5-2007

Linear and Non-Linear Control of a Quadrotor UAV

Andrew Neff
Clemson University, aneff@ieee.org

LINEAR AND NON-LINEAR CONTROL OF A QUADROTOR UAV

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science
Electrical Engineering

---

by
Andrew Neff
May 2007

Accepted by:
Dr. Timothy Burg, Committee Chair
Dr. Darren Dawson
Dr. Samuel Sander

ABSTRACT

This thesis describes two controllers designed specifically for a quadrotor helicopter unmanned aerial vehicle (UAV). A linear controller and a non-linear controller are discussed for use on the quadrotor helicopter using feedback that is obtained from microelectromechanical systems (MEMS) and global positioning system (GPS) sensors.

The linear controller is an orientation based PID controller that controls the angles of the quadrotor UAV. The controller was first simulated and the results displayed graphically using FlightGear. Experiments were conducted using this controller on a DraganFlyer X-Pro quadrotor helicopter to prove the proposed method used for closing the feedback loop.

The non-linear controller is developed using Lyapunov stability methods. The design goal for this controller is to add a two degree-of-freedom camera postioner to the quadrotor for a total of six degree-of-freedom camera actuator. The UAV will track three desired translational velocities and three angular velocities using only translational and rotational velocities for feedback. Simulations were conducted to verify this controller.

ACKNOWLEDGEMENTS

I would like to acknowledge my committee and my professors for all of their help through my college career.

DEDICATION

I dedicate this to Yomiko, for all of her encouragement.

TABLE OF CONTENTS

Table of Contents (Continued)

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

List of Figures (Continued)

Figure Page

CHAPTER 1
INTRODUCTION

Background

An unmanned aerial vehicle (UAV) refers to any flying vehicle that does not require a live pilot on the aircraft, typically an airplane or helicopter. UAV research is a growing field because of the emerging affordable technology, allowing for UAVs to be deployed in numerous new applications. Sending UAVs into dangerous situations prevents the endangering of human lives while accomplishing tasks such as visual inspections, following a target [4], scouting, and many more applications.

This thesis will focus on the control design for a particular UAV, the quadrotor helicopter, because of its simple design and its ability to control its torques. This particular type of UAV is an underactuated helicopter that is, there are only four inputs to control the six degrees-of-freedom. The quadrotor design is covered in [6] and the analysis for controlling a quadrotor is described in [11]. Since the quadrotor has only four control inputs, two degrees-of-freedom are coupled in the sense that the translational position depends on the orientation of the aircraft.

Another key aspect to developing these control systems is the sensors that measure position and orientation of an aerial vehicle. One of the most prevalent sensor systems in use is the global positioning system (GPS) based on satellite signals. GPS supplies measurements of the three translational positions and velocities with respect to the earth. However, to measure the three orientations another technology is required. Using a combination of microelectromechanical systems (MEMS) gyroscopes, accelerometers, and magnetometers, the orientation of a UAV can be determined. Thus, by combining GPS sensor with an array of attitude sensors, the six DOF position and orientation of an aircraft can be determined.

Previous Work

There is a lot of work being done on UAVs with the availability of affordable UAVs and UAV sensors such as MEMS Gyros and GPS sensors. The Jang and Tomlin paper [8] discusses the use of a single GPS sensor for use on UAV tracking. It is important to have a method for a UAV to know where it is and how it is oriented so that a controller can have a feedback signal. The Hamel and Mohony paper [3] defines a dynamic model for an X-4 Flyer. The X-4 Flyer is another quadrotor similar to the DraganFlyer X-Pro used at Clemson University. The dynamic model proposed in [3] treated the quadrotor helicopter as a rigid body that has the ability to thrust and torque itself in midair.

The Chitrakaran and Dawson paper [5] designs an autonomous landing system using a vision based controller. The controller used includes a method for handling the underactuated quadrotor and the coupling between the translational and rotational forces. Procerus Technologies has a Vision-Centric [15] approach to many targeting systems. They have developed an OnPoint Targeting system that include 5 different tracking methods. Among these difference methods, include a Fly by Camera Control approach where the UAV can be controlled from the frame of reference of the camera instead of the UAV frame.

Thesis Outline

A MEMS and GPS sensor are used on a quadrotor helicopter in the development of two completely different control systems. The first is a PID controller that will utilize the angular sensors for controlling the orientation of a quadrotor in flight, according to a set of desired orientations. Hovering a helicopter is a very demanding task that requires many small adjustments when flying by hand and can only be effectively accomplished by an experienced pilot. Using the proposed orientation controller, the user simply specifies an orientation, and the control works to achieve it. The desired orientations used can be generated by numerous means, but they will be generated by a joystick for these experiments.

The second control system will focus on the visual inspection application of UAVs. When a helicopter is flown, an operator will typically: watch the helicopter as it moves and actuate the motors for local uses or watch video taken from a pilot perspective with an on-board video camera. In either case, it is difficult to manually keep the UAV stable. A new approach to this control problem was presented in [5] where the UAV and the camera positioning unit are considered to be a single robotic unit. From this perspective, a controller can be developed which will simultaneously control both the UAV and the camera positioning unit in a complementary fashion. Here, this control approach is exploited to provide a new perspective for piloting the UAV. This perspective, which shall be referred to as the fly-the-camera perspective, presents a new interface to the pilot. In this proposed approach, the pilot commands motion from the perspective of the on-board camera - it is as though the pilot is riding on the tip of the camera and commanding movement of the camera ala a six-DOF flying camera. This is subtly different from the traditional remote control approach wherein the pilot processes the camera view and then commands an aircraft motion to create a desired motion of the camera view.

If there is a camera mounted on the UAV, then the orientation and the position of the UAV affect the orientation and position of the camera. Since the camera is very important during visual inspections, it was decided that instead of making the control inputs control the UAV, a non-linear controller will be used to control the movements of the camera, actuating the UAV in the process. The control inputs will not be controlling a particular torque to either keep the UAV still, to rotate it, or to move the UAV, but the input will simple tell the camera to move in a particular direction, handling the current orientation of the UAV in the background. In addition, using an actuated camera and the UAV will create a fully actuated system, giving complete control over all six degrees-of-freedom.

This thesis is divided into two main chapters. Chapter 2 covers the development of a PID controller that uses the quadrotor orientation feedback to control quadrotor. A method for implementing a closed-loop sensor feedback system using wireless transmitters will be described. The experiments for the PID controller include a simulation of the controller and experiments implementing the PID controller on the DraganFlyer X-Pro.

Chapter 3 is dedicated to a non-linear controller that uses only translational and angular velocities for feedback. A two degree-of-freedom camera positioner will be mounted onto the quadrotor helicopter to make a combined UAV camera platform that is fully actuated in all six degrees-of-freedom. Desired velocities will be given relative to the camera frame, creating a fly-by-camera interface.

Notation

The math for explaining robotics and their systems can involve many points of views, or frames of reference. With the existence of two or more frames, quantities such as rotation between two frames will be expressed as

$$\Theta_N^A \in \mathbb{R}^3$$

where Θ_N^A are the three roll, pitch, and yaw angles of rotation of frame N with respect to A. A position will be expressed as

$$x_{EB}^N \in \mathbb{R}^3$$

denoting the position of frame B relative to frame E expressed in the orientation of frame N. The quantities x_{EB}^N can be expressed in other frames by using a rotation matrix

$$R_N^A \in SO\left(3\right),$$

where R_N^A is the matrix that will transform coordinates defined in frame N to frame A. So the quantity x_{EB}^N can be expressed in frame A by saying

$$R_N^A x_{EB}^N = x_{EB}^A. \tag{1.1}$$

CHAPTER 2
LINEAR CONTROL

Introduction

Unmanned Aerial Vehicles (UAVs) can be used to complete a variety of tasks. The quadrotor unmanned aerial vehicle can be used for civilian and military tasks. They can go places too dangerous for humans and in places too small for a person [12]. The quadrotor UAV is inherently unstable, thus a system is required to control the four actuators on the quadrotor to achieve a desired position and orientation.

