University of Cincinnati

Date: 10/28/2014

I, Vijaykumar Sureshkumar FNU , hereby submit this original work as part of the requirements for the degree of Master of Science in Aerospace Engineering.

It is entitled:

Autonomous Control of A Quadrotor UAV Using Fuzzy Logic

Student's name: **Vijaykumar Sureshkumar FNU**

This work and its defense approved by:

Committee chair: Kelly Cohen, Ph.D.

Committee member: Elad Kivelevitch, Ph.D.

Committee member: Bruce Walker, Sc.D.

13928

Autonomous Control of a Quadrotor UAV Using Fuzzy Logic

A thesis submitted to the
Graduate School
Of the University of Cincinnati
In partial fulfillment of the
Requirements for the degree of

Master of Science

In the Department of Aerospace Engineering
Of the College of Engineering and Applied Science

By
Vijaykumar Sureshkumar FNU

B.E. Mechanical Engineering University of Mumbai

October 13, 2014

Committee Chair: Kelly Cohen, Ph.D.

ABSTRACT

UAVs are being increasingly used today than ever before in both military and civil applications. They are heavily preferred in "dull, dirty or dangerous" mission scenarios. Increasingly, UAVs of all kinds are being used in policing, fire-fighting, inspection of structures, pipelines etc. Recently, the FAA gave its permission for UAVs to be used on film sets for motion capture and high definition video recording. The rapid development in MEMS and actuator technology has made possible a plethora of UAVs that are suited for commercial applications in an increasingly cost effective manner. An emerging popular rotary wing UAV platform is the Quadrotor A Quadrotor is a helicopter with four rotors, that make it more stable; but more complex to model and control. Characteristics that provide a clear advantage over other fixed wing UAVs are VTOL and hovering capabilities as well as a greater maneuverability. It is also simple in construction and design compared to a scaled single rotorcraft. Flying such UAVs using a traditional radio Transmitter-Receiver setup can be a daunting task especially in high stress situations. In order to make such platforms widely applicable, a certain level of autonomy is imperative to the future of such UAVs.

This thesis paper presents a methodology for the autonomous control of a Quadrotor UAV using Fuzzy Logic. Fuzzy logic control has been chosen over conventional control methods as it can deal effectively with highly nonlinear systems, allows for imprecise data and is extremely modular. Modularity and adaptability are the key cornerstones of FLC. The objective of this thesis is to present the steps of designing, building and simulating an intelligent flight control module for a Quadrotor UAV.

In the course of this research effort, a Quadrotor UAV is indigenously developed utilizing the resources of an online open source project called Aeroquad. System design is comprehensively dealt with. A math model for the Quadrotor is developed and a simulation environment is built in the MATLAB/Simulink framework. The Fuzzy flight controller development is discussed intensively. Validation of the math model developed is presented using actual flight data. Excellent attitude tracking is demonstrated for near hover flight regimes. The responses are analyzed and future work involving implementation is discussed.

.

This Page Intentionally Left Blank

ACKNOWLEDGEMENTS

Author would like to thank Dr. Kelly Cohen from the University of Cincinnati for all his guidance, support, and assistance.

Author would like to thank Michael Piasecki, Joseph Pawelczyk and Jeffrey Field at Dragonfly Pictures, Inc for their support and expertise and use of facilities to bring significant milestones to this research.

This work is dedicated to my Mom, Dad and Brother, without whose unconditional love and support, would be a mere work of fiction. Special mention to my friends and loved ones alike for making life here in the United States feel like home.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Unmanned Air Vehicles are being increasingly used for missions that are undesirable for humans [1]. Flying a rotary wing UAV using traditional radio receiver and transmitter (RX/TX) can be a daunting task especially in high stress situations. A certain level of autonomy is imperative to the future of such UAVs [2]. A Quadrotor is a helicopter with four fixed pitch propellers arranged symmetrically about a central housing. It is an under actuated system with six degrees of freedom (DOF) and four control inputs. It is simple in design and construction as compared to a scaled single rotorcraft, but presents an interesting control challenge [3]. Fuzzy logic control has been chosen to address this control problem, as it can effectively deal with highly nonlinear systems and provides inherent robustness. Modularity and robustness are key advantages to the fuzzy logic approach. Autonomous control refers to operation where the UAV is sent high level position commands such as GPS coordinates and executes the operation without the intervention of the RC Pilot at any stage. **The subject of this thesis is the autonomous control of a Quadrotor UAV using Fuzzy Logic**.

In this chapter, the motivation is developed and several possible applications described. Furthermore, the advantages of VTOL UAVs over fixed wing UAVs are stated. The research objective is discussed along with a short description of the organization of the thesis.

1.1 Motivation

Increasingly over the years, autonomous robots and vehicles are being used to perform missions that are considered “dull, dirty, and dangerous” in both military and civil operations, such as operations in nuclear power plants, for the exploration of Mars, to investigate behind enemy lines in battle, wild-fire surveillance, border patrols, and weather forecasting [1,4]. From

military applications like ISR, drone strikes etc. to civilian ones such as sports photography, film making, anti-poaching, fire-fighting etc., the sphere of influence for UAVs has grown dramatically. Unlike a decade ago, UAVs are now being developed for as little as $ 400, depending on the intended usage, making it an extremely cost effective solution to a myriad of commercial applications.

As the trend develops toward the increasing use of UAVs; autonomy becomes of prime importance. There is a growing need to substantially decrease the work-load of the ground based operators and provide wide spread access to cost effective surveillance. Flying a rotary wing UAV using traditional radio receiver and transmitter (RX/TX) can be a daunting task especially in high stress situations. It has been identified that a main concern for UAV growth is autonomous and intelligent control [5]. The intelligent flight control module based on fuzzy logic proposed here would better emulate human pilot behavior; helping UAVs to act and react more like their manned counterparts.

An important civilian application concerns wildfires, which cause destruction of thousands of acres, millions of dollars in damage, and cost many people their lives [6]. Situational awareness is the perception of environmental elements (wildfires, here) with respect to time/space and future projection is a key concept in combating wildfires and limiting damage, while also making the most effective use of resource allocation. UAV's are being used more often in these situations, and it is expected that their numbers will continue to grow [2]. Presently, wildfires are suppressed with the help of ground resources such as fire engines and fire crews and aerial resources such as manned fixed wing and rotary wing aircraft. Fighting large active fires with multiple "hot spots" is still an extremely challenging problem. It requires real time management of resources so as to keep the system at the highest level of synergy and

optimality. Recent developments have shown the effectiveness of using UAVs in this scenario. NASA's unmanned Ikhana UAV has onboard infrared sensors that can penetrate thick smoke and haze and it is capable of communicating information about size, intensity, and movement of fires over a long period of time. This acquired data is overlaid on Google Earth maps and then transmitted in real-time to firefighting commanders [7]. This allows for optimal resource allocation from a dynamic standpoint. The SIERRA project launched by UC's MOST AERO Labs is actively pursuing the above goal by making use of a fixed wing autonomous UAV to provide real time imaging data that is processed by an imaging algorithm; rendering an accurate fire growth pattern and spread prediction based on the current topography and environmental factors[8].

Other commercial applications include wildlife monitoring, cattle herding, aerial photography, film-making, cell tower inspection, archaeology and surveyors, precision agriculture, first responders, aerial delivery systems, pipelines and off shore oil rigs inspection. Cyberhawk, a UK based aerial surveillance firm, is using RC piloted fixed wing and rotary wing UAVs to provide inspection services to petroleum and utilities companies such as Shell, ExxonMobil, Marathon Oil etc. [9]. In 2012, Cyberhawk received a UK business efficiency award for an estimated $ 7 million savings in inspecting a drilling derrick. French firm, EDF Energy is using an autonomous "photogrammetric UAS" to create 3-D terrain maps [10]. More recently, Amazon showed off its drone delivery concept system called "PrimeAir" that utilizes a Multirotor UAV to make quick aerial deliveries, thus changing our notion of how products are shipped completely [11]. It is clear from the industry examples above; UAVs have already entered the civilian market and will continue to do so in large numbers globally.

1.2 VTOL vs. Fixed-Wing

VTOL aircraft have several key advantages over fixed wing aircraft, though it comes at a significantly higher consumption of power. Small fixed-wing UAVs such as the Dragon Eye, Hornet and Wasp has exhibited excellent flight abilities as well as high levels of autonomy [12, 13]. These UAVs while light weight, still have to fly fairly fast to provide sufficient lift [14].This necessitates more space to turn and limits their ability to operate in confined areas such as urban environments and indoor spaces [15]. The ability to hover in place is of prime importance in surveillance operations, photography etc. The ability to take off and land vertically in limited amount of space is also a useful advantage over fixed wing aircraft [12, 13]. These features of VTOL aircraft allow them to conduct multiple profile missions in almost every kind of environment, indoor and outdoor.

VTOL aircraft do however require high thrust to weight ratios and the propulsion systems are normally quite complicated [16]. This is also true for single or dual rotorcraft, additionally also requiring highly complicated control surface mechanisms such as collective pitch control of the blades, longitudinal and lateral cyclic as well as collective pitch control of tail rotor/aft rotor. This mechanical complexity results in high manufacturing and maintenance costs as well as advanced control logic. Some of the above are eliminated in the so called multi-rotor class of UAVs; they consist of an even number (normally) of fixed pitch rotors arranged around a central cabin and rely on differential thrust for control. Thus they are simple in construction and significantly cheaper to build but are more difficult to control due to their inherent instability. Attitude and positional control is achieved by varying the speed of the individual rotors, equal speed increase or decrease of all rotors means translation in the Z axis. Figure 1 below depicts a few popular fixed wing UAVs that are in use by the military.

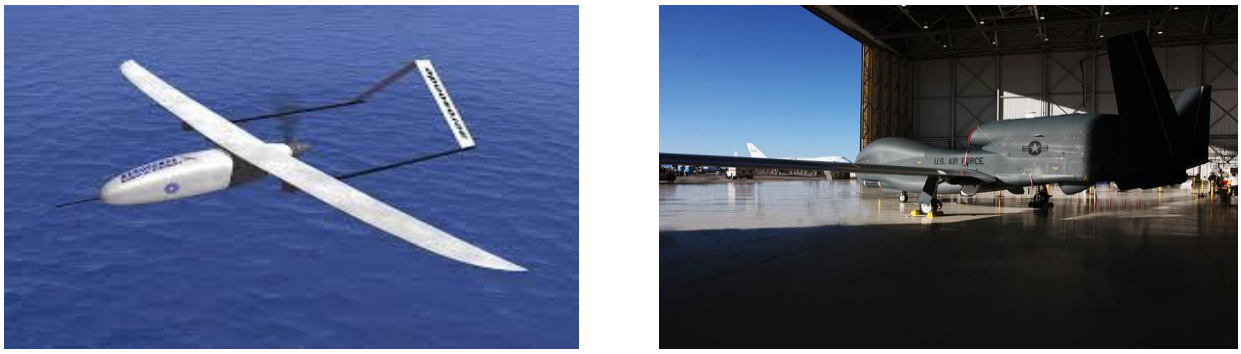

Fig. 1 Examples of Fixed Wing UAVs [17, 18]

Fig. 2 Examples of Single Rotary Wing UAVs [19, 20]

Figure 2 shows the Hornet UAV, which is a micro single rotor UAV, developed by Prox Dynamics AS of Norway, and in use by the British Army [21]. Also Pictured is the Camcopter S-100 UAV, developed by Schiebel, based in Vienna, Austria [22]. It boasts full autonomy capabilities, a payload capacity of 110 lbs., a top speed of 130 Knots and an endurance of 6 hours.

Figure 3 below shows a typical multirotor, the Draganflyer X8 which is a popular photography as well as research platform having 8 rotors in total [14]. The DP-12 Rhino, developed by Dragonfly Pictures, Inc. based in a Philadelphia, USA is a 400 lb. fully

autonomous tandem rotor UAV that features a payload capacity of 150 lbs., a top speed of 112 knots and a range of 300 nautical miles.

Fig. 3 Examples of Multi rotor and tandem rotor UAVs [23, DP-12 Rhino (Courtesy DPI)]

The Quadrotor is of the multi-rotor class of UAVs and has four fixed pitch propellers, two spinning clockwise and two spinning counter clockwise to negate net yaw moment. The high power consumption characteristic of VTOL aircraft especially UAVs, prevented the proliferation especially of the multi-rotor UAV. However with the recent advances in battery technology, significantly improving the energy density as well as more efficient actuators and low cost MEMS, this class of UAVs has seen rapid growth. The academic community too has embraced the multi-rotor UAV to a large extent and there exists an astounding amount of research in this field that has benefitted the author greatly in understanding this fascinating UAV. Figure 4 shows the Arducopter which is based on the open source project of the same name and the Arduino platform [24, 25].

Fig. 4 Example of a Typical Quadrotor : Arducopter [26]

1.3 Thesis Objectives

The objective of this thesis is to propose a methodology for the autonomous control of a Quadrotor UAV using Fuzzy Logic.

With the end goal of the thesis stated, there are several pre milestones that were achieved for the successful completion of the research goal. These milestones are listed below:

1. Design, build and fly an indigenous low cost Quadrotor UAV.
2. Test and modify the wireless communications module.
3. System modeling (Thrust and Torque response).
4. Develop effective Mathematical Model for the Quadrotor.
5. Develop Quadrotor Simulator in MATLAB/Simulink based on the above.
6. Develop and tune Fuzzy Flight Controller.
7. Generate an equivalent PID controller for the Quadrotor and compare the controllers.
8. Discuss implementation issues and future work.

In the course of this research effort, the Quadrotor developed went through multiple design and hardware changes as the learning process evolved. This resulted in significantly better performance both in simulations as well as actual PID controlled flight. It is worth mentioning here that the Quadrotor was designed, tested and flown utilizing the excellent resources of the online open source project called Aeroquad, that provides a stock PID based flight controller that can be tuned with minimal effort [27]. It also provided libraries for communications with the various sensors on board and has consequently sped up the development to a large extent. Each of the above sub objectives is discussed in subsequent chapters in detail.

1.4 Background

The rapid development in sensor, actuator and battery technology has overseen the advent of the multi-rotor UAV, most notably the Quadrotor. This popular rotary wing UAV platform is gaining sharp focus in realizing micro air vehicle concepts. The complex phenomena demonstrated by the Quadrotor, has been embraced by researchers and generated several areas of interest including autonomy, waypoint navigation, obstacle avoidance, robust trajectory control etc. The Quadrotor has four rotors placed at the front, left, right and back end of a cross frame, symmetrical about the center of mass of the vehicle. The speed of each individual rotor is varied to produce differential thrust and the control is achieved in this fashion. Two rotors rotate clockwise while the other pair rotates counter clockwise thereby maintaining net zero moment. Roll and pitch control is achieved by varying the speeds of two rotors at a time at opposite ends of the frame, i.e. front and back rotor speed control produces pitching and left and right rotor speed control produces rolling motions respectively. The Quadrotor has significantly lower kinetic energy in each rotor, thereby limiting damage in case of a collision. Also the vehicle's

dynamics are good for agility and increased payload [16]. However, the dynamics are also quite unstable and the Quadrotor presents an interesting control challenge.

CHAPTER 2: LITERATURE REVIEW

In this chapter, a review of literature related to the research objective is presented. A review of the commercially available flight controllers is given and which platform was selected for this research and the reasons for the same. A bibliographic survey of the astounding amount of research done in this field is presented thus cohesively rounding out the literature survey. The research that exists for the modelling and control of Quadrotors does not provide a suitable basis for comparison. This is true because several of the methods and results hold only for the specific UAV researched and implemented. The common control methodology applied to Quadrotors overwhelmingly is PID control and is reviewed extensively. The chapter ends with a section devoted to Fuzzy Logic control.

The online open source project, Aeroquad, was extensively used in the beginning to get a head start on the development of the indigenous Quadrotor, the sensors, Arduino microcontroller and an electronic shield were purchased as to assemble kit that made up the flight controller board i.e. the IMU and the microcontroller which runs the flight software. All the initial flying, testing and code development was done using this flight controller that could be easily modified. It was however partially damaged in a crash and subject to random voltage drops in subsequent testing and was henceforth abandoned. In the final stages of this research, access to several different flight controllers was made available, courtesy of an internship at Dragonfly Pictures, Inc. (DPI) which is an UAS developer. 3D Robotics' APM, DJI's NAZA and Wookong controllers and Micropilot's MP 2128^{HELI} were all being actively used on a variety of UAV configurations. The author choose to move to using the APM, Arducopter flight controller because DPI has substantial experience with it, though it was used on an Octarotor UAV and also because since it was based on the same Arduino platform, the developed code could be used

as is for future implementation. APM is based on a nested PID loop structure and when tuned for it, is capable of autonomous flight. A very interesting basis for comparison now became possible and this is further discussed in Chapter 5.

2.1 Commercially Available Flight Controllers

The following subsections are devoted to the fight controllers that are available, their capabilities, a review and the reasons for choosing the particular flight controllers for this study. The author has hands on experience with the below mentioned flight controllers and thus has practical knowledge of the gaps in technology that the developed FLC module would fill.

2.1.1 Aeroquad Project

Aeroquad is an online open source hardware and software project dedicated to the construction of RC Quadrotors [27]. The flight software being open source is readable and modifiable to a large extent. It is also constantly updated by developers around the world thus making it the ideal research platform. At the time when this research was undertaken, the flight software was based on the low cost and easy to use Arduino platform [25]. The resources of this project was extensively used in the beginning to get a head start on the development of the indigenous Quadrotor; the sensors, Arduino microcontroller and an electronic shield were purchased as to assemble kit that made up the flight controller board i.e. the IMU and the microcontroller which runs the flight software. Figure 5 show the flight controller that is the result of the Aeroquad project at this time.

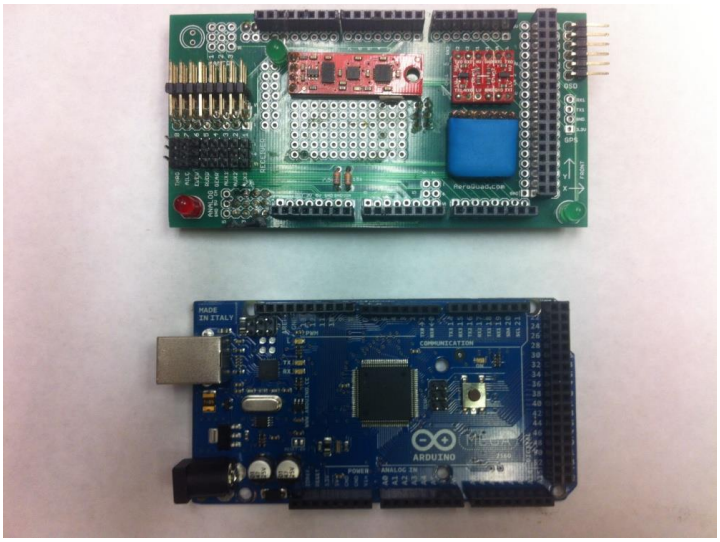

Fig.5 Aeroquad Sensor Shield (above) and Arduino MEGA 2560 Microcontroller

All of the initial testing, flying, code development and tuning was done using the flight controller shown in Figure 8. In early 2014, while testing altitude hold features in strong winds, it was partially damaged in a crash and subject to random voltage drops and sensor outages in subsequent testing and was deemed beyond repair and henceforth abandoned.

2.1.2 ArduPilot Mega (Arducopter)

The APM autopilot suite started as a DIY project similar to the Aeroquad project back in 2007, since then it has matured into one of the most impressive low cost flight controllers for multi-rotor UAV's in particular. 3D Robotics officially adopted the APM project and drove it towards what it is today, able to turn any RC car, boat or UAV (fixed and rotary wing) into an autonomous vehicle complete with waypoint navigation capabilities.

The hardware package is sold by 3DR along with peripherals such as GPS, telemetry radios, ultrasonic rangefinders etc. The APM 2.6 was adopted in the latter stages as the flight controller for this project because of its availability (courtesy DPI) and the experience DPI had with it. Reliability and ease of use were key to the seamless transition, as it was based on the

same Arduino platform being used before, all of the code development for future implementation stayed usable and relevant. The APM 2.6 comes pre soldered with sensors and breakout pins and thus ended the need for further hardware development for this research. It has enabled excellent flying results and also importantly facilitated the sensor data and other flight data collection with ease. The recursive PID tuning process thus became better with the analysis of flight data, whose collection and analysis was something the author struggled with in the Aeroquad version. The Quadrotor developed along with the APM 2.6 flight controller achieved fully autonomous flight utilizing the existing PID control structure onboard the controller. Figure 6 below shows the APM 2.6 flight controller module and board.

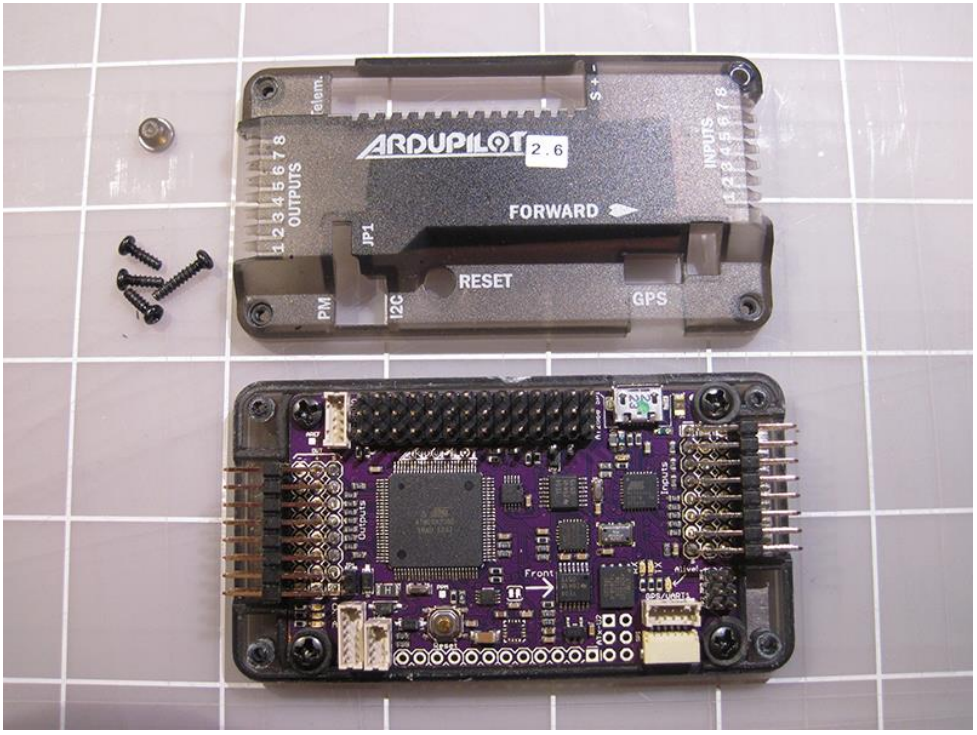

Fig. 6 APM 2.6 Flight controller board (Courtesy DPI)

2.1.3 MicroPilot

MicroPilot, based in Canada, is one of the earliest UAV autopilot developer established in 1994. Their autopilots are compact, lightweight, and powerful, having flown everything from one-pound MAVs/backpacks to high-speed, turbine-powered drones. As one of the leading commercial developers, their autopilots cost in the range of $ 3000 - $ 10000, gearing them more towards military applications than commercial ones. They support highly detailed ground control station software called Horizon and feature full autonomy from takeoff to landing. It is also based on the nested PID control loop architecture and allows for in-depth tuning of a multitude of parameters. DPI has been using the Micropilot line of UAV autopilots on their large single rotor UAV the DP-5X Wasp, their small tandem rotor UAV the DP-6 and the larger 400 lb. tandem rotor UAV the DP-12 Rhino to great success. Figure 7 shows the MP 2128^{HELI} along with an external compass and an integrated Ublox GPS sensor.

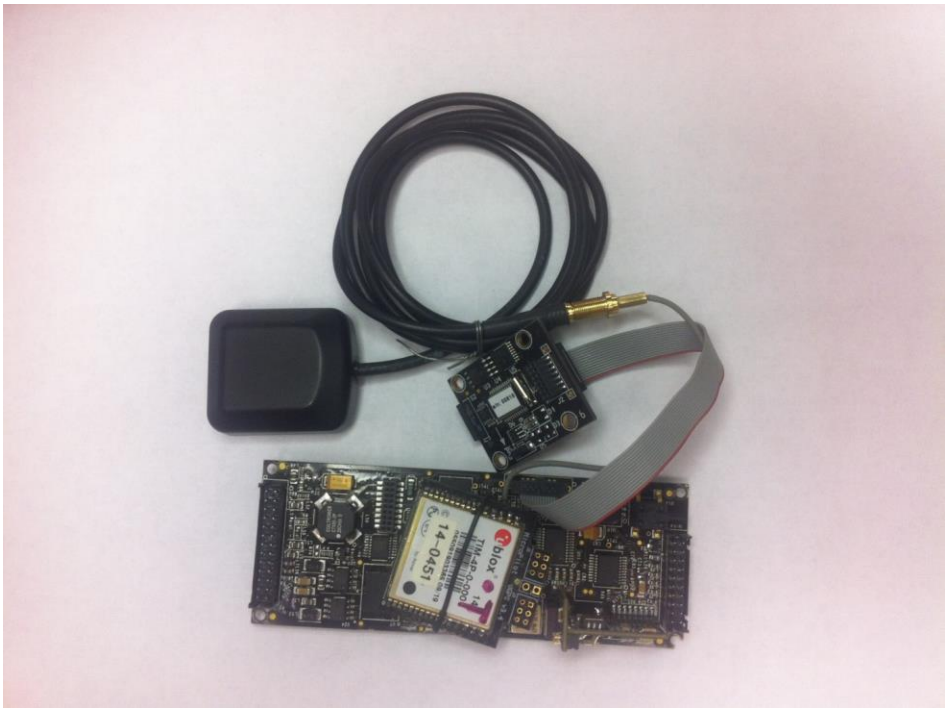

Fig. 7 MicroPilot's MP 2128^{HELI} Autopilot Board with Integrated Ublox GPS and External Compass (Courtesy DPI)

2.1.4 DJI WOOKONG M

DJI are currently the largest manufacturer of multi-rotor UAV's in the world. Their line of UAV's like the Phantom Quadrotor, S800/900 Hexrotor and S1000 Octarotor along with a variety of camera gimbals are one of the best camera/photography aerial platforms in the market. Their UAVs fly either the NAZA or WOOKONG range of autopilots which are widely used multi-rotor flight controllers. The Wookong flight controller is however, very easy to use and adopt, standard tuning parameters exist and are well implemented.

The Wookong range of autopilots is the more professional flight controller and costs about $ 1000. Over and above the NAZA's features, it also offers an IPad based ground control station, built-in gimbal stabilization, precision position and altitude hold etc. It is the flight controller of choice for professional aerial photographers, shown in Figure 8 below.

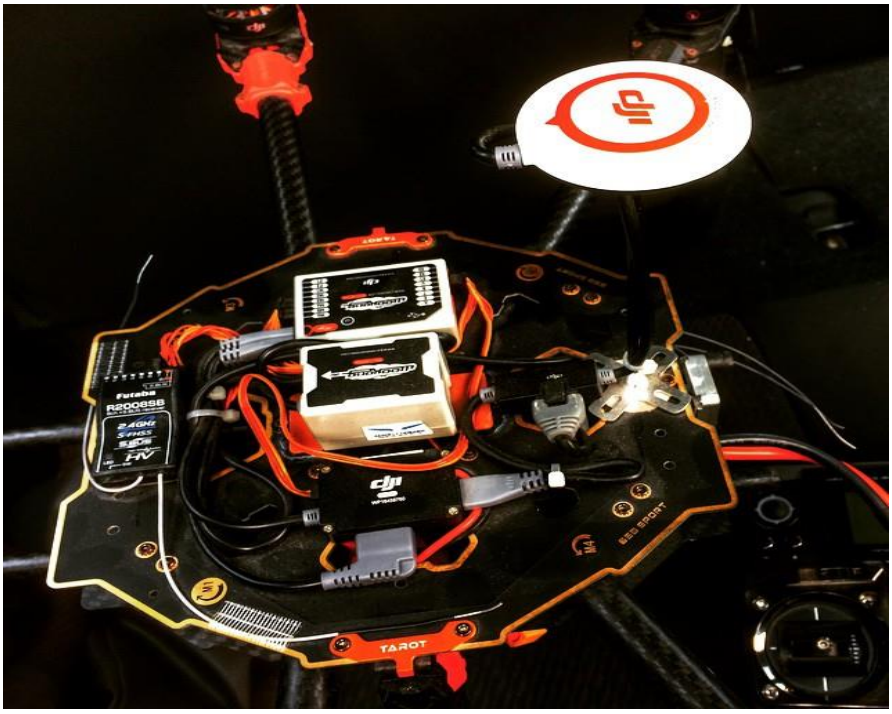

Fig. 8 DJI's WOOKONG Flight Controller with External GPS and Customized Motor Mixer (Courtesy DPI)

2.2 Bibliographic Survey

The Quadrotor's basic dynamic model is the starting point for most of the research but some work introduces complex aerodynamic properties as well [28, 29]. A variety of control sciences have been applied, most notably PID controllers for attitude and autonomous control as well as waypoint navigation [30, 31, 32, 33]. Back stepping control methods [34, 35], nonlinear H∞ [36], LQR controllers [33] and other nonlinear controllers with nested saturations [37, 38], have been applied with varying degrees of success. Almost all of the above controllers require accurate information from the MEMS like accelerometers, gyroscopes, GPS, barometer etc. [39, 40].

