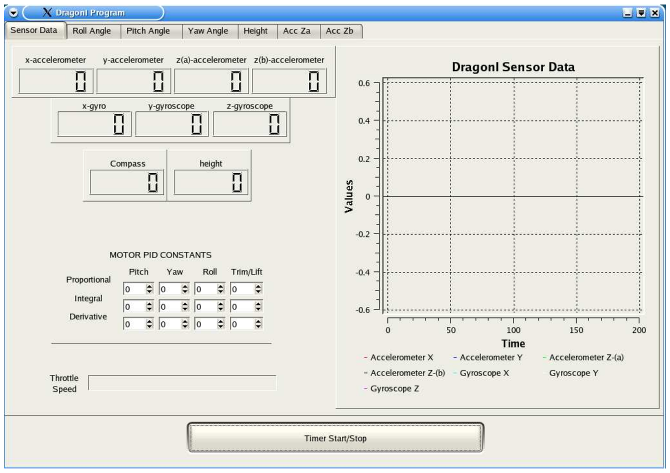


Figure 3.18: Screen shot of the DragonI GUI application

3.5.1 Functionality

One of the main reasons why we would like to make use of the graphical user interface is to provide the user with some feedback on the responsiveness of the PID controller as well as providing the user with a means to tune the PID constants in real-time and control the the Draganflyer via a joystick during semi-autonomous control. Although it would be possible to create a simple text based application to do some of the basic functionality mentioned above. The user will not be able to get a true feel of how well their PID is working tuned. As this project develops it would also be possible to incorporate things such as live video feeds from the wireless camera onboard the DragonI into our application.

3.5.2 Design

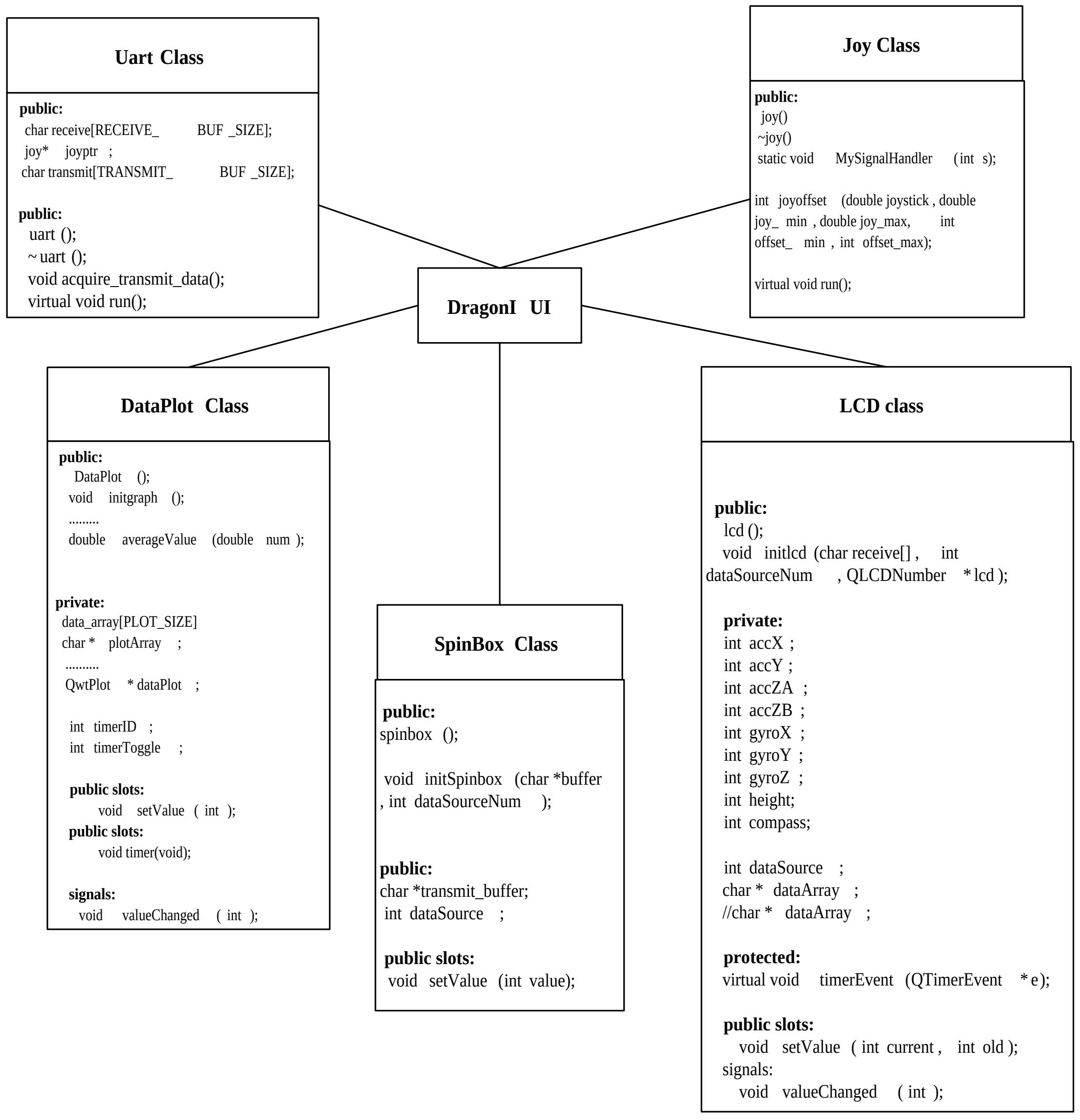


Figure 3.19: Class diagram of the base station's GUI

The DragonI Class - dragonI.cpp

The DragonI class is the main class which contains references to all objects within the GUI application. It is the main interface class which allows the main application to interact with the graphing class (DataPlot.cpp) , joystick input class (joy.cpp) , LCD display class (lcd.cpp) , UART interface class (uart.cpp) & the class responsible for getting the inputs of the PID constants from the user (SpinBox.cpp). (see figure 3.19 for class diagram of the GUI)

Joystick Class - joy.cpp

The joy class (Joystick Class) extends the QThread class which is part of the standard QT library. By extending the Qthread class , the joy class is able to run as a separate thread which continuously reads the joystick and places the data on a buffer which serves as input to the DragonI.

Serial Communication Class - uart.cpp

The UART class also extends the QThread class. It acts as an intermediary between the RPCBoard and the GUI by running as a separate thread continuously polling the serial USB line as well as the joystick data buffer. The UART class makes use of the serial port library we have written in order to communicate with the USB port so that we could transmit setpoints to the RPCBoard as well as receive sensor data being sent back from the DragonI .

Data Plotting Class - DataPlot.cpp

The DataPlot class is primarily responsible for the graphing of the data being received from the transmit & receive buffers. The widget used to display the graphs were obtained from QWT (a SourceForge project developing useful QT widgets).

LCD class - lcd.cpp

The LCD class is used to print raw data coming from the receive buffers. These give the user the exact value of the sensor outputs coming from the DragonI.

Serial port library - serial.c

The serial port library which we had implemented were based on the examples provided by Michael R. Sweet - "Serial Programming Guide for POSIX Operating Systems". They provide basic functionality to access the serial port.

Chapter 4

Sensor Characterisation

Since autonomous control is a central feature of our system, we need to perform attitude and heading estimation with the help of onboard sensors. This chapter studies the different characteristics of the sensors. The aim of the sensor characterisation is to establish strengths and shortcoming of the sensors as applied to the system at hand.

4.1 Experimental Setup

The DragonI was mounted on a *Universal Joint*(please refer to Section 7.1 for more detail). This setup allowed different experimental parameters such as tilt, heading and rotor speed to be varied for different sensors with minimal restrictions to the system. The aim of the setup was to eliminate any restrictions and simulate behaviours close to what would be experienced in real world. Different parameters were investigated for different sensors.

4.2 Accelerometers

4.2.1 Experimental Parameters

When investigating the characteristics of the the accelerometers, two parameters were looked into:

Tilt: The DragonI was tilted at different angles in order to test its response to change in angle. The following angles were observed $\pm 10°$, $\pm 20°$, $\pm 30°$, $\pm 40°$, $\pm 50°$, $\pm 60\circ$, $\pm 70°$

Speed: The Rotor speed was increased, with the aim to observe the introduced noise.

4.2.2 Results

Sensor characterisation experiments on the accelerometers revealed that the sensors have adequate sensitivity to tilt. The accelerometers were able to detect tilts of about 1° in resolution.

Furthermore it was revealed that the accelerometers are very sensitive to the rotor speed. As the speed was increased the noise in the data increased. At fast speeds the error between the actual angle and the angle calculated by accelerometers would grow to large values such as 60° see Figure 4.1.

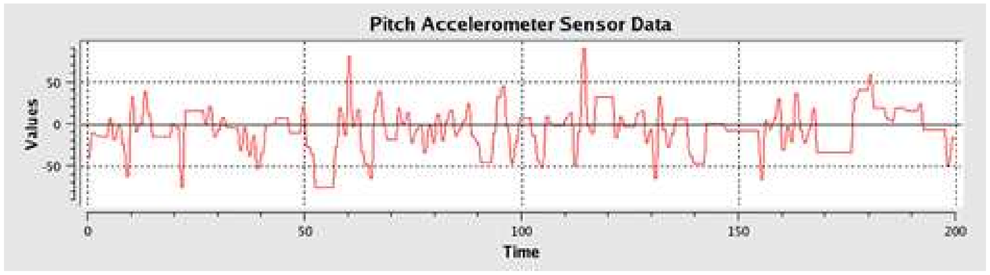


Figure 4.1: Accelerometer output: $Angle = 0°$, $RotorSpeed = fast$

4.3 Rate Gyroscope

4.3.1 Experimental Parameters

In the investigation of the rate gyroscopes, the following parameters were looked into:

Rotor Speed: The Rotor speed was increased, with the aim to observe the introduced noise.

Rotation: The reaction of the gyroscope on its axis of rotation.