The quadrotor has a six degree-of-freedom (DOF) rigid body that is positioned by changing the relative speed of the four rotors. These speed differences of the rotors can produce torques about the roll, pitch, and yaw axes in addition to the thrust produced as the sum of the four rotating blades. Since the helicopter is underactuated, it is only able to translate in one direction, up and down, while rotating about all three axes. The remaining two translational axes depend on the upward force coupled with the orientation of the UAV.

The basic components for building an orientation or position controller include the quadrotor, a sensor for feedback, and a method for closing the loop. In this chapter, a method for closing the control loop wirelessly is covered and tested. This wireless loop requires that there be no computer on the quadrotor weighing it down, only the sensors and the hardware needed for wireless communication are used on the quadrotor itself. To test out the wireless link, an orientation control system is developed using a PID controller which will allow a pilot to easily control the quadrotor. This will allow for a less experienced pilot to control the quadrotor without problem.

This chapter is divided up into six additional sections after the introduction. The system model section will cover the dynamics of the quadrotor that

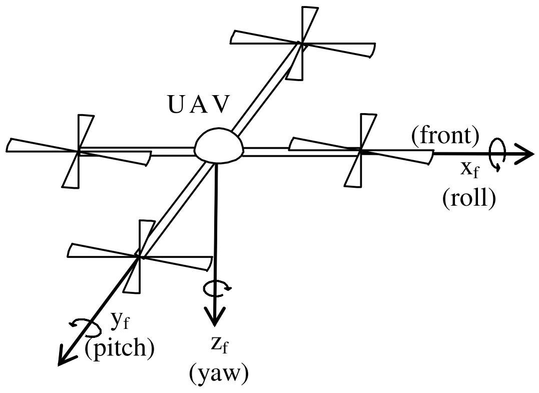


Figure 2.1 Yaw, pitch, and roll definitions.

will be used in the simulations, as well as the kinematics and coupling effect of the quadrotor design. The control method section will cover how these sensors will be used in a PID system to control the quadrotor. Simulation and implementation will cover the software used for the simulation and controller and implementing it on the DraganFlyer X-Pro quadrotor helicopter. Observations and results will cover how well the controller works followed by the conclusion.

System Model

DraganFlyer Quadrotor Overview

Figure 2.2 displays the different effects of certain rotor combinations. The depiction in Figure 2.2.a shows all four rotors spinning at an equal rate which results in an upward force in the z-direction. Since the rotors on the DraganFlyer X-Pro can only spin in one direction, the forces from each rotor and the sum of all four rotors will always be added up in the negative z-direction, according to Figure 2.1. If all the rotors spin faster then the craft will rise and if all spin slower then the craft will settle.

The intriguing aspect of a quadrotor is the manner in which the torques, that can be used to move the quadrotor, are generated. The four rotors can be grouped into two sets, group A consisting of the front and back rotors and group B consisting of the left and right rotors. Both rotors in group A spin counter-clockwise while both rotors in group B spin clockwise, shown in Figure 2.2. Pitch will be defined as rotation about the y-axis, roll as rotation about the x-axis and yaw as rotation about the z-axis, as seen in Figure 2.1. To achieve pitch torque, the front and back rotors in group A must spin at different speeds. To pitch clockwise, the front rotor speed is decreased and the rear rotor speed increased while keeping the left and right rotors in group B constant, as depicted in Figure 2.2.b. The front rotor is increased and the back rotor is equally decreased so that the total sum of the four rotor forces remain the same. The same method is used for generating a roll torque in the clockwise direction as seen in Figure 2.2.c. The third body torque is applied using a different method; instead of using the thrusting forces of the rotors as done for roll and pitch, rotating in the yaw direction uses torque couples. Since group A spins counter-clockwise and group B spins clockwise, the quadrotor creates a clockwise couple and counter-clockwise couple. When all four rotors spin at the same speed, the couples cancel out and there is no yaw rotation. But when group B slows down, and group A speeds up, there will be a counterclockwise rotation as illustrated in Figure 2.2.d.

The DraganFlyer X-Pro is designed for each rotor to spin in one direction only. Because of this restriction, there are certain situations in which all torques cannot be arbitrarily applied. The first example in Figure 2.3.a is when all four rotors are stopped. If roll is the desired torque, one motor cannot be decreased while the opposite is increased as an undesired yaw force will be introduced. This yaw force can be cancelled out with the other two rotors, but then an undesired upward force is generated. If the motors could spin backwards, then the roll torque could be achieved by simply spinning the left rotor and right rotor in opposite directions. It is rare to have all four rotors stopped while flying. The example in Figure 2.3.b is far more likely. When trying to achieve a large yaw torque, two of the motors will shut off, making

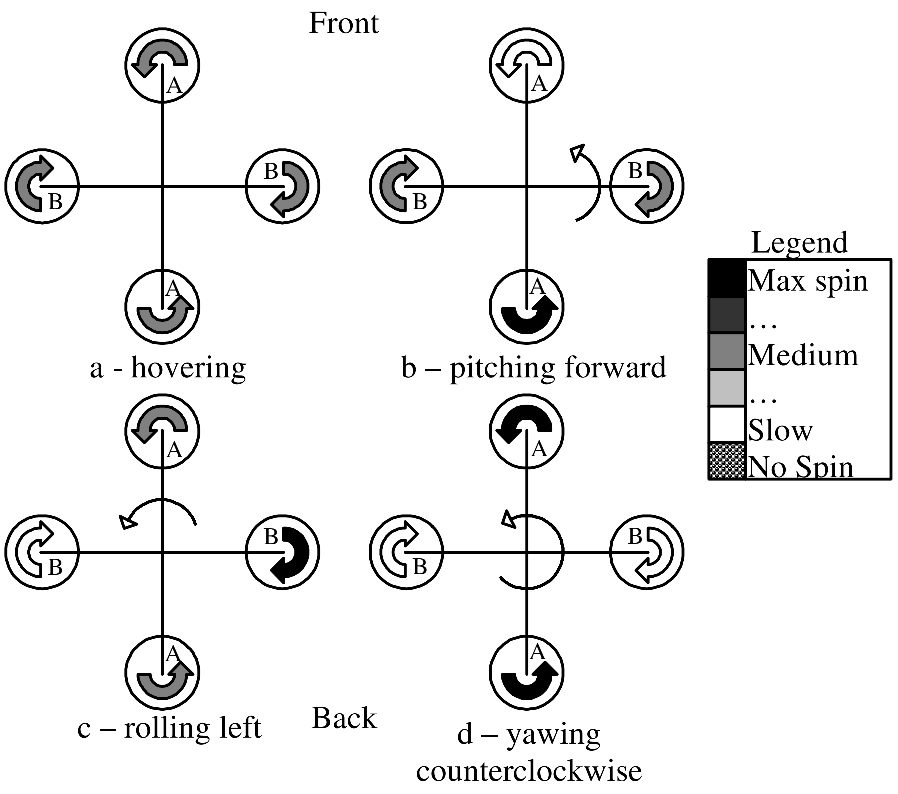


Figure 2.2 Quadrotor method of applying torques to produce motion.