Bouabdallah et al. [41] utilized a Lyapunov Function's nonlinear control technique to stabilize the quadrotor's attitude, and compare the real system behavior against a respective simulation. Bouabdallah et al. [33] extended their work on the OS4 project, comparing PID controllers against modern LQ adaptive optimal control methods for the attitude control problem. Tayebi and McGilvray [30], performed advanced studies on control techniques for orientation control of a Quadrotor using quaternion math for attitude representation, Lyapunov stability criterion and a PD2 feedback structure. Kim et al. [42] compared the performance of four control techniques: LQR, LQR with gain scheduling, feedback linearization and sliding mode control. Their experimental results verified that LQR with gain scheduling was more robust in light of modeling uncertainties whereas the sliding mode control showed better performance for a well modeled system. Voos [43] applied feedback linearization in a nested control loop structure, with the inner loop performing attitude stabilization and the outer loop for position control. Experimental results were achieved with the commercially available Draganflyer Quadrotor platform. This control hierarchy also works well with PID based nested control.

2.3 PID Control

The Proportional, Integral Derivative control is by far the most common feedback control algorithm used by UAV flight control systems. It is in fact also the most widely used control methodology in practice today across a whole spectrum of industries [44]. It easy implementation, ease of use across different disciplines and low processor cost make PID control based applications ubiquitous. A PID controller calculates an *error* value as the difference between a measured state variable and a desired state to be achieved. The controller attempts to minimize the *error* by adjusting the process through use of a manipulated variable. As described above, almost all the autopilot systems available in the market today for multirotor UAVs use a nested PID control loop structure. The inner attitude loops are nested within the outer/higher level position control loops, these outer loops are also sometimes again nested within waypoint navigation control loops and the system is built up in layers from a controls perspective. Figure 9 below depicts a typical PID control loop. Table 1 describes the three functions carried out by a typical PID controller. Both the Aeroquad and APM platforms used in this research project utilize the PID control methodology described above.

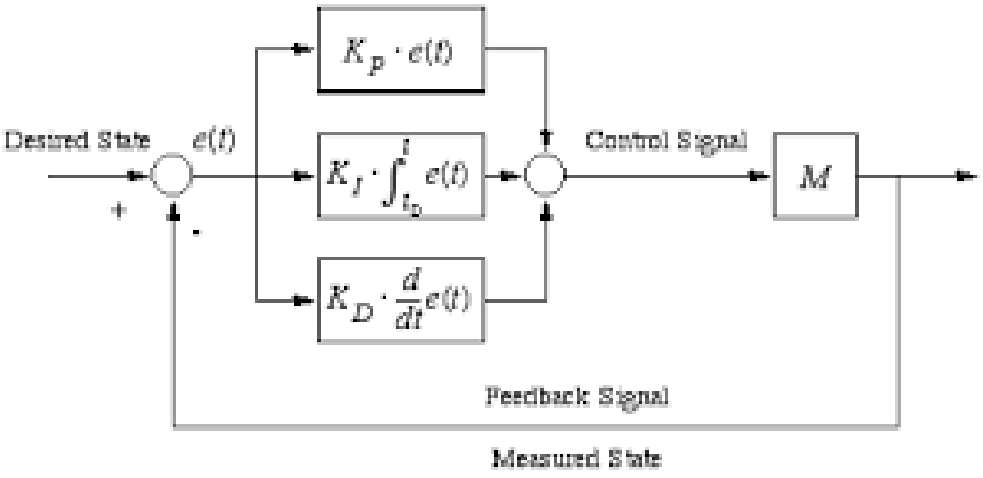

Fig. 9 Typical PID Control Loop

Table 1 PID Definition

Sub System	Math	Definition
P - Proportional	$K_p \times error$	This term achieves the main control drive in the system and drives the error towards zero.
I - Integrator	$K_i \times \int error \times dt$	The integral term reduces the final steady state error in the system
D - Derivative	$K_d \times \frac{derror}{dt}$	This is an anticipatory control action that normally counteracts any sudden changes in the system that the K_p and K_i terms react to.

In the absence of knowledge of the underlying process, a PID controller has historically been considered to be the most useful controller [45]. The least utilized term is K_d, which is almost always set to zero because of its poorly understood action on a specific control problem. Derivative action predicts system behavior and thus improves settling time and stability of the system [46, 47]. An ideal derivative is not causal, so that implementations of PID controllers include an additional low pass filtering for the derivative term, to limit the high frequency gain and noise [48]. Derivative action is seldom used in practice though - by one estimate in only 20% of deployed controllers, because of its variable impact on system stability in real-world applications [48].

In the latter stages of this project, the APM 2.6 flight controller board was used to fly the Quadrotor and having excellent data logging capabilities, it provided an excellent platform to learn more about and tune the existing PID attitude and position control loops.

2.3.1 3DR Hexrotor PID Tuning

It was the author's intention to lay the ground work for studying modularity for the Fuzzy Logic controller developed at a later stage. In the interest of the above research goal, the preliminary work of building a new Hexrotor UAV utilizing the same APM 2.6 controller used to fly the Quadrotor was accomplished at DPI. The ultimate goal was to build a modular simulator tool for the whole class of multi rotor UAVs so that the control algorithms developed may be tested extensively before embarking on the arduous implementation process. DPI has shown considerable interest in commercializing the autonomous controller developed by the author, once all key concerns such as robustness, modularity etc. have been addressed. Figure 10 below shows the 3DR Hexrotor that was developed, tuned and flown at DPI by the author.

Fig. 10 3DR Hexrotor Developed at DPI

It should be noted that this is not to be considered as a tangent to the research goal rather as an extension that brings significance to this research. DPI had already built, tested and flown an Octarotor UAV, thus with the development of the Hexrotor, it became possible to evaluate all interesting multi rotor UAV configurations. Due to a lack of time, the simulation work for the Hexrotor remains pending and will be remain part of the future work to be accomplished at DPI

by the author. Here the PID tuning effort for the Quadrotor and Hexrotor that was simultaneously accomplished is recorded. The following is representative of both the Quadrotor and Hexrotor developed and is thus referred generally as the multi rotor in this subsection only.

The initial K_p, K_i and K_d gains that were set for the roll and pitch controllers proved to be quite poor and this is illuminated by the data logs of the initial flights. Figure 11 and 12 below shows the flight log for desired roll vs. the actual roll and desired pitch vs. the actual pitch, flying in the so called "Stabilize mode", the RC transmitter's stick position directly translates to a requested roll/pitch/yaw attitude and throttle percentage.

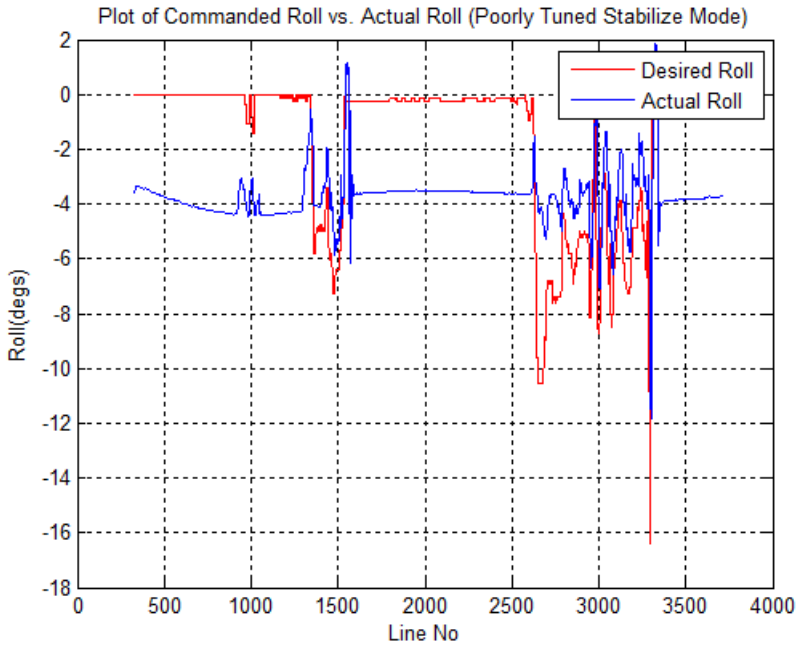

Fig. 11 Desired Roll vs. Actual Roll (Stabilize Mode)

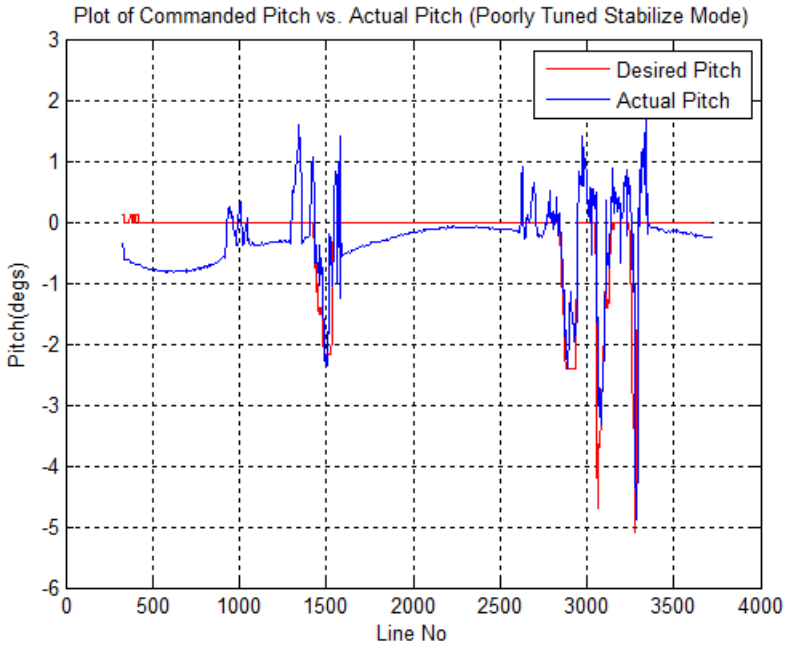

Fig. 12 Desired Pitch vs. Actual Pitch (Stabilize Mode)

It is clear from the data graphed above that the PID's have been incorrectly tuned, or demonstrate poor attitude tracking.

It was indeed quite difficult to control the multi rotor at all let alone pursue aggressive maneuvering or attempt an autonomous flight using the stock PID settings. The autonomous PID controller calls the inner attitude loop controller in turn and hence if the multi rotor's inner attitude loops are poorly tuned, a fully autonomous flight cannot be achieved.

The tuning process progressed according to the following guidelines:

1. Lock pitch and roll gain values, meaning that both sets of gains will be the same as the multi rotor is perfectly symmetrical about Z axis.

2. Set all gains to zero and program the knob on the transmitter to be the K_p tuning channel allowing for min value of 0 and max value of 0.1 (APM Forum).

3. The stock gains for the rate PID controller (inner loop) for the multirotor are K_p=0.1; K_i=0.05 and K_d=0.

4. Take off in stabilize mode and keep turning the knob thus increasing K_p until the multirotor starts to oscillate.

5. Back off about 20% of that final oscillating K_p point and verify the actual value using the Mission Planner ground control station that connects to the APM over the telemetry modules.

6. Now enable data logging and start tuning the K_i and K_d terms in successive flights until a trend is established i.e. towards stability/instability and inspect logs.

Following the step 5 iteratively and carefully inspecting the logs to observe the attitude tracking, resulted in arriving at an extremely good PID gain for the multi rotor. The attitude tracking achieved was excellent and allowed very aggressive maneuvering and hence the ability to record very critical data at the extreme boundaries of the flight envelope which are analyzed later. Conclusions about the advantages and limitations of the PID controller are drawn.

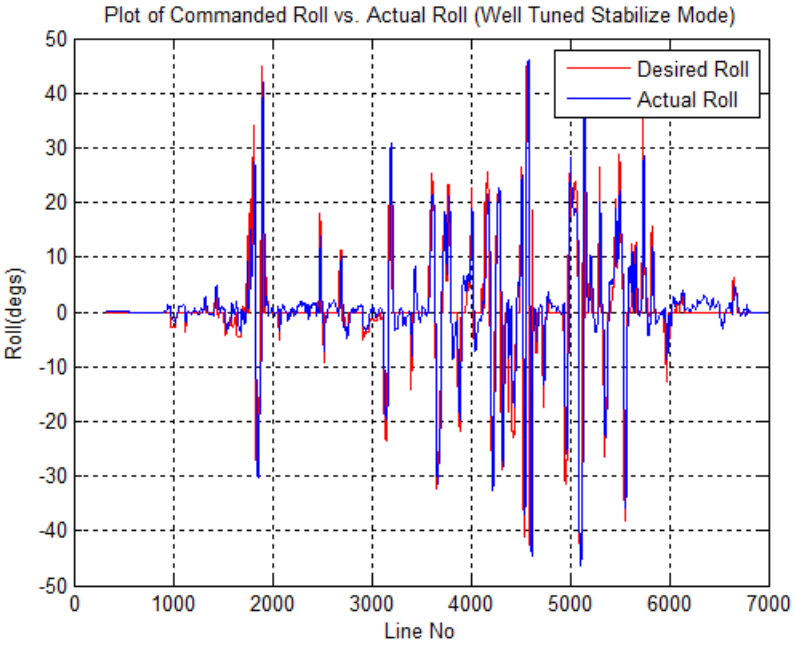

Fig. 13 Desired Roll vs. Actual Roll (Well-Tuned Stabilize Mode)

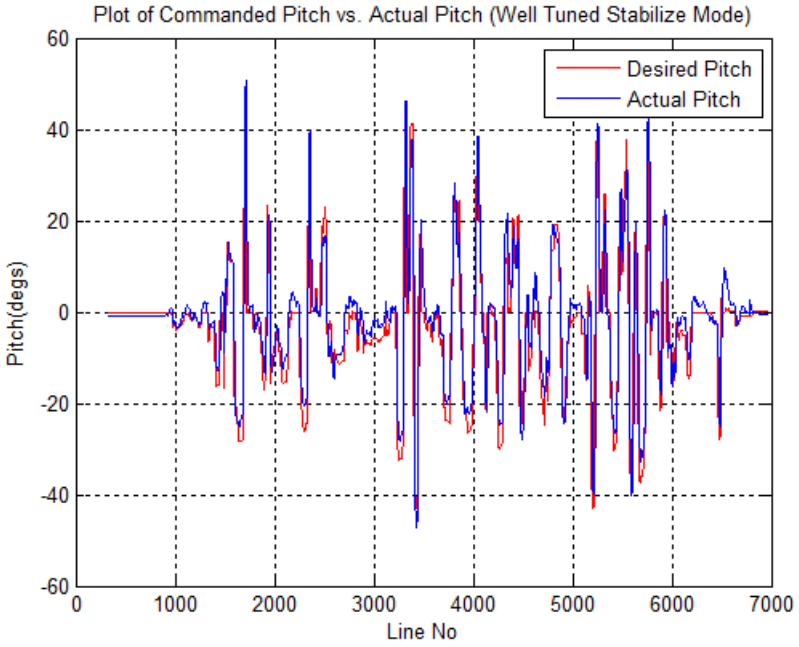

Fig. 14 Desired Pitch vs. Actual Pitch (Well-Tuned Stabilize Mode)

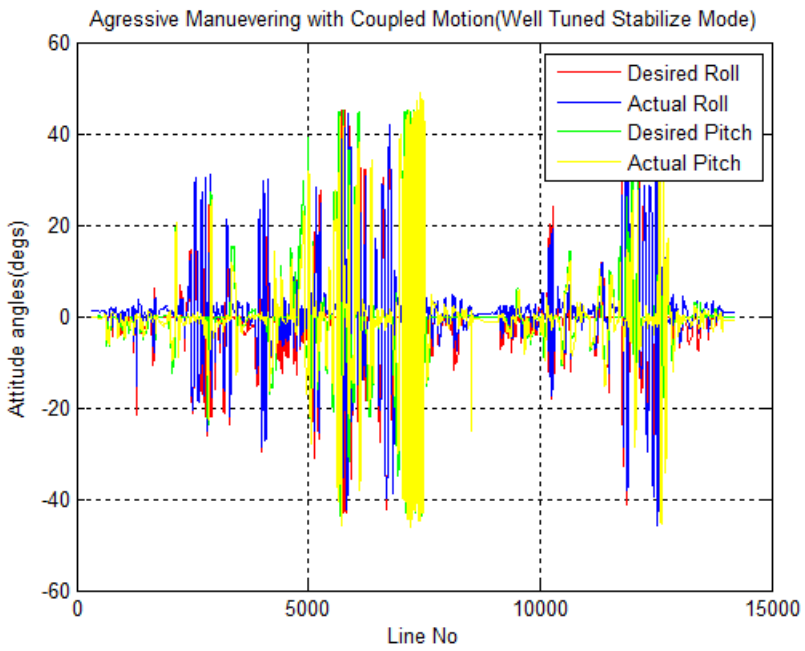

Fig. 15 Flight Example of Aggressive Maneuvering (Well-Tuned Stabilize Mode)

Figures 13 and 14 demonstrate the excellent attitude tracking achieved by following the above tuning guidelines, the highest roll and pitch angles safely achieved are - 45° to + 45°. Figure 15 shows the data log from a 7 min long flight example demonstrating coupled aggressive maneuvering, it is seen that the results are quite impressive. However the PID Altitude hold controller failed to maintain altitude in steep roll and pitch motions thereby demonstrating a drawback of PID controllers; the PID values do not hold for the entire flight domain merely for near hover flight regime. In the example above the altitude fluctuations were large enough that pilot throttle input was required to remain at a particular altitude thereby defeating the purpose of the Altitude Hold mode.

The major limitation of PID control is that there is no one set of controller gains that suits all purposes i.e. K_p, K_i and K_d need to be varied over the entire flight domain continuously. In our specific case, the particular set of PID gain values are not effective over the entire flight

envelope. This is especially true in fast forward flight, wherein it is regularly observed that the Hover PID gains are too high for speedy flight and cause oscillations. One way to get over this has been to make use of Gain scheduling techniques which are difficult to setup correctly but have shown decent performance across different flight regimes. Figure 16 below shows the GPS speed data variation over a typical flight.

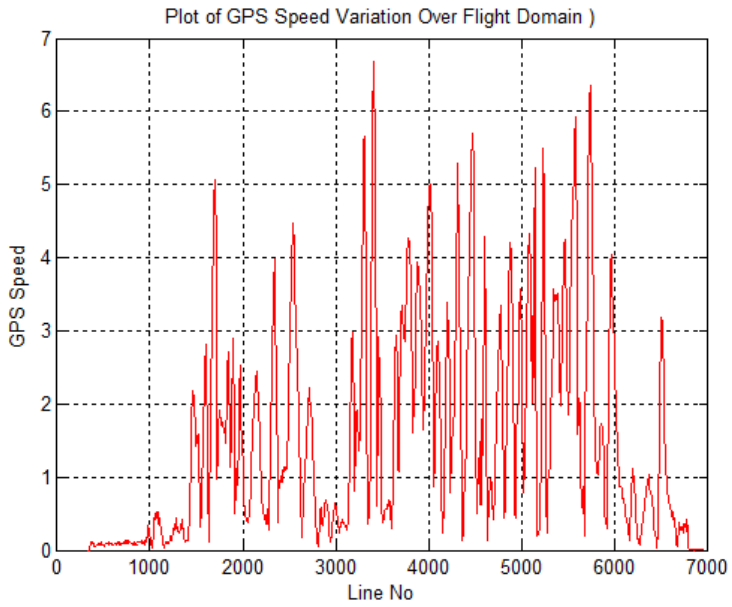

Fig. 16 GPS Speed (m/s) Data Log

Figures 17 and 18 shows the typical PWM motor commands over the entire flight for the Quadrotor and Hexrotor respectively.

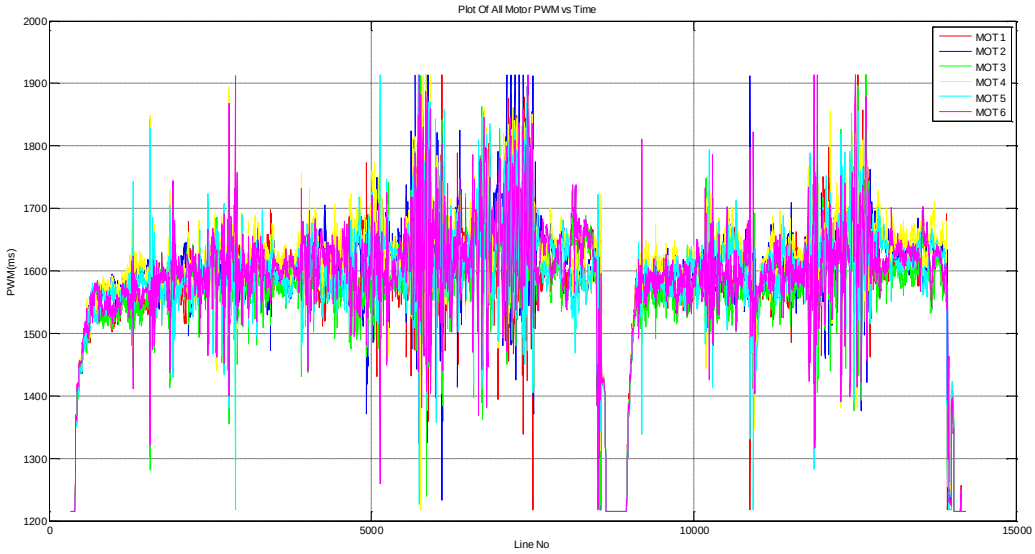

Fig. 17 Plot of Motor commands for Hexrotor during typical flight

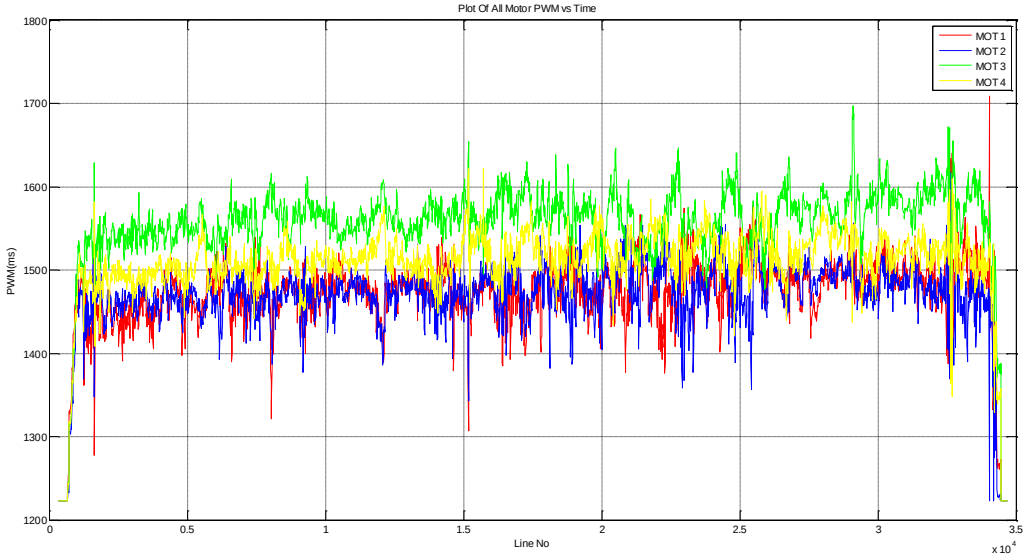

Fig. 18 Plot of Motor commands for Quadrotor during typical flight

2.4 Fuzzy Logic

Fuzzy logic is a misnomer. Developed by Lotfi.A.Zadeh, it is firmly grounded in mathematical theory [49]. FLC is a digital control methodology that emulates human decision making by allowing partial truths i.e. multivalued logic. It allows for the imprecision inherent in real world scenarios.

The output of a fuzzy system is smooth and continuous thereby lending itself to the control of continuously variable systems such as electric motors. In general, FLC is used when the system to be controlled is nonlinear, time variant and ill defined.

Fuzzy logic maps crisp inputs to crisp outputs according to the process shown in Figure 19 below.

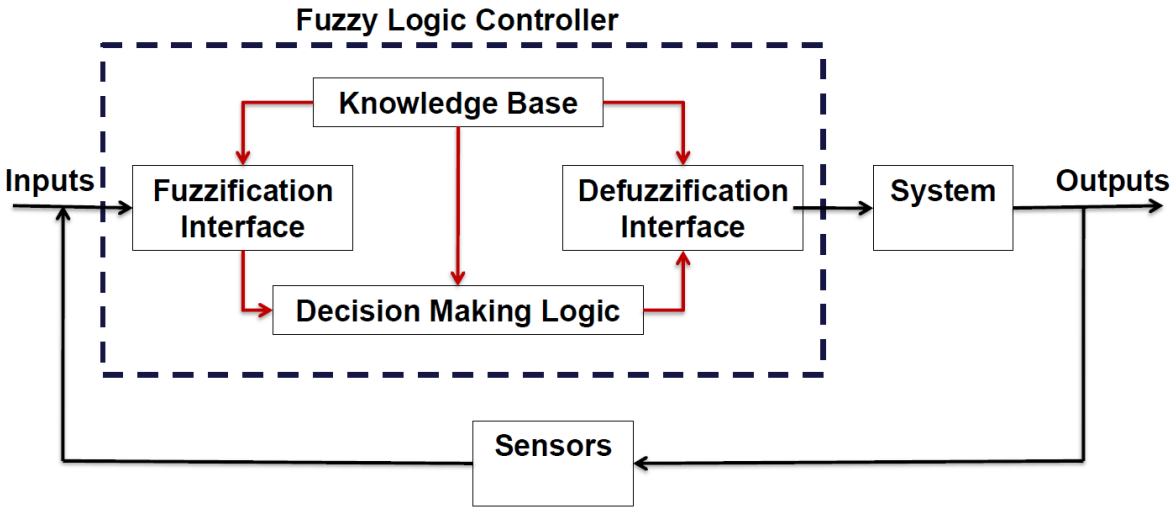

Fig. 19 Fuzzy Logic Control Process

The crisp inputs are first fuzzified i.e. converted into fuzzy membership functions, these input and output membership functions are then related by the inference system based on the rule base developed and then the fuzzy output is defuzzified to produce crisp outputs. Fuzzy logic

offers a form of multivalued logic that is in sharp contrast to the binary logic in place today in computational world. It allows for the partial truths that are inherent in real world scenarios. The fuzzy logic control process is described below in more detail.

A membership function is a function that assigns the degree of membership of a particular value say X of its entire domain. The value of MF always lies between the interval [0 1] where 0 implies no membership and 1 implies complete membership. This is in contrast to typical, binary values that can either be only 0 or 1 i.e. a particular value X must belong completely or not at all to a particular classification.

Furthermore, the domain of a value is typically broken into several membership functions that correspond to the ranges over which the input can be divided. These membership functions typically take the form of triangles, trapezoids, bell curves, Gaussian functions and/or sigmoids. The assignment of the variable to membership functions is based on the designer's knowledge, reasoning, intuition etc. MF's can also be computer generated using artificial neural networks, genetic algorithms etc. Fuzzy logic control for complex systems in this respect suffers from the non-transferability of the heuristic understanding and experience gained from one control engineer to the other. However, this heuristic basis acquired serves the control engineer well when applying simple system control to complex systems.

It is worth mentioning here that designing FLC's is an extremely recursive effort where multiple simulations are done to observe performance and tweak the system by adjusting membership function shapes and relative placement on the Universe of discourse.