4.3.2 Results

Some interesting behaviour of the gyroscopes were observed in this experiment. An undesired behaviour of the gyroscope was observed when it was rotated and then stopped. Normally it is expected, after the initial rise, the value will return to zero. But it was not the case, the data retrieved from the Flight Control Unit showed that the value will over shoot the zero and then attenuates to zero. To eliminate the fact that the amplifier circuit was interfering with the output of the gyroscopes, an oscilloscope was used to get the raw output of the gyroscope and the amplified output. The data showed the same behaviour in the unamplified signal(see Figure 4.2).

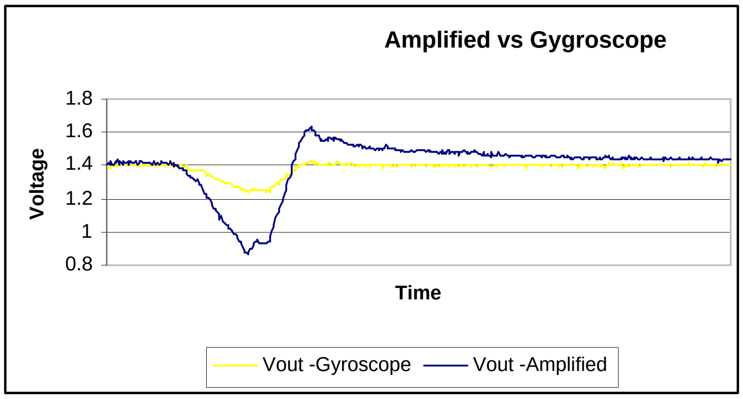


Figure 4.2: Gyroscope overshoot

Moreover we noticed that the gyroscopes measuring the rate of change around the $X - axis$ & $Y - axis$ were less prone to noise due to rotor speed while the $Z - axis$ gyroscope detected noise (See Figure 4.3). It was also noted that the gyroscopes respond fairly quickly to changes in the angle of the DragonI. They were more responsive than the accelerometers.

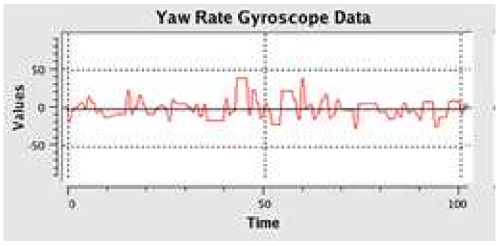


Figure 4.3: Gyroscope Z-axis output: $Angle = 0°$, $RotorSpeed = fast$

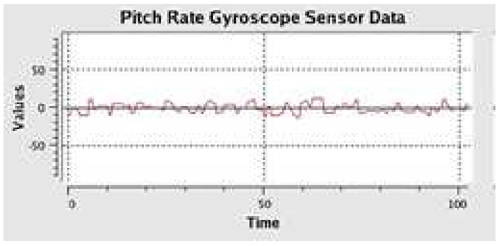


Figure 4.4: Gyroscope X-axis output: $Angle = 0°$, $RotorSpeed = fast$

4.4 Electronic Compass

4.4.1 Experimental Parameters

The parameters of interest for the Electronic Compass were:

Rotor Speed: The speed of the rotors were increased, with the aim to observe the introduced noise due to the magnetic fields created by the motors.

Heading & Tilt: Different headings were used - 0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°, and the compass was tilted at different angles while keeping the heading constant - ±10°, ±20°, ±30°, ±40°, ±50°, ±60○, ±70° to observe the effect of tilt on the heading.

4.4.2 Results

The magnetometer passed the Rotor Speed test with flying colours. No sign of interference from the electric motors was noticed in the signal. But the heading test as expected failed. The data sheets for the electronic compass mention a divergence of 2 degrees per degree of tilt. The observed divergence was random, and non linear. Figure 4.5 shows the results for roll and pitch axis. The curves for both roll and pitch show the same shape.

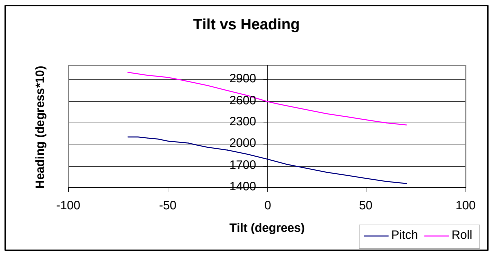


Figure 4.5: Compass heading vs tilt: $RollHeading = 2601°$ & $PitchHeading = 1800°$

While this behaviour was noticed for the headings, it was necessary to do more tests on the electronic compass having different combinations of the pitch and tilt. To get a better feel of the sensitivity of the compass to tilt, another experiment was set up. This time the heading was kept constant, then the DragonI was tilted on the pitch and roll simultaneously. The results were graphed[1] in 3D to look for irregularities in the correlation between tilt and the deviation from the heading.

[1]Using Graphis, by Kylebank Software Ltd

Figure 4.6 shows the 3D representation of the data collected along with 2D representation of the same data for roll and pitch angles(Figures 4.8 & 4.7). The graphs clearly show an indeterministic behaviour of the sensor. This is not a desired behaviour but this means that the sensor cannot be used as is. Some intelligent filtering is needed to extract the required information. One possible solution is to mount the compass on a mechanical gimbal in order to compensate for the tilt.

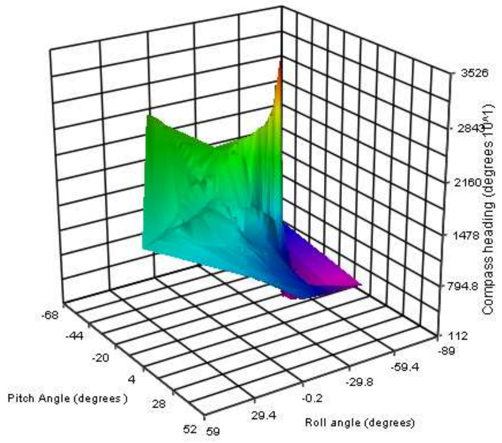


Figure 4.6: Compass heading vs tilt: $Heading = 174.6°$

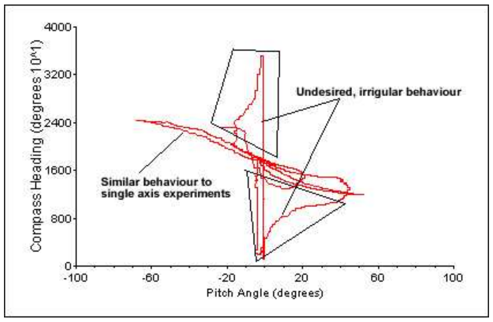


Figure 4.7: Compass heading vs pitch: $Heading = 174.6°$

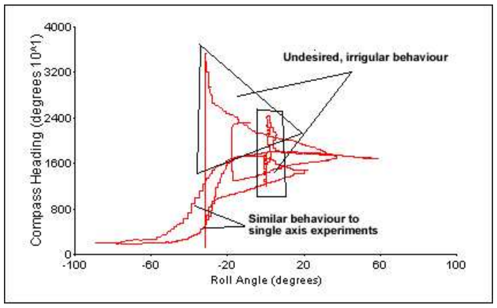


Figure 4.8: Compass heading vs roll: $Heading = 174.6°$

4.5 Ultrasonic Range Finder

4.5.1 Experimental parameters

The Ultrasonic Range Finder was tested at different heights and different surfaces. Tests have confirmed characteristics published by the manufacturer. Anything below $3cm$ in range would cause the data for the ultrasonic range finder to become invalid, as it can be seen in Figure 4.9, the values are indeterministic.

4.5.2 Results

In the operating region (i.e. $3cm$ to $6m$) the signal were fairly consistent with some spikes which may have been caused by echos received from distant objects of the previous ping.

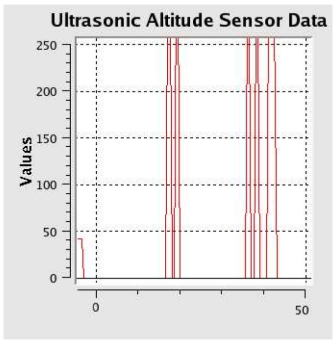


Figure 4.9: Ultrasonic output: $Height = 0cm$

4.6 Remarks

The sensor characterisation tests illustrated the strengths and shortcomings of the sensors tested. While the tests confirmed the sensors adhere to the specifications published by the manufacturer, it highlighted the need for filtering to get a sensible data from them. From the sensor characterisation experiments ,the following points were concluded:

- The accelerometers suffer from slow response time and high output noise, hence in the short run they do not provide good estimate of the attitude of the DragonI. However since the noise in the signals are white noise with zero mean they provide better estimates in the long run.

- The gyroscope provides a fast response time with low output noise, since the signals have to be integrated over time to estimate attitude of the system, even very small errors in measurement are amplified which results in divergence of the estimated value from the real value. This is also known as the random walk problem. This property is called the drift rate of the gyroscope. Hence in the short run they will give more superior results than the accelerometers. Moreover the unexpected behaviour observed above(see Figure 4.2) creates problems in the integration process, which needs to be addressed.

- The electronic compass however is not effected by any environmental noise, but is very sensitive to tilt.

- The ultrasonic range finder gives reliable data within the specified operating zone. The minimum height of $3cm$ will put a burden on the system as the ultrasonic range finder is sitting at the bottom of the DragonI. Careful consideration needs to be taken into account to rectify the situation.

Chapter 5

Attitude Estimation Algorithm

Accurate attitude determination is one of the most important tasks of any UAV project. For this purpose a variety of sensors are used to estimate the attitude of the DragonI. These sensors include rate gyroscopes, accelerometers electronic compass and ultrasonic range finder. More than one of these sensors measure the same physical property. An engineering approach would be to employ state estimators to fuse the signals from different sensors in order to produce an accurate estimate of the system state.

The roll and pitch can be obtained both from the accelerometers and the gyroscopes. As discussed in previous chapter the gyroscopes give an excellent result in the short run, but in the long run it suffer from the random walk problem, while the accelerometers are not reliable in short run they have a stable response in the long run. Therefore both these sensors measure the same physical property which have desirable output in mutually exclusive regions. To utilise the strengths and eliminate the shortcomings of the sensors the signals from the two sensors can be fused to obtain a superior estimate of the attitude of the system. Similarly the signals from the gyroscope and the electronic compass can be fused to get a better estimate of the heading.