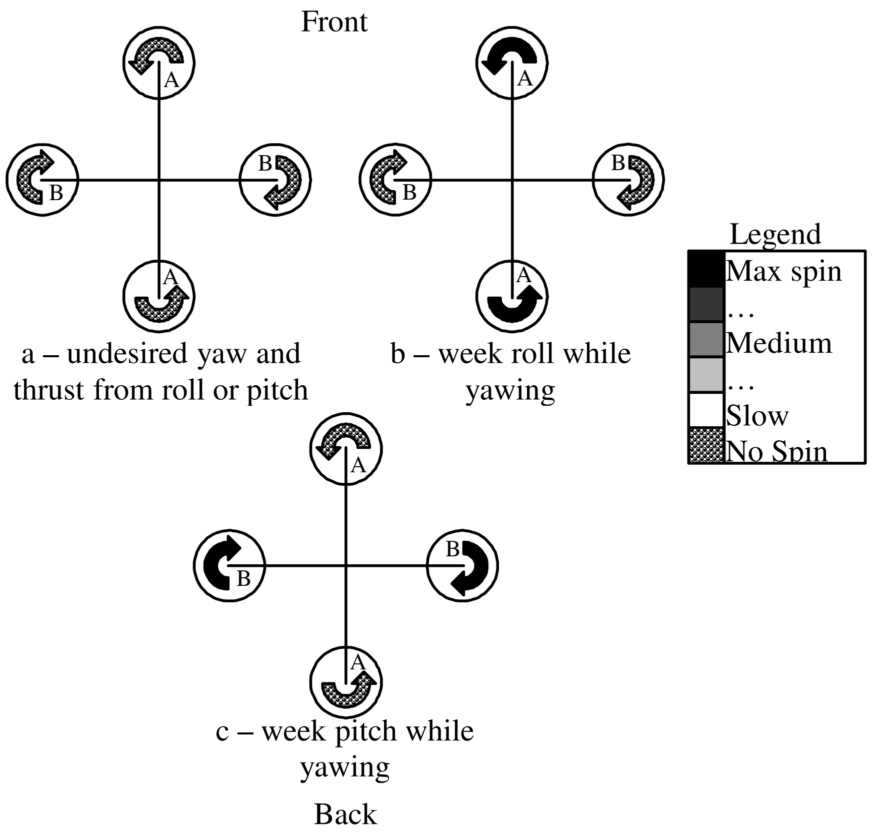

Front
A
B
B
A
a – undesired yaw and thrust from roll or pitch
A
B
B
A
b – week roll while yawing
Legend
Max spin
…
Medium
…
Slow
No Spin
A
B
B
A
c – week pitch while yawing
Back

Figure 2.3 Peculiarities resulting from mono-directional rotor motors and blades.

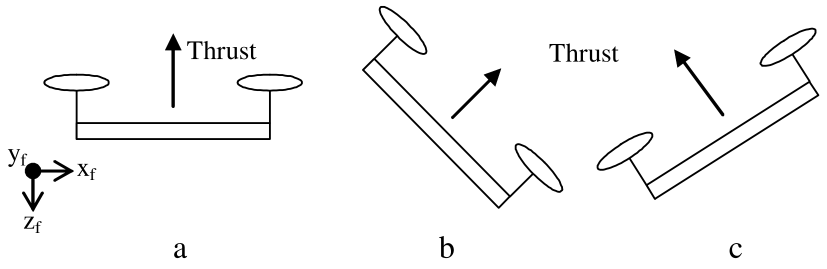


Figure 2.4 Series of motions the quadrotor executes while moving

roll impossible or extremely weak. In order to apply a roll torque, speeding up the left rotor only will start creating a stray yaw torque in addition to an additional upward force. The same is true for pitch in Figure 2.3.c. One method to prevent such a situation is to set a minimum rotor speed. That way the quadrotor can still apply at least a small amount of torque in the roll, pitch, and yaw directions without introducing undesired forces and torques.

As stated, the quadrotor is underactuated, although it is still free to move in all of its six degrees-of-freedom (DOF). An example of quadrotor motion is shown in Figure 2.4. In Figure 2.4.a, the quadrotor is hovering. To move in the x-direction, the quadrotor must pitch clockwise to direct a component of the forward direction as seen in Figure 2.4.b. To come to a stop, the quadrotor must pitch back as seen in Figure 2.4.c to bring the quadrotor's velocity down to zero. Once the quadrotor horizontal motion has stopped, it returns to the horizontal state, Figure 2.4.a. This is the coupling between the pitch angle and the x-direction which is used to move in the forward direction. The same coupling is seen between the roll angle and the y-direction. Note that the rotor speeds must increase in Figures 2.4.b and 2.4.c above Figure 2.4.b as the thrust component that counters gravity is reduced.

Quadrotor Dynamics Model

As discussed above, the quadrotor UAV, such as the DraganFlyer X-Pro quadrotor [18], is an inherently underactuated system. While the angular

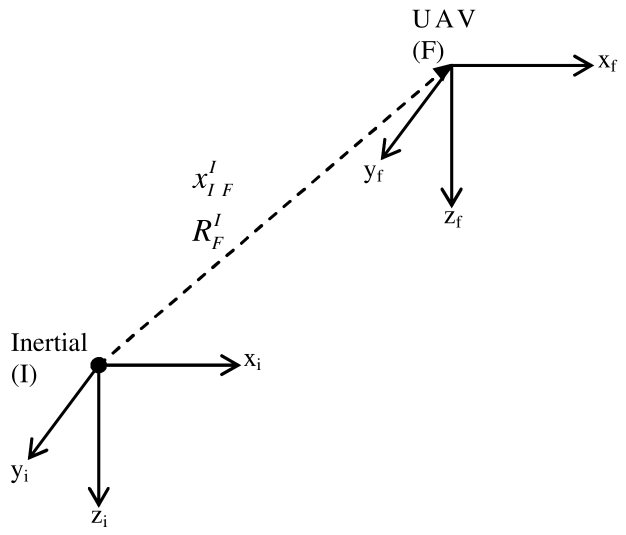


Figure 2.5 The quadrotor helicopter coordinate frames

torques are directly actuated, the translational forces are only directly actuated in the z-direction. The forces and torques are expressed as

$$\begin{array}{rcll} F_f^F & = & \begin{bmatrix} 0 & 0 & u_1 \end{bmatrix}^\intercal & \in \mathbb{R}^3 \\ F_t^F & = & \begin{bmatrix} u_2 & u_3 & u_4 \end{bmatrix}^\intercal & \in \mathbb{R}^3 \end{array} \tag{2.1}$$

where $F_f^F(t)$ refers to the UAV translational forces expressed in the UAV frame F and $F_t^F(t)$ are the UAV torques expressed in the UAV frame, as seen in Figure 2.5.

Rigid body dynamics are used for the UAV dynamics because the quadrotor is a rigid body that can thrust and torque freely in space. The four equations to describe the UAV's rigid body dynamics are [3] in

$$\dot{x}_{IF}^I = R_F^I v_{IF}^F \tag{2.2}$$

$$m\dot{v}_{IF}^F = -mS\left(\omega_{IF}^F\right) v_{IF}^F + N_1(\cdot) + mgR_I^F e_3 + F_f^F \tag{2.3}$$

$$\dot{R}_F^I = R_F^I S\left(\omega_{IF}^F\right) \tag{2.4}$$

$$M\dot{\omega}_{IF}^F = -S\left(\omega_{IF}^F\right) M\omega_{IF}^F + N_2(\cdot) + F_t^F \tag{2.5}$$

where $\dot{x}_{IF}^I(t) \in \mathbb{R}^3$ is the time derivative of the position of the UAV frame with respect to the inertia frame expressed in the inertia frame orientation,

$v_{IF}^{F}(t) \in \mathbb{R}^3$ is the translational velocity of the UAV with respect to the inertia frame, M, expressed in the orientation of the UAV frame, $\omega_{IF}^{F}(t) \in \mathbb{R}^3$ is the angular velocity of the UAV, $R_F^I(t) \in SO(3)$ is the rotational matrix that transforms the vectors from the UAV frame, F, to the inertia frame, g is the gravitational constant, $m \in \mathbb{R}$ is the mass of the UAV, and $M \in \mathbb{R}^{3x3}$ is the constant moment of inertia matrix for the UAV. $S(\cdot) \in \mathbb{R}^{3x3}$ represents a skew symmetric defined as [20]

$$S(\omega) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \text{ where } \omega = [\omega_1, \omega_2, \omega_3]^\intercal \in \mathbb{R}^3. \tag{2.6}$$