The placement and relative placement of the membership functions on the universe of discourse is far more influential on the result than their respective shapes. As a consequence of

this, the control engineer developing a FLC spends more time on range setting and tweaking the functions rather than selecting the geometric shape of the MFs.

The next step in designing the FLC is to develop the set of 'if-then' rules that make up the rule base which relates the above two inputs to the outputs. This is the most crucial phase in the design process and also where the experience gained from previous examples comes into play.

Once a set of rules is created, they can be evaluated based on the crisp inputs given. If the value of the crisp input falls within the domain of the membership function, then the rules involving those membership functions "fire" and are evaluated. The degree to which a rule fires corresponds to the degree of membership of the crisp value in the membership function [50]. Finally, all the outputs from the evaluated rules are totaled and defuzzified (Figure 20). Defuzzification can be done in various ways. Several commonly used ones are: max membership output, where the output is found as the corresponding value of the maximum membership value (in Figure 24 the value would be 0); centroid method (in Figure 20 the value would be 5.8); and weighted average method (the mean of each output membership function is used to calculate an overall average) [51].

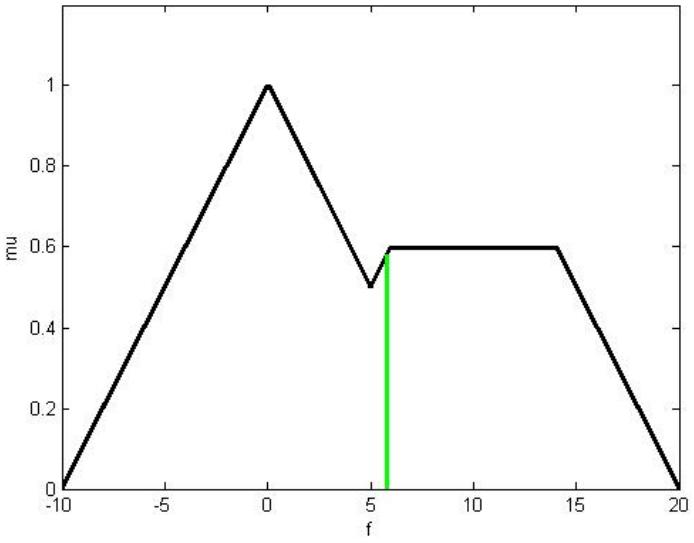

Fig. 20 Defuzzification of Totaled Outputs

CHAPTER 3: SYSTEM DESIGN & MODELING

This chapter provides a complete description of the system and general principles involved in the construction of the Quadrotor UAV. The system components selected, basis for this selection and the evaluation of the design parameters is summarized. The wireless communication between the Quadrotor and a PC is discussed and justification for the propeller selection is given. The Quadrotor went through multiple design iterations throughout this research endeavor and an overview of the different builds is provided. System modeling is presented, specifically the thrust and torque response of the Motor-Propeller combination is extracted.

3.1 System Description

The Quadrotor developed has four rotors symmetrically distributed around a central cabin that houses the electronics. The two cross arms have been constructed out of square 6061 Aluminum alloy tubes measuring ¾ inches x 24 inches. The 6 DOF IMU is mounted using vibration dampers on top plate to reduce sensor noise due to vibrations as far as possible. Motor mounts have been fashioned out of aluminum and carbon fiber and lock nuts have been used to secure them to the frame. This ensures that vibrations don't shake loose the actuator components. The main battery is integrated into the frame at the base position to ensure stability but not placed too low as this compromises the maneuverability of the craft [52]. It was decided to use a separate smaller flight battery to power the Arduino microcontroller and the sensor shield. Motor to motor distance is exactly 60 cm, and the total weight is 1.776 Kgs. It is built and flown as a '+' configuration. This final design was motivated by sluggish responses in preliminary simulations. A significant weight reduction, a robust, repeatable and low cost frame and smaller

inertia values were successfully achieved. Figure 21 shows the Quadrotor system in its entirety. This system was flown for an aggregate of 10+ hours exhibiting good hovering capabilities and stable flight characteristics. A GPS was not integrated with this Aeroquad flight controller as it was not capable of autonomous flight based on its own software.

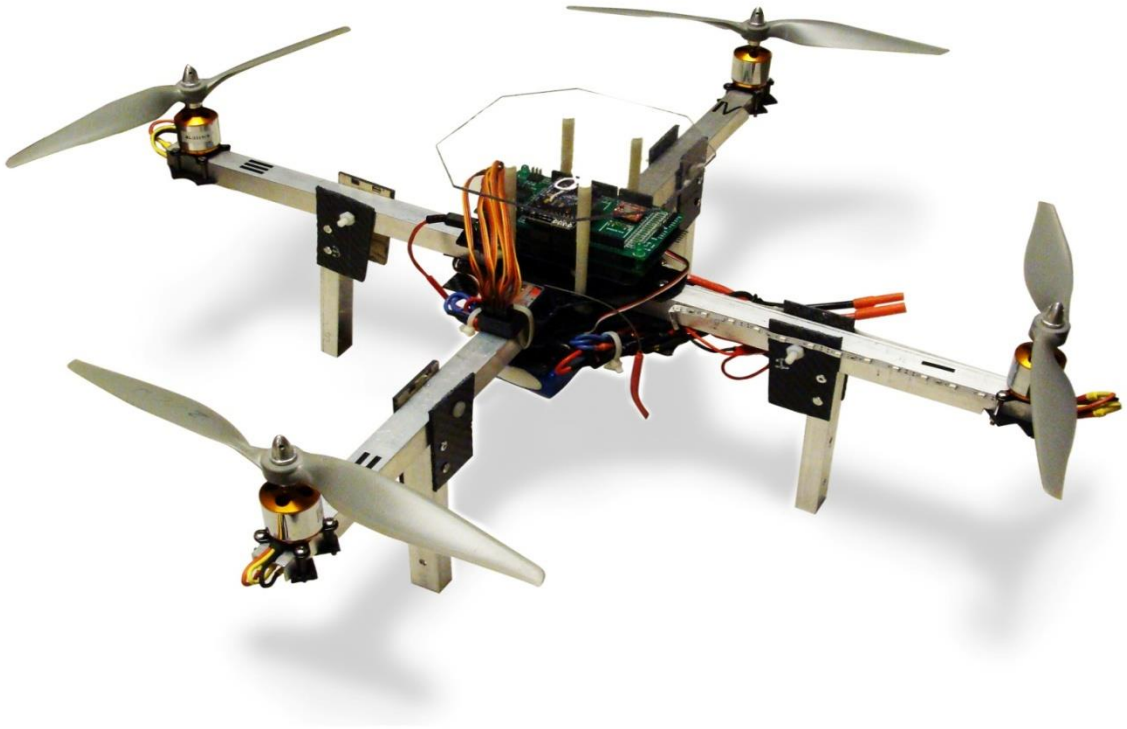

Fig. 21 Quadrotor System

Figure 22 shows the side view and the vertical layout of the Quadrotor can be observed, there are a total of three carbon fiber plates. The top plate is secured to the upper surface of the two cross arms by aluminum rivets and provides structural rigidity and the flight controller is mounted on it. The mid plate is secured to the bottom surface of the two cross arms. The bottom plate is attached via nylon standoffs to the mid plate and the ESC's are mounted between these

two plates along with the power harness. The main flight battery is attached to the bottom of this plate using Velcro straps.

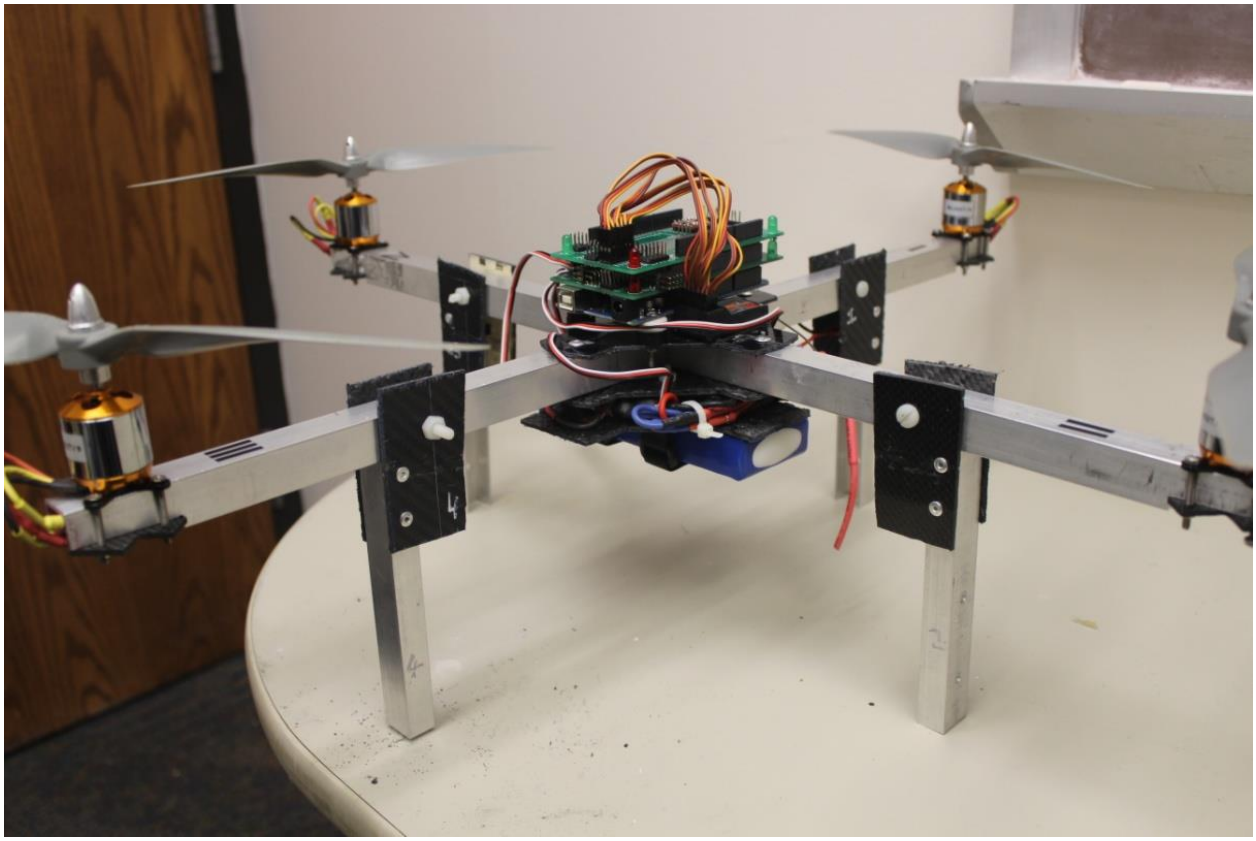

Fig. 22 Side View of Quadrotor

The individual components are listed in Table 2, along with their description and mass properties. The description is a non-exhaustive one, though sufficient in itself to the more familiar. One of the goals being to develop a low cost platform with a high level of confidence, all of the precision machining of the aluminum arms, the carbon fiber plates and the motor mounts were done at the University's workshop by the author. This lead to significant savings compared to buying a frame kit which would then have to be manipulated to fit the avionics. The total cost of the final variant of the Quadrotor is $ 700.

Table 2 Component Specification

Component	Specification	Weight
2 Aluminum square arms	¾" x ¾"x 24"	450 g
4 Landing gear	Aluminum + Carbon fiber	173 g
4 BLDC motors	950 Kv, 200 Watts	292 g
4 ESCs	Exceed RC proton 30A	164 g
4 fixed pitch propeller	APC SF 12x3.8 2CW, 2CCW	88 g
Main lithium polymer battery	Turnigy 11.1 V, 4000 mAh	353 g
Secondary battery	E-flite 7.4 V, 620 mAh	37 g
9 DOF IMU shield	ADXL 345 – triple axis accelerometer ITG 3200 – triple axis gyroscope HMC 5833L – triple axis magnetometer	40 g
Microcontroller	Arduino MEGA 2560	35 g
2 Xbee modules	1 Xbee Pro 900 MHz with Xbee shield(on board) 1 Xbee Pro 900 MHz with USB explorer dongle	11 g
Carbon fiber plates, wiring Standoffs, screws and lock nuts	Miscellaneous frame components	133 g
	Total Weight	1776 g

3.2 Wireless Communication

Figure 23 shows the IMU which is equipped with a triple axis accelerometer ADXL 345, a triple axis gyroscope ITG 3200 and a triple axis magnetometer HMC 5833L. Two Xbee Pro 900 MHz modules have been used to facilitate wireless transfer of signals. This module is ideal for solutions where RF penetration and absolute transmission distance are paramount. A shield is used to communicate with the Arduino and it transmits data to another module that is connected to the ground station via an USB explorer dongle. Data is imported in real time into MATLAB

environment for future hardware-in-the-loop simulations. This particular module offers the least interference to the 2.4 GHz band that is used by the RX/TX setup.

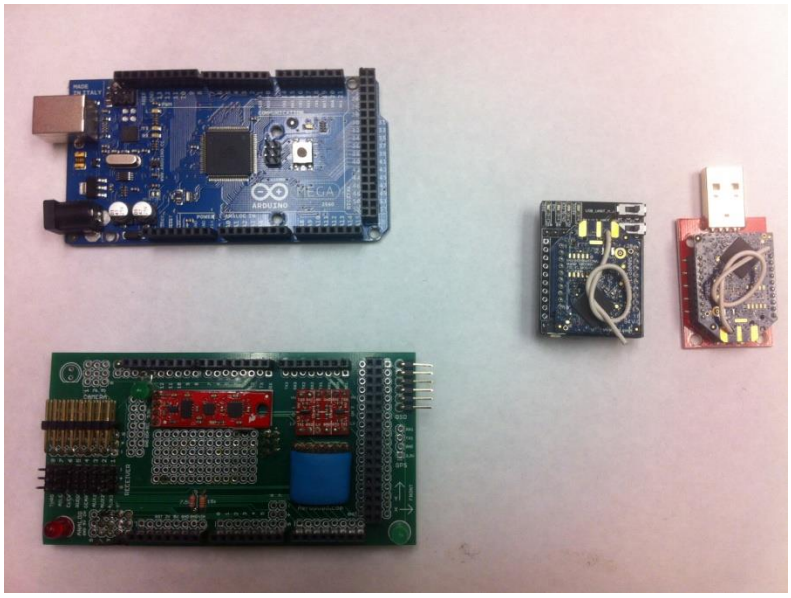

Fig. 23 Xbee Pro 900 MHz modules shown with Aeroquad IMU shield and Arduino Microcontroller

The open source flight software uses DCM [53] techniques to fuse sensor data into an estimation of the attitude of the craft. A 4^{th} order low pass filter is used to filter the accelerometer and gyroscope data. The excellent functionality that the software provides has been utilized extensively for handling communications. The Arduino along with the IMU outputs filtered accelerometer and gyroscopic data as well as roll, pitch and heading angle estimates. We will propose in later sections how the control loops are to be evaluated.

3.3 Propeller Selection and Calculations

There are several rules of thumb when it comes to propeller selection and evaluation. There are also practical considerations such as availability of clockwise and counter-clockwise

propellers for multirotor UAV's in general. Price point and availability of data sheets for the selected propeller is also of prime importance. Reliability and repeatability was also considered.

The thrust to weight ratio set as 2:1 was a good starting point in that it ensures enough thrust exists for desired performance without the Quadrotor being too agile/difficult to fly. The flying weight was initially set to be around 1 kg with the plywood frame of Build 1.0 above. This eventually went up to 1.776 kg through the design evolution. The brushless motor selected was capable of producing a wide range of RPM with a low power draw and high efficiency at hover RPM. The University of Illinois at Urbana-Champaign has an extensive propeller database; this includes wind tunnel measurements for nearly 140 propellers used on small UAVs and model aircraft. The propellers were purchased off-the-shelf from retail outlets and were unmodified for those tests. Measurements include thrust and torque coefficient data over a range of advance ratios for specific RPMs.

This very useful resource was banked upon to find the right propeller according to the above constraints of motor, price point, availability etc. The Aeroquad open source project for multi-rotor UAV as well as several blogs of experienced radio control pilots and hobbyists was researched for starting points. For larger Quadrotor's that carry payloads, large propellers and low-kv motors tend to work better. These have more rotational momentum, and will more easily maintain stability.

Another online resource provides a digital calculator for the various flight parameters and how they are affected by different motor and propeller combinations. A comparison of the different propellers shortlisted along with pertinent information is given below in Table 3.

Table 3 Propeller calculations

Propeller	Volts(V)	Amps(A)	Thrust(g)	Power(W)	Flight time(mins)
APC 10x5 E	12.31	14.20	839.73	174.80	4.6
APC 10x7 E	10.83	12.04	699.40	130.39	3.9
APC 10x4.7 SF	11.06	9.31	470.90	102.97	4.7
APC 11x5.5 E	10.78	13.86	791.53	149.41	3.6
APC 12x3.8 SF	12.12	17.91	1206.49	217.07	7.8

Propeller 10x5 E & 10x7 E had very high pitch angles and did not meet the prerequisites stated above. APC 11x5.5 E & 12x3.8 SF were shortlisted and evaluated further, the poor flight times associated with the former as well as non-optimal power consumption led to the APC 12x3.8 SF as the propeller of choice. Its easy availability and reasonable price point made it an attractive option.

3.4 Build Versions

This research endeavor was borne out of work done towards a concept for an emergency road able vehicle capable of VTOL. The concept took the form of a Quadrotor, and a build of a scale model commenced. As the first build was completed, the scope of work involved in realizing the proposed strategy was enormous. The project focused on autonomous intelligent control and envisioned a future module for obstacle avoidance. Developing and implementing an indigenous controller would contribute significantly to the research already being carried out at MOST Aero labs at UC , while also demonstrating the efficacy of fuzzy logic control. This section details the design iterations that the Quadrotor went through before its final form and issues that were addressed.

Fig. 24 Quadrotor Build 1.0 – Proof of Concept

The first Build 1.0 (see Figure 24) featured a plywood frame and a center to center distance of 0.46 m and 11" x 5.5" fixed pitch propellers. It was constructed purely to demonstrate the Quadrotor concept and to allow investigation of design objectives. The salient points are good vibration absorption and excellent payload fraction; its inherent fragility did not permit a test flight.

It basically facilitated a tangible understanding of how the components would be collocated with respect to each other. The horizontal arms played an important role in absorbing the vibrations and keeping the IMU data relatively the same in ON and OFF conditions. As the initial payload was expected to be a large road vehicle, emphasis was on keeping the frame as light as possible. The basic layout of the components remained the same in the next two subsequent builds, i.e. the ESC's were located along the arm length, thus keeping the wire length short and eliminating the need for another layer in the central housing for the ESC's, thus saving considerable weight and complication.

Fig. 25 Quadrotor Build 1.1

Figure 25 shows the first design that achieved flight using a RX/TX radio setup. Carbon fiber arms were machined and overlaid on the wooden framework to reinforce it. Precision machining of the carbon fiber proved to be a very time consuming and difficult operation, however attempting to outsource this work would have necessitated the need for accurate Solidworks drawings and significant resources which would have driven the cost upwards. There was also the question of future modifications being made necessary due to poor flight characteristics.

At this particular point in the project, there were several electronic problems associated with the IMU board, attributed in part to bad soldering practices. An oscilloscope and a multimeter were used to scope the signals to the various sensors on board for fluctuating voltages and bad connections and after identifying the same, corrective measures were taken. The horizontal arms helped absorb vibrations, thus keeping sensor noise to a minimum. However the flex allowed by the arms led to unstable and undesirable flight characteristics.

Fig. 26 Quadrotor Build 1.2

This led to the build 1.2 pictured above in Figure 26 which incorporated sectioned aluminum square tubes to both stiffen the arms and produce more durable landing gear. Stability was achieved at the cost of increased weight.

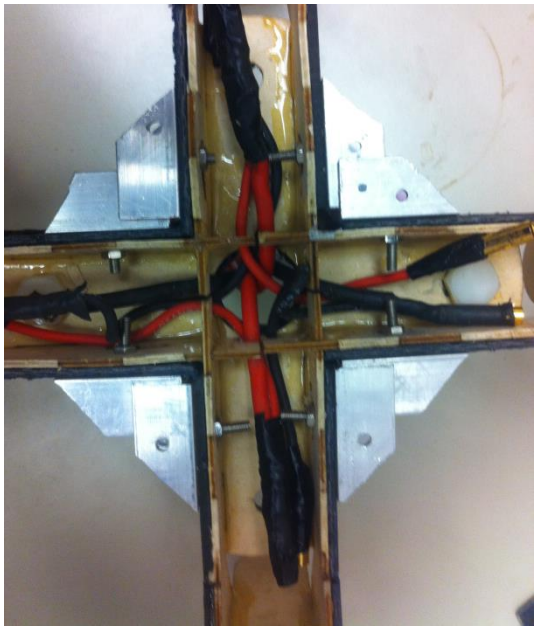

Fig. 27 Top View of Quadrotor Build 1.2 (Top Plate removed)

Figure 27 shows the top view of this build showing the layout of the wires and the cross aluminum sections that reduced the flex in the arms by integrating the load in the arms onto the whole frame. This prototype was the first to be modeled, however the responses obtained from the simulation showed an extremely sluggish behavior. Thrust to weight ratio went below the 2:1 threshold set from research into design concepts.

A complete overhaul of the system was proposed and the result was the Quadrotor as shown in Figure 21 at the start of the chapter. Increasing the span length improved stability as well as allowing the use of slightly bigger propellers. A 1" increase in propeller diameter increased thrust by over 20%. The ESC's were incorporated into the central housing. Also a repeatable, reliable, low cost frame was developed which can be used as a starting point for all future projects. This iterative design process facilitated a basic understanding of the dynamics of a Quadrotor, the factors affecting it and design optimization.

3.5 System Modeling

The mathematical model presented was used to develop a simulation environment in MATLAB/Simulink to facilitate the development of proposed control strategy. Experimental modeling was favored over theoretical modeling for the motor-propeller combination, to generate real world input-output relationship that incorporates miscellaneous losses for the simulator. The Quadrotor is a complex mechanical system; it collects numerous physical effects from the aerodynamics and mechanics domain [33]. It should be noted that ill-defined and complex phenomena such as ground effect, gyroscopic effects, airflow interactions, propeller flapping etc. were studied but not modeled due to their complexity. It is hoped that the inherent robustness fuzzy logic control embodies will tackle these shortcomings. Future development

insists on a more comprehensive model that will lead to significant improvement in the results obtained.

Two experiments were setup to accurately quantify important model parameters such as thrust and torque generated with respect to the pulse width modulated output values of the actuators. The experimental modeling allowed incorporating and taking into account air interaction effects on thrust efficiency and friction losses. Figure 28 shows the experimental thrust setup, the Quadrotor is weighted down on all four arms at a distance from the center to minimize the roll and pitch motions that inevitably occur at high PWM values. Due to manufacturing tolerances, at a specific PWM value, each motor-propeller combination spins at slightly different speeds causing unequal lift and thus making reading of the weight scale difficult and imprecise. A program was written in the Arduino IDE that sends incremental PWM values to all the motors with a delay built in to allow for recording of the scale readings. Characterizing the motor's in this way, while time consuming lends high confidence to the expected thrust and torques generated during future simulation and validation.

Fig. 28 Experimental Thrust Setup

Figure 29 below shows the thrust vs. PWM plot from the experimental data collected using the setup above

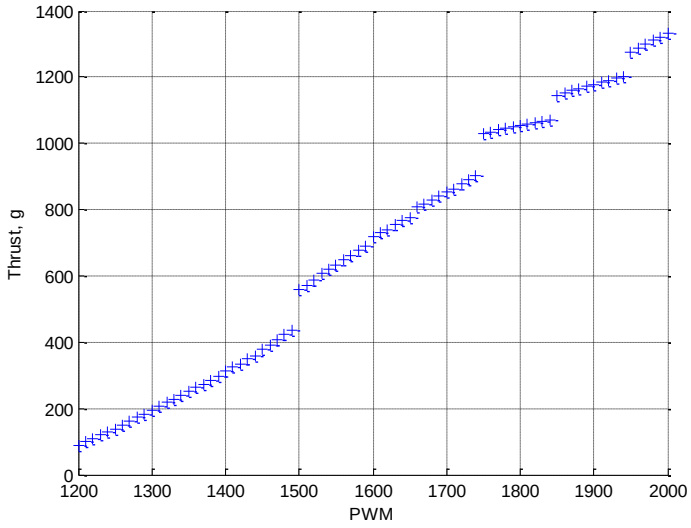

Fig. 29 Thrust vs. PWM plot

When the Quadrotor's altitude is approximately at or below the same distance as the rotor diameter, there is, depending on airfoil and aircraft design, an often noticeable ground effect. This is caused primarily by the ground interrupting the wingtip vortices and downwash behind the wing. When a wing is flown very close to the ground, wingtip vortices are unable to form effectively due to the obstruction of the ground [54, 55]. During the experimental validation process for the Quadrotor Simulator, it was observed that the thrust data collected did not lend itself well to validation during the entire flight envelope rather only during the takeoff and landing phases. This is consistent with the fact that during these phases ground effect has a large part to play depending on the distance from the ground and as such the data collected above is suitable for validation only during these times. The rotational speed of a brushless DC motor is heavily dependent on the voltage supplied across its leads. During flight, the voltage of a lithium

polymer battery constantly keeps dropping, this leads to higher current in the motor windings i.e. the speed controller tries to achieve the desired rotational speed but is unable to do so as they are not closed loop speed controllers. Thus in the experimental setup above, the batteries were replaced with fresh ones at the midpoint, thus the higher voltage caused higher peaks in the thrust values obtained.

The experiment was conducted in this way to best mimic the actual flight condition but ultimately failed in its objective of producing usable thrust data. A completely new thrust setup was envisioned to accurately collect thrust data, while also measuring RPM and Power, while keeping the voltage constant thus allowing for the complete characterization of the motor-propeller combination. The revised setup uses an external constant DC voltage power source and a teeter-totter method for accurately determining the thrust generated.

Fig. 30 Revised Experimental Thrust Setup

Figure 30 portrays the revised experimental setup with a constant voltage DC power supply configured to provide 11.1 V (3S LIPO nominal voltage), a Futaba transmitter and receiver setup to provide the variable PWM values on the throttle channel, an inline PWM meter and Watts up meter facilitates measurement of all variables of interest such as PWM, current, power. An accurate scale rests at one end of the teeter-totter square steel tube with a counterweight designed to keep the arm perfectly balanced on the scale. A tachometer is used along with reflective tape on the propeller to facilitate RPM measurement. The electronic speed controller (ESC) has been throttle calibrated so as to provide smoother flight characteristics.

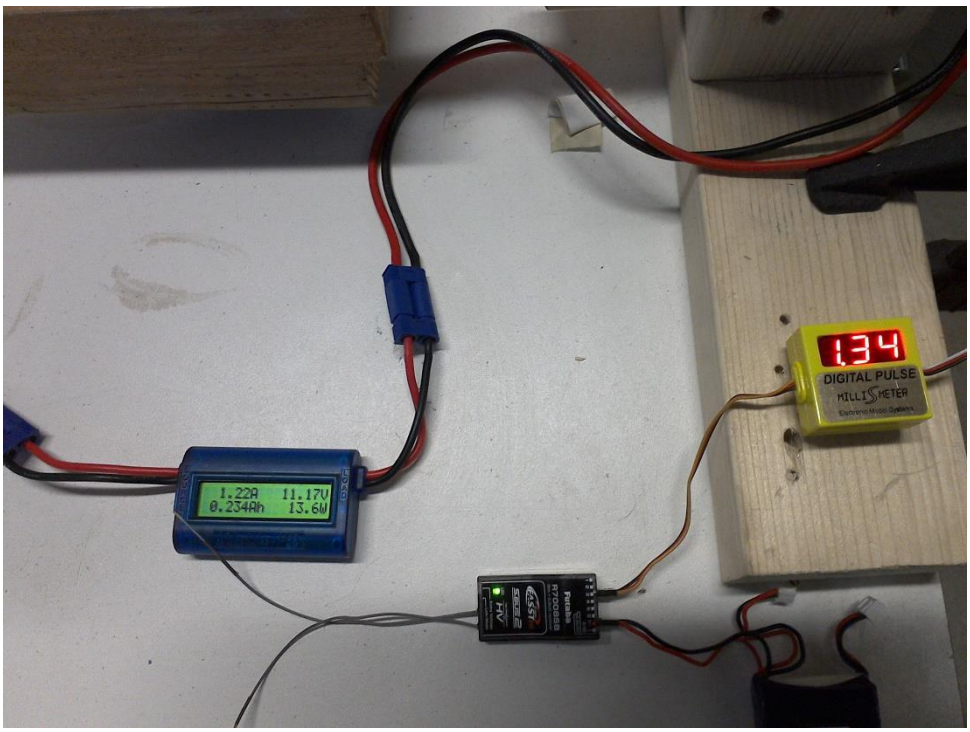

Fig. 31 Inline PWM and Watts Up meters and RX setup

Figures 31 and 32 show the component level connections and overall setup in greater detail. The tachometer readings eliminated all uncertainty from the experiment by providing evidence for the discontinuities observed earlier due to higher batter voltages and also provided a basis for comparison against the propeller database of the University of Illinois at Urbana-Champaign.