5.1 Considerations

Sensor fusion is a challenging technical barrier because fusion must take place in near-real time in order to be useful for stabilisation. Hence when looking at various filtering methods to be used for sensor fusion, some considerations had to be taken in to account. Simplicity and low computational complexity were the primary concern when choosing the algorithm, in light of it's eventual adaption into real time system. Thus ensuring that algorithm of choice involved:

- Performing relatively inexpensive integer operations, as opposed to hardware intensive floating point operations
- Have a fast response time to changes in true state
- Filter out the noise in the signals

The following sections outline the algorithms that were considered.

5.2 Moving Averages

5.2.1 Implementation

One of the simplest algorithms to implement was to average the accelerometer and gyroscope signals and take a weighted average of these two signals. Moving averaging was used to average the accelerometer and gyroscope signals. This algorithm can be implemented recursively i.e. the new average is calculated based on previously calculated value. Now suppose at an instant k we want to average over n latest samples of signals S_i, the average is given by the following formula:

$$\overline{S}_k = \frac{1}{n} \sum_{i-k-n+1}^{k} S_i \tag{5.1}$$

Similarly in time k-1, the average of latest n samples is given by

$$\overline{S}_{k-1} = \frac{1}{n} \sum_{i-k-n}^{k} S_i \tag{5.2}$$

Therefore,

$$\overline{S}_k - \overline{S}_{k-1} = \frac{1}{n}\Bigg[\sum_{i-k-n+1}^{k} S_i - \sum_{i-k-n}^{k} S_i \Bigg] = \frac{1}{n}\Bigg[S_k - S_{k-n} \Bigg] \tag{5.3}$$

Rearranging equation 5.3 gives:

$$\overline{S}_k = \overline{S}_{k-1} + \frac{1}{n}\Bigg[S_k - S_{k-n} \Bigg] \tag{5.4}$$

This method is called the moving window averaging with n being the window size. This method was used to act as a low pass filter to filter out the noise in the accelerometer and gyroscope data. As illustrated in Figure 5.1 the output of the Low Pass Filters (LPF) for the gyroscope and accelerometer are fused together by using a weighting. The value of the weighting signifies the amount of trust that can be put in the particular sensor.

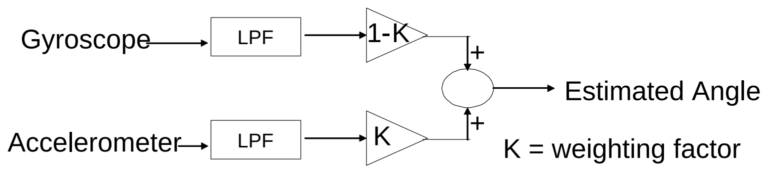


Figure 5.1: Moving Window Averaging diagram

5.2.2 Results

Once the algorithm was implemented various values of K (i.e. weighting) and window size were used to evaluate the effectiveness of the algorithm. As the value of K was increased, that is more weight was put on the accelerometer measurement, the bias of the gyroscope was reduced, but in the long run the estimate would still deviate from the true state. At the same time the noise in the accelerometer signals gained dominance. To counteract this the window size had to be increased, but larger window size resulted into a slow response time from the filter. Soon it was realised this algorithm is not going to work for the system. There were two fundamental problems with the algorithm:

1. To filter the noise in the signals adequately, a larger window size was needed. This meant that a fast response time was compromised for a smoother signal.

2. No matter what value of gain K was used it could not eliminate the eventual drift of the gyroscope, and as a result the estimate would drift as well.

The algorithm was relatively low on computational costs, but it could not filter the noise adequately without compromising the response time. Moreover the estimated angle drifted as a result of integration of gyroscope signal. Figure 5.2 shows the averaged accelerometer signals and the averaged gyroscope signals, averaged with a window size of 32 samples. The last graph shows the estimate of the system state obtained by fusing the accelerometer and gyroscope signals. It is clearly visible that the gyroscope is causing the estimate to drift over time.

While the algorithm failed for the roll, pitch and heading estimation, our hard work did not go in vain. The moving window averaging algorithm proved to be effective for the estimation of height. The reasons being the height does not have fast dynamics and it is more tolerant to delays.

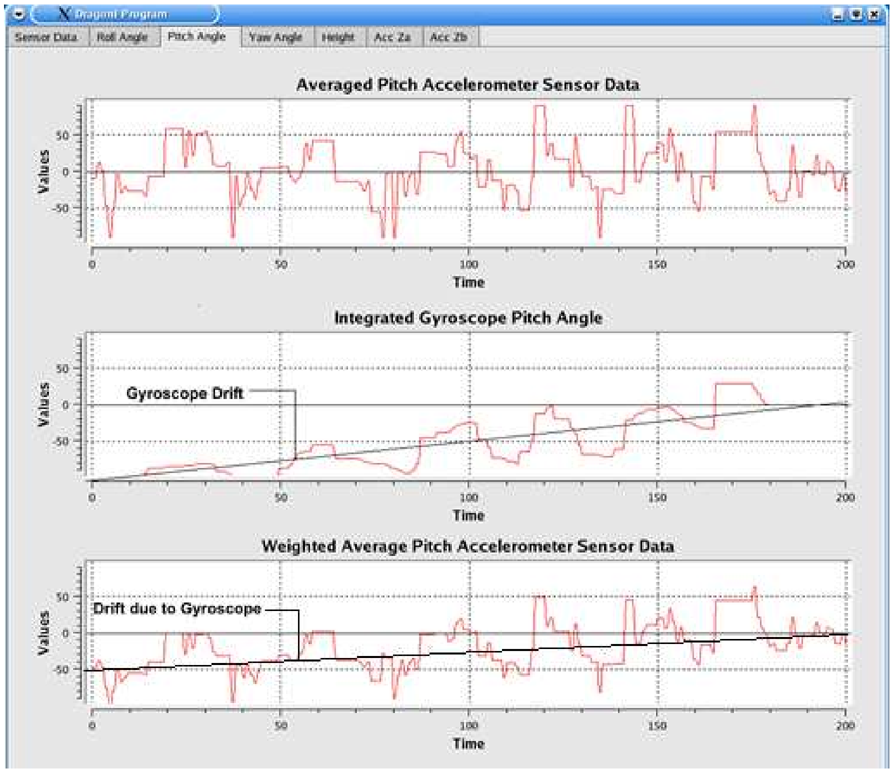


Figure 5.2: Moving Window Averaging filter output

5.3 Kalman Filters

The next algorithm that we looked at was Kalman filtering. A Kalman filter is an optimal recursive filter. Since Kalman filter is recursive, it "does not require all previous data to be kept in storage and reprocessed every time a new measurement is taken"[8]. The Kalman filters make some basic assumptions:

1. The system is linear
2. The noise is white Gaussian noise

Kalman filter can provide optimal filtering and hence optimal attitude estimation as it incorporates all of the available information. The estimation process uses all of the measurements made by the system, regardless of their precision. It uses "knowledge of the system and measurement device dynamics, the statistical description of the system noises, measurement errors and uncertainty in the dynamics models, and any available information about initial condition of the variable of interest"[8] to generate an optimal estimation of the states.

Figure 5.3 shows a typical application of Kalman filters. As illustrated the system modeled has errors in it, just the fact that no two physical components can be identical can cause errors in the system and the system does not behave as it was modeled. The measuring devices have their own errors. The system is effected by its controls, in our case the joystick when under human control, and the autonomous controller when the system is being controlled autonomously. To extract the true state of the system the estimator can use the observed measurements to get an optimal estimate of the system state.

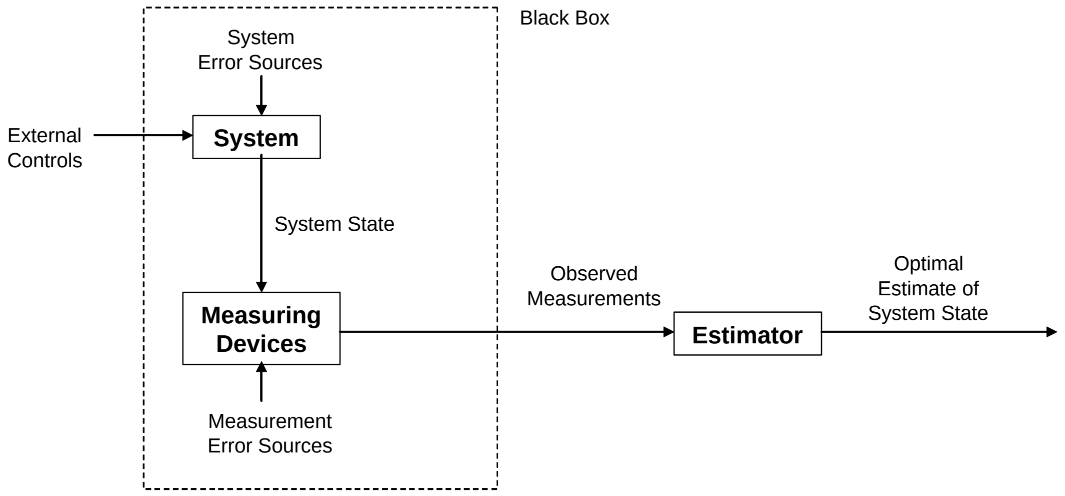


Figure 5.3: Typical Kalman filter application

Kalman filter is suitable for use in any system, where process and the measurement models can be described linearly. For non-linear systems such as ours, Extended Kalman filters can be used. An Extended Kalman filter is one which applies the Kalman Filter to the linear approximation of the non-linear system[11]. Since Kalman filter combines all of the measurements of the system, it is a suitable filter for sensor fusion.

To use the Kalman filter, first of all one must find a representation for the states of interest i.e. the states that have to be tracked and estimated, then a representation of the dynamics of the process that relates the transition from current state to the next state has to be devised. Finally a model for the measurements is to be found. Once this is achieved the Kalman filter works in two steps

1. **Prediction step:** Using the system dynamics model, make a prediction what should be the state of the system in time the next time step.

2. **Correction step:** Whenever there is a measurement available, the prediction is corrected

by blending prediction and residual between prediction and the measurement.