Both $N_1\left(x_F^I, R_F^I, v_{IF}^I, \omega_{IF}^I, t\right) \in \mathbb{R}^3$ and $N_2\left(x_F^I, R_F^I, v_{IF}^I, \omega_{IF}^I, t\right) \in \mathbb{R}^3$ are the unmodeled non-linear terms in the translational and rotational dynamics, respectively. Gravity is shown separately in (2.3) so that it can be analyzed separately from the unmodeled dynamics. Out of the dynamics equations, (2.2) is the easiest to understand. The time derivative $\dot{x}_{IF}^I(t)$ is the same as the velocity of the UAV, except for the orientation in which it is expressed. The transformation matrix, $R_F^I(t)$, simply changes the orientation frame, as (1.1) shows. Similarly, (2.4) has to change orientation frames and relate $\omega_{IF}^F(t)$ to the time derivative of $\dot{R}_F^I(t)$. Sometimes a matrix similar to [5]

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} -b & -b & -b & -b \\ 0 & db & 0 & -db \\ db & 0 & -db & 0 \\ k & -k & k & -k \end{bmatrix} \begin{bmatrix} \bar{\omega}_1^2 \\ \bar{\omega}_2^2 \\ \bar{\omega}_3^2 \\ \bar{\omega}_4^2 \end{bmatrix} \tag{2.7}$$

is used where $\bar{\omega}(t)$ are the torques of each rotor on the quadrotor, and $d, b, k \in \mathbb{R}^1$ are constant parameters based on the rotor design and placement. Equation (2.7) describes the relationship between the four rotor torques on the quadrotor and the forces and torques of the quadrotor from (2.1). With the DraganFlyer X-Pro (and most quadrotor RC helicopters) this calculation is done internally and the joystick inputs are mapped to $u(t)$ instead of $\omega(t)$.

Quadrotor Kinematic Model

Many of the equations, such as (2.2)-(2.4), will need either $R_F^I(t)$ or $R_I^F(t)$. While (2.4) expresses how to get $R_F^I(t)$, it involves integrating an $SO(3)$ matrix, which will not yield another $SO(3)$ matrix due to numerical integration method errors. However, the integration can be done on the roll, pitch, and yaw angles. A Jacobian will be required in order to satisfy the equation

$$\omega_{IF}^F = J_F \dot{\Theta}_{IF}^F \tag{2.8}$$

which can then be used to solve for

$$\Theta_F^I = \int_0^t J_F^{-1} \omega_{IF}^F dt \tag{2.9}$$

where $\Theta_F^I(t) \in \mathbb{R}^3$ represents the roll, pitch, and yaw angles between the UAV frame and inertia frame. The Jacobian matrix used, $J_F^{-1}\left(\Theta_F^I(t)\right)$ is defined as [2]

$$J_F^{-1} = \begin{bmatrix} 1 & \sin\psi\tan\theta & \cos\psi\tan\theta \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi/\cos\theta & \cos\psi/\cos\theta \end{bmatrix},\ \Theta_T^I = \begin{bmatrix} \psi \\ \theta \\ \phi \end{bmatrix}. \tag{2.10}$$

As long as the θ term is not near $\pm\frac{\pi}{2}$, the yaw, pitch, roll representation will not reach singularity. To convert between $R_F^I(t)$ and $\Theta_F^I(t)$,

$$R_F^I = \begin{bmatrix} \cos\phi\cos\theta & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \sin\phi\cos\theta & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\phi \\ -\sin\theta & \cos\theta\sin\psi & \cos\theta\cos\psi \end{bmatrix} \tag{2.11}$$

is used [20].

Control Method

Because of the coupling explained in the previous sections, it was decided to make the system control the pitch and roll angles at the expense of controlling the x and y position directly. By controlling these angles, commanding the quadrotor to move forward will be the same as commanding the quadrotor to pitch forward in the pattern displayed in Figure 2.4. Since the yaw angle is not coupled with a direction, it can rotate without affecting the position and the yaw velocity which will facilitate the design of a controller that will allow for the user to command the quadrotor to rotate the yaw at a certain rate instead of to a certain angle.

Since the quadrotor UAV controls its torques directly, it is easier to control the angle. Using a proportional-derivative-integral (PID) controller, the non-linearities from (2.4) and (2.5), primarily the $S\left(\omega_{IF}^{F}\right)M\omega_{IF}^{F}$ and the unmodeled non-linearities $N_2\left(\cdot\right)$, will be ignored. By using the torque $F_t^F\left(t\right)$ as a control signal, a feedback system can be developed based on

$$u = -k_p e - k_d \frac{de}{dt} - k_i \int e \tag{2.12}$$

where $u\left(t\right)$ is the control signal, $e\left(t\right)$ is the error signal, k_p is the proportional gain, k_d is the differential gain, and k_i is the integral gain. The error signal used, $e_\theta\left(t\right) \in \mathbb{R}^3$, consists of

$$e_{roll} = \theta_{roll} - \theta_{rolld} \tag{2.13}$$

$$e_{pitch} = \theta_{pitch} - \theta_{pitchd} \tag{2.14}$$

$$e_{yaw} = \theta_{yaw} - \hat{\theta}_{yawd} \tag{2.15}$$

where $\hat{\theta}_{yawd}\left(t\right)$ is

$$\hat{\theta}_{yawd} = \int \tilde{\theta}_{yawd}. \tag{2.16}$$

Based upon these signals, the feedback signals used are

$$u_1 = \tilde{u}_1 \tag{2.17}$$

$$u_2 = -k_{p\_roll} e_{roll} - k_{d\_roll} \frac{de_{pitch}}{dt} - k_{i\_roll} \int e_{roll} \tag{2.18}$$

$$u_3 = -k_{p\_pitch} e_{pitch} - k_{d\_pitch} \frac{de_{pitch}}{dt} - k_{i\_pitch} \int e_{pitch} \tag{2.19}$$

$$u_4 = -k_{p\_yaw} e_{yaw} - k_{d\_yaw} \frac{de_{yaw}}{dt} - k_{i\_yaw} \int e_{yaw}. \tag{2.20}$$

The desired trajectories that will be generated will create the values for $\tilde{u}_1\left(t\right)$, $\theta_{rolld}\left(t\right)$, $\theta_{pitchd}\left(t\right)$, and $\tilde{\theta}_{yawd}\left(t\right)$.

This approach takes a PID linear approach and applies it to the non-linear system. It is not known if the non-linearities cause any significant instabilities, however the goal of this was to verify the wireless closed looped system will work.

Simulation and Implementation

Quadrotor Model Simulation Parameters

In order to have an accurate simulation of the DraganFlyer X-Pro, certain parameters have to be measured. The mass and inertia matrix are needed for the dynamics equations and the actual maximum thrust and torques produced by the quadrotor are needed to set realistic limits on the simulation.

To measure the mass, the helicopter was weighted using a spring scale. To measure the total force the helicopter can produce, a spring scale is used to measure the amount of force one rotor can create when spinning at maximum speed. This quantity multiplied by four will yield the full thrust ability. In addition, the distance from the rotor to the center of the UAV will yield the total roll and pitch torque that can be generated with one rotor spinning at its maximum velocity while the opposite rotor stopped, as in Figure 2.2.b. A similar measurement was made when two opposite rotors were spinning at maximum speed and the other two rotors were off, as in Figure 2.3.b, to estimate the maximum yaw torque. All of these measurements are displayed in Table 2.1. The inertia matrix was not measured and was estimated based off of [9][7] where the vehicle was half the weight of the DraganFlyer, so the values of the inertia matrix were doubled to

$$M = \begin{bmatrix} 1.3 & 0 & 0 \\ 0 & 1.3 & 0 \\ 0 & 0 & 2 \end{bmatrix} kg \cdot m^2. \tag{2.21}$$

Simulation

To simulate this controller, there are three main tasks that must be implemented as seen in Figure 2.6. The first block is the helicopter dynamics

Table 2.1 DraganFlyer X-Pro Parameters

Parameter	**Value**	**Units**
Mass	2.041	kg
Additional Weight (battery and sensors)	.68	kg
Max Thrust	35.586	N
Max *Roll Torque*	4.067	Nm
Max *Pitch Torque*	4.067	Nm
Max *Yaw Torque*	2.034	Nm

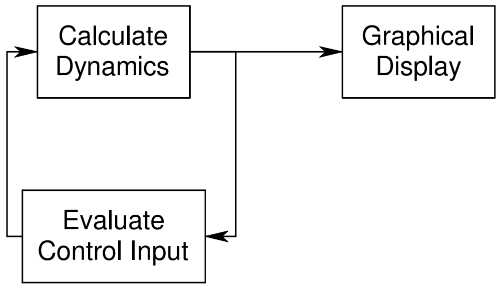


Figure 2.6 Simulation diagram

that must be simulated based on (2.2)-(2.5). Then the control input must be formulated based on sensor readings that come from the dynamics equations. The control input will then be fed back into the dynamics equations to close the loop. The third portion of the simulation is displaying the position and orientation of the quadrotor in a 3D-simulation program called FlightGear.