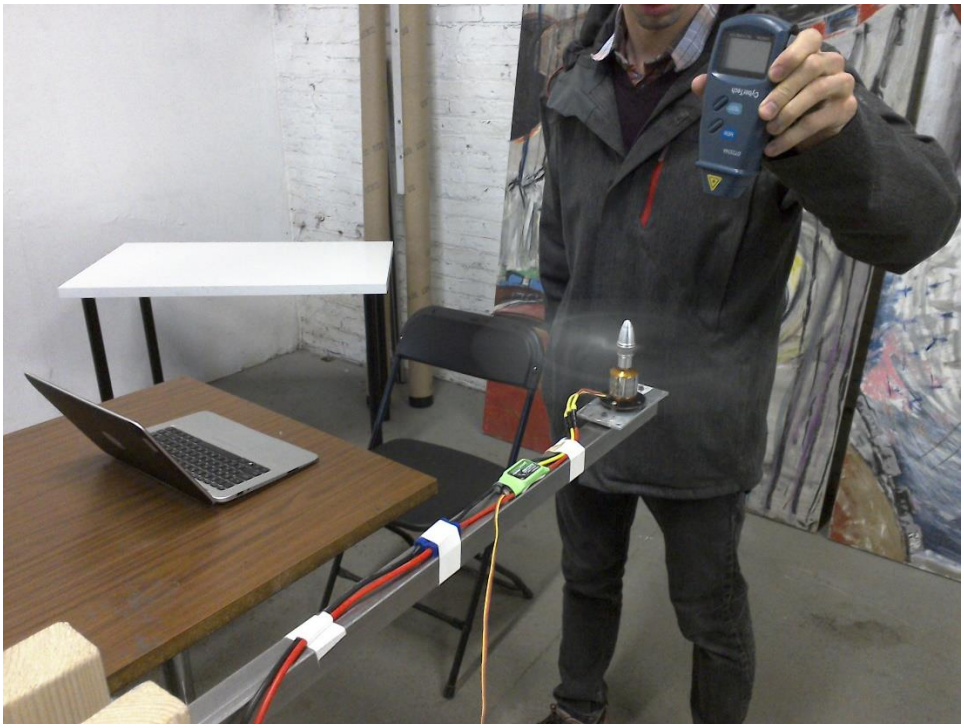

Fig. 32 RPM measurement using Non-Contact Laser Tachometer

Figure 33 below shows the plot of thrust vs. PWM and we observe an almost linear response over the entire range of PWM values.

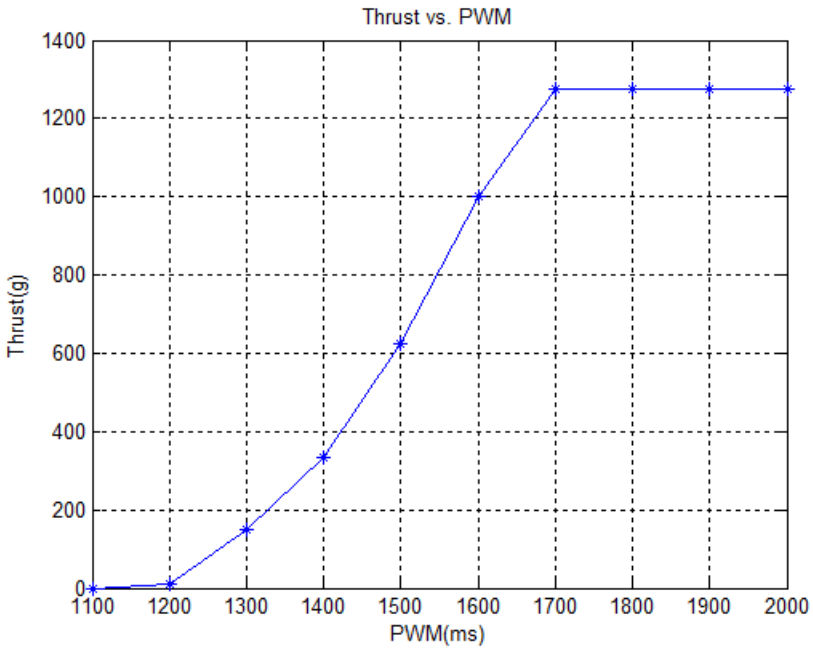

Fig. 33 Plot of Thrust (g) vs. PWM (ms)

Figure 34 shows the plot of thrust generated versus the rotational speed of the motor-propeller combination. Figures 35 and 36 represent the complete characterization of the drive system.

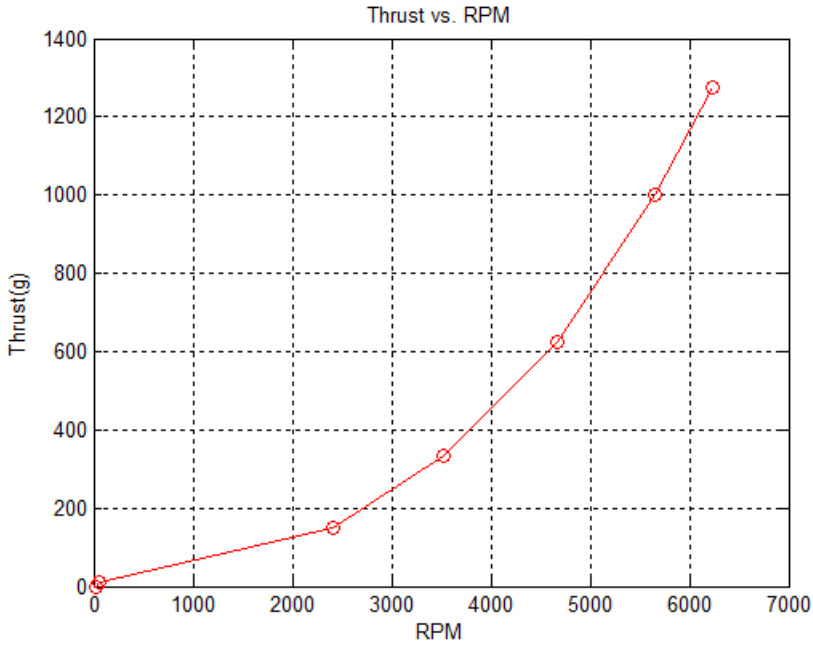

Fig. 34 Plot of Thrust (g) vs. RPM

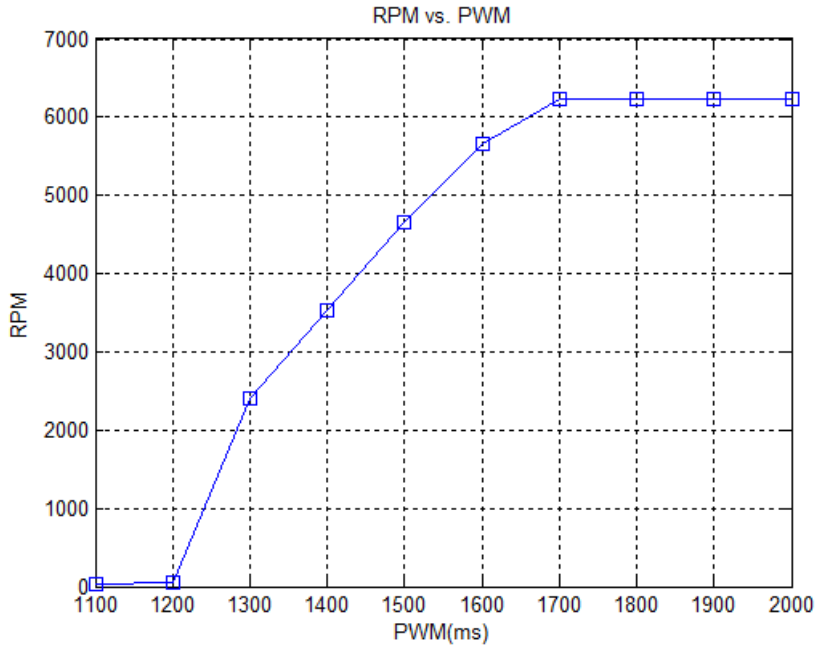

Fig. 35 Plot of RPM vs. PWM (ms)

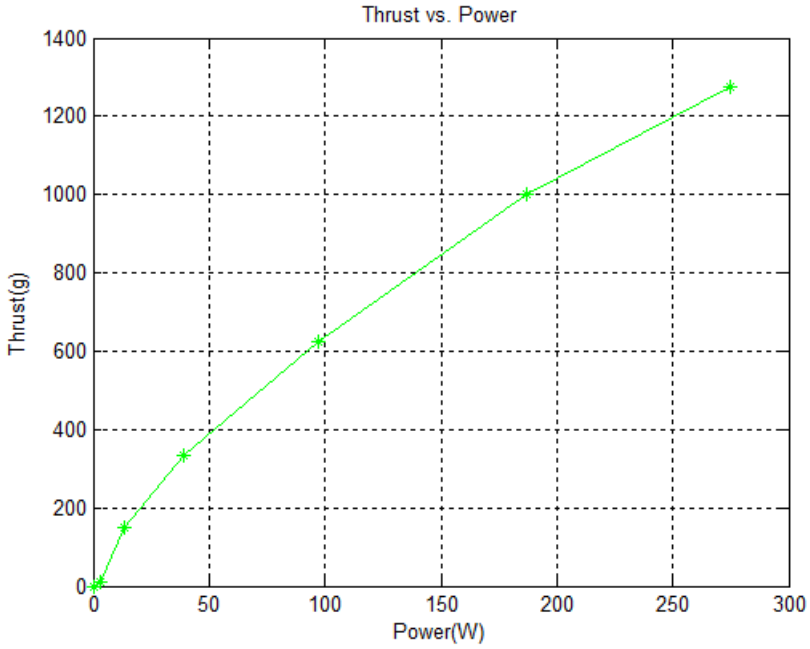

Fig. 36 Plot of Thrust (g) vs. Power (W)

Table 4 below represents all the data collected for this motor-propeller combination. The data reflects the average of 10 experimental runs. The experiment was tightly controlled even in testing duration as the

motor performed differently at different temperature conditions. Each set of data was collected in exactly two minutes and a delay of 15 minutes was given to ensure the motor cooled down to room temperature.

Table 4 Thrust Test Data

PWM	Thrust (g)	Amps(A)	Power(W)	RPM	Efficiency (g/W)
1200	10	0.27	3.05	514	3.28
1300	147.5	1.23	13.7	2410	10.77
1400	332.5	3.48	38.8	3521	8.57
1500	625	8.81	97.3	4666	6.42
1600	1000	17.39	187	5661	5.35
1700	1275	25.35	275	6235	4.64
1800	1275	25.35	275	6235	4.64
1900	1275	25.35	275	6235	4.64
2000	1275	25.35	275	6235	4.64

A torque stand setup shown in Figure 37 along with a high precision balance was used to generate Torque vs. PWM data, shown as a scatter plot in Fig. 38.

Fig. 37 Torque Stand Setup

The torque values obtained from such small brushless out runner motors are understandably quite small and hence the yaw response of the Quadrotor UAV in general is quite poor. This is the input to the simulation model and allows for the calculation of the torque produced by the motor-propeller combination at different commanded PWM values. This is true of the entire class of multi-rotor UAVs; however this has not proved to be a deal breaker in terms of performance and usefulness.

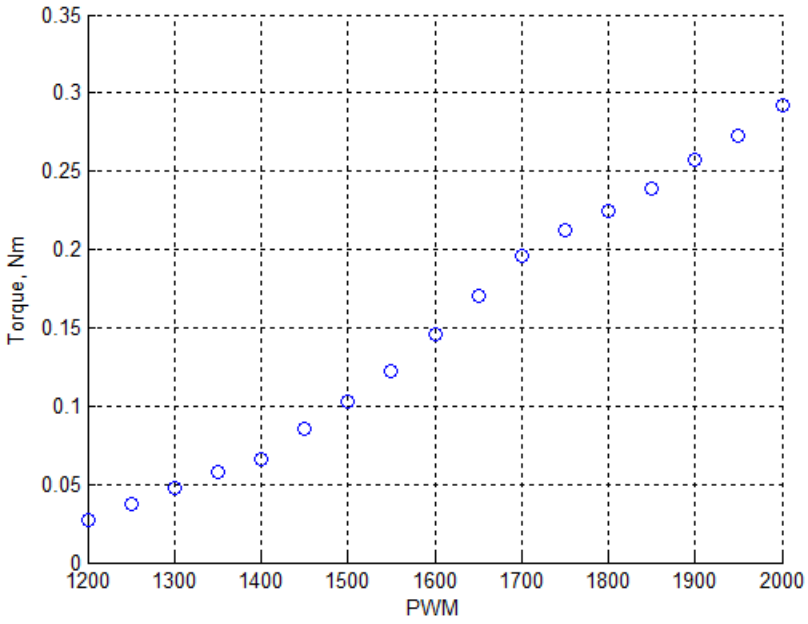

Fig. 38 Torque vs. PWM Plot

CHAPTER 4: MATH MODEL & METHODOLOGY

The math model developed is described in detail in this chapter. The following subsections describe the dynamics of the Quadrotor along with the forces and moments generated by the motor-propeller combination. In this chapter, the overall control system is described in detail, including control inputs and outputs, the decision-making rule base, and the control scheme implemented.

4.1 Math Model

This section details out the development of a mathematical model of a Quadrotor. The complete derivation of the kinematics and dynamics, along with an understanding of reference frames and coordinate systems, is provided. Forces and torques acting on the Quadrotor and their effects are interpreted.

4.1.1 Reference Frames

Axis systems provide a reference point or origin and a sense of positive displacement [56]. The use of several different reference systems is essential to make analysis simpler. The coordinate frames are transformed into one other through rotations and translations. Newton's equations of motion are given in the inertial frame as are GPS readings, future work like waypoint navigation etc. will also be specified in this frame. Sensors constituting the onboard IMU give readings with respect to the body frame. Forces and torques acting on the Quadrotor are also evaluated in the body frame. The following reference frames shown in Figure 39 are adopted:

1. The inertial frame, $F_i = (\vec{x}_i, \vec{y}_i, \vec{z}_i)$, is an Earth-fixed coordinate system with its origin located conveniently at the base station. It is the conventional North, East and Down (NED) frame.

2. The body frame, $F_b = (\vec{x}_b, \vec{y}_b, \vec{z}_b)$, has its origin at the COG of the Quadrotor with its x-axis pointing forward along motor 1, y-axis pointing out to right along motor 2 and z-axis pointing out the belly.

3. The vehicle-carried NED frame, $F_v = (\vec{x}_v, \vec{y}_v, \vec{z}_v)$, has its axes aligned with the inertial NED frame but has its origin at the COG of the Quadrotor.

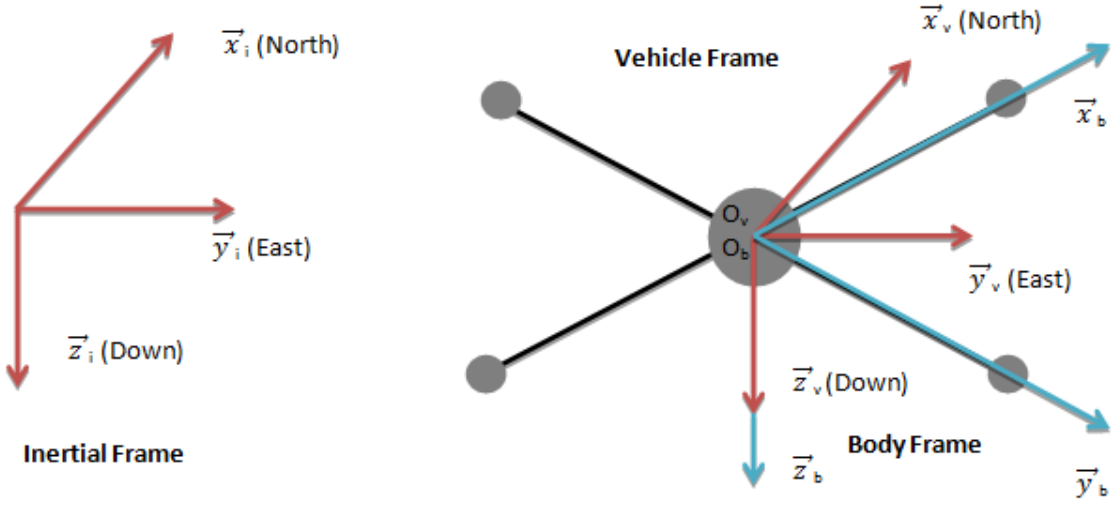

Fig. 39 Reference Frames

Each frame can be obtained from another by performing translations and rotations. The vehicle NED frame can be transformed into the inertial frame by undergoing a pure translation. The body frame can be obtained from the vehicle frame by a series of rotations in a specific order. Let us define two more intermediate frames to illustrate this, the frame $F_{v\text{-}1}$ is obtained by

rotating F_v positively about $\vec{z}_v$ by the yaw angle ψ. The frame F_{v-2} is obtained by rotating this F_{v-1} about the $\vec{y}_{v-1}$ by the pitch angle θ in a positive sense. Now if this frame is rotated about the $\vec{x}_{v-2}$ by the roll angle φ, we obtain the body frame. The rotation matrices are derived below.

The transformation from F_v to F_{v-1} can be written as

$$F_{v-1} = R_v^{v-1}(\psi) F_v \quad \text{, where} \qquad R_v^{v-1}(\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The transformation from F_{v-1} to F_{v-2} is given by

$$F_{v-2} = R_{v-1}^{v-2}(\theta) F_{v-1} \text{ , where} \qquad R_{v-1}^{v-2}(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$

And finally the transformation from F_{v-2} to F_b is given by

$$F_b = R_{v-2}^{b}(\phi) F_{v-2} \text{ , where} \qquad R_{v-2}^{b}(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}$$

Thus the complete rotation matrix from the vehicle frame F_v to the body frame F_b is written as:

$$R_v^b(\phi,\theta,\psi) = R_{v-2}^b(\phi) R_{v-1}^{v-2}(\theta) R_v^{v-1}(\psi)$$

$$= \begin{pmatrix} \mathrm{c}\theta\,\mathrm{c}\psi & \mathrm{c}\theta s\psi & -\mathrm{s}\theta \\ \mathrm{s}\phi\,\mathrm{s}\theta c\psi - c\phi s\psi & \mathrm{s}\phi s\theta s\psi + c\phi c\psi & s\phi c\theta \\ c\phi s\theta c\psi + s\phi s\psi & c\phi s\theta s\psi - s\phi c\psi & c\phi c\theta \end{pmatrix}$$

Where $c\theta = \cos\theta$ and $s\theta = \sin\theta$

4.1.2 Kinematics

The 6 DOF equations of motion (EOM) are more easily formulated in the body frame due to the fact that, the on board IMU measurements are in this frame, the control forces are given in this frame and the symmetry of the Quadrotor simplifies equations [41]. The Euler angles are defined in the vehicle frame F_v and its variations F_{v-1} and F_{v-2}. The following vector notation will be used at all times.

$$S_i = \begin{bmatrix} x_n \\ y_e \\ z_d \end{bmatrix} \qquad V_b = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad \omega_b = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \qquad \omega_v = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

The velocity in the various frames of reference can be related using the rotation matrix derived above as

$$\begin{bmatrix} \dot{x}_n \\ \dot{y}_e \\ \dot{z}_d \end{bmatrix}_{F_i} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_{F_v} = R_b^v(\phi,\theta,\psi) \begin{bmatrix} u \\ v \\ w \end{bmatrix}_{F_b} \quad \textbf{(1)}$$

Where, $R_b^v = \left[R_v^b\right]^{-1} = \left[R_v^b\right]^T$

The angular rates p, q and r are defined in the body frame whereas the Euler angles are defined in the various vehicle frames, specifically the yaw angle ψ is defined in F_v, the pitch angle θ in the F_{v-1} and the roll angle φ in the F_{v-2}. It is shown in [57] that

$$R_{v-2}^b\left(\dot{\phi}\right) = R_{v-1}^{v-2}\left(\dot{\theta}\right) = R_v^{v-1}\left(\dot{\psi}\right) = I$$

$$\begin{aligned} \begin{bmatrix} p \\ q \\ r \end{bmatrix} &= R_{v-2}^b\left(\dot{\phi}\right)\begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_{v-2}^b(\phi)R_{v-1}^{v-2}\left(\dot{\theta}\right)\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_{v-2}^b(\phi)R_{v-1}^{v-2}(\theta)R_v^{v-1}\left(\dot{\psi}\right)\begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \\ &= \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}\begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}\begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \quad \textbf{(2)} \\ &= \begin{pmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{pmatrix}\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \end{aligned}$$

Inverting this we get

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{pmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3}$$

4.1.3 Dynamics

Differentiation in a moving axis system is given by,

$$\frac{dS}{dt} = \dot{S} + \omega_b \times S \tag{4}$$

Where S is the position vector of a particle relative to the origin of the moving body frame and ω_b is the angular rate that the body frame is moving with respect to an inertial frame. Newton's laws of motion are only valid in inertial frames, thus we need the above relation to formulate our equations of motion in the body frame. Let F_{ext} be the force vector containing all external forces including gravity. Mass being constant, applying the above equation and the chain rule of differentiation, we can concisely write the linear EOM as,

$$F_{ext} = \frac{d\left(mV_b\right)}{dt} = m\dot{V}_b + \omega_b \times \left(mV_b\right) \tag{5}$$

Thus using our previously defined notation, this becomes

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{F_{ext}}{m} \tag{6}$$

Seckel [58] demonstrates how to develop the equations of angular motion. Let $M_{ext} = [L\ M\ N]^T$ be the moment vector containing all external moments developed, then the angular EOM can be written as,

$$M_{ext} = J\dot{\omega}_b + \omega_b \times \left(J\omega_b \right) \tag{7}$$

The Quadrotor is completely symmetrical about all its axes; this conveniently renders the cross terms in the inertia matrix J to be zero. Thus $J_{xy} = J_{yz} = J_{xz} = 0$ and our inertia matrix J is given by,

$$J = \begin{pmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{pmatrix} \qquad J^{-1} = \begin{pmatrix} \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_z} \end{pmatrix}$$

The moments of inertia are formulated according to [59] with the Quadrotor modeled as a spherical dense center of mass M_c and radius R_c and four point masses m_{mot} located at the arm length l.

$$J_x = \frac{2M_cR_c^2}{5} + 2l^2m_{mot} = 0.0296Kgm^2$$

$$J_y = \frac{2M_cR_c^2}{5} + 2l^2m_{mot} = 0.0296Kgm^2$$

$$J_z = \frac{2M_cR_c^2}{5} + 4l^2m_{mot} = 0.0566Kgm^2$$

Thus the angular EOM can be written as,

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} pq \end{bmatrix} + \begin{bmatrix} \frac{L}{J_x} \\ \frac{M}{J_y} \\ \frac{N}{J_z} \end{bmatrix}$$

(8)

4.1.4 Forces and Moments

The forces and moments developed are due to gravity and the thrust and torques developed by the four propellers.

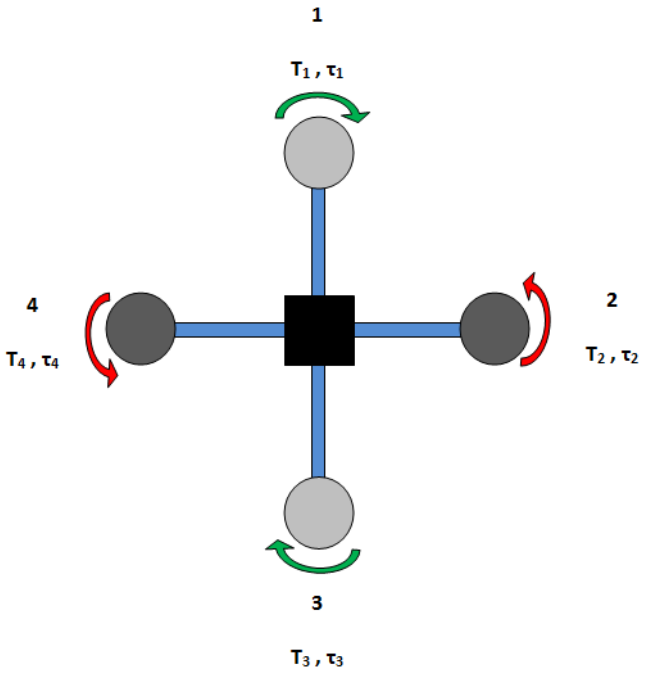

Fig. 40 Top View of Quadrotor

Figure 40 shows the top view of the Quadrotor and the forces developed by each motor. Figure 41 shows the forces and torques acting on the Quadrotor and positive sense of rotation for roll, pitch and yaw.

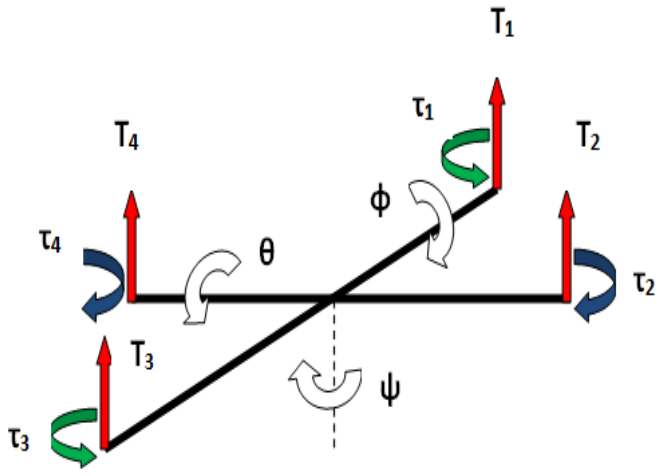

Fig. 41 Force and Moment Definitions

The total thrust force acting on the Quadrotor in the body Z axis is,

$$T = T_1 + T_2 + T_3 + T_4$$

The rolling torque L is produced by the thrust difference between motors 2 and 4 as,

$$L = l\left(T_4 - T_2\right)$$

The pitching torque M is produced by the thrust difference between motors 1 and 3 as,

$$M = l\left(T_1 - T_3\right)$$

The yawing torque N is produced by the total difference in clockwise and counterclockwise torques generated by all four motors as,

$$N = \tau_2 + \tau_4 - \tau_1 - \tau_3$$

F_{ext} thus has only a f_z component i.e. –T, but the gravitational force also needs to be given in the body frame,

$$\begin{aligned} F_{ext} &= R_v^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix} \\ &= \begin{bmatrix} -mg\sin\theta \\ mg\cos\theta\sin\phi \\ mg\cos\theta\cos\phi \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix} \end{aligned} \quad \textbf{(9)}$$

Thus equations (1)-(9) become,

$$\begin{bmatrix} \dot{x}_n \\ \dot{y}_e \\ -\dot{z}_d \end{bmatrix} = \begin{pmatrix} \mathrm{c}\theta\,\mathrm{c}\psi & \mathrm{s}\phi\,\mathrm{s}\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & \mathrm{s}\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ s\theta & -s\phi c\theta & -c\phi c\theta \end{pmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{10}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \begin{bmatrix} -g\sin\theta \\ g\cos\theta\sin\phi \\ g\cos\theta\cos\phi \end{bmatrix} + \frac{1}{m}\begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix} \tag{11}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{pmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{12}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} pq \end{bmatrix} + \begin{bmatrix} \frac{L}{J_x} \\ \frac{M}{J_y} \\ \frac{N}{J_z} \end{bmatrix} \tag{13}$$

Equations **(10) - (13)** provide the complete 6 DOF EOM in the body frame of reference

4.2 FLC Methodology

The intelligent control strategy proposed is presented in detail in this section. The Quadrotor was flown with a traditional TX-RX, utilizing the open source flight software available, to gain better insight into the dynamics of such an interesting UAV. The open loop model was run multiple times: dynamic responses were observed and tracked to achieve the heuristic understanding required before delving into the Fuzzy logic control process. The limits for the pitch and roll angles were derived via flight tests. Wireless telemetry also allowed the observation of typical acceleration and angular velocity values which helped in range setting of the FLC's (Fuzzy logic controller).