While Kalman filtering could be more superior than other filtering algorithms, it was going to be computationally expensive and a slow process introducing latency in the system and hence not meeting real-time requirements of the system. For this reason the Kalman filter was deemed to be unsuitable for the system.

5.4 Weighted average Infinite Impulse Response (IIR) Filters

5.4.1 Implementation

Research about Kalman Filters led the way to IIR filters. The weighted average IIR filters are computationally inexpensive algorithms. The algorithm does not need any pre-filtering of the signals and hence there is no compromise in the response time. The algorithm works by adding a fraction of the error between accelerometer and gyroscope to the integrated gyroscope data.

Figure 5.4 shows the heart of the algorithm. The algorithm integrates the gyroscope signals to estimate the state of interest. The value of the estimate from the gyroscope is corrected with measured value of the state from the accelerometers This is done by multiplying the residual between gyroscope and the accelerometer by a constant K called the weighting factor of the filter. The value of weighting factor K determines whether the estimate obtained from gyroscope is to be trusted more or the accelerometer signal. Since the accelerometer data is more noisy than the gyroscope data, the weighting is usually small. This makes sure that the estimate is not effected by the noise in the accelerometer. Since in the long run accelerometer data is more reliable(see Section 4.2 for more details), this makes sure that the estimate does not diverge from the true state, hence it eliminates the drift problem associated with the gyroscopes.

$$
\begin{aligned}
&angle_{estimate} = angle_{estimate} + data_{gyro}dt \\
&angle_{estimate} = angle_{estimate} + (angle_{acco} - angle_{estimate})K \\
&Where\ K\ =\ weighting\ factor
\end{aligned}
$$

5.4.2 Results

The results obtained from this algorithm are fast. The key for this algorithm to work is to choose the right value of weighting factor K. To obtain the right value of K several experiments were run. The value of K was changed starting from 0.5 (i.e. 50% weighting on each of the

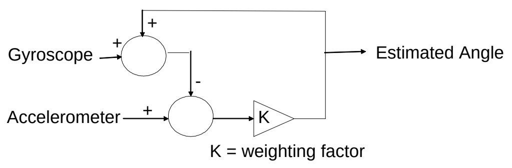


Figure 5.4: Weighted Average IIR diagram

signals) and was divided by five everytime. To evaluate the system for a particular value of K, the rotors were run at a medium speed, then the DragonI was positioned horizontaly such that it is parallel to the ground. It is then rotated 90 degrees and held in that position for a period of 1 seconds then rotated back so that the DragonI is back to its original horizontal position. This procedure was repeated 10 times for each value of K. The accelerometer signals and the output of the weighted IIR filters were graphed and visual inspections were conducted into the response time of the system and noise level present in the signal. Once it was noticed that any further reduction in the value of K shows signs of drift, the value was multiplied by five and used as the weighting factor of the graph.

Figure 5.5 shows the output of the filter, the first graph shows the output of the accelerometer and the second graph shows the output of the filter. Evidently the filter smoothes out the noise from the accelerometer and at the same time eliminates the gyroscope bias. The resultant attitude estimation was far more superior than the moving window averaging. A better noise reduction was achieved without compromising the response time, there is no drift due to gyroscope.

5.5 Weighted average IIR Filters with fuzzy logic

5.5.1 Implementation

Weighted average IIR algorithm gave a good result for roll and pitch estimation, it was not a sufficiently good algorithm for the yaw estimation. Recall from Section 4, the electronic compass is very sensitive to the tilt and hence needs to be tilt compensated. To tilt compensate an electronic compass successfully a three axis electronic compass is required. A three axis electronic compass is one that senses the earths magnetic field in x, y and z directions. Using these three components it is possible to tilt compensate the electronic compass heading. Since our electronic compass is only two axis, standard methods of the tilt compensation were not applicable.

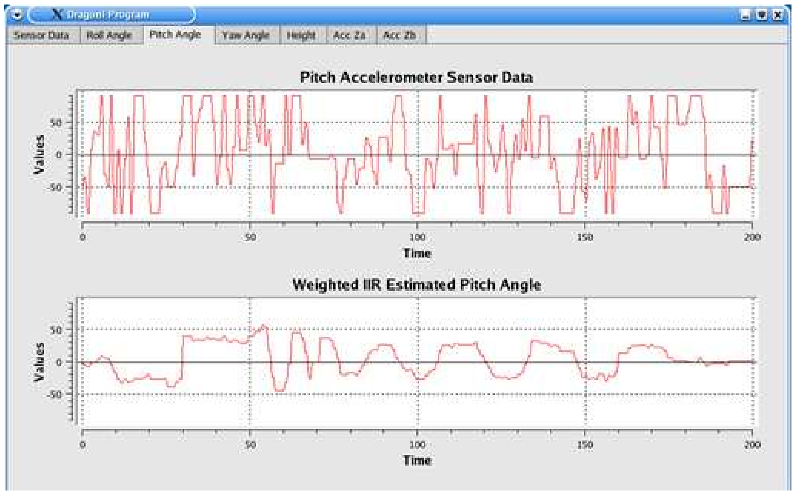


Figure 5.5: Weighted Average IIR filter output

To filter out the undesired behaviour of the electronic compass, a fuzzy logic was integrated into the previously implemented IIR filter. The fuzzy logic took the advantage of the fact that when the electronic compass is level, the readings are very accurate and as the DragonI starts tilting in any direction, the data becomes unreliable. The more it is tilted the more unreliable the data becomes. At the same time the gyroscopes gives us the rate of angular change which can be integrated over time to get the angle of the yaw. As discussed before the gyroscope drifts over time. Using this knowledge, the weighting of the IIR filter was dynamically calculated based on the estimated tilt and pitch of the system, and hence an adaptive filter was implemented to fuse these two signal.

Table 5.5.1 shows the rules that were implemented for the filtering of the electronic compass. It illustrates as the angle of the pitch or roll increases the system lowers the value of K, which in turn puts less and less trust into the electronic compass eventually the value of the electronic compass is discarded and the data from gyroscope is used to estimate the yaw angle.

1. If roll is less than $Threshold_1$ and pitch is less than $Threshold_1$ then K is 0 .5
2. If roll is less than $Threshold_2$ and pitch is less than $Threshold_2$ then K is 0.05
3. If roll is less than $Threshold_3$ and pitch is less than $Threshold_3$ then K is 0.005
4. for all other cases K is 0

Table 5.1: Rules of the fuzzy logic algorithm

Where $0 < Threshold_1 < Threshold_2 < Threshold_3 < 1$

5.5.2 Results

Although the filter is not an optimal, it is better than any other option that is available to us. A few tests were run to see if the filter performs reasonably. This algorithm is still under testing as this reporting is being written.

5.6 Conclusion

Different filtering techniques were investigated for estimating roll, pitch, heading and height of the DragonI. After running several tests it was found that:

- The moving averages algorithm was not suitable for the roll pitch and yaw estimation. Signal smoothing could only be achieved by introducing latency in the response time of the state. These states are vary dynamic and any subtle changes are of importance to the control system. In the case of the height estimation the moving window averaging was sufficient as small changes in the state are not as critical as is the smoothness of the signal.
- The Kalman filters were not used as it is computationally expensive algorithm.
- Weighted IIR filters proved to be a better filtering method for the system. The algorithm is efficient, and achieves good results without introducing any computational complexity or compromising the response time. Therefore it met all of the requirements of the ideal filtering method for the system.
- Augmentation of the fuzzy logic to the Weighted IIR filter made it possible to filter out the undesired behaviour of the electronic compass to some extent.

Chapter 6

Control and Control Architecture

6.1 Considerations

When designing the control architecture there are several issues that have to be considered. Some of the issues that had to be considered are:

Efficiency: Since the control loop is being executed continuously, it has to be efficient so that it could handle real time events. Hence simplicity of the algorithm and low computational costs were taken in as primary considerations when designing the control loop.
Cross-coupling: Cross-coupling happens when a change in one part of the system effects some other part, and vice versa. These two parts are said to be cross-coupled. From controls perspective, one would like to have all of the control parameters decoupled. This gives the controller the flexibility to control each parameter separately without considering the effects on other parameters.

In the DragonI, the issue of cross-coupling arises in the context of controlling roll, pitch and yaw angles. The roll and pitch angles are closely coupled to yaw. As the DragonI is pitched or rolled, it start to yaw. The reason behind this behaviour is the the aerodynamic drag effecting the system. Aerodynamic drag "is a retarding force acting on a body moving through a gas parallel to the motion of the body"[1]. The aerodynamic drag increases with the square of the wind speed[2], and hence with square of the speed of the rotors in our case.

Why does this effect surfaces out during roll or pitch? If we recall from Section 2.2.2, to achieve roll or pitch the relative speed of front(left) and back(right) rotors are increased(decreased).

[1]NASA: http://topex-www.jpl.nasa.gov/glossary.html
[2]Aerodynamics of Wind Turbines: http://www.windpower.org/en/tour/wtrb/drag.htm

Since the drag is proportional to the square speed, even though same amount of speed is subtracted from one rotor and is added to the other the resultant drag will not be equal to zero and the system being in equilibrium previously is taken out of the equilibrium state and it starts to yaw as now the drag from roll(pitch) is no longer equal to drag from pitch(roll). Hence cross coupling is an important issue that needed to be considered, when determining the control architecture.

Responsiveness: Since the system is a real-time system and has to respond to real-time events the response time of the control system is a critical factor.

Complexity: Complexity of the system ultimately effects the responsiveness of the system. The more complicated the control system becomes, the less responsive it will become, as it will have to perform complex calculations.

6.2 Proportional Integral Differential (PID) Control

PID's are essentially algorithms which are primary used in control application.These algorithms try to control the output of a system by minimising the difference or error between the setpoint (desired value) & current point (observed value). One of the classic application example of PID control would be the oven thermostat. The thermostat's primary function would be to maintain a certain temperature within the oven by varying the amount of heat the oven's heating element could give out in order the maintain the set temperature within the oven.