For calculating the dynamics, equations (2.2)-(2.5) are rewritten as

$$\begin{aligned} \dot{x}_{IF}^{I} &= R_F^I v_{IF}^F & (2.22)\\ \dot{v}_{IF}^{F} &= -S\left(\omega_{IF}^F\right) v_{IF}^F + gR_I^F e_3 + \frac{N_1(\cdot) + F_f^F}{m} & (2.23)\\ \dot{\Theta}_{IF}^{F} &= J_F^{-1}\omega_{IF}^F & (2.24)\\ \dot{\omega}_{IF}^{F} &= M^{-1}\left(-S\left(\omega_{IF}^F\right) J_F\omega_{IF}^F + N_2(\cdot) + F_t^F\right). & (2.25) \end{aligned}$$

Equation (2.2) remains unchanged and (2.3) is divided by m to solve for $\dot{v}_{IF}^F(t)$. Equations (2.4) and (2.5) are replaced with the roll, pitch, yaw version of the equation from (2.8) instead of rotation matrices. Using the yaw, pitch, roll representation results in a singularity at $\theta_{pitch}(t) = \pm\frac{\pi}{2}$. The solution to this problem is to avoid $\pm\frac{\pi}{2}$. The equations are then integrated on both sides using an Adams Integrator at a 1000 Hz update frequency.

The software platform used is QNX Real-Time Operating system [16] running a QMotor program [17] written in C++. The entire program consists of seven parts outlined in Figure 3.12. The program starts by initializing all variables in "Start". Then in "Calculate Dynamics" (2.22)-(2.25) are utilized to find $x_{IF}^{I}(t)$, $v_{IF}^{F}(t)$, $\Theta_{F}^{I}(t)$, and $\omega_{IF}^{F}(t)$. In "Evaluate Control Input" the program uses the trajectories for $\theta_{rolld}(t)$, $\theta_{pitchd}(t)$, $\tilde{\theta}_{yawd}(t)$, and $\tilde{u}_1$ from "Read in joystick values and generate trajectories" to evaluate (2.17)-(2.20). "Saturate Control Inputs Based on UAV Parameters" uses the parameters from Table 2.1 to make sure the control inputs do not exceed realistic values, and if they do, it saturates the value to the maximum limits. A minimum rotor speed is also utilized to prevent the quadrotor coupling peculiarities due to the motors only spinning in one direction. Not only are the singularities prevented, but the effects of the coupling between the different torques are also included in the control inputs. "Update System States" is where all the positions and velocities in the inertia, UAV, and camera frames are computed for use in other calculations in the next iteration. The resulting position and velocity are sent to FlightGear by "Send UDP Packets" and received and shown on the screen by "Virtual Simulation." The code for the QNX program can be found at http://www.ece.clemson.edu/crb/research/uav/simulation.zip. In the simulate.cpp file, the "Send UDP Packets" is accomplished by

d_flight_UDP->PackitSend(*Latitude*, *Longitude*, *Altitute*, *Roll*, *Pitch*, *Yaw*);

where all numbers are in radians except *altitude* which is in meters and d_flight_UDP is of type FlightUDP. A UDP is send to a PC on a specific port defined in the constructor in FlightUDP.cpp as

d_UDP_client=new UDPClient("192.168.1.100",4444,*d_timeout*);

where the UDP packet is sent to the FlightGear IP address, 192.168.1.100 in this case, on port 4444 and *d_timeout* is a struct of type *timeval* set to 50ms. The UDP packet is of type FGNetFDM defined by net_fdm.hxx [1]. The only use for the second computer is the FlightGear simulation, because a simulation such as FlightGear requires| almost all of a computer's CPU, video card, and input/output system [1]. To execute the FlightGear simulator at GSP airport, the following command is used

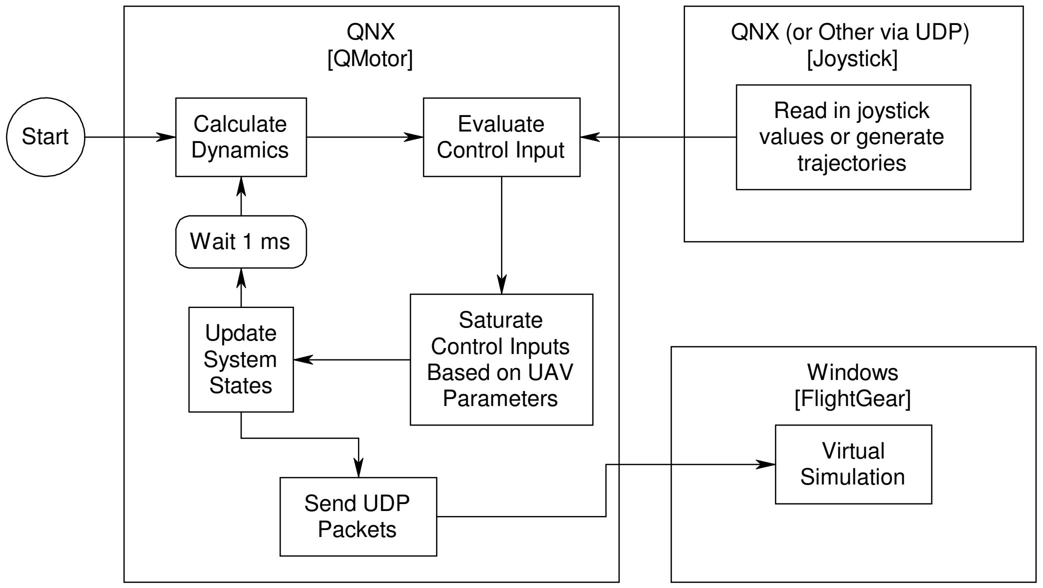


Figure 2.7 Two Computer Simulation Flow

```
fgfs.exe –fg-root="C:\Program Files\FlightGear\data"
–fg-scenery="C:\Program Files\FlightGear\data\Scenery;
C:\Program Files\FlightGear\scenery" –airport-id=KGSP
–aircraft=draganfly –control=joystick –enable-game-mode
–disable-splash-screen –disable-random-objects –disable-sound
–disable-ai-models –disable-clouds –fog-disable
–geometry=1680x1050 –timeofday=noon –fdm=external
–native-fdm=socket,in,20,192.168.1.1,4444,udp
```

where 192.168.1.1 is the IP of the QNX PC sending the UDP packets on port 4444. The rest of the options load the other setting such as environment, location, aircraft model, and resolution.

For the actual experiments, the "Calculate Dynamics" section is replaced with "Read in Sensor Values" to get the actual position and velocity values for the feedback loop.