Fuzzy logic manipulates inputs using the heuristically developed rule base and converts them into outputs. An overview of the fuzzy control process is shown in Figure 42 below.

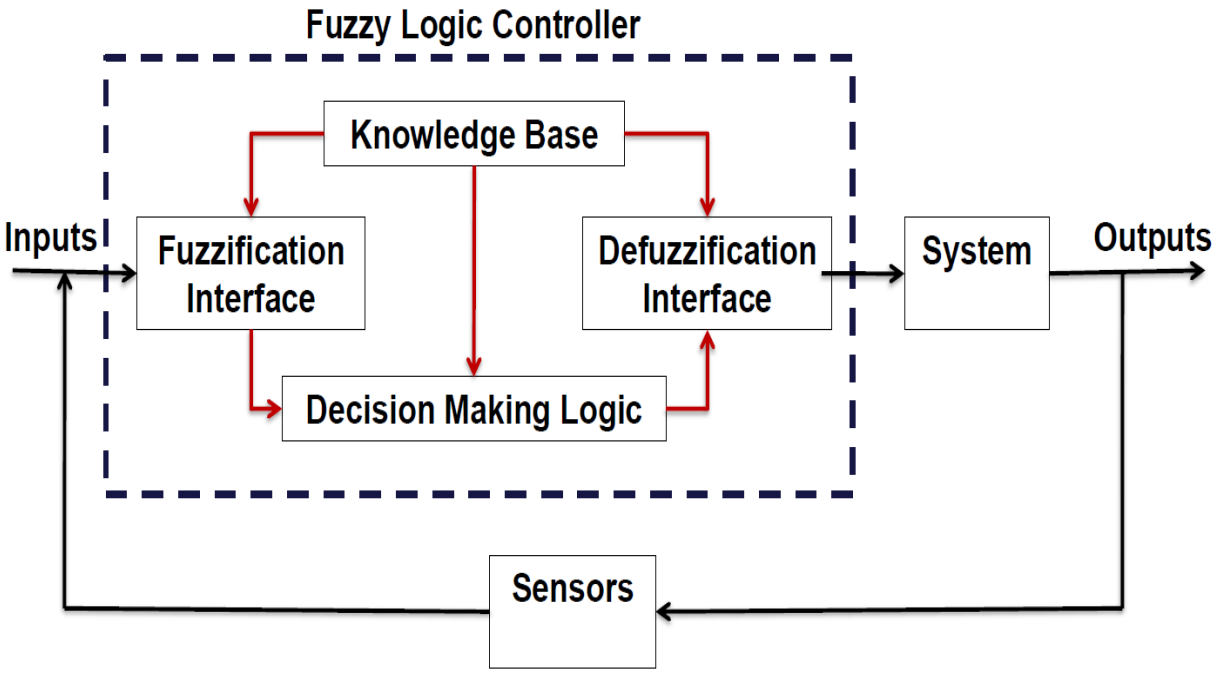

Fig. 42 Control Process Overview

The flight controller must perform two separate tasks simultaneously:

1. Stabilize the attitude of the Quadrotor.

2. Control the position of the Quadrotor by manipulating its attitude.

The mathematical model illuminates the fact that X and Y motions are coupled with pitching and rolling motions respectively. Thus in all six states namely X-Coordinate, Y-Coordinate, Z-Coordinate, heading angle (ψ), roll angle (φ) and pitch angle (θ) have to controlled simultaneously. Accordingly six FLC's have been designed; one FLC to control each state; the FLC X and FLC Y are cascaded with FLC θ and FLC φ respectively, whereas FLC Z and FLC ψ are standalone controllers. Each FLC has two inputs namely the error and error rate and one output which is the Δ PWM value. MATLAB's Fuzzy Logic toolbox was used to develop the controllers, which use the Mamdani-type inference method and the centroid method for defuzzification.

A predefined bias "offset" is used to counteract the weight of the Quadrotor. Translation in the +X direction is achieved by a pitch down i.e. nose down movement, hence PWM_1 = offset – uX and PWM_3 = offset + uX, while the other motors are unaffected i.e. PWM_2 = PWM_4 = offset. This keeps the net lift constant and the net yaw torque zeroed. Similarly ±Y translation is achieved by rolling right/left respectively, altering the speeds of motors 2 and 4, and keeping that of 1 and 3 constant.

Z translation is achieved by increasing/decreasing the speeds of all the motors by the same amount i.e. uZ. Control of the heading angle requires an unbalanced torque, this is manipulated by either increasing speeds of the clockwise motors (1, 3) and decreasing that of the counterclockwise motors (2, 4) by the same amount, or vice versa, thus producing an

anticlockwise or clockwise net moment. The desired roll and pitch angles are always fed as zero so as to maintain the stability of the craft, when a X or Y translation is desired, the FLC X or FLC Y anticipate the desired pitch and roll angles continuously and the FLC θ or FLC φ achieve these angles by manipulating the speeds of the motors as above. The roll and pitch angles anticipated have set limits and are chosen empirically in such a way so as to always stabilize the attitude of the craft. It should be noted that in this research effort, priority has been given to achieving quick stable responses and not to aggressive maneuvering.

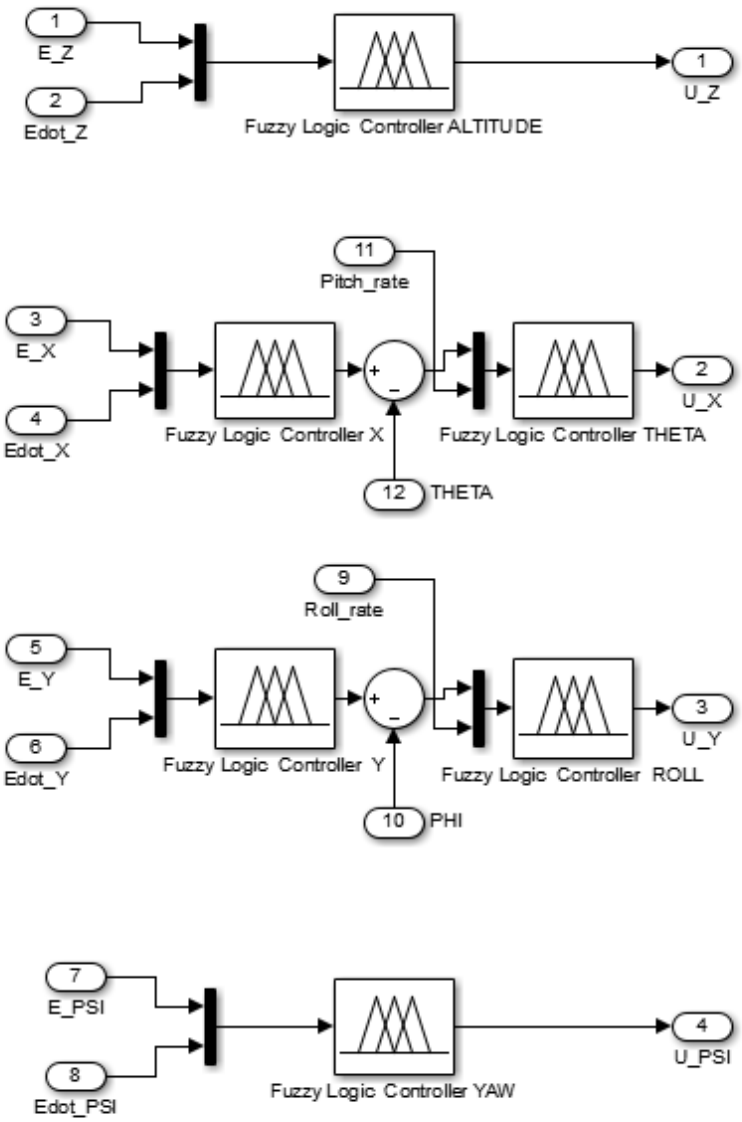

Fig. 43 Control Scheme Implementation

The control scheme presented in the Figure 43 above results directly from an understanding gained during initial flight tests, the logic described above and from the interpretation of coupling of motions by studying the formulated mathematical model.

4.3 Membership Functions and Rule Base

Each of the three attitude FLC's and the altitude FLC is setup in the following manner: Two inputs to the fuzzification interface i.e. the error and error rate of the particular state to be

controlled and one output which is the motor output control value. The two positional FLC's namely FLC X and FLC Y are setup slightly differently: Two inputs to the fuzzification interface i.e. the error and error rate of the X and Y state to be controlled and one output which are the required Pitch and Roll angle to achieve that position respectively. We shall see in the next simulation chapter just how the outputs are combined before being sent to the motor. For example, considering the FLC X positional controller, its inputs and output would b:

1. INPUT_1 error_X = Des_X – Actual_X
2. INPUT_2 error_rate = Actual_X_Velocity
3. OUTPUT Required Pitch angle θ

This output of the positional controller is then fed to the attitude controller thus closing the outer and inner loops. Thus say we give a positional command of X = 5 m, keeping other states as is, then the FLC X controller is called and outputs the required pitch angle which along with the current pitch rate is fed into the FLC θ which then outputs the motor command for a pitch moment. This approach is congruent to the existing PID control methodology used for autonomous control; inner loop controls attitude and outer loop controls position and feeds an error into the inner attitude loop.

The error input to all the FLCs consists of five membership functions and is normalized to the range [-1 1] shown in Figure 44:

1. Negative High
2. Negative Low
3. Zero

4. Positive Low

5. Positive High

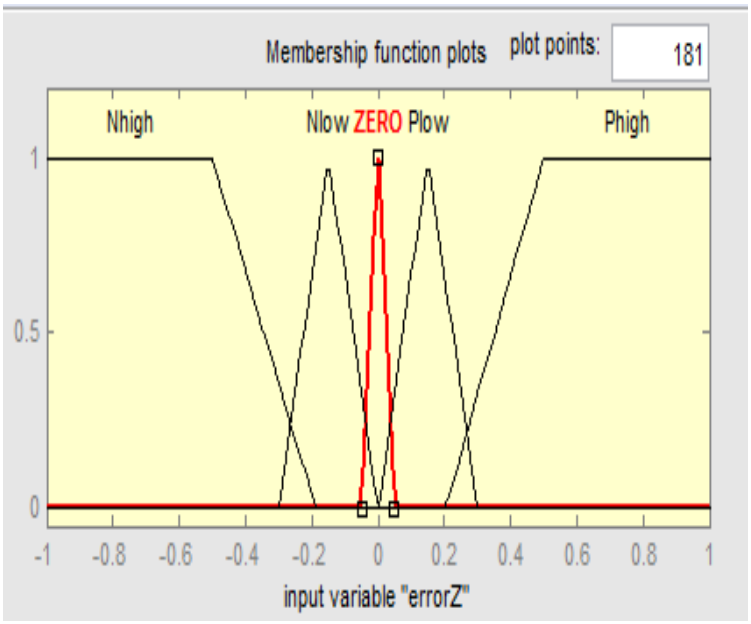

Fig. 44 Member ship Functions for Input "Error"

The error rate input for all the FLCs also consists of five membership functions. Physically, the error rate input for the attitude FLCs signifies the angular rates while the error rate input for the positional FLCs signifies the linear velocities.

Thus the range setting of these inputs represents the physical limits imposed on the system in terms of its linear and angular velocities. This range is set based on heuristic understanding and also on flight data collected that gives a basis for the range of achievable velocities and safe operation. Figure 45 shows the membership functions of the input "error_rate".

1. Negative Fast
2. Negative Slow
3. Zero
4. Positive Fast
5. Positive Slow

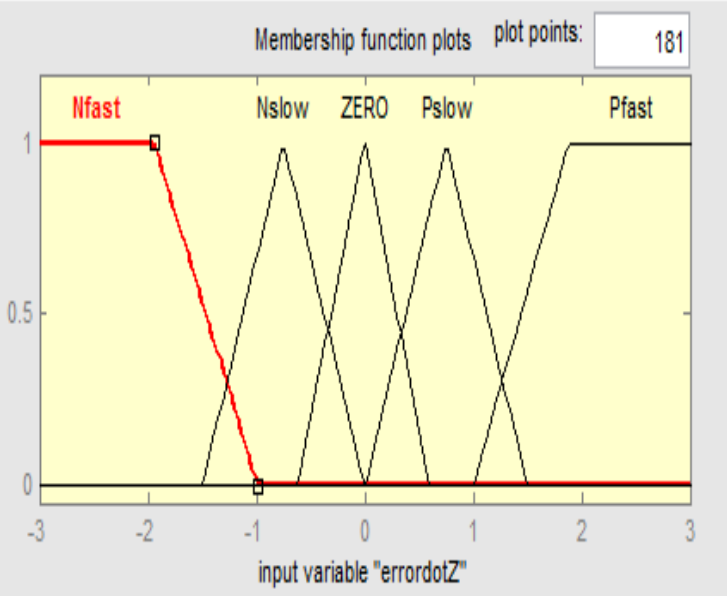

Fig. 45 Member ship Functions for Input "Error_Rate"

The output of each of the FLCs comprises of the following five membership functions. The range setting for the outputs of the FLCs was a very recursive and iterative effort. Enough control authority had to be given to each FLC, while keeping the control performance satisfactory. Figure 46 shows the output membership functions.

1. Negative Big

2. Negative Small

3. Zero

4. Positive Small

5. Positive Big

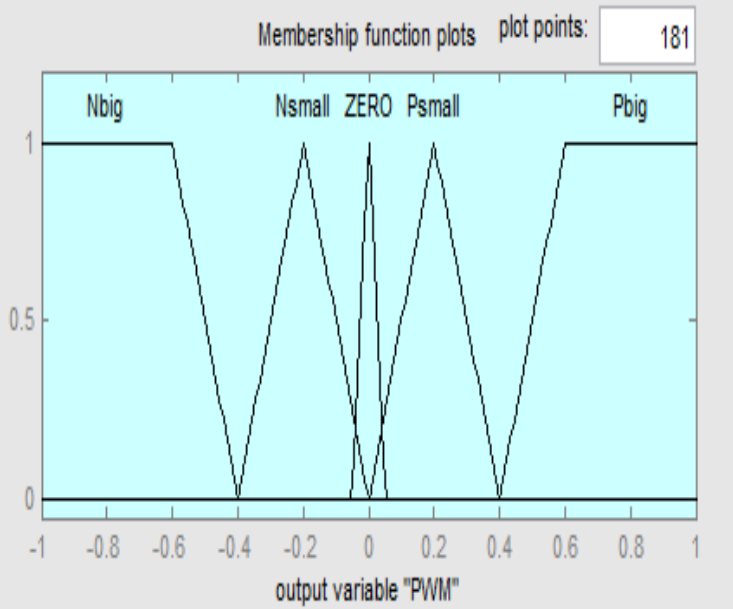

Fig. 46 Member ship Functions for Output "Δ PWM"

The control scheme implemented is meant to be used as a one input at one time control strategy i.e. perturbations in either X, Y, Z axis and heading are commanded one at a time. Having said this, the controller performed exceptionally well to two input at a time scenarios and these are illustrated in the chapter on simulations in detail. Altitude and heading are changed simultaneously and the controller was still able to achieve its goal. A simultaneous X and Y perturbation also gave excellent results in terms of control performance. The tuning of the membership functions and the rule base follows the flowchart depicted in Figure 47 below [50].

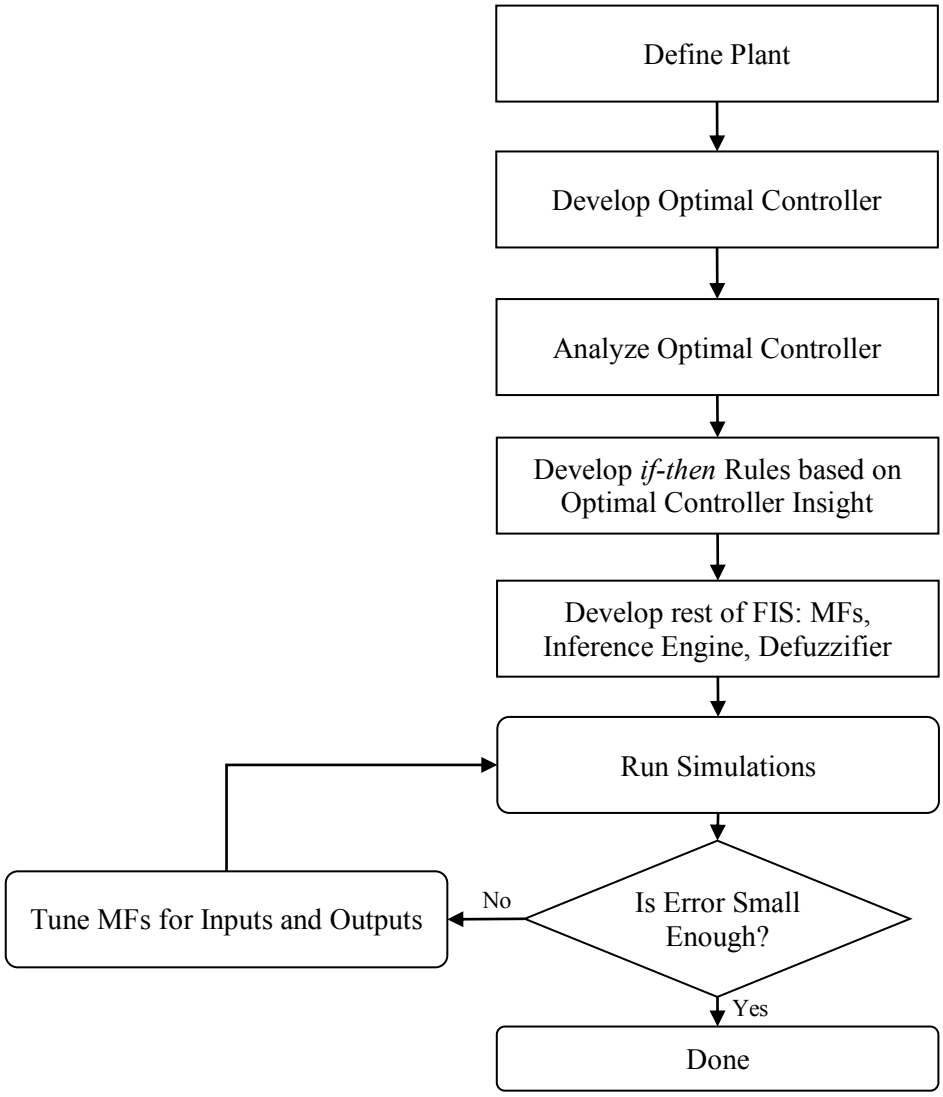

Fig. 47 Flowchart Process of Defining Rules and Tuning Membership Functions [50]

Rules are developed as if-then statements based on heuristics and experience. For example, **IF** errorZ is "Nhigh" and errordotZ is "Pfast" **THEN** PWM is "Nbig". The heuristic interpretation of this rule is as follows: if actual altitude is much higher than desired, thus error is negative high, and the error rate i.e. velocity in Z direction is positive fast meaning a high upward velocity, then the output is negative big, i.e. a sharp decrease in PWM value for each motor thereby causing quick deceleration. A total of 25 rules have been developed for each FLC. The complete rule base is presented in the table 5 below using the linguistic variables previously defined.

Table 5 Rule Base

E / Ė_dot	Nhigh	Nlow	Zero	Plow	Pfast
Pfast	Zero	Psmall	Psmall	Pbig	Pbig
Pslow	Nsmall	Zero	Psmall	Psmall	Pbig
Zero	Nbig	Nsmall	Zero	Psmall	Pbig
Nslow	Nbig	Nsmall	Nsmall	Zero	Psmall
Nfast	Nbig	Nbig	Nbig	Nsmall	Zero

4.4 Implementation Issues

Real time implementation of the controller on the actual Quadrotor requires solving a number of concerns. These include real time data acquisition and processing, timing of the control process so motor commands are sent at least at 10 millisecond intervals, data packet losses and interfacing between the onboard Arduino microcontroller and ground station software. The development of such autonomous vehicles is a time consuming process that requires a wide range of aerospace, computer engineering and control skill [60].

Three separate approaches to implement the controller were researched. The first approach involved writing code in C# that allowed communication and catching data packet loss. Simple though it might be to write and execute, bringing MATLAB's fuzzy logic control files of the .fis type into this environment posed a serious, time consuming problem.

Future interest in this subject matter dictates that this research be easily reproduced here at the University of Cincinnati, so as to enable future development with regards waypoint following algorithms and obstacle avoidance techniques by students. In light of these facts, the second approach studied and partially implemented was to use MATLAB Simulink's Real Time Windows Target to implement the controller. The advantages in using this toolbox are widespread student familiarity with MATLAB, academic availability, easy to use GUI's and added functionality using embedded function blocks.

Real Time Windows Target provides a real-time engine for executing Simulink models on a PC and blocks that connect to a range of I/O boards. It enables the creation and control of a real-time system for rapid prototyping [60]. It has two modes of execution, Normal mode and an external mode. The external mode option along with Simulink coder generates executable C code from the Simulink model developed and provides high performance, which is extremely necessary for our application. The need to develop C code from scratch is eliminated completely.

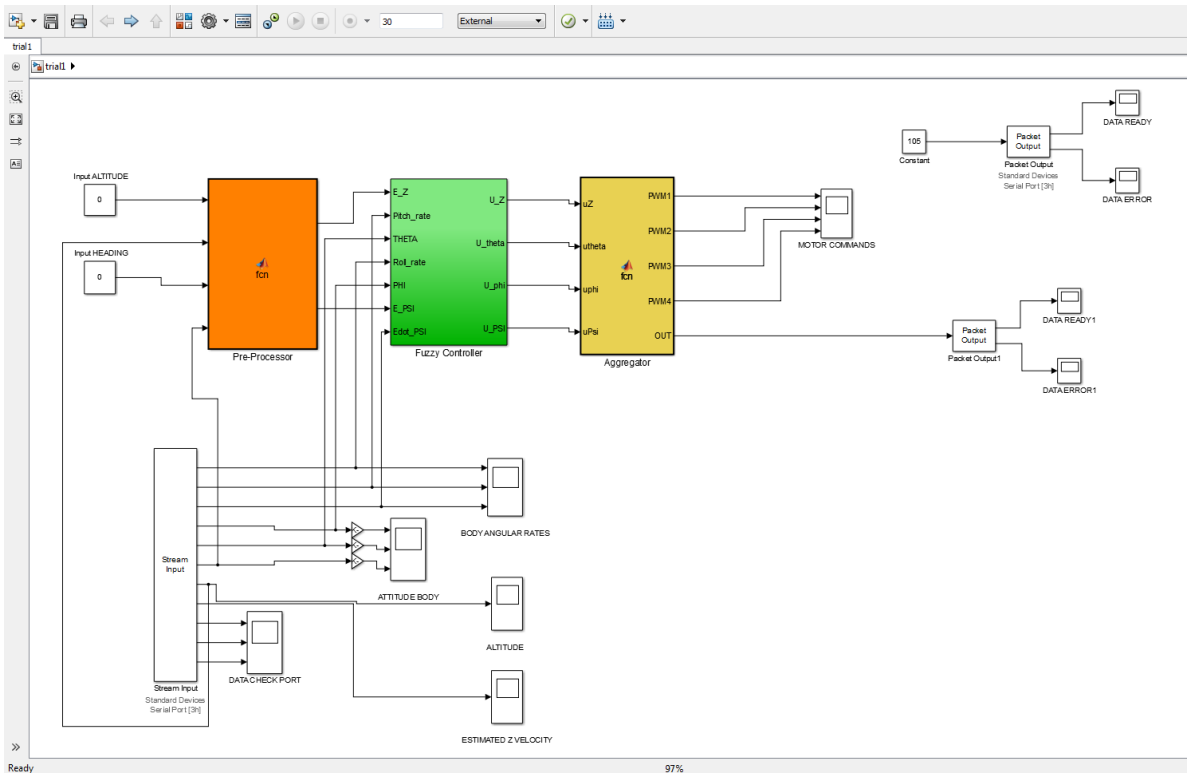

Fig. 48 MATLAB Simulink Real Time Data Acquisition and Partial Control Model

Figure 48 above shows the real time model developed in Simulink and the associated blocks used for communication. Two *Packet output* blocks are used to send binary data to the Quadrotor, one to send the binary equivalent of 'i' i.e. 105, so as to start the serial telemetry feature. The other to send motor commands (untested). A *Stream Input* block is used to read the formatted ASCII data. The parameters read are roll rate, pitch rate, yaw rate, estimated roll, pitch angles, heading, barometer altitude and estimated Z velocity.

After configuring the blocks to allow MATLAB to assert RTS flow control (XBee default communication), we build the model incrementally and then connect to the target hardware i.e. Arduino. We can now specify the simulation time (fixed time step discrete) and

start our real time data acquisition. In addition, every I/O block has data error, data ready and 'Missed Ticks' ports, so that we may keep tabs on the status of the communication.

Using this engine successfully required a number of changes to the software side on the Arduino. The open source flight software uses DCM [53] techniques to fuse sensor data into an estimation of the attitude of the craft. Serial telemetry data types had to be changed in order to allow Simulink access. The string delimiter had to be changed from the 'comma' default. Baud rate of '111111' was used to allow for high performance yet avoid data loss. Furthermore, the model had to be run multiple times to ensure correct and consistent data collection. Real time synchronization block was added to facilitate control loop timings.

The PWM command values that are generated will be sent back to the Quadrotor in binary data packets so as to speed up the update time. The low confidence exhibited; evidenced by data packet losses and partial string decomposition; by the real time model did not permit actual testing on the Quadrotor, however data acquisition was run successfully during a RC operated flight.

While this approach was a partial success, the important conclusion taken away from this was that only some form of embedded control would be practical. There is too much latency introduced into the system by the above approach and data packet loss would cause skipped motor commands that would lead to huge instability. Synchronization problems come into effect due to increased latency. The controllers must be embedded within the microcontroller and timing of the control loops must be internal with the ground station only tasked with receiving flight data and sending high level commands like waypoints etc.

This approach has been researched and found to be most applicable; incorporating the .fis control files into the Arduino itself and send commands via MATLAB. Keeping the manual override of the transmitter in place, this methodology however is expected to cause necessary programming downtime and the need for proficiency in the Arduino IDE C like language. Time did not permit the implementation of this approach, however the Arduino code developed for the controllers is reproduced in the appendix.

CHAPTER 5: SIMULATION RESULTS

This chapter highlights the simulation model built in the MATLAB/Simulink environment and describes the different modules of the simulator. The simulation results are discussed in detail. Experimental validation of the simulator was carried out using flight data which was recorded during flights achieved with the COTS flight controller APM 2.6. High coherence between the simulation and flight data is seen for near hover flight regimes. A large number of open loop simulations were performed initially to better understand the quantitative effect of changing PWM values and how they affect the dynamic states of the Quadrotor. Subsequently each loop was iteratively closed starting with the inner attitude loops, then the altitude controller loop was closed and then finally the X and Y position controller loops were closed. Fine tuning the parameters of the FLC's is a time consuming recursive process involving a lot of back and forth tuning as closing any subsequent loops meant that the control authority is redistributed each time and it took several hundred simulations before all the loops showed optimal tuning. An indication of the excellent tuning achieved is the ability of the controller to handle tasks over and above its intended operation such as coupled inputs given simultaneously.

5.1 Quadrotor Simulink Model

The closed loop controller and the Quadrotor simulator have been implemented in MATLAB Simulink. The inputs to the Quadrotor block are the four motor PWM values, it outputs the linear and angular accelerations that are twice integrated to obtain linear and angular velocities and positions. The EOM are embedded in the Quadrotor block of the simulator. Figure 49 below shows the Quadrotor Simulink model.