There are three terms which make up PID control:

- P - Proportional Control
- I - Integral Control
- D - Derivative Control

These three terms are then multiplied by their respective constants and then summed up to give the final PID Control output.

$$PID = K_p \cdot error + K_i \cdot \int error\ dt + K_d \cdot \frac{d\ error}{dt} \tag{6.1}$$

Before going any further with PID control. One must understand the effects of each of these PID terms. Each of these terms adds to the overall effect of the PID output. Each term also have their own associated problems however when they are summed up together their behaviours will tend to compliment each others short comings.

6.2.1 Proportional Control

Proportional Control is essentially the difference between the desired value & measured value.

$$Proportional = setpoint - currentpoint \tag{6.2}$$

The difference is sometimes referred to as the *"error"*. Therefore the size of the P control term changes with the size of the error. The P term has the largest effect on the PID controllers behaviour due to the fact that its output is based purely on the error and nothing else. Going back to the oven thermostat analogy mentioned earlier. If the oven temperature is 270 degrees (measured temperature) and we have set the temperature to be 300 degrees (desired temperature). The P controller would tell us that the oven temperature must increase by 30 degrees. Therefore the value of 30 is returned to the PID controller from the P term.

The same concept would work with the DragonI. The onboard attitude estimation module measures tilt(pitch & roll angle) of the craft using inertial sensors(see Chapter 5 for more details). The data coming from the attitude estimation module will act as our observed value or currentpoint. The desired value or setpoint will be transmitted by the user via a joystick (for autonomous behaviour the system itself will decide what the setpoints are). If the DragonI is set to a hovering position the setpoints will be set to zero degrees. Any difference between the measure angle & the setpoint will be outputted by the Proportional controller.

Some problems with a proportional-only controller include:

- **Asymptotic behaviour:** This is where a term(in this case the P term) approaches the setpoint but never quite reaches there & if it does reach the set point, it will do so slowly. Basically as our currentpoint starts getting closer to the setpoint, the amount of error the P controller returns becomes negligible.
- **Positive feedback:** If the value of the proportional constants are large. We would start to observe the P controller trying to compensate for the error in an exaggerated manner. If this is the case, we would start to observe the DragonI swinging wildly trying to make up for the size in error. Because the K_p (Proportionality Constants) are high, the P controller will overshoot the setpoints and in attempt to re compensate for this overshoot will move in the opposite direction and again overshoot the setpoint causing the system to oscillate. This oscillation is what we call positive feedback or ringing effect.
- **Steady state effects:** If there are forces not controlled by the system, such as wind. The attitude of the DragonI will be affected. For example, if the DragonI is in a hover state &

the wind is blowing from the left to right causing the DragonI to tilt slightly to the right. The error will begin to grow. Because the error is growing, the right motors will start to speedup in order to compensate for the tilt. However, the motors will only speedup as far as the the amount of tilt introduced by the system by the cross wind effect.

6.2.2 Integral Control

The Integral Controller will give out the sum of errors over time. It is used to deal with Asymptotic behaviour as well as make up for steady state errors. Usually there will be errors which are too small to be picked up by the P controller. The Integral Controller will sum up these small error into something which will have a greater impact on the system output. The integral will also take care of any bias present in the system by accumulating these errors and correcting itself over time. On the DragonI, the integral term is determined by the following equations:

$$Integral = Integral + (Proportional \times (period\ between\ sampling)) \tag{6.3}$$

A problem associated with the integral controller is the fact that sometimes the integral would tend to accumulate to a large number when there is a force acting on the system which does not allow any change to take place . This problem is called “integral wind-up”. An example of this would be when we hold down the one end of the DragonI’s blades (lets say the front blades) when it is attempting to hover. The integral on the front blade would accumulate to a very large number. Upon releasing the front blade the DragonI would take time to degrade the large integral which had built up when the systems front blade was held down.

6.2.3 Differential Control

The primary function of the Differential controller is to dampen any changes taking place within the system. The Derivative term will take care of the positive feedback problem present within the proportional controller. Because the derivative controller looks at the rate of change in error. Its output would vary depending on how quick the change in error is. The Derivative controller would therefore prove to be an effective tool in dampening the output of the proportional controller. The Derivative controller when implemented in the microcontroller looks similar to the following equations.

$$D = \frac{current\ error - previous\ error}{period\ between\ sampling} \tag{6.4}$$

6.3 PID Tuning

Calculating the Proportional, Integral & derivative terms are easy as described above. The main challenge in PID Control is tuning PID constants (K_p, K_i, K_d). By tuning the constants we will ensure that the system behaves in a desired manner by having the error number as small as possible (difference between setpoint & currentpoint). There are several methods in which PID can be tuned. We will discuss 2 of these technique below.

- Ziegler Nichols - Open& Closed loop method
- Trial & Error

6.3.1 Ziegler Nichols Method

The Ziegler Nichols tuning method is one of the well know PID tuning methods which was developed in the early 40's. There are two approaches for setting the PID constants by using the Ziegler Nichols tuning method.

- **Open Loop Method:** Determines the PID parameters from the process parameters alone (Process gain, deadtime, time constant etc) by performing an impulse test on the system. (see figure 6.1)
- **Closed Loop Method:** Determines the gain at which a loop with proportional only control will oscillate, and then derive the gain of the integral & derivative constant terms from this gain as well as the period of the oscillation.

Ziegler Nichols - Open loop method

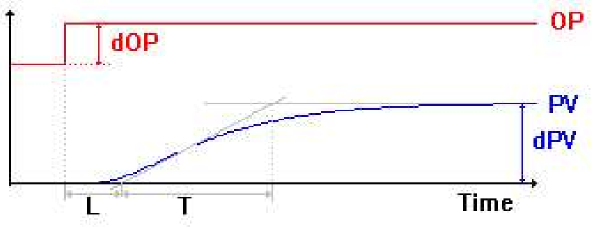
The following is a list of procedures to tune the PID constant with the open loop Ziegler Nichols method:

1. Setup & perform an open loop test (e.g. a step test) by changing the output by a small amount.
2. Determine the process parameters: Process gain, deadtime, time constant (see below: draw a tangent through the inflection point and measure L and T as shown below in diagram 6.1)

3. Calculate the parameters according to the Ziegler Nichols - Open loop method tuning chart (see Table 6.1) where $K = \frac{TimeConstant}{ProcessGain \times Deadtime}$. Please note that $K_i = \frac{K_p}{T_i}$, $K_d = K_p \cdot T_d$

	K_p	T_i	T_d
P Control	K		
PI Control	$0.9 \times K$	$3.3 \times Deadtime$	
PID Control	$1.2 \times K$	$2 \times Deadtime$	$0.5 \times Deadtime$

Table 6.1: Ziegler Nichols - Open loop method tuning chart



Process gain	=	dPV / dOP
Deadtime	=	L
Time Constant	=	T

Figure 6.1: Diagram showing Ziegler Nichols - Open loop method tuning experiment

Ziegler Nichols - Closed loop method

In order to tune the PID according Ziegler Nichols - Closed loop method , the following procedure must take place.

1. Turn the PID controller to P-only mode, i.e. turn both the Integral and Derivative modes off by setting their constants to zero.

2. Increase the gain of the proportional constant, K_p slowly and observe the output response. Note that this requires changing K_p in step increments and waiting for a steady state in the output, before another change (increase or decrease) in K_p is implemented.

3. Keep playing with the proportional constant K_p until a sustained periodic oscillation in the output (or close to it) is obtained, mark this critical value of K_p as K_u**, the ultimate gain**. Also, measure the period of oscillation, P_u **, which is known as the ultimate period** (see Figure 6.2 below).

4. By using the values of the ultimate gain , K_u , and the ultimate period, P_u , we can derive the other PID tuning constants by referring to the Ziegler Nichols - Closed loop method tuning chart (see Table 6.2). Please note that $K_i = \frac{K_p}{T_i}$, $K_d = K_p \cdot T_d$

	K_p	T_i	T_d
P Control	$\frac{K_u}{2}$		
PI Control	$\frac{K_u}{2.2}$	$\frac{P_u}{1.2}$	
PID Control	$\frac{K_u}{1.7}$	$\frac{P_u}{2}$	$\frac{P_u}{8}$

Table 6.2: Ziegler Nichols - Closed loop method tuning chart

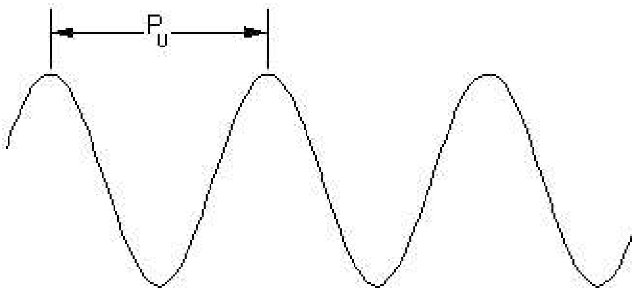


Figure 6.2: Diagram showing Ziegler Nichols - Closed loop method tuning experiment

6.3.2 Trial & Error Method

The trail an error or some may refer to as “tune-by-feel” method basically involves tuning the PID based on what is being observed in the system and tuning the PID constants based on any short comings of the observed system. The trial & error method basically starts of by

1. Turn the PID controller to P-only mode, i.e. turn both the Integral and Derivative modes off by setting their constants to Zero. (Just like the Ziegler Nichols - Closed loop method)

2. Set the proportional constant K_p to a low value

3. Increase the proportional constant K_p by small increments until a sustained oscillation with constant amplitude is observed (i.e K_u is obtained).
4. Set the proportional constant K_p to half the value of K_u & Increase the derivate constantK_d until the the any noticeable oscillation is dampened.
5. Increase the integral constant K_i to remove any asymptotic behaviour observed in the system

6.3.3 Other tuning techniques

Other Tips for speeding up the tuning process include the following:

- **Start with the proportional constants** (As stated in the Ziegler Nichols Method)- Since the proportional constant has the most significant effect on the PID term, it makes sense to tune this term first. By simply setting the constants of the integral & derivative term to zero, our PID controller now effectively becomes a Proportional-only controller after which we can tune the Proportional term until the desired behaviour is observed.
- **Only vary one constant at a time** - When tuning the PID controller we must realise that any change to the value of each of the three PID constants will effect the way the other two PID components behave. For example if the derivative constant was increased such that any changes made to the system will be dampened, therefore the system will take longer to reach the desired value.
- **Tuning the PID constants on the fly** (real-time) - Before the PID is properly tuned, it is very likely that there will be many changes to the value of these constants. Since the PID controller sits on the microcontroller which sits on the helicopter . It makes sense to tune the PID controller in real-time rather than having to wait for the program to be recompiled & uploaded onto the circuit board. Not only does this save us time, it would also allow us to see the effects of changing the PID constants instantly as changes are taking place in real-time.
- **Logging the PID outputs** - This is extremely useful technique especially for debugging purposes. By graphing the outputs of the PID controller we can take note of its behaviour and work out any problems which can effect the behaviour of the overall system.