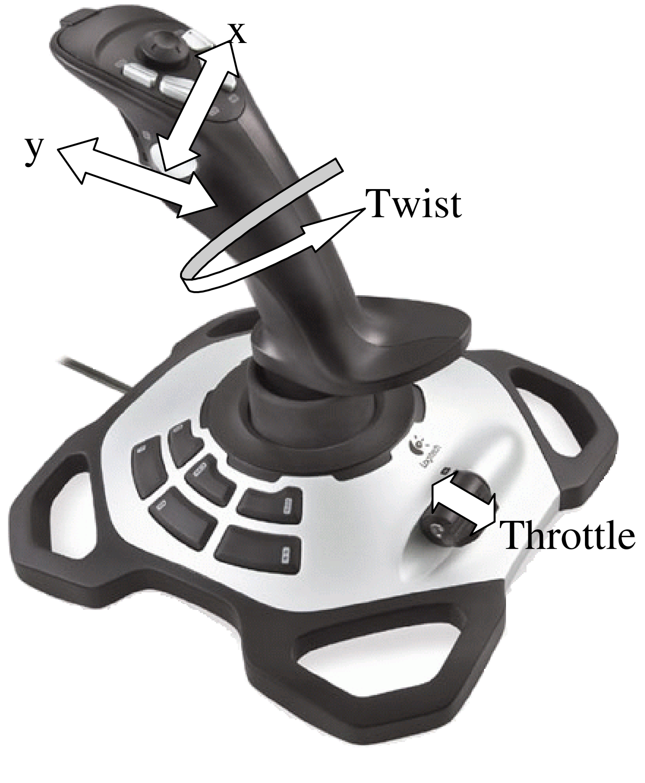


Figure 2.8 Logitech Wingman Extreme 3D Pro joystick

Input

The desired trajectories can be generated by any means. The method used for this experiment used a Logitech Extreme 3D Pro joystick [19], seen in Figure 2.8, to create four inputs labeled as x, y, twist and throttle. The x will be used to generate $\theta_{rolld}\left(t\right)$, y for $\theta_{pitchd}\left(t\right)$, twist for $\tilde{\theta}_{yawd}\left(t\right)$ and throttle for $\tilde{u}_1\left(t\right)$. The x, y, and twist can be easily controlled with one hand while using the other hand for throttle.

FlightGear

FlightGear is an open-source flight simulator [1]. It can run on many computer platforms including Linux and Windows. FlightGear offers a versatile package with multiple input-output (I/O) systems and an interface for importing custom helicopter models. FlightGear is used to display the position and

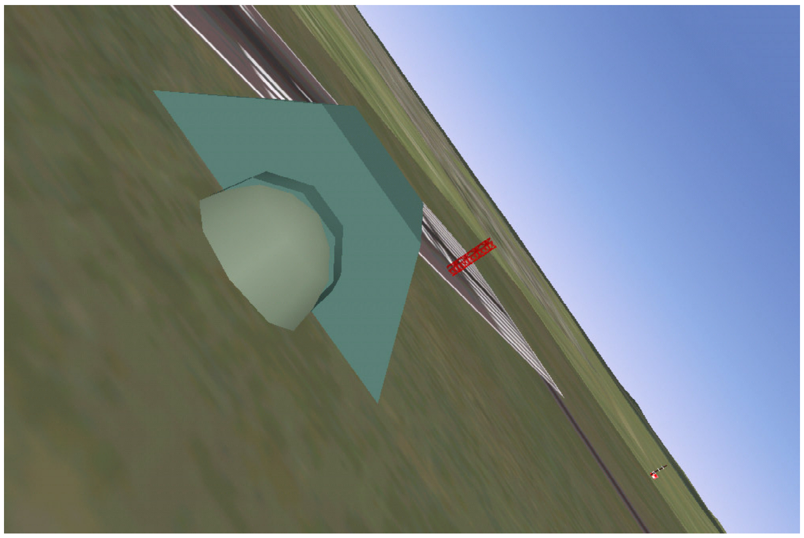


Figure 2.9 First person view of quadrotor while moving left

orientation and show the output in a 3D virtual world using the DraganFlyer X-Pro helicopter model [18].

FlightGear has as least three Flight Dynamics Models (FDM) built in. The user can use any of these FDMs on a real airplane model. However, since this paper uses its own dynamics equations, it was decided to use a custom model using the dynamic equations (2.2)-(2.5) instead of FlightGear's dynamics. FlightGear's I/O system allows it to receive UDP packets of FDM calculations in real-time. This network FDM (net_fdm) allows any computer on the network to perform the dynamics calculations, such as (2.22) through (2.25). This provides a live fully visual representation of what is going on inside the QMotor simulations.

FlightGear is used to show the helicopter during flight. A first person and third person mode on FlightGear will show the quadrotor helicopter and the angles as which it rotates as it moves around. Figure 2.9 shows the first person mode and Figure 2.10 shows an exampled of third person mode. By using FlightGear, a fully visual system can be used to test out the controller.

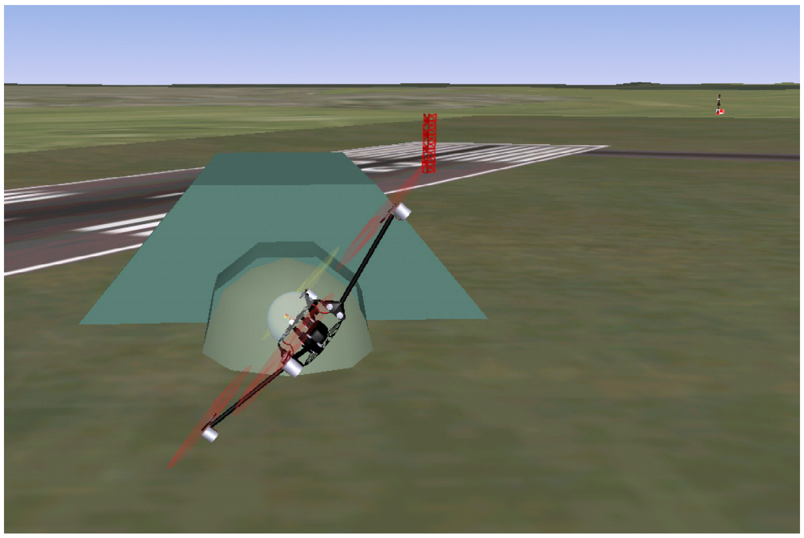


Figure 2.10 External view of quadrotor while moving left

Sensors

For an experiment, a sensor had to be picked so that all the feedback signals required could be measured. After searching though many gyroscope sensors, GPS, and Inertia Measurement Units (IMU), the MIDG II sensor [13] was picked. The MIDG II sensor includes a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer and a Differential Global Position System sensor. The MIGDG II Inertia Navigation System (INS)/GPS uses these sensors to determine its orientation and position in addition to their associated velocities. The sensor was also picked for its light weight of 55 grams. The sensor uses an XTend RS-232/RS-485 RF Modem [14] to transmit to a ground computer. Using software written to receive and parse the MIDG II data, the measurements from the MIDG II can be relayed back to the ground computer.

Using the MIDG II sensors, tests were done to evaluate the GPS sensor on the MIDG II sensor using the ANT-GPS-UC-SMA GPS antenna. The experiment covered in Appendix A discouraged the use of GPS since it was not always getting 5 Hz update frequency, often going seconds with no update at all. Until this problem can be fixed, it would be very difficult to use GPS. Therefore it was decided not to use GPS for feedback in this experiment.

The sensor uses the gyroscopes to measure the angular rates as the sensor rotates about the x-, y-, and z-axes. Mathematically, to get the orientation the sensor must integrate the angular rates. However, there is an unknown initial condition problem with this in addition to a bias drift error that will occur from imperfections in the sensor values. In order to find out the initial conditions and correct for drift, a second method of measurement is required. The accelerometer can measure the gravity vector and can be used to determine roll and pitch angles. To measure the yaw angle, the magnetometers are use to measure which way north is, using north as zero degrees yaw [10]. Using this redundant method, a Kalman filter uses the second method to determine a bias correction to accurately determine the orientation. The accelerometers will introduce additional error if they are accelerating sideways, however this is usually a small error compared to acceleration measured due to gravity. The magnetometer will introduce additional error when it is in the presence of a magnetic field that is not due to the Earth. The main source of this kind of magnetic field is the field generated from the high current of the DraganFlyer X-Pro. This problem cannot be overcome with the DraganFlyer X-Pro design, so a modified firmware was used to enable the magnetometers to determine the initial bias for yaw, then stop using the magnetometers and allow for a small drift error over time (approximately 1 degree per minute), covered in Appendix A.