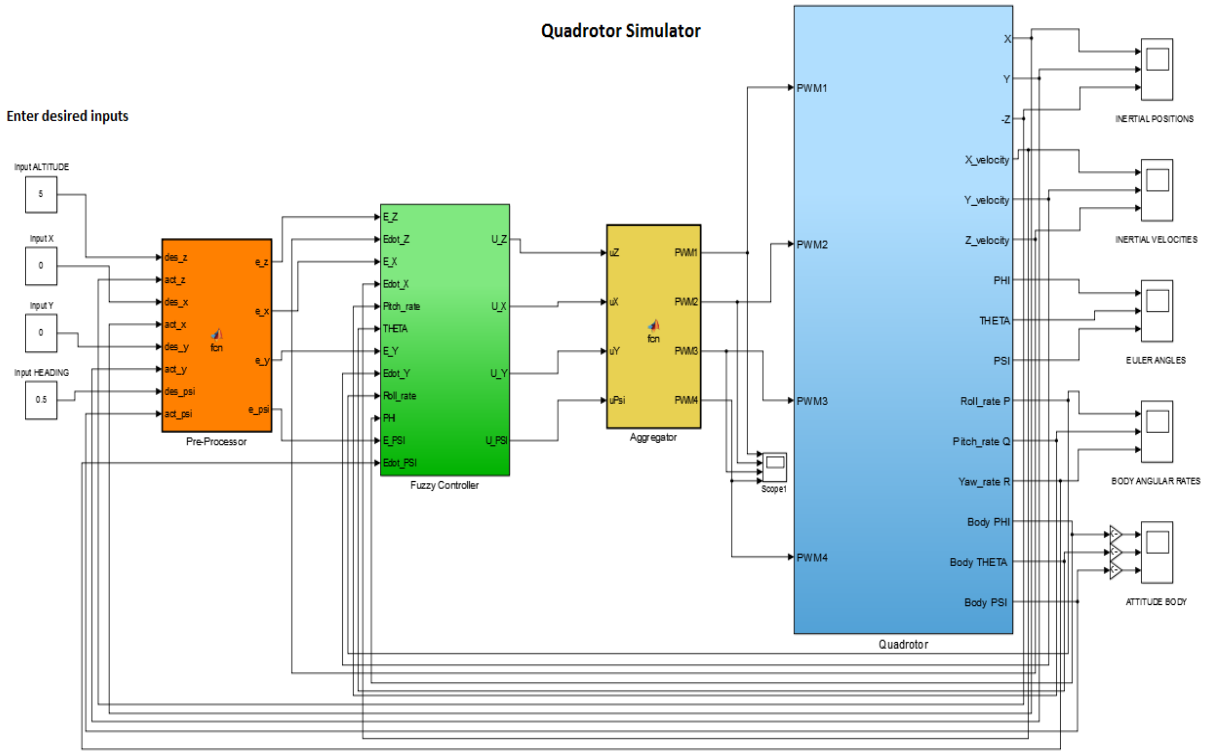

Fig. 49 Quadrotor Simulink Model

The model can effectively be broken into four major blocks. They are as follows:

1. Pre-processor Block
2. Fuzzy Logic Controller Block
3. Aggregator Block
4. Quadrotor Block

The Pre-processor receives the user defined inputs of altitude, X coordinate, Y coordinate and heading angle and calculates, normalizes and outputs the errors to the controller block. The normalized error value that is sent to the fuzzy controller block is in the range [-1 1] and negative altitude commands are not permitted and throws an error warning.

The controller block contains the six FLC's and takes as its inputs the errors and the error rates of all the states and outputs the control variables Uz, uX, uY and uPSI. The control scheme shown above in the methodology chapter Figure 43 is implemented in this block. The altitude and heading FLCs are standalone controllers while the FLC X and FLC θ are cascaded i.e. the output of the FLC X drives the input for the FLC θ, similarly the FLC Y controller is cascaded with the FLC φ controller.

The aggregator block fuses all the control actions into individual PWM motor values as shown below. This logic represents the positive sense of displacement and rotation and how the controller's outputs combine to drive the Quadrotor to a particular position and attitude in space.

$$PWM_1 = offset + uZ - uX + uPSI$$
$$PWM_2 = offset + uZ - uY - uPSI$$
$$PWM_3 = offset + uZ + uX + uPSI$$
$$PWM_4 = offset + uZ + uY - uPSI$$

The Quadrotor block consists of the embedded 6 DOF EOM that were derived in Chapter 3. The block outputs are the linear positions and velocities and the attitude and attitude rates.

5.2 Experimental Validation

The validation of the math model developed is of prime importance to this research endeavor. It is clear that the simulation model must be validated with actual flight data to lend high confidence to the results obtained for the fuzzy logic controller developed. A specific methodology was derived in order to go about the validation process in a comprehensive manner.

With implementation efforts for the fuzzy logic controller still in early stages, it was decided to conduct the validation with the already functional PID based stock controller which was part of the open source APM 2.6 flight controller board. Code was written in the Arduino IDE environment to allow for the logging of the controlled motor outputs at 50 Hz for post processing. This development work took significant investment of time to achieve clean coherent data that could be imported into MATLAB & Simulink as an input to the Quadrotor Simulink model.

The validation model presented below in Figure 50 shows a MATLAB time series data block that feeds the logged motor PWM signals into the Quadrotor dynamic model. The mask over the Quadrotor dynamics' block from Figure 49 (Blue block 'Quadrotor') has been lifted in the above figure in the interest of clarity and supporting future researchers. The validation model consists of five distinct blocks, namely the Timeseries data block mentioned above, the flight dynamics block that converts the motor PWM inputs into forces and moments in the body frame, the 6 DOF quaternion block that preexists in Simulink, the three axis inertial measurement unit block that mimics the IMU onboard the APM 2.6 for accelerometer and gyroscopic data (courtesy DPI) and lastly the atmosphere model block that incorporates environmental effects. The model is an outdoor simulation model that takes into account air density, temperature and pressure. Modeling of wind effects was not done primarily to keep complexities out of the model and ensure flight test data is free from external forces as far as possible.

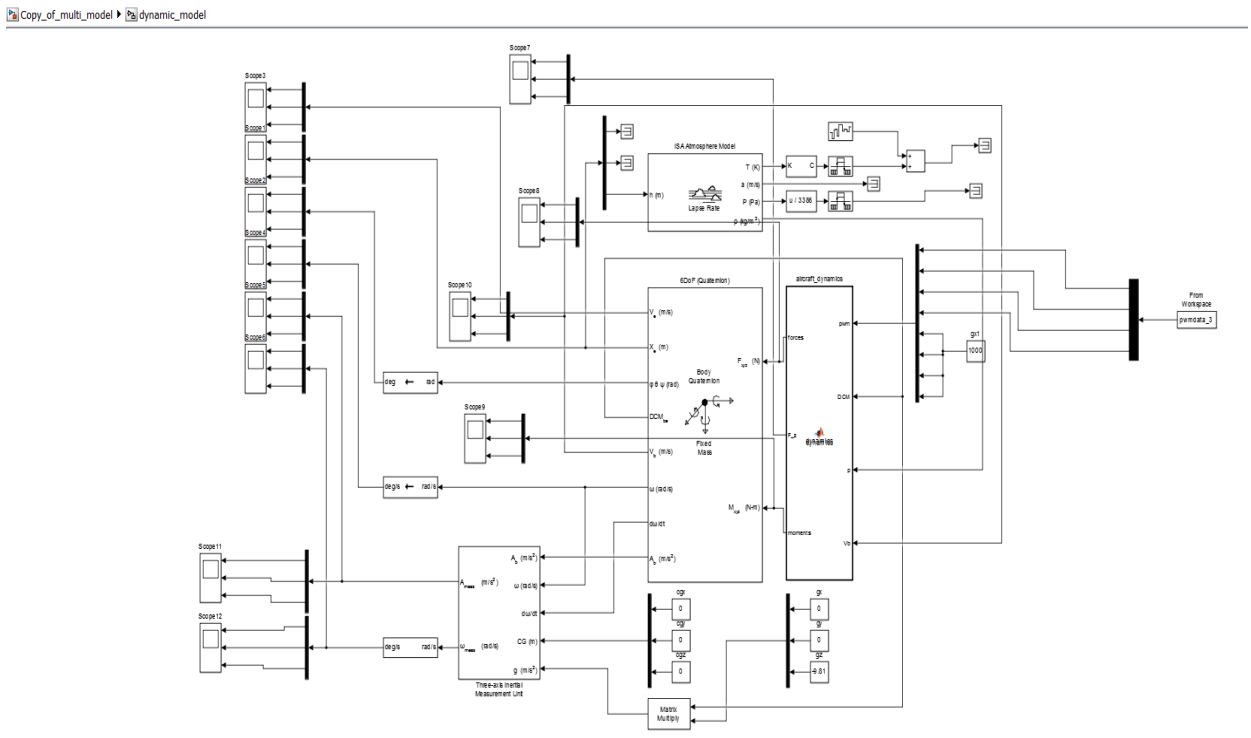

Fig. 50 Validation Simulink Model

An experimental flight of about one minute was conducted for validation purposes and the flight envelope consisted of non-aggressive maneuvers and stable hovering with pure altitude perturbations. Non aggressive maneuvering here refers to not allowing sustained velocity to build up, i.e. sharp short roll and pitch maneuvers as the model is validated near hover conditions. The motor inputs that were logged are included in the appendix. Roll and pitch i.e. movements from the X-Y plane were performed separately and not overlapped in order to identify and correct the model for discrepancies.

It should be mentioned here that fairly accurate results have been obtained while keeping in mind certain factors such as the data logging could only be accomplished at 50 Hz for attitude as well as motor output simultaneously, but the actual controller updates the motor commands at

100 Hz, thus causing loss of significant number of data points. This is the reason why the validation for now cannot be performed for aggressive flight conditions. Also the PID controller employed cannot be simulated accurately as the internal workings of the APM 2.6 remain unidentifiable. Thus while it would make more sense to simulate using the pilot's joystick inputs i.e. throttle, aileron, elevator and rudder, this is not possible at this moment in the development cycle. It is hoped that future validation work would follow the above thought process.

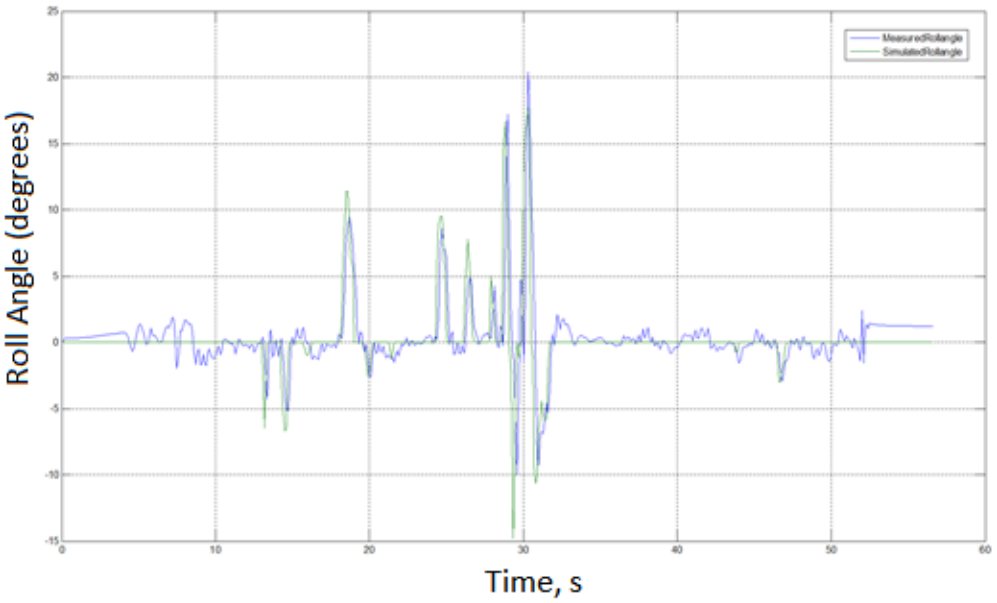

Fig. 51 Measured roll angle (phi_flight) vs. Simulated roll angle (phi_sim)

Figure 51 shows the measured and simulated roll angles in degrees vs. time in seconds. It is seen quite clearly that the simulation follows both the trend and the magnitude of the flight roll angles. The small roll angles do not show as much coherence as larger angles simply because the IMU data is quite jittery and noise inclusive and the moments produced under a particular value are neglected by the simulator for accuracy reasons. However, the simulator captures the flight dynamics to a very high extent. Figure 52 below shows the measured pitch angle vs. the simulated pitch angle.

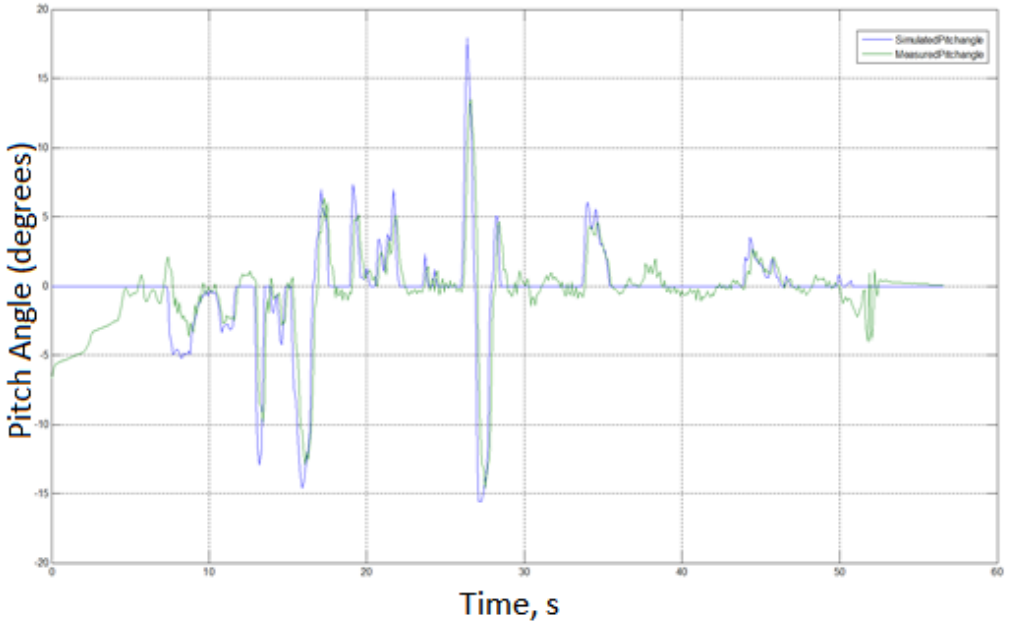

Fig. 52 Measured pitch angle (theta_flight) vs. Simulated pitch angle (theta_sim)

Similar to the roll angle tracking, it is observed that a significantly high coherence exists between the actual flight attitude data and the simulated body angles.

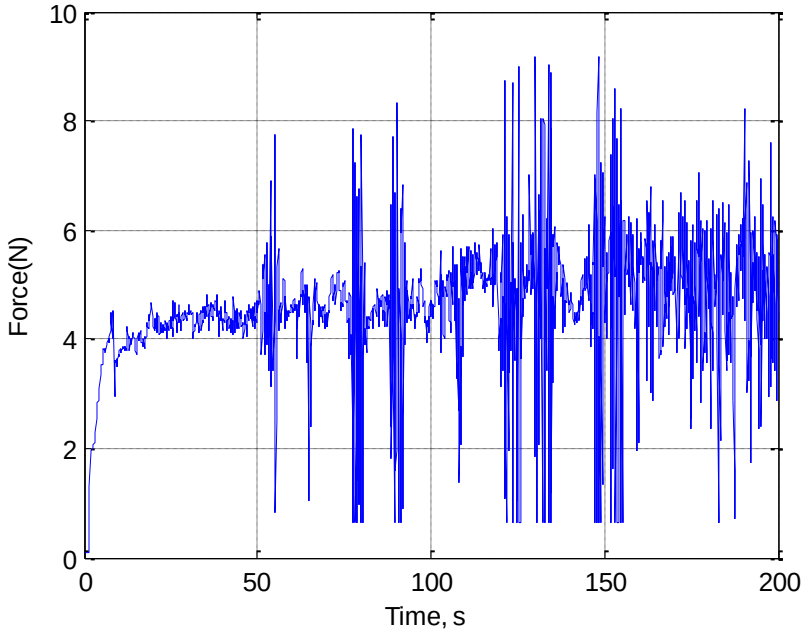

Fig. 53 Plot of forces developed by Rotor 1 for the validation flight example

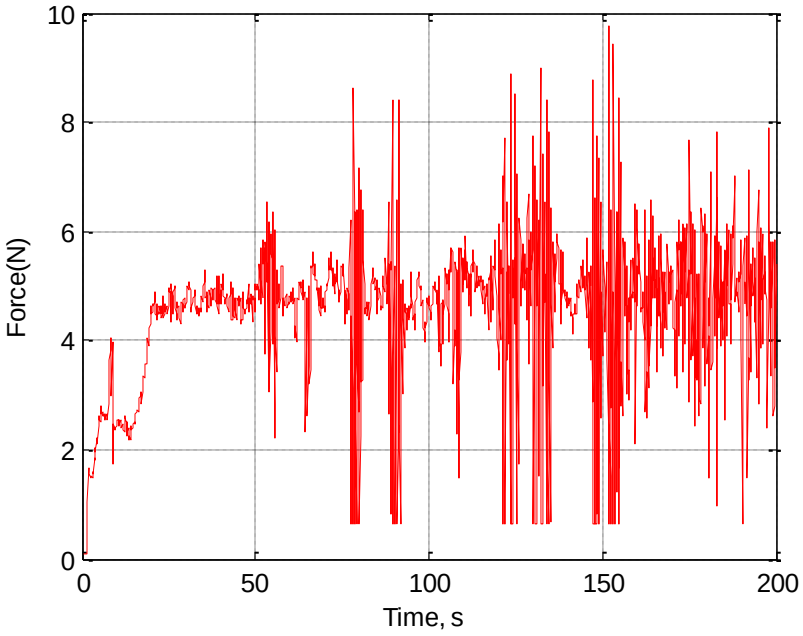

Fig. 54 Plot of forces developed by Rotor 2 for the validation flight example

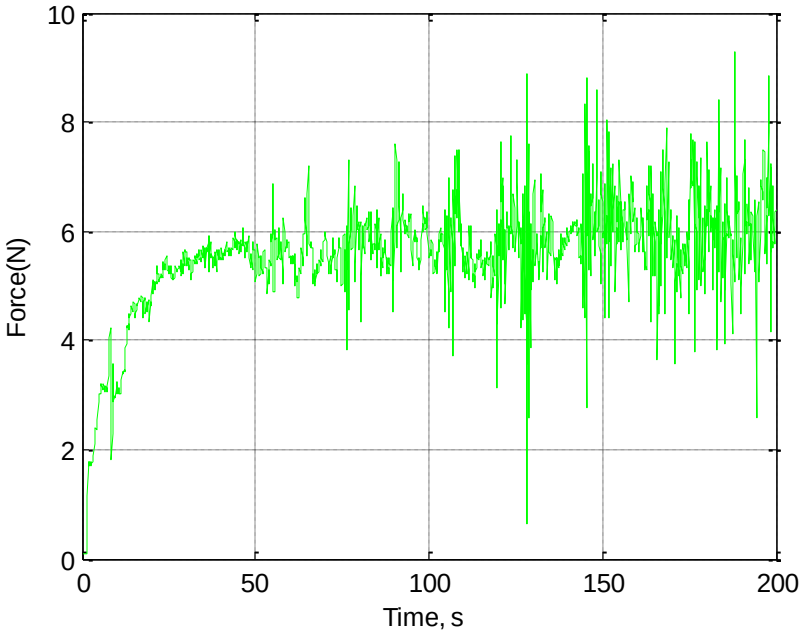

Fig. 55 Plot of forces developed by Rotor 3 for the validation flight example

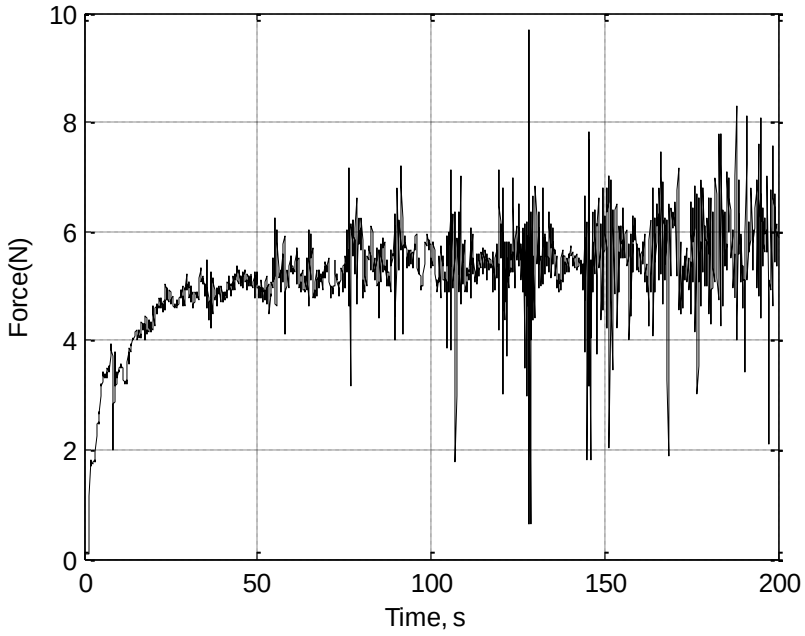

Fig. 56 Plot of forces developed by Rotor 4 for the validation flight example

Figures 53, 54, 55 and 56 show plots for the rotor forces that are developed for the above flight. The rotor forces developed are between magnitudes of 0 – 7 N which are typically expected, the hover rotor force for each rotor is experimentally seen to be 4.75 N. Some considerations were made in order to account for the lower frequency of data logging that proved invaluable in cutting off low degree oscillatory behavior. It should be noted here that comparing simulations to flight test data and extracting meaningful results is a very time intensive and recursive effort.

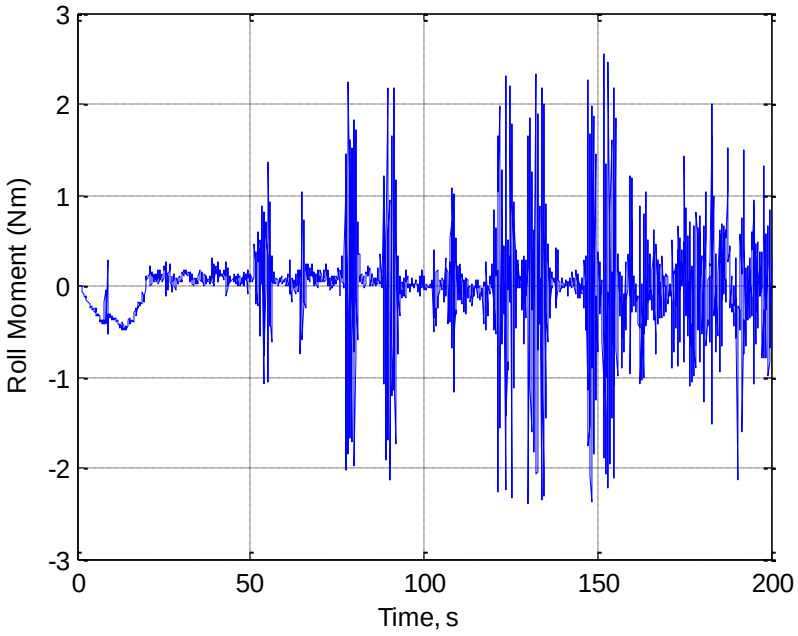

Fig. 57 Plot of Roll moment for the validation flight example

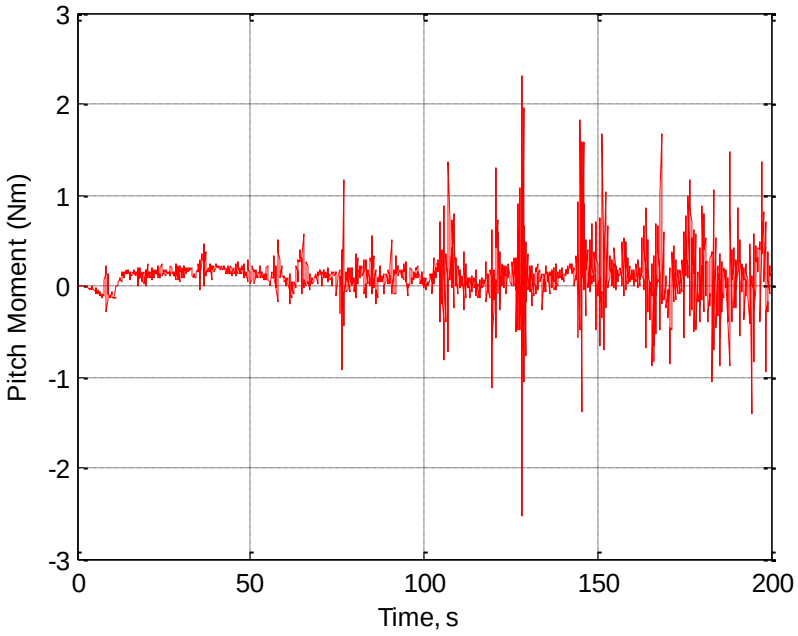

Fig. 58 Plot of Pitch moment for the validation flight example

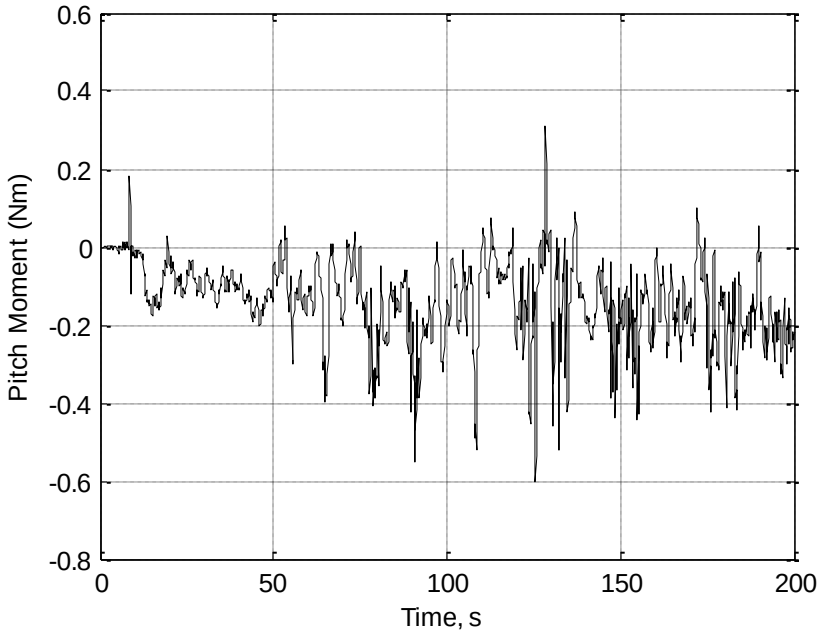

Fig. 59 Plot of Yaw moment for the validation flight example

Figures 57, 58 and 59 show the moments generated in the body frame of reference for the flight example above. The plot of the moments developed in each axis shows highly oscillatory behavior due to the fact that the roll and pitch angles computed form the IMU data by the flight controller are rarely stable, even non zero when not in actual flight and thus the controller always produces corrective action. This results in the actual PWM motor outputs to vary a lot even in stationary hover conditions. To account for this, a cutoff moment value is set so as to only capture slightly larger oscillations, in the range of 4-40 degrees in the roll and pitch axes. This is why the simulator is able to appreciably capture the real dynamics of the system. Moments in the yaw axis were developed due to the varying motor outputs but the offset set for this axis more or less zeroed out the final effects and this is in accordance to what was observed during the actual flight test.

5.3 FLC Autonomous Control

This subsection illuminates the results obtained for the FLC autonomous controller. The controller is intended to be utilized such we can give successive positional and heading commands in order to carry out a desired trajectory in real time i.e. for example rising to an altitude of 10 m and then traversing a square trajectory of side 5 m. Future work will include a GUI in MATLAB where top level commands may be sent to the Quadrotor. The response of an altitude command of 10 m is shown in Figure 60. It can be seen that the roll and pitch angles are within 2-3 degrees and are stabilized fairly quickly (see Figure 61). The system reaches its intended position within 8 seconds.