6.3.4 Remarks

The tuning methods mentioned above all have their own merits. However, the method we have chosen to tune the DragonI was via the trail & error method mainly because the Ziegler Nichols method might cause us to run into difficulties such as:

- It may be difficult to determine the slope at the inflection point accurately, especially if the measurement is noisy.
- The methods tends to be sensitive to controller calibration errors.

We figured that it was best that we tune the DragonI using the trial and error method as well as using the tuning techniques listed above (See Section 6.3.3) to tune the DragonI's PID constants based on what we observe and changing the constants and noting down the response observed in the system. Although this may be time consuming, we figured that this is the only way tuning can be done. (see chapter **??** for experimental setup as well as the results of the PID experiments conducted)

6.4 Control Architecture

6.4.1 Design

The control architecture is comprised of four PID controllers, integrated in such a way that maneuverability is achieved on the DragonI by controlling the inputs to the four motors of the DragonI. Hence the output of the control system is four Pulse Width Modulated values (PWM) which are fed into the motors.

The PID controllers are implemented on each of the four axis of the DragonI, namely roll, pitch, yaw and lift. The decision to have a controller on each axis, rather than having controllers on each motor was based on the fact that the control on an axis involved a differential change to a set of motors. For example to pitch forward the speed of front motor has to be reduced at the same time the speed of the back motor has to be increased by the same amount in order to keep the thrust of the DragonI constant. Figure 6.3 is a graphical illustration of the control architecture.

The controller on the pitch axis takes as input the error in pitch (i.e the difference between the desired pitch and the current pitch). The error is processed by the PID algorithm producing a value which is added to the front motor PWM and subtracted from the back motor PWM. The controller on the roll axis functions in the same way acting on the roll parameters.

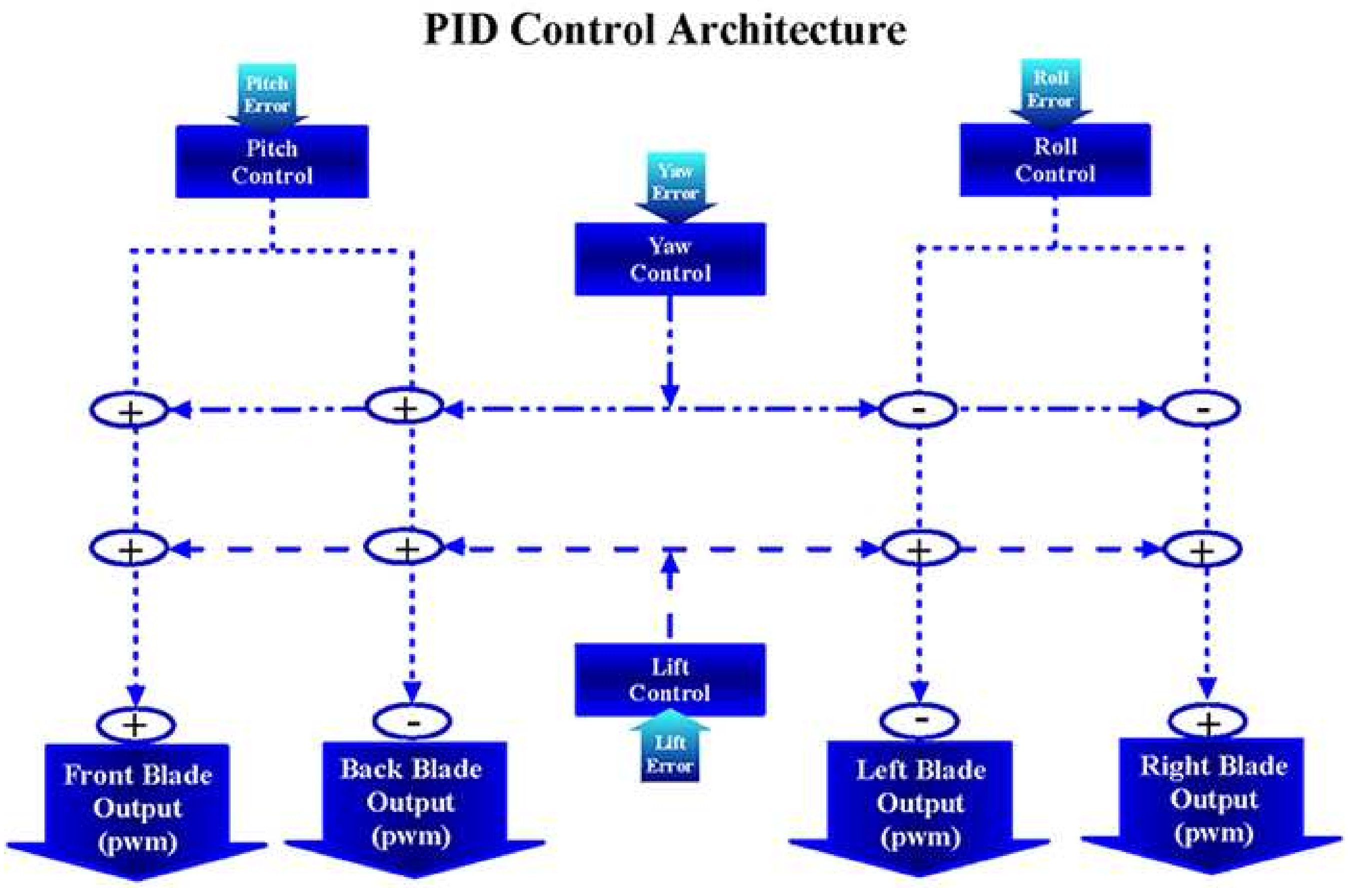


Figure 6.3: Control Architecture

The controller on the yaw axis looks for errors in the yaw, and outputs PWM values to keep the orientation of the DragonI same as that of the desired orientation. To achieve this the value of output from the PID controller is added to the front and back motors, while it is being subtracted from the left and right motors.

The lift controller increased the lift based on the error on the height of the DragonI. To change the collective of the system, all of the motors have to be simultaneously increased/decreased therefore holding the roll pitch and yaw constant.

Therefore the resultant control for each motor consists of a combination of the four controller outputs giving the following equations:

$$PWM_{front\ motor} = PID_{lift} + PID_{pitch} + PID_{yaw}$$

$$PWM_{back\ \ motor} = PID_{lift} - PID_{pitch} + PID_{yaw}$$

$$PWM_{left\ \ motor} = PID_{lift} + PID_{roll}\ \ - PID_{yaw}$$

$$PWM_{right\ motor} = PID_{lift} - PID_{roll} \quad + PID_{yaw}$$

6.4.2 Implementation of PID Controllers

Once the control architecture was finalised, the implementation of the architecture on the AVR present itself. The main reason behind choosing PID controllers was their ability to react rapidly to changes in the system. In the pursuit of this aim three different ways of PID implementation were explored.

First Implementation

The first implementation was based on the literature of PID, the only inputs to the PID algorithm were estimates of the attitude of the system (i.e roll, pitch, yaw and height) for the corresponding controller, with following equation for the PID

$$PID_x = K_{p\ x} \cdot error + K_{i\ x} \cdot \int error\ dt + K_{d\ x} \cdot \frac{d\ error}{dt} \tag{6.5}$$

where $x \in \{roll, pitch, yaw, height\}$

To get the integral and the differential components of the controller, numerical integration and differentiation methods were used. After running a few tests we realised, in order to have a quick response, the proportional component has to be fairly significant which caused a ringing effect. In theory the differential component of PID stops this ringing effect (see Section 6.2.3). It was observed that the differential component only helped with small ringing effects.

A careful look at the numerical differentiation algorithm employed in the system showed that even though nothing was wrong with the method, it was not able to capture the rate of change fast enough. The reason is that the the values coming from the analog to digital converters are discrete, not continuous and hence small changes was not reflected in our calculations

Second Implementation

Spending time in improving the differentiation algorithm was soon considered to be a dead end. It was realised that the solution the problem had been sitting in front of us all along. The rate gyroscope on the three axis (roll, pitch and yaw) already provided us with the rate of angular change. The PID controller was modified to use the signal from the gyroscope as the differential input, giving the following equation:

$$PID_x = K_{p\ x} \cdot error + K_{i\ x} \cdot \int error\ dt + K_{d\ x} \cdot gyroscope_x \tag{6.6}$$

where $x \in \{roll, pitch, yaw\}$

The new PID implementation provided a better control of the system. But still it was far from being complete. As the lift was increased (i.e. the throttle) the PID would loose effectiveness. Essentially this is because for a particular value of error, the PID needs to output a small number when the throttle is low, while for the same error at higher throttle the same PID output will be insignificant and hence ineffective.

Third Implementation

The PID controllers were modified to counteract the problem in the second implementation of the PID. Obviously the solution involved making the output of the PID to be proportional to the throttle. Our approach to the problem was to find the maximum allowable error and the error for the PID

$$PID_x = K_{p\ x} \cdot \frac{error \cdot throttle}{error_{MAX}} + K_{i\ x} \cdot \int \frac{error \cdot throttle}{error_{MAX}}\ dt + K_{d\ x} \cdot \frac{gyroscope_x}{error_{MAX}} \cdot throttle \tag{6.7}$$

where $x \in \{roll, pitch, yaw\}$

The resultant controller was far more superior than the first two implementations. As we increased the throttle we noticed that the PID effectiveness remains constant unlike the second implementation.

6.5 Failsafe Control

One of the important features of any autonomous system is the failsafe control. There are three failsafe controls within our system. At the highest level sits the abort mission. On the receipt of this signal, which is achieved by pressing a designated abort button on the joystick. The base station starts transmitting the abort command to the RPC board which in turn sends a zero throttle value to the DragonI.