The MIDG II sensor with a modified firmware will be mounted onto the quadrotor for determining the angle of the quadrotor. The GPS sensor is not used because it was determined that the lapses between updates were too unpredictable for this experiment. The throttle will be controlled manually, since the emphasis of this controller is controlling the three angles.

To complete the feedback loop, wireless transmitters and receivers are used for sending commands to the quadrotor and receiving feedback. To send the commands, the PCM 9XII R/C controller is used. The signals for throttle, roll, pitch, and yaw ($u_1(t)$ through $u_4(t)$) are attached to digital to analog converters (DAC) on a ServoToGo BreakOut Board on a PC. The MIDG II serial communication goes through an XTend modem that also converts from the MIDG II's RS-422 signal to an RS-232 signal. This way only the sensor, transmitter and receiver are mounted onto the quadrotor helicopter, weighing much less than half a pound. All the control calculations are done on the ground PC, allowing for a much more complicated controller algorithms to be implemented in the future.

Experimental Setup

To perform experiments on the quadrotor UAV, an approach similar to the simulation is used, as seen in Figure 2.11. The major difference in the experiment is the "Read Sensors" section. Instead of calculating the dynamics

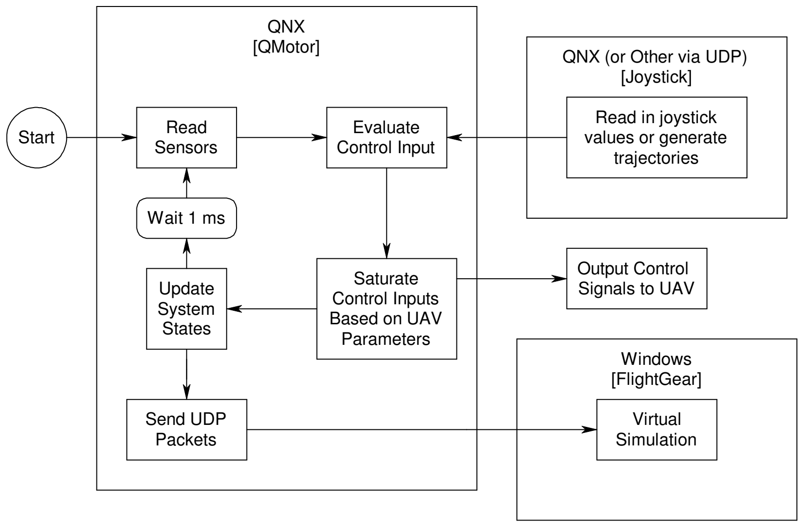


Figure 2.11 Two Computer Experiment Flow

of the quadrotor helicopter based on the dynamic model, the actual orientation of the DraganFlyer X-Pro is read in at 50 Hz using the MIDG II server. The other difference is the control inputs are sent to the DraganFlyer in the "Output Control Signals to UAV." This is accomplished by sending the voltage signals directly to the DraganFlyer remote control on a 0 to 5 volt scale via the Servo-to-go board. The DraganFlyer X-Pro remote uses potentiometers to read in the thumb stick positions seen in Figure 2.12. "Output Control Signals to UAV" takes the control signal and sets the potentiometer signal to the specified voltage, ignoring the actual thumb stick position.

To experiment inside, a trainer is used as seen in Figure 2.13. The DraganFlyer X-Pro is mounted to a dowel stick and joint allowing the UAV to move freely in the z direction, roll, pitch, and yaw. A string potentiometer is used to measure the z position while the MIDG II sensor measures the orientation angles. An array of power supplies are also included in the trainer to allow for the DraganFlyer to be powered without using the Lithium Polymer batteries. As the controller evolves, the trainer is removed and the DraganFlyer is tied down to a weight and allowed to move freely around the room.

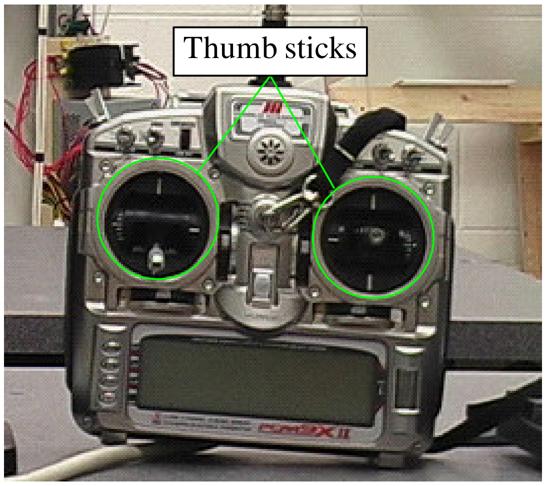


Figure 2.12 DraganFlyer X-Pro Remote Controller

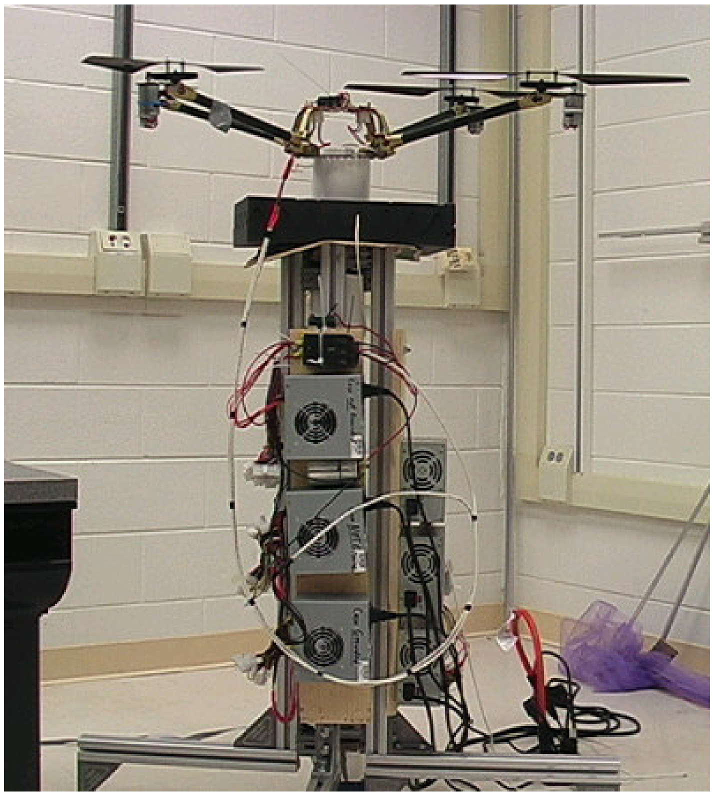


Figure 2.13 DraganFlyer X-Pro Trianer with power supply and vertical sensors

Observations and Results

Simulation

The simulation used the dynamic model of the quadrotor helicopter to test the two-computer simulation system. The gains found in the simulation would not be in the same units of the quadrotor because the simulated dynamics use torque input while the actual quadrotor uses a voltage to control the torques, and little is known about the exact relationship of the voltage to the helicopter rotor speeds. The simulation for the PID shows the simulation works for a PID and can work for a more complicated controller, covered in Chapter 3.