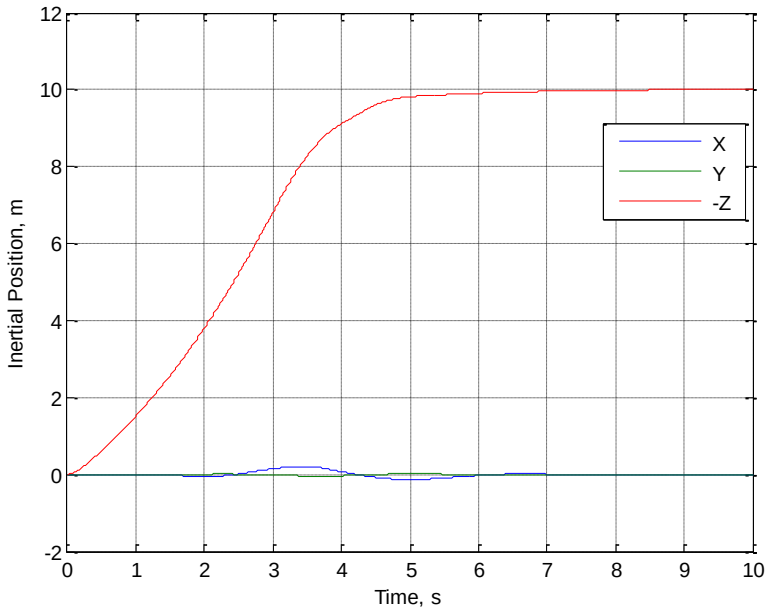

Fig. 60 Commanded Altitude of 10 m

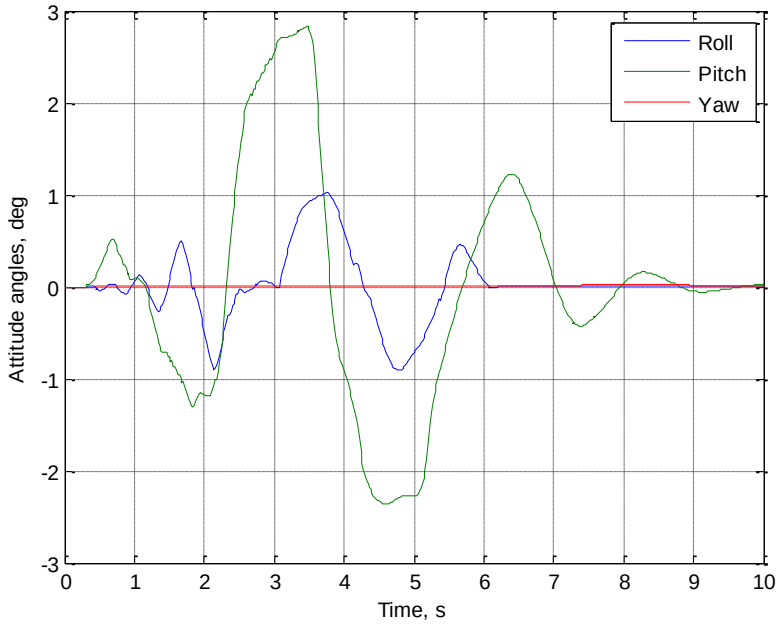

Fig. 61 Attitude of the Quadrotor for Commanded Altitude

Figure 62 shows the response to an X coordinate perturbation of 5m and the body angle plot is depicted in Figure 63. A quick stable response is obtained in under 5 seconds, while importantly not producing unwanted motion in other axes. The small non zero Z position value is indicative of a hover position in space at the datum line. Compared to experimental results with the off the shelf PID based flight controller, where large altitude (1-2 feet) perturbations are the norm when moving in any other axes, the performance is excellent.

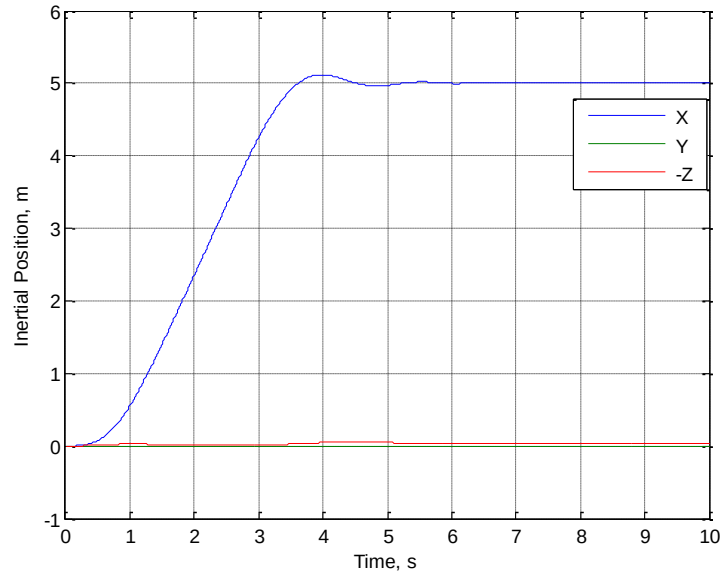

Fig. 62 Commanded X Position from 0 to +5 m

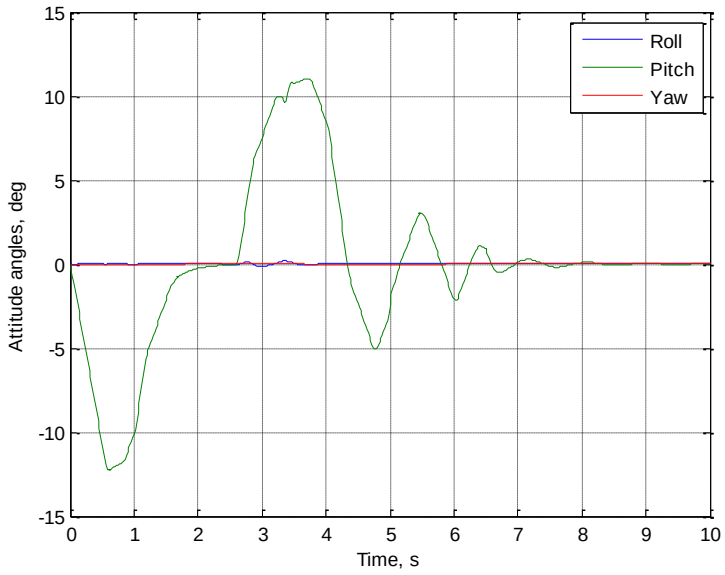

Fig. 63 Body Angle Plot for Commanded X Position from 0 to +5 m

The body angle plot above shows the smooth nature of the pitch angle commanded, this was especially difficult to achieve due to the fact that there is no tangible connection between the top level position controller and the low level attitude controller output.

There is no equivalent control surface plot between the above mentioned controller action to analyze and it required the trial and error method to find an acceptable solution. This is where fuzzy logic control suffers from non-transferability of the control engineer's experience. Having said that, once this heuristic understanding was gained, it was a fairly easy process to reproduce for a Hexrotor UAV which has similar rigid body dynamics. This is an ongoing project at DPI to build an autonomous velocity based waypoint navigation Hexrotor UAV of which the author is a part.

The next simulation is conducted specifically to overburden the FLC controller and view the detrimental control response. Multiple commands are introduced in three states simultaneously namely, a X position perturbation to -5 m, a Y position perturbation to +5 m and a heading angle perturbation of +30 degrees.

Physically, this represents the manipulation of all three attitude angles simultaneously while also controlling the linear position of the Quadrotor. The inherent robustness that fuzzy logic control embodies is on display in the results below. While a slight degradation is noticed in the heading angle control, the attitude and position of the Quadrotor is perfectly controlled to the desired commands. While the attitude response is not as smooth as the previous run, it is an extremely good performance considering the controllers have been over tasked. The motor control saturation limits are still well within bounds and the response is achieved in about 6 seconds. Figures 64 and 65 show the above scenario simulated response.

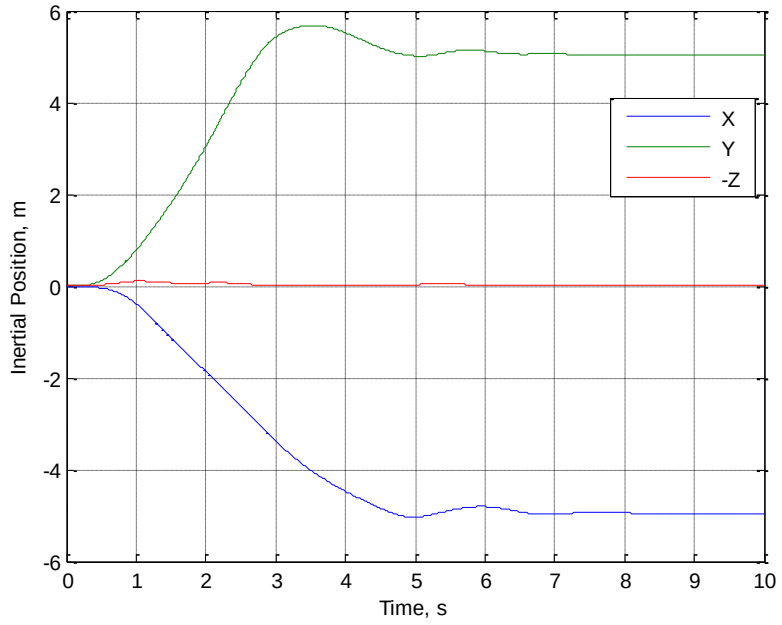

Fig. 64 Commanded X 0 to -5 m, Y 0 to +5 m and Heading 0° to 30°

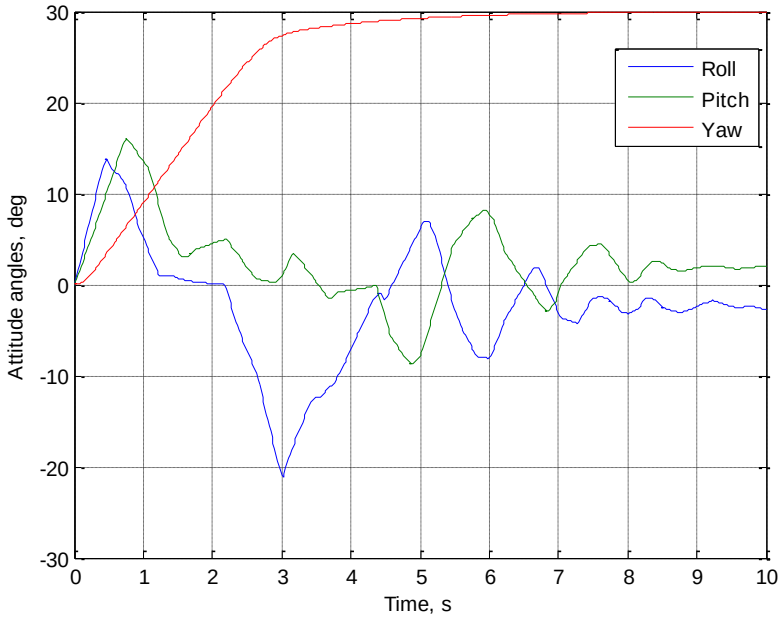

Fig. 65 Body Angle Plot for Commanded X 0 to -5 m, Y 0 to +5 m and Heading 0° to 30°

5.4 FLC vs. PID (Attitude Control)

This subsection deals with the comparison of the Fuzzy logic controller against the PID controller that is embedded in the off the shelf flight controller i.e. the APM 2.6. The PID gains for the Quadrotor were derived via flight tests according to the procedure established in the PID control tuning subsection. For this to be an accurate as well as real world comparison, it became necessary to decode the entire open source flight software to develop an exact representation of the flight software. Several new modules such as a sensor fusion block, motor mixer and ISA atmosphere block were developed to add to the PID simulation model.

The simulation model is split into the control and dynamics part so that tweaks to the control system could be done faster by opening not the entire simulation model but just the required control module. Figure 66 below shows the PID control architecture that is an exact representation of the flight software for the APM 2.6.

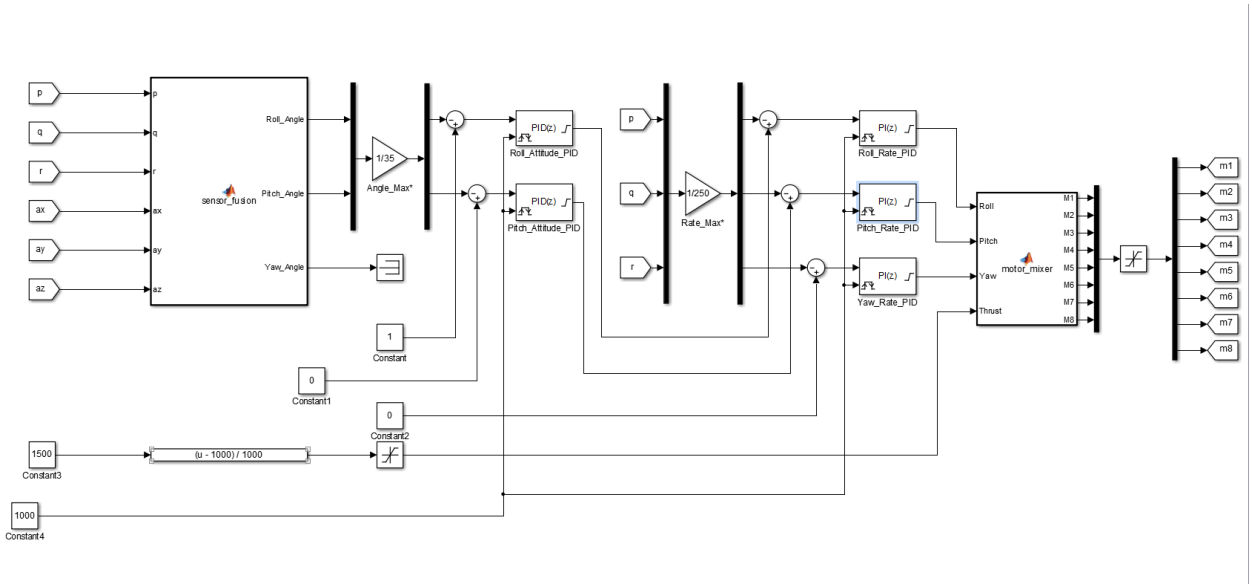

Fig. 66 PID Control Architecture of APM 2.6

The sensor fusion block has as its inputs the IMU sensor information i.e. body angular rates from the 3-axis gyroscope and body linear accelerations from the 3-axis accelerometer. The

outputs of this block are the Euler angles computed according to the DCM implementation of the APM 2.6. The roll and pitch angles have a 38° saturation limit and the yaw is controlled as a desired yaw rate rather than an absolute angle. As mentioned in the Literature review of the APM 2.6, the PID control features a cascaded loop structure with an inner rate loop and outer attitude loop. This is a representation of the "Stabilize" mode of the APM 2.6 which was used for all experimental validation efforts.

The PID and FLC are compared in the following two cases, where in case 1 a roll angle is commanded and the response is compared; in case 2 a pitch angle is commanded and the response is analyzed. It was not possible to compare the yaw performance due to the fact that the PID and FLC controllers are setup differently for this state, the PID control request a yaw rate and the FLC expects a commanded yaw angle.

1. Case 1 : Roll angle (phi) = 38°, Pitch angle (theta) = 0° , Yaw angle (psi) = 0°, Yaw Rate = 0 deg/s

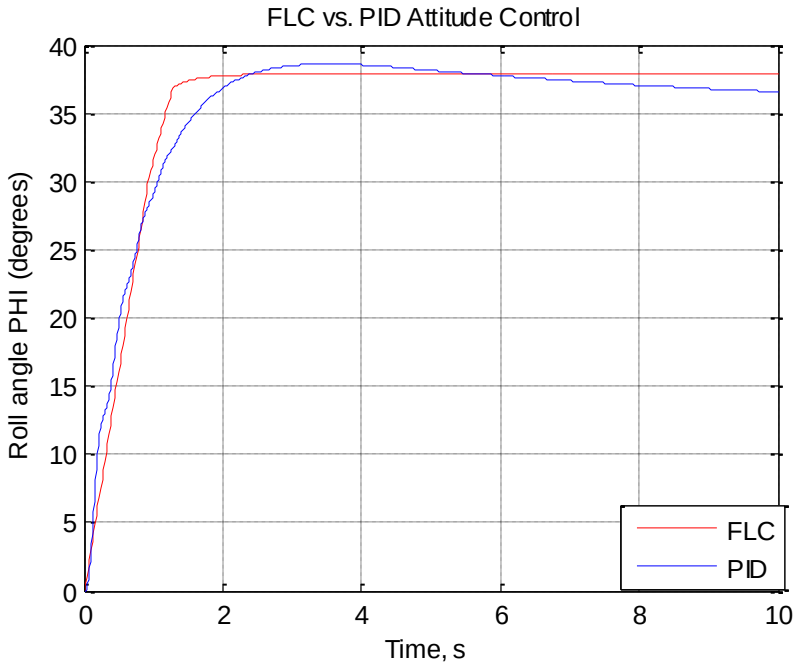

Fig. 67 FLC vs. PID Attitude Control for Commanded Roll Angle

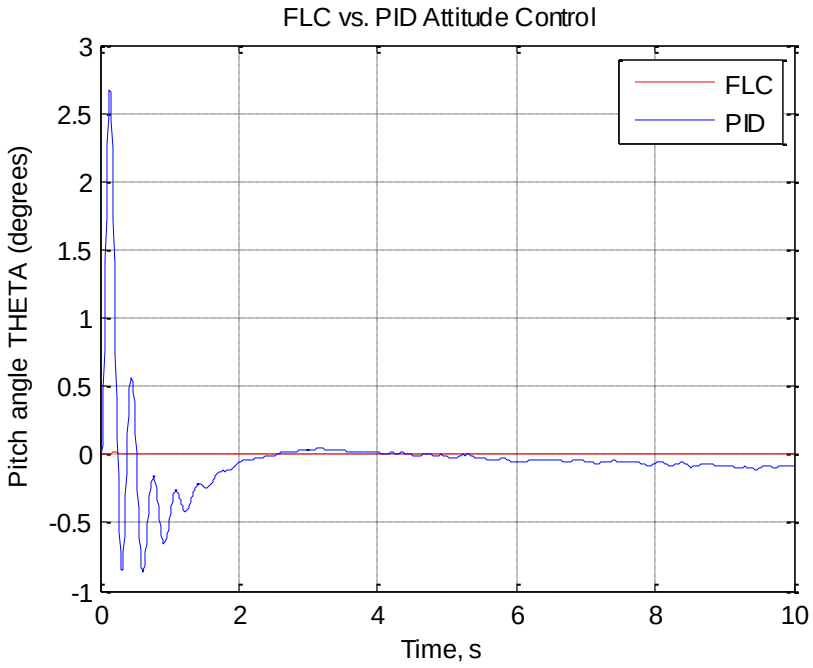

Fig. 68 FLC vs. PID Attitude Control Stabilization for Pitch during Roll Command

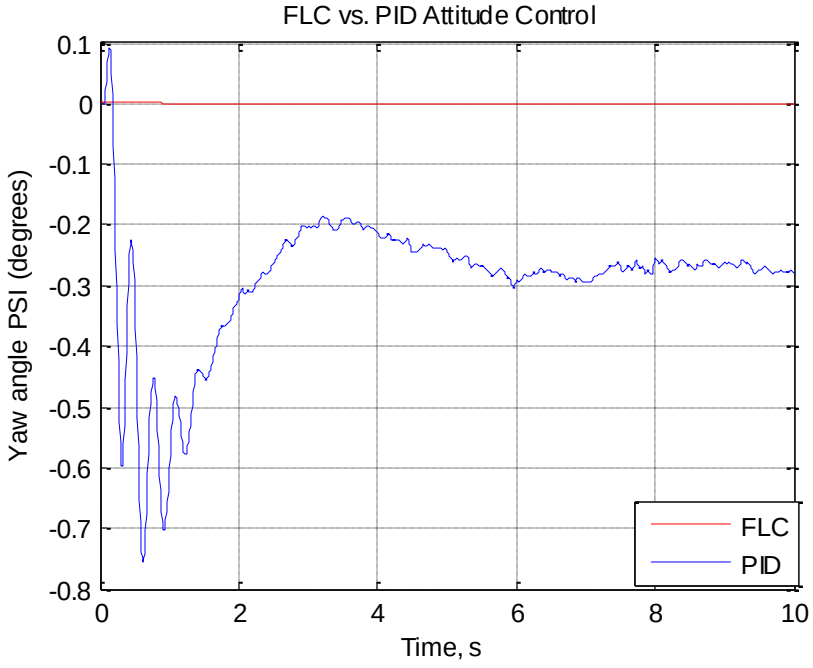

Fig. 69 FLC vs. PID Attitude Control Stabilization for Yaw during Roll Command

Figure 67 shows the roll response for a commanded roll angle of 38°, it is clear that the FLC outperforms the PID controller in both how quickly the command is achieved and the magnitude of the steady state error. However the PID control tuning is slightly non optimal and there wasn't enough time to perform a comprehensive tune, the gains of the PID controller here are the exact gains used on the actual APM 2.6 flight controller. The PID controller shows a large period small oscillation in the commanded axis. Figures 68 and 69, show the stabilization of the Euler angles during the commanded roll, again the FLC outperforms the PID controller, the magnitude of the Euler angles stabilized for the FLC was in the range of $10^{\wedge -3}$. The steady state error for the FLC is much lower by entire order, this brings significant confidence during future work involving experimental flight using the FLC.

2. Case 2 : Roll angle (phi) = 0°, Pitch angle (theta) = 30° , Yaw angle (psi) = 0°, Yaw Rate = 0 deg/s

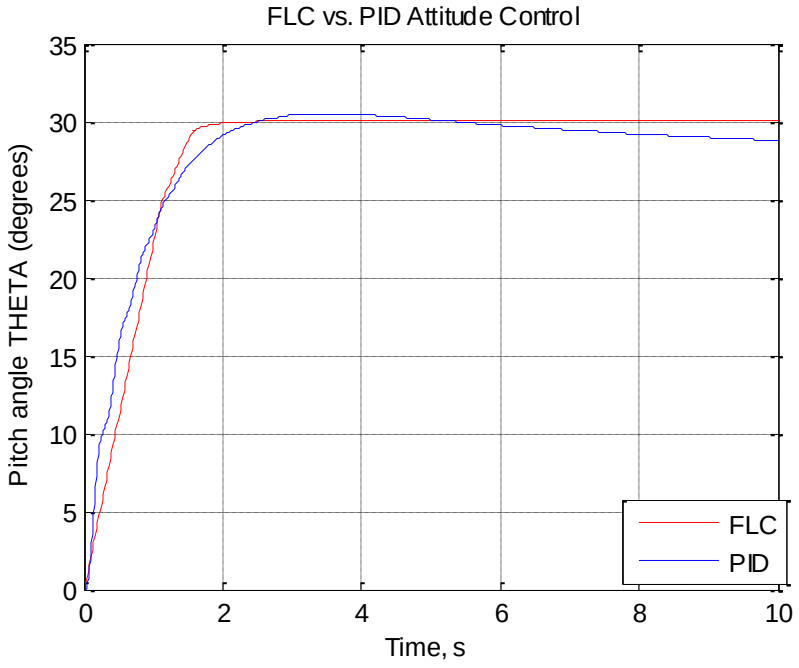

Fig. 70 FLC vs. PID Attitude Control for Commanded Pitch Angle

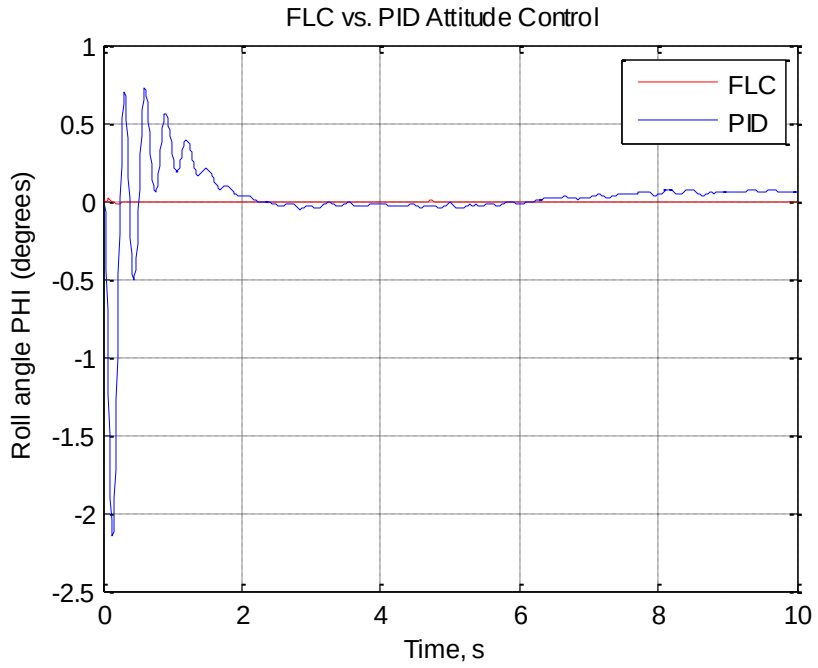

Fig. 71 FLC vs. PID Attitude Control Stabilization for Roll during Pitch Command

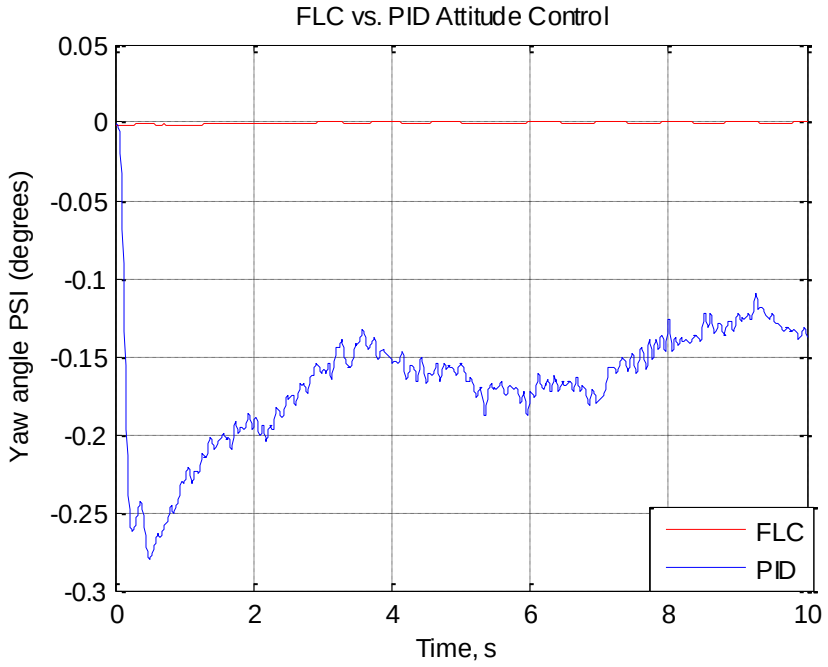

Fig. 72 FLC vs. PID Attitude Control Stabilization for Yaw during Pitch Command

Figure 70 shows the pitch response for a commanded pitch angle of 30°, the response is very similar to the roll response above. This is expected as the PID gains for both these axes are

the same, owing to the symmetry of the Quadrotor and all multi rotor UAVs in general. The FLC outperforms the PID controller in both how quickly the command is achieved and the magnitude of the steady state error.

Figures 71 and 72 shows the response of the Euler angles during the commanded Pitch response. It is observed that the FLC produces a more stable response than the PID controller across the board in roll and pitch.