On the second level is the user control, the DragonI can be brought under human control pressing a designated button of the the joystick. This takes over the DragonI's controller, by sending setpoints to the PID controller of the DragonI.

Embedded on the DragonI is transmission failure control. On the event of loss of transmission the DragonI enters the emergency control. Essentially it intercepts all normal operations of the DragonI, sets the system into hover mode and gradually drops the throttle and lands.

Chapter 7

System Evaluation

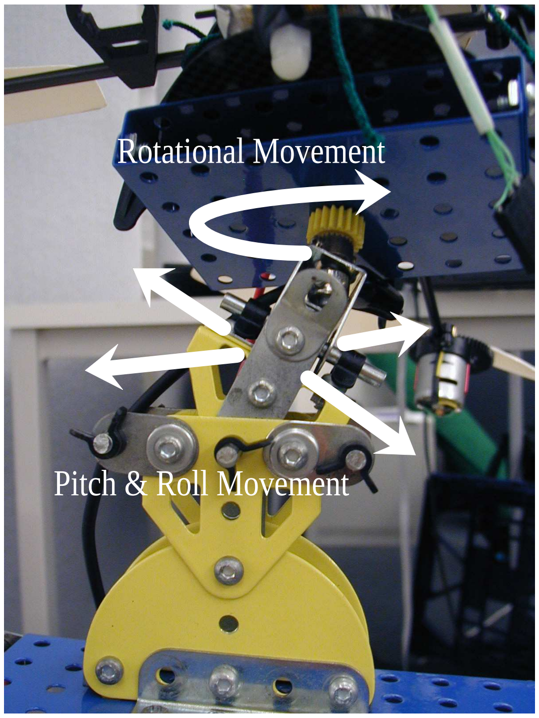
So far we have been talking about different parts of the system and the experiments performed to evaluate their functionality. In this process some choices were made to make the system work such as our choice of attitude estimation algorithm, control algorithm and so on, but is the system working? This is the question we are trying to address in this chapter.

In order to evaluate the effectiveness of the system implemented, three different platforms were used. Each of these platforms put a different degree of restriction on the system and hence our attempt was to start from a fairly restricted setup where only few things can go wrong, then to a moderately restricted setup and finally in an ideal unrestricted setup, when we say ideal it means that the setup environment is stable (i.e. there are no sudden wind gusts). The sections below explain each one of these setups, namely the universal joint, the pulley system and the free flight and presents their results. Finally a conclusion is drawn from these setups on how well the system performs.

7.1 Universal Joint

7.1.1 Experimental Setup

The universal joint is made from a $Meccano^{TM}$ set. As shown in Figure 7.1 the universal joint was modeled on an inverted pendulum. It has 3 degrees of movement (yaw , pitch & roll) with its angle of movement on the pitch and roll axis limited to 60 degrees. Although the joint gives the DragonI the freedom to pitch, roll and yaw, the ability to lift has been restricted. An elastic stretch cord was also erected over the universal joint which is suspended by the ceiling. The purpose of this stretch cord is so that the DragonI is never resting at an extreme angle when DragonI's collective is set to zero (i.e resting at a pitch angle or roll angle).



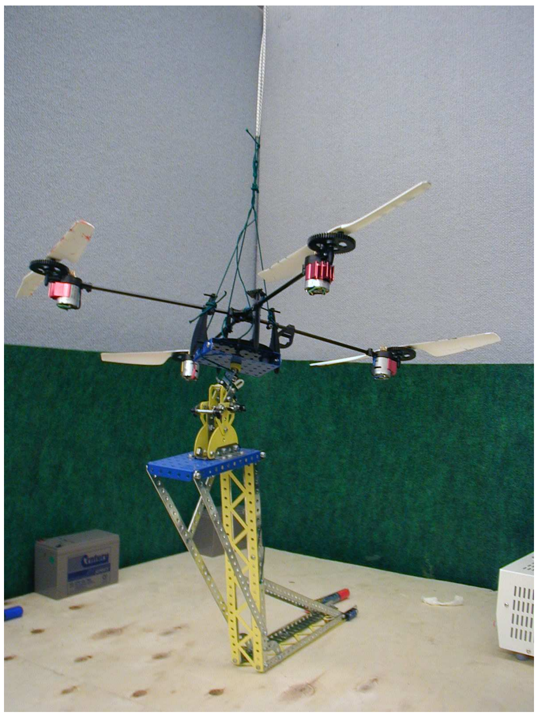


Figure 7.1: Universal joint setup: Showing it's maneuverability

The universal joint is bolted onto a wooden platform approx 15 inches off the ground. The reason for this is because when motor speed is set to zero, the frame of the helicopter tends to rest at an angle. Hence if it is not at a specific height above the ground. The blades would hit the floor. Furthermore at that height, the turbulence created by the ground effect is minimised.

In order to determine the optimal PID constants. There has to be a series of trial & error experiments. The problem with the trail & error approach is that the DragonI must be subjected to a number of test runs, most of which would be unsuccessful.This would also mean a lot of test flight crashes which is not feasible financially & from safety point of view would be considered dangerous. The universal joint was created specifically for this purpose. It acts almost like a cradle, holding the DragonI in place.

Most of the tests on the system were run on this platform. Once it was deduced that the right parameters were found then we moved to the next testing platform.

7.1.2 Experimental Procedure

To test the system, the DragonI was positioned on the universal joint . The speed of the DragonI was then increased until there was enough thrust to allow the DragonI to hover. Once the system stabilised itself then the desired pitch and roll were changed using the joystick under manual control. One of the buttons on the joystick provides instantaneous change of control from the human to PID, setting the desired roll and pitch values to zero and hence to hover mode. This procedure was used to get the impulse response of the system.

7.1.3 Results

Figures 7.2 &7.3 show the experimental results. When analysing the results it is worth to note that the desired values are plotted from the data within the base station, while the PID output is sent wirelessly, as a result of which the latency in the response of the PID controller may be exaggerated.

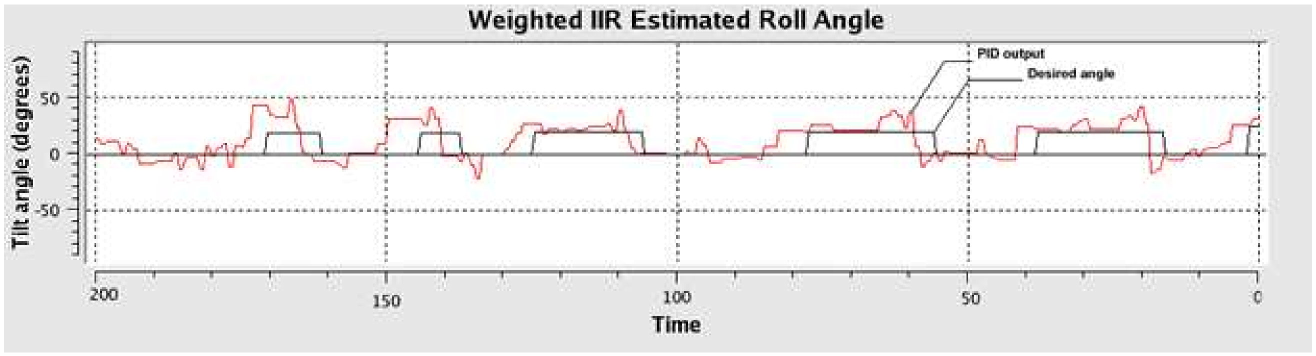


Figure 7.2: PID response for roll

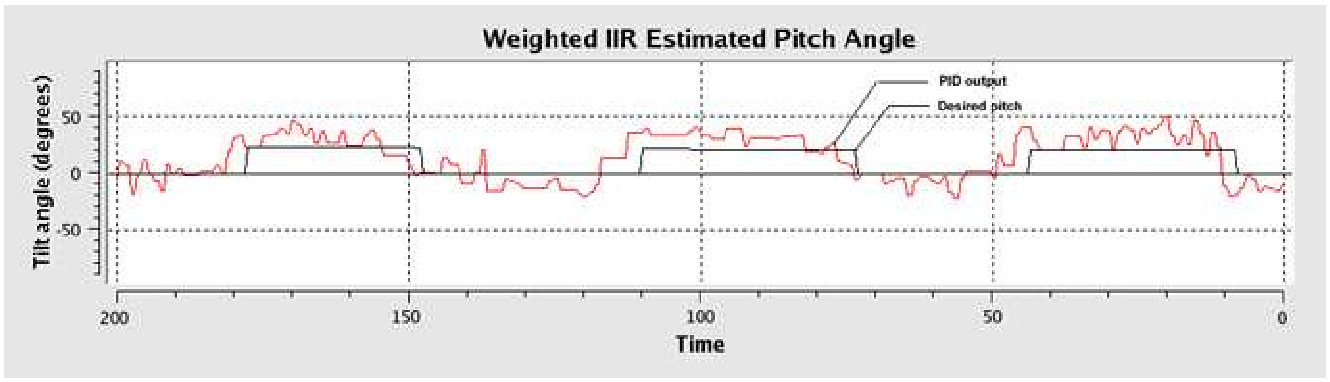


Figure 7.3: PID response for pitch

From the figures it can be deduced that the PID controllers are able to achieve reasonable response to the changes in the system. There is evidence of overshoot in both of the PIDs. The roll PID seems to be more fine tuned than the pitch PID. The pitch control needs more investigation for its behaviour. We were in the process of investigating the pitch PID that one of the wireless modules failed. This rendered our analytical system useless and as a result we were not able to conduct any further thorough investigation and tuning of the pitch PID. The wireless modules are being tested as we are writing this report. Any further results obtained when the problem is rectified will be appended to the report at a later stage.