3. Case 3 : Roll angle (phi) = 30°, Pitch angle (theta) = 30° , Yaw angle (psi) = 0°, Yaw Rate = 0 deg/s

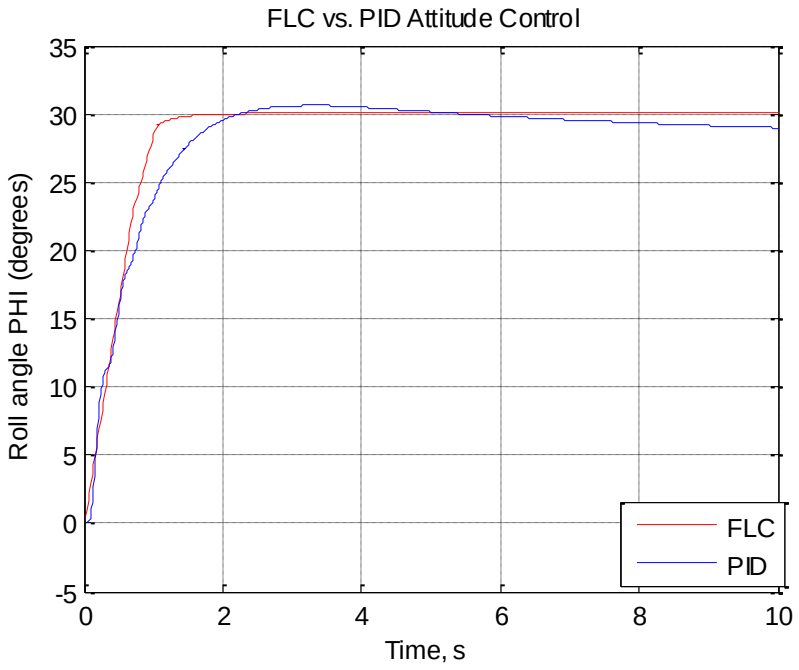

Fig. 73 Roll Response for FLC vs. PID Attitude Control for Commanded Roll and Pitch Angles

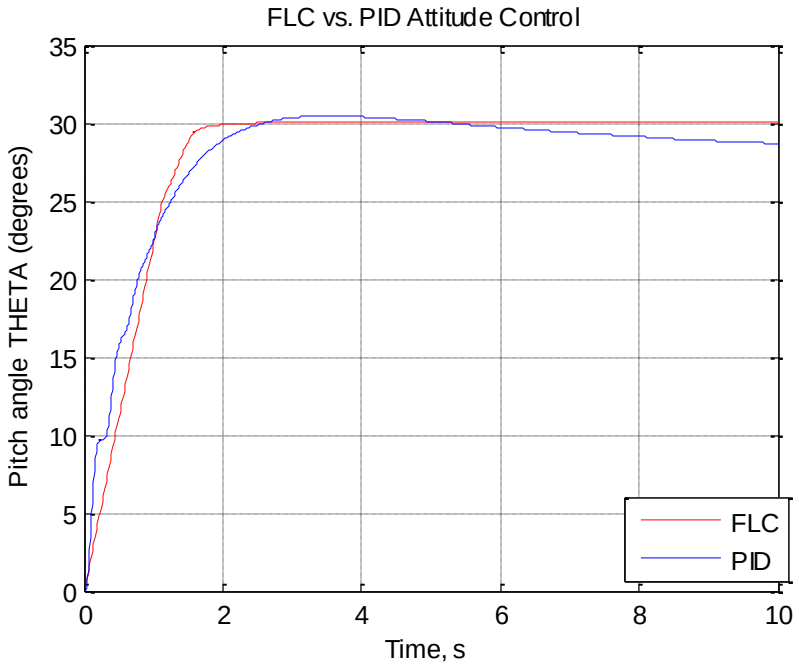

Fig. 74 Pitch Response for FLC vs. PID Attitude Control for Commanded Roll and Pitch Angles

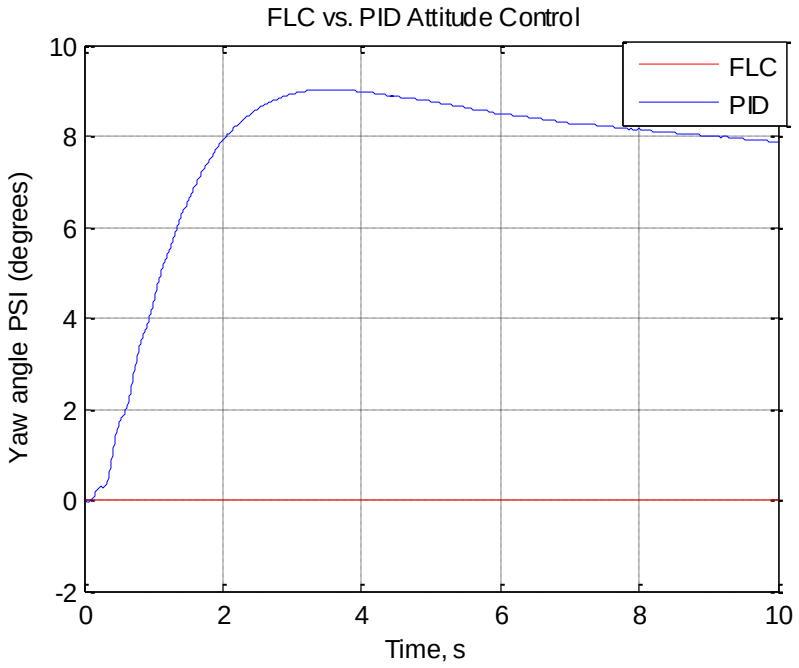

Fig. 75 Yaw Response for FLC vs. PID Attitude Control for Commanded Roll and Pitch Angles

Figures 73 and 74 show the roll and pitch responses respectively for a coupled input scenario i.e. commanded roll and pitch of 30°. The FLC as before performs better in a coupled input scenario than the PID controller. Figure 75 shows the yaw response and here the PID controller, struggles to keep the yaw axis stabilized, there is a 6-8° steady state error that gradually dies down, the FLC is able to stabilize the yaw axis in the scenario above. This is typically observed in flight as well with the APM 2.6, a small yaw drift exists even at the hover condition and is fairly noticeable when rolling and pitching i.e. a diagonal movement is attempted. It requires small pilot corrective input and is a drawback of the PID controller. However the new APM flight controller does incorporate a compass and a heading hold function that locks the yaw response down. It is clear from the above three cases, that there is a legitimate claim for a better controller for the Quadrotor UAV and that the FLC would bring significant benefits when implemented

Chapter 6: CONCLUSIONS & FUTURE WORK

This thesis deals with the mathematical modeling and control of a Quadrotor UAV. An intelligent fuzzy flight controller has been proposed, designed and built using MATLAB's fuzzy logic toolbox. A simulation environment was developed using the EOM derived and simulations show the mathematical model to be realistic in modeling position and orientation. Responses prove the effectiveness of the six FLC's in controlling position as well as achieving stability.

The presented mathematical model consists only of the basic structures of the Quadrotor's dynamics. Several phenomena were not modeled due to their complexity and associated uncertainty. It is hoped to be proved experimentally that the inherent robustness fuzzy logic control embodies will deal with these shortcomings.

Experimental validation is carried out with actual flight data thus lending high confidence to the math model developed. It is seen that the tracking of the attitude data is particularly encouraging; there are of course several considerations with respect to the simulations that have been made. The high coherence demonstrated brings significance in this research effort and the simulation results for the fuzzy flight controller. It is worth mentioning here in the interest of redundancy that the simulator gives good results for near hover flight regimes and low velocities, the inherent robustness that FLC demonstrates should account for controllability across the entire flight envelope.

The next step towards achieving a fully autonomous flight would be to validate the simulation model across the entire flight envelope and thus ascertain the true performance of the FLC in simulation. This work requires the modelling of several key aerodynamics effects such as a true blade element momentum theory model, ground effect, blade flapping dynamics and also

fluid effects such as propeller drag, wind resistance and a complete body drag model. The simulation model also needs to incorporate different configurations of multi rotor UAVs to be truly useful in determining a control strategy and validating its performance before advancing on the embedded code for the controller.

Future work is being directed towards achieving a completely autonomous flight in controlled indoor conditions. Wireless communication using the Xbee modules and incorporation of data packet loss redundancies has been successfully achieved. A more comprehensive simulation environment including wind disturbances and sensor noise inclusion is being developed. The development of preliminary embedded code for the FLCs is complete and the process of stitching the entire flight software code together is going to be undertaken shortly. Position feedback is to be achieved using the GPS and ultrasonic rangefinder now successfully integrated into the APM 2.6 Flight IMU. The implementation of the proposed control strategy will allow comparison with the simulation results obtained and would bring significant benefits to this research endeavor.

REFERENCES

1. Wegener, S., Schoenung, S., Totah, J., Sullivan, D., Frank, J., Enomoto, F., Frost C., Theodore,C., "UAV Autonomous Operations for Airborne Science Missions," *AIAA 3rd Unmanned Unlimited Technical Conference Proceedings*, 20-23 Sep., 2004, Illinois.

2. Sabo, Chelsea, and Cohen, Kelly, "Fuzzy Logic Unmanned Air Vehicle Motion Planning," *Advances in Fuzzy Systems*, Vol. 2012, No. 13, Jan. 2012.

3. Fnu, Vijaykumar Sureshkumar, and Kelly Cohen. "Intelligent Fuzzy Flight Control of an Autonomous Quadrotor UAV," University of Cincinnati, *52nd AIAA Aerospace Sciences Meeting*, AIAA SciTech 2014, National Harbor, Maryland, 13-17 Jan., 2014.

4. Tsach, S., Peled, A., Penn, D., Keshales, B., Guedj, R., "Development Trends for Next Generation UAV Systems," Israel Aircraft Industries Ltd., *AIAA Infotech@Aerospace Conference Proceedings*, May, 2007.

5. "Air Vehicles Vision 2009," Air Vehicles Directorate, Air Force Research Laboratory, 2009.

6. "California Wildfires at a Glance," *Associated Press,* 27 Oct., 2007. http://www.boston.com/news/nation/articles/2007/10/27/california_wildfires_at_a_glance/.

7. "Wildfire Imaging Flights By NASA's Ikhana UAV Conclude," *NASA,* 29 Oct., 2007. http://www.nasa.gov/centers/dryden/news/Features/2007/wildfire_socal_10_07.html.

8. Sabo, C., Cohen, K., Kumar, M., Abdallah, S., “Path Planning for a Fire-Fighting Aircraft using Fuzzy Logic,” University of Cincinnati, *47th AIAA Aerospace Sciences Meeting*, Orlando, Florida, 5-8 Jan., 2009.

9. Harris, A. "Rising to the Challenge" *Engineering & Technology* Vol. 7, No. 10, pp. 56-58, 2012

10. Stifter, M., and F. Milano. "An example of integrating open source modelling frameworks: The integration of GIS in PSAT," *Power & Energy Society General Meeting, IEEE*, pp. 1-5, 2009.

11. D'Andrea, Raffaello. "Guest Editorial Can Drones Deliver?," *Automation Science and Engineering, IEEE Transactions* Vol. 11, No. 3, pp. 647-648, 2014

12. “Unmanned Air Vehicles Roadmap, 2002-2027,” Office of the Secretary of Defense, 2002.

13. “Unmanned Air Vehicles Roadmap, 2005-2030,” Office of the Secretary of Defense, 2005.

14. McKerrow, Phillip. "Modelling the Draganflyer four-rotor helicopter," *Proceedings of the 2004 IEEE International Conference on Robotics and Automation,* New Orleans, Los Angeles, Vol. 4, pp. 3596-3601, 2004..

15. Wu, H., Zhou, Z. Sun, D., "Autonomous hovering control and test for micro air vehicle." *Proceedings of the 2003 IEEE International Conference on Robotics and Automation Robotics and Automation,* Taipei, Taiwan, Vol. 1, pp. 528-533, 2003.

16. Hanford, S. D., Long, L. N., Horn, J. F., "A small semi-autonomous rotary-wing unmanned air vehicle (UAV)," *AIAA Infotech@Aerospace Conference and Exhibit*, Arlington, Virginia, September, 2005.

17. Aerosonde. Aerosonde "laima" fixed wing UAV. http://uavm.com/uavfixedwing/fwaaiaerov.html Online; accessed 23-March-2015

18. Lockheed Martin. Global Hawk Drone. http://commons.wikimedia.org/wiki/File:Global_Hawk_Drone_(4020342776).jpg Online; accessed 17-February-2015

19. Prox Dynamics AS. Black Hornet Nano Helicopter UAV. http://commons.wikimedia.org/wiki/File:Black_Hornet_Nano_Helicopter_UAV.jpg Online; accessed 28-March-2015

20. Schiebel. Sciebel Camcopter S-100. http://commons.wikimedia.org/wiki/File:Schiebel_Camcopter_S-100_at_ILA_2010.jpg Online; accessed 28-March-2015

21. Wood, A., "Drone Operator," The Rosen Publishing Group, 2013.

22. Kochan, A., "Automation in the sky," *Industrial Robot: An International Journal*, Vol. 2, No. 36, pp.468-471, 2005.

23. Draganfly. Draganflyer X8. http://maisonbisson.com/blog/post/12366/dragonflyer-x6-uav-remote-control-helicopter-is-sneaky-awesome/ Online; accessed 26-March-2015

24. Drones, D. I. Y., "Arducopter," 2013.

25. Team, Arduino., "Arduino Mega homepage," 2011.

26. 3D Robotics. 3DR Iris. https://instagram.com/p/yN1oP7qfRu/ Online; accessed 27-April-2014

27. Carancho, T. "AeroQuad–The open source quadcopter/multicopter," 2011.

28. Hoffmann, G. M., Huang, H., Waslander, S. L., Tomlin, C. J., "Quadrotor helicopter flight dynamics and control: Theory and experiment," *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*, Vol. 2, South Carolina, Aug., 2007.

29. Huang, H., Hoffmann, G. M., Waslander, S. L., Tomlin, C. J., "Aerodynamics and control of autonomous Quadrotor helicopters in aggressive maneuvering," *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, pp. 3277-3282, May., 2009

30. Tayebi, A., McGilvray, S., "Attitude stabilization of a four-rotor aerial robot." *Proceedings of the 2004 IEEE Conference on Decision and Control,* Vol. 2, pp. 1216-1221, 2004.

31. Dikmen, I. C., Arısoy, A., Temeltas, H., "Attitude control of a quadrotor," *4th International Conference on Recent Advances in Space Technologies*, pp. 722–727, 2009.

32. Zuo, Z., "Trajectory tracking control design with command-filtered compensation for a quadrotor," *Control Theory & Applications, IET*, Vol. 4, No.11, pp. 2343-2355, 2010.

33. Bouabdallah, S., Noth, A., Siegwart, R., "PID vs LQ control techniques applied to an indoor micro quadrotor," Proceedings *of the 2004 IEEE/RSJ Conference on Intelligent Robots and Systems,* Vol. 3, pp. 2451-2456, Sep., 2004.

34. Madani, T., Benallegue, A., "Backstepping control for a quadrotor helicopter," *Proceedings of the 2006 IEEE/RSJ Conference on Intelligent Robots and Systems,* pp. 3255-3260, Oct., 2006

35. Zemalache, K. M., Beji, L., Marref, H., "Control of an under-actuated system: application to a four rotors rotorcraft," *Proceedings of the 2005 IEEE International Conference on Robotics and Biomimetics,* pp. 404-409, Dec., 2005.

36. Raffo, G. V., Ortega, M. G., Rubio, F. R., "An integral predictive/nonlinear H∞ control structure for a quadrotor helicopter," *Automatica*, Vol. 46, No. 1, pp. 29-39, 2010

37. Castillo, P., Lozano, R., Dzul, A., "Stabilization of a mini rotorcraft with four rotors," *IEEE Control Systems Magazine*, Vol. 25, No. 6, pp. 45-55, 2005

38. Escareno, J., Salazar-Cruz, S., Lozano, R., "Embedded control of a four-rotor UAV," *American Control Conference,* Vol. 4, No. 11, pp. 3936-3941, Jun., 2006.

39. Martin, P., Salaun, E., "The true role of accelerometer feedback in quadrotor control," *Proceedings of the 2010 IEEE International Conference on Robotics and Automation,* pp. 1623-1629, May., 2010

40. He, R., Prentice, S., Roy, N., "Planning in information space for a quadrotor helicopter in a GPS-denied environment," *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, pp. 1814-1820, May., 2008

41. Bouabdallah, S., “Design and control of quadrotors with application to autonomous flying,” PhD dissertation, École Polytechnique federale de Lausanne, Zurich, Switzerland, 2007

42. Kim, T., Stol, K., Kecman, V., “Control of 3 DOF Quadrotor Model,” *Robot Motion and Control, pp.* 29-38, 2007.

43. Voos, H. “Nonlinear control of a quadrotor micro-UAV using feedback-linearization,” *Proceedings of the IEEE International Conference on Mechatronics,* pp. 1-6, Apr., 2009.

44. Kebriaei, Reza; Frischkorn, Jan; Reese, Stefanie; Husmann, Tobias; Meier, Horst; Moll, Heiko; Theisen, Werner. "Numerical modelling of powder metallurgical coatings on ring-shaped parts integrated with ring rolling". *Material Processing Technology* **213** (1): 2015–2032.

45. Bennett, Stuart., “A history of control engineering”, London, United Kingdom, 1986. IET. p. p. 48

46. "Introduction: PID Controller Design". University of Michigan. http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID Online; accessed 29-March-2015

47. Tim Wescott. "PID without a PhD". EE Times-India. Oct., 2000. http://igor.chudov.com/manuals/Servo-Tuning/PID-without-a-PhD.pdf Online; Accessed 27-March-2015

48. Ang, K.H., Chong, G.C.Y., and Li, Y. “PID control system analysis, design, and technology”, *IEEE Trans Control Systems Tech*, 13(4), pp.559-576, May., 2005.

49. Correspondent, K. S., “Designing with fuzzy logic,” *IEEE Spectrum,* pp. 42-44, Nov., 1990.

50. Sabo, C., “UAV Two-Dimensional Path Planning in Real-Time using Fuzzy Logic,” M.S. Thesis, University of Cincinnati, Cincinnati, Ohio, Jul., 2011.

51. Ross, T., “Fuzzy Logic with Engineering Applications,” Wiley, West Sussex, United Kingdom, 2010.

52. Bresciani, T., “Modelling, identification and control of a quadrotor helicopter,” Department of Automatic Control, Lund University, 2008.

53. Premerlani, W., Bizard, P., "Direction cosine matrix imu: Theory,"*DIY DRONE: USA,* pp. 13-15, 2009

54. Gleim, I., “Pilot Flight Maneuvers,” Aviation Publications, Ottawa, Ontario, Canada, 1982.

55. Dole, C. E., Lewis, J. E., “Flight theory and aerodynamics: a practical guide for operational safety,” John Wiley & Sons, USA, 2000.

56. Dreier, E. M., “Introduction to Helicopter and Tiltrotor Simulation,” AIAA Education Series, pp. 27, Virginia 20191, USA, 2007.

57. Beard, R. W., “Quadrotor dynamics and control,” Brigham Young University, 2008.

58. Seckel, E., “Stability and control of airplanes and helicopters,” Academic Press, New York, 1964.

59. Raza, S. A., Gueaieb, W., “Intelligent Flight Control of an Autonomous Quadrotor”. *InTech*, 2010.

60. Yang, H. C., Sababha, B., Acar, C., Rawashdeh, O.,”Rapid Prototyping of Quadrotor Controllers using MATLAB RTW and dsPICs,” *Proceedings of the AIAA Infotech Aerospace Conference,* Georgia, USA, 2010.

APPENDIX

C++ Source Code for FLC φ

```
/contents of .c file source code
#include "quad_phi_new.h"
// Not Function
float _not(float x)
{
        return (float)(1.0-x);
}
// Generates the values in the range [min,max] with increments
// defined by 'step'.
void generate(float minimum, float maximum, float st, float* values,int length)
{
        float x = minimum;
        int i;
        for(i=0;i<length;i++,x+=st)
                values[i] = x;
}
// Sets all elements in the array with the same value
void set(float* values,int length,float value)
{
        int i;
        for(i=0;i<length;i++)
                values[i] = value;
}
// Returns the smallest element of an array
float min_array(float* x, int length)
{
   if(length <= 0) return (float)0;
        float m = x[0];
        int i=1;
        for(;i<length;i++)
                m = min(m,x[i]);
        return m;
}
// Returns the largest element of an array
float max_array(float* x, int length)
{
   if(length <= 0) return (float)0;
        float m = x[0];
        int i=1;
        for(;i<length;i++)
                m = max(m,x[i]);
```

```
        return m;
}
// Centroid of points
void centroid(float* x, float* y, int length, float* Cx, float* Cy)
{
        float a = 0;
        float tx = 0;
        // float ty = 0;
        float t = 0;
        int i = 0;
      int l = length - 1;
        for(;i<l;i++)
        {
                t = ((x[i] * y[i+1]) - (x[i+1] * y[i]));
           a += t/2;
                tx = (x[i] + x[i+1]) * t;
                // ty = (y[i] + y[i+1]) * t;
        }
        *Cx = (tx /(6 * a));
        // *Cy = (ty /(6 * a));
}
// Triangular Member Function
float trimf(float x, float a, float b, float c)
{
   return (float)max(min((x - a) / (b - a), (c - x) / (c - b)), 0);
}
// Trapezoidal Member Function
float trapmf(float x, float a, float b, float c, float d)
{
   return (float)max(min((x - a) / (b - a), min(1.0, (d - x) / (d - c))), 0);
}
// Aggregation code for membership functions that require 3 setting values
void aggregate3(float* rules, int* ruleIndexes, int ruleIndexLength, float* xArr, float *yArr, int
resultLength,_mffloat3 mf,float a,float b,float c)
{
        int i=0, j=0;
        for(;i<resultLength;i++)
        {
                for(j=0;j<ruleIndexLength;j++)
                {yArr[i] = max(yArr[i],min(rules[ruleIndexes[j]],mf(xArr[i],a,b,c)));}
        }
}

// Aggregation code for membership functions that require 4 setting values
void aggregate4(float* rules, int* ruleIndexes, int ruleIndexLength, float* xArr, float *yArr, int
resultLength,_mffloat4 mf,float a,float b,float c,float d)
```

```
{
	int i=0, j=0;
	for(;i<resultLength;i++)
	{
		for(j=0;j<ruleIndexLength;j++)
		{yArr[i] = max(yArr[i],min(rules[ruleIndexes[j]],mf(xArr[i],a,b,c,d)));}
	}
}

void quad_phi_new(float* inputs, float* outputs)
{	// Transformation of inputs to fuzzy inputs.
	// Input variable errorPHI
	float Inputs1[5];
	// Input term 'ZERO'for variable 'errorPHI'
	Inputs1[0] = trimf(inputs[0],-0.025,0,0.025);
	// Input term 'Phigh'for variable 'errorPHI'
	Inputs1[1] = trapmf(inputs[0],0.1019,0.25,0.675,0.88);
	// Input term 'Nlow'for variable 'errorPHI'
	Inputs1[2] = trimf(inputs[0],-0.15,-0.075,0);
	// Input term 'Plow'for variable 'errorPHI'
	Inputs1[3] = trimf(inputs[0],0,0.075,0.15);
	// Input term 'Nhigh'for variable 'errorPHI'
	Inputs1[4] = trapmf(inputs[0],-0.935,-0.535,-0.25,-0.0939);
	// Input variable errordotPHI
	float Inputs2[5];
	// Input term 'Nfast'for variable 'errordotPHI'
	Inputs2[0] = trapmf(inputs[1],-1.768,-1.385,-0.648,-0.3275);
	// Input term 'ZERO'for variable 'errordotPHI'
	Inputs2[1] = trimf(inputs[1],-0.209,0,0.198);
	// Input term 'Pfast'for variable 'errordotPHI'
	Inputs2[2] = trapmf(inputs[1],0.3325,0.627,1.4,1.77);
	// Input term 'Nslow'for variable 'errordotPHI'
	Inputs2[3] = trimf(inputs[1],-0.5,-0.25,0);
	// Input term 'Pslow'for variable 'errordotPHI'
	Inputs2[4] = trimf(inputs[1],0,0.25,0.5);
	// Build all the conditions.
	// Variable to store the condition result
	float R[25];
	float CondR[2];
	// Handle conditions for rule 1
	CondR[0] = Inputs1[4];
	CondR[1] = Inputs2[0];
	R[0] = 1 * min_array(CondR,2);
	// Handle conditions for rule 2
	CondR[0] = Inputs1[4];
	CondR[1] = Inputs2[3];
```

```
R[1] = 1 * min_array(CondR,2);
// Handle conditions for rule 3
CondR[0] = Inputs1[4];
CondR[1] = Inputs2[1];
R[2] = 1 * min_array(CondR,2);
// Handle conditions for rule 4
CondR[0] = Inputs1[4];
CondR[1] = Inputs2[4];
R[3] = 1 * min_array(CondR,2);
// Handle conditions for rule 5
CondR[0] = Inputs1[4];
CondR[1] = Inputs2[2];
R[4] = 1 * min_array(CondR,2);
// Handle conditions for rule 6
CondR[0] = Inputs1[2];
CondR[1] = Inputs2[0];
R[5] = 1 * min_array(CondR,2);
// Handle conditions for rule 7
CondR[0] = Inputs1[2];
CondR[1] = Inputs2[3];
R[6] = 1 * min_array(CondR,2);
// Handle conditions for rule 8
CondR[0] = Inputs1[2];
CondR[1] = Inputs2[1];
R[7] = 1 * min_array(CondR,2);
// Handle conditions for rule 9
CondR[0] = Inputs1[2];
CondR[1] = Inputs2[4];
R[8] = 1 * min_array(CondR,2);
// Handle conditions for rule 10
CondR[0] = Inputs1[2];
CondR[1] = Inputs2[2];
R[9] = 1 * min_array(CondR,2);
// Handle conditions for rule 11
CondR[0] = Inputs1[0];
CondR[1] = Inputs2[0];
R[10] = 1 * min_array(CondR,2);
// Handle conditions for rule 12
CondR[0] = Inputs1[0];
CondR[1] = Inputs2[3];
R[11] = 1 * min_array(CondR,2);
// Handle conditions for rule 13
CondR[0] = Inputs1[0];
CondR[1] = Inputs2[1];
R[12] = 1 * min_array(CondR,2);
// Handle conditions for rule 14
```

```
CondR[0] = Inputs1[0];
CondR[1] = Inputs2[4];
R[13] = 1 * min_array(CondR,2);
// Handle conditions for rule 15
CondR[0] = Inputs1[0];
CondR[1] = Inputs2[2];
R[14] = 1 * min_array(CondR,2);
// Handle conditions for rule 16
CondR[0] = Inputs1[3];
CondR[1] = Inputs2[0];
R[15] = 1 * min_array(CondR,2);
// Handle conditions for rule 17
CondR[0] = Inputs1[3];
CondR[1] = Inputs2[3];
R[16] = 1 * min_array(CondR,2);
// Handle conditions for rule 18
CondR[0] = Inputs1[3];
CondR[1] = Inputs2[1];
R[17] = 1 * min_array(CondR,2);
// Handle conditions for rule 19
CondR[0] = Inputs1[3];
CondR[1] = Inputs2[4];
R[18] = 1 * min_array(CondR,2);
// Handle conditions for rule 20
CondR[0] = Inputs1[3];
CondR[1] = Inputs2[2];
R[19] = 1 * min_array(CondR,2);
// Handle conditions for rule 21
CondR[0] = Inputs1[1];
CondR[1] = Inputs2[0];
R[20] = 1 * min_array(CondR,2);
// Handle conditions for rule 22
CondR[0] = Inputs1[1];
CondR[1] = Inputs2[3];
R[21] = 1 * min_array(CondR,2);
// Handle conditions for rule 23
CondR[0] = Inputs1[1];
CondR[1] = Inputs2[1];
R[22] = 1 * min_array(CondR,2);
// Handle conditions for rule 24
CondR[0] = Inputs1[1];
CondR[1] = Inputs2[4];
R[23] = 1 * min_array(CondR,2);
// Handle conditions for rule 25
CondR[0] = Inputs1[1];
CondR[1] = Inputs2[2];
```

```
R[24] = 1 * min_array(CondR,2);
// Basic output processing
int res = 100;
float step;
float xMin;
float xMax;
float x[res];
float y[res];
float t;
int value[25];
// Output variable PWM_phi
xMin = -250;
xMax = 250;
t=0;
step = (xMax - xMin) / (float)(res - 1);
generate(xMin, xMax, step, x, res);
set(y, res, 0);
value[0] = 2;
value[1] = 3;
value[2] = 4;
value[3] = 9;
value[4] = 14;
aggregate4(R,value,5,x,y,res,trapmf,-470,-420,-150,-100);
value[0] = 0;
value[1] = 6;
value[2] = 12;
value[3] = 18;
value[4] = 24;
aggregate3(R,value,5,x,y,res,trimf,-12.5,0,12.5);
value[0] = 15;
value[1] = 20;
value[2] = 21;
value[3] = 22;
aggregate4(R,value,4,x,y,res,trapmf,100,150,285,372.5);
value[0] = 1;
value[1] = 7;
value[2] = 8;
value[3] = 13;
value[4] = 19;
aggregate3(R,value,5,x,y,res,trimf,-100,-50,0);
value[0] = 5;
value[1] = 10;
value[2] = 11;
value[3] = 16;
value[4] = 17;
value[5] = 23;
```

```
        aggregate3(R,value,6,x,y,res,trimf,0,50,100);
        centroid(x, y, res, &outputs[0], &t);
}
```

Header File for FLC φ

```
/Contents of .h file quad_phi_new.h header file

typedef float (*_mffloat2)(float,float,float);
typedef float (*_mffloat3)(float,float,float,float);
typedef float (*_mffloat4)(float,float,float,float,float);
typedef double (*_mfdouble2)(double,double,double);
typedef double (*_mfdouble3)(double,double,double,double);
typedef double (*_mfdouble4)(double,double,double,double,double);

/*************************************************************************
 * A Fast, Compact Approximation of the Exponential Function
*************************************************************************/
#define M_LN2 0.69314718055994530941
union
{
        double d;
        struct{ int j,i;}n;
}_eco;
#define EXP_A (1048576/M_LN2)
#define EXP_C 60801
#define _exp(y) (_eco.n.i = EXP_A * (y) + (1072693248 - EXP_C), _eco.d)
```