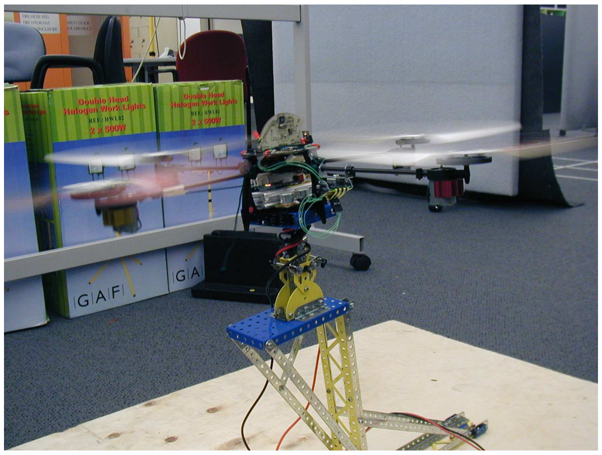


Figure 7.4: DragonI self stabilised using PID controllers

Figure 7.4 is showing the DragonI self stabilised using the PID controllers. Videos of this test are included in the CD accompanying this report. The relevant videos for this section are:

DragonI No PID: In this video the behaviour of the system is shown where there is no PID control.

DragonI PID 01: In this video the DragonI is started from an extreme angle, and then the throttle speed is increased. As it can be seen in the video the system is able to attain a hovering state.

7.2 Pulley System

7.2.1 Experimental Setup

The universal joint gave the DragonI three degrees of freedom, to get a better appreciation of the system, all six degrees of freedom were desired. Figure 7.5 shows the setup that allowed us to gain all six degrees of freedom, where the DragonI is suspended in air using an elastic chord. Lateral movement (sideways) is achieved with the use of a pulley running on a string.

The advantages of this setup is that it puts almost no restraint on the movement of the DragonI given it stays with in a certain radius.

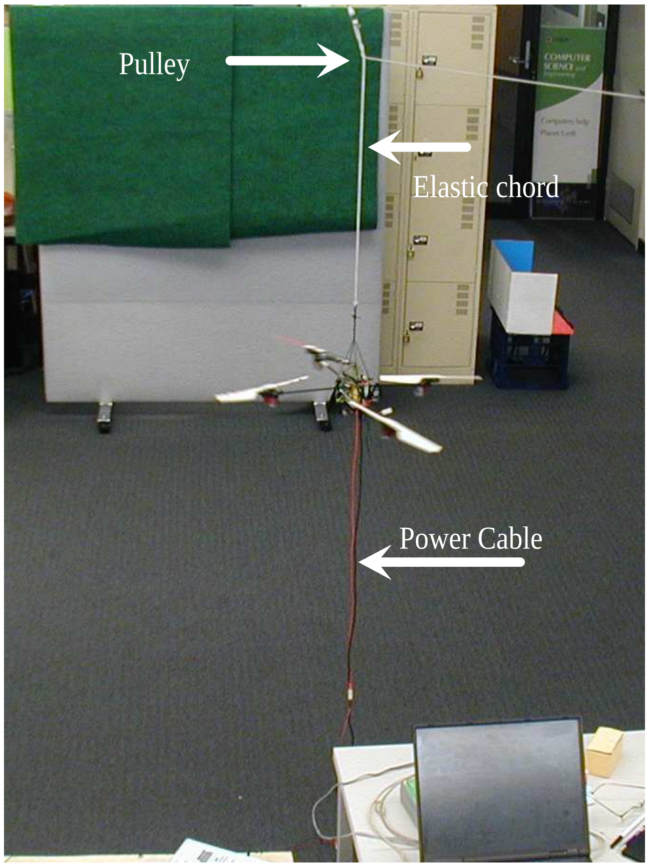


Figure 7.5: The pulley system

7.2.2 Experimental procedure

The DragonI was suspended on the elastic chord, the speed of the DragonI was increased until a slight slack in the elastic chord was noticed, which signified that the DragonI was under the control of its control system and not being supported by the elastic chord. During the PID tuning stages the values of the PID coefficients were dynamically changed until the system was stabilised.

7.2.3 Results

There were no graphs recorded for this test prior to the failure of the wireless modules. The only results that we have in hand are the last know PID coefficients that stabilised the system. These values were plugged into the DragonI's control and with the help of onboard timers the throttle speed was increased gradually to a predefined value. Since we do not have any means of getting data from the DragonI at this stage video clips were taken. The relevant video clips for this experiment are:

DragonI PID 04: In this video, the DragonI is suspended with an elastic chord, the purpose of the elastic chord is to stop the DragonI from drifting and hitting objects on the sides. Once it drifted too far it was yanked back to the center. The fact that there is slack in the chord shows that the DragonI is hovering.

DragonI PID 05: In this video the speed of the DragonI was set about 60% so it gets more lift. As it can be seen in the video there is more slack in the chord and hence a better evidence of the fact that the hovering is maintained without any help form the chord. Once again the chord was used to not let the DragonI drift.

7.3 Free Flight

We were not able to run free flight tests on the systems, partly because we do not want to break the DragonI before it any official demonstration is made.

7.4 Discussion

From the results we have been able to obtain, it can be seen that the PID controllers are able to stabilise the system and sustain hovering. So is the system working? Yes we can certainly see a system that has grown form just some pieces of hardware setting on the shelves, to something that can achieve a challenging task of stablising a helicopter. Every small decision we made during the course of the project contributed to its eventual success.

Chapter 8

Conclusion

8.1 Applications

While the main objectives of our thesis is to develop a platform that provides access to the internal states of the Draganflyer and can be used as a research testbed for Artificial Intelligece experiments, it has a direct application for a myriad of other purposes.

There are countless number of alternative applications, of which some are outlined below:

- **Search and Rescue:** The DragonI can be used for search and rescue operations. The camera onboard can be used to send pitures from disaster zones and hence it can either be analysed by a human or an automated system to locate victims.

- **Inspection:** Since DragonI is small in size and cost of building it is relatively lower than helicopters, it can be used to inspect areas where it is not possible for a normal helicopter to reach because of its big size. Moreover there might be other potential life threatening risks posed to the pilot. Such inspection may include inspection of dams, caves, possible biological disaster zones.

- **Law Enforcement:** Once again the small size and low cost makes UAVs such a the DragonI a very appealing platform to be used for law enforcement activities. Not only it costs less but also it something goes wrong and it crashes, it will have minimal damage to the immediate environment.

- **Recreational:** DragonI can be used for receational purposed. Since the states of the Flight Control Unit is accessible by the user, curious users may not only use it for normal flying activities but also for perfoming somthing out of ordinary.

8.2 Future Work

From current results, the DragonI has demonstrated that the platform has quite a number of future potentials. Work remains to be done on implementing a more accurate form of attitude estimation such as Kalman Filtering as well as comming up with techniques to work out the lateral drift during flight such that hovering over a certain spot could be achieved much more easily. Also the DragonI could be used in the field of AI to demonstate topics such as reenforcement learning as well as behaioural cloning which could potentially one day replace the current PID based control architecture. Other potential porjects for this platfom include the use of the onboard cameras to track ground targets, speech based control as well as redesigning the circuit board to include the use of more sensors to expand the capabilities of the DragonI

8.3 Concluding Remarks

At the beginning of this thesis the idea of an Unmanned Aerial Vehicle was proposed. The orginal system that was presented to us had its share of problems that had to be addressed, such that it could be used as a suitible platform for research purposes.

Completed in this project was the comprehensive characterisation and utilisation of the interial sensors to be employed, implementation of differnt filtering algorithms that would provide reliable estimation of the systems states, formulation of a robust communication protocol,design and implementation of the systems hardware and software components in accordance with the requirements as well as coming up with a suitable control architecture to achieve stable flying platform.

Bibliography

[1] Mark Diel Varun Ganapathi Jamie Schulte Ben Tse Eric Berger Andrew Y. Ng, Adam Coates and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

[2] M. Tischler B. Mettler and T. Kanade. System identification of small-size unmanned helicopter dynamics. In *American Helicopter Society, 55th Forum*, 1999.

[3] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *K. Furukawa, D.Michie and S. Muggleton (Eds.), Machine intelligence 15*, pages 103–129. Oxford University Press, 1999.

[4] M.D. Bruce J. Weimer. The bxflyer. In *Seattle Robotics Society Newsletter - Encoder.* http://www.seattlerobotics.org/encoder/200311/weimer/bxflyer.html, June , 2003.

[5] Pennsylvania State University's Rotorcraft Centre. Semi-autonomous control of a draganflyer. *http://www.psu.edu/dept/rcoe/Research Projects/Task 8/Presentation 8.ppt*, 2001 - Present.

[6] Gordon Wyeth Gregg Buskey, Jonathan Roberts. A helicopter named dolly - bhavioural cloning for autonomous helicopter control. In *Proceedings of the 2003 Australasian Conference on Robotics & Automation*, 2003.

[7] U. W. Brandenburg M. Musial and G. Hommel. Development of a flight control algorithm for the autonomously flying robot marvin.

[8] Peter S. Maybeck. Stochastic models, estimation, and control, volume 1. *Academic Press, Inc*, 1979.

[9] Massachusetts Institute of Technology (MIT). The phaeton project. *http://clohessy33.mit.edu/16.821/docs/ Presentation_Draft_2.ppt.pdf*, May 4, 2004.

[10] A. Dzu P. Castillo and R. Lozano. Real-time stabilization and tracking of a four rotor mini-rotorcraft. In *Proceedings of European Control Conference, Cambridge UK*, pages 1–4, 2003.

[11] Greg Welch and Gary Bishop. An introduction to the kalman filter. *University of North Carolina, Department of Computer Science*, 197.

Appendices are included in the Accompanying CD

Appendix A

RPC Data Sheet

Appendix B

SRF08 Ultra sonic range finder Data Sheet

Appendix C

LMC660 Operational Amplifier Data Sheet

Appendix D

Voltage Regualor

Appendix E

FTDI setial to USB Data Sheet

Appendix F

IRLZ44NS MOSFET Data Sheet

Appendix G

IR4426 Gate Driver Data Sheet

Appendix H

HMC6352 Electronic Compass Data Sheet

Appendix I

ATmega128 MIcrocontroller Data Sheet

Appendix J

ATmega32 Data Sheet

Appendix K

CG-L43 Gyroscope Data Sheet

Appendix L

ADXL311 Accelerometer Data Sheet

Appendix M

Software

Appendix N

Protel Design Schematics

Appendix O

Experiment Videos

Appendix P

AVR Studio 4