Experiment

In early attempts of calculating $\frac{de_\theta(t)}{dt}$ in (2.18)-(2.20), $\frac{de_\theta(t)}{dt}$ was calculated using

$$\frac{de_\theta}{dt} = \frac{d\theta}{dt} - \frac{d\theta_d}{dt}. \tag{2.26}$$

The gyro values from the MIDG II sensor were used to obtain $\frac{d\theta(t)}{dt}$. However the gyro signals are so noisy, that the feedback signals become extremely noisy. This is probably the reason that most IMU manufacturers use a Kalman filter. Instead of using the gyro values, the numerical derivative of $e_\theta(t)$ is calculated and filtered to obtain $\frac{d\theta(t)}{dt}$. This appears to give the most stable results.

The first experiment done was a flight around a room, using the Logitech joystick for input. The desired velocities can be seen in Figure 2.14 with the roll and pitch varying between $\pm.2$ radians and the yaw varying up to $\pm.5$ radians/sec. In this experiment, the gains are set to $k_p = \begin{matrix} 6 & 6 & 6 \end{matrix}^{\mathsf{T}}$, $k_d = \begin{matrix} .2 & .2 & 2 \end{matrix}^{\mathsf{T}}$, and $k_i = \begin{matrix} .01 & .01 & .1 \end{matrix}^{\mathsf{T}}$. By looking at the error plots of $e_\theta(t)$ in Figure 2.15, it can be seen that the error peaks when there is a sudden change in the desired angles. After each peak in error, it settles in a damped sinusoidal pattern. However, the roll and pitch do not fully attenuate and settle in a sine pattern instead of settling at zero. This was observed when k_p was too high for a given k_d and k_i. This problem was improved in

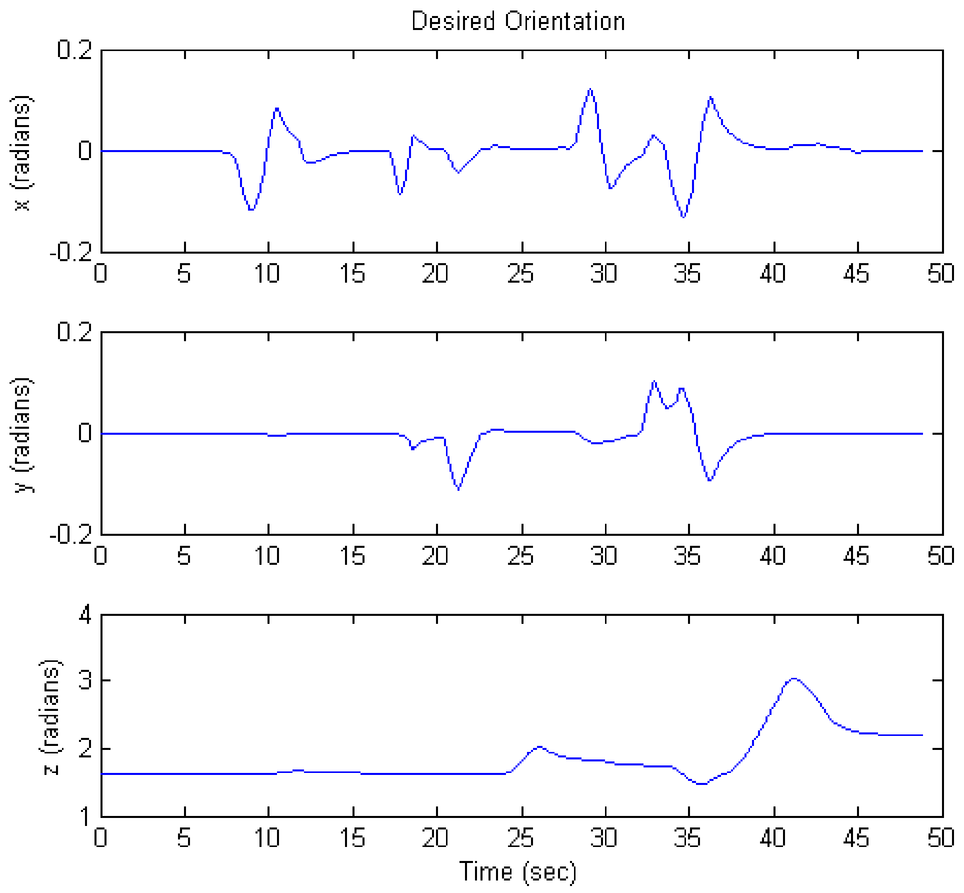


Figure 2.14 Desired orientations of quadrotor: Experiment 1

experiment 2 where the gains were set to $k_p = \;\; 4 \;\; 4 \;\; 2 \;\; ^T$, $k_d = \;\; .5 \;\; .5 \;\; .5 \;\; ^T$, and $k_i = \;\; .5 \;\; .5 \;\; 0 \;\; ^T$. In experiment 2, the desired angles were kept constant while the quadrotor was jolted in midair. To see how the quadrotor reacted with these gains, the error plots in Figure 2.16 show a jolt occurring at 10 seconds, where it takes approximately 3 seconds after the disturbance to settle in the shape of 1 and a half cycles of a damped sinusoid. In this case, the error settles at zero instead of a sine wave. There is a small drifting error in the roll and pitch, but this is on a very small magnitude (less then .2 degrees). Even though the quadrotor is not a linear system, the PID controller works well for the quadrotor in these experiments.

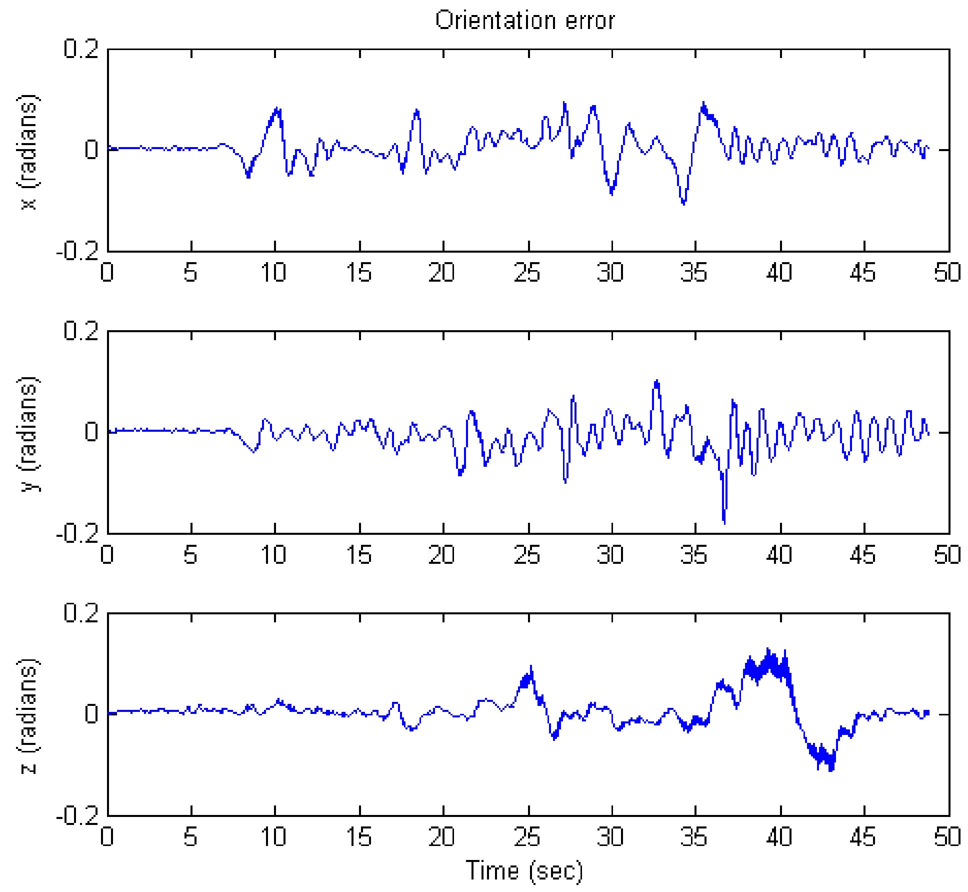


Figure 2.15 Orientation error of quadrotor: Experiment 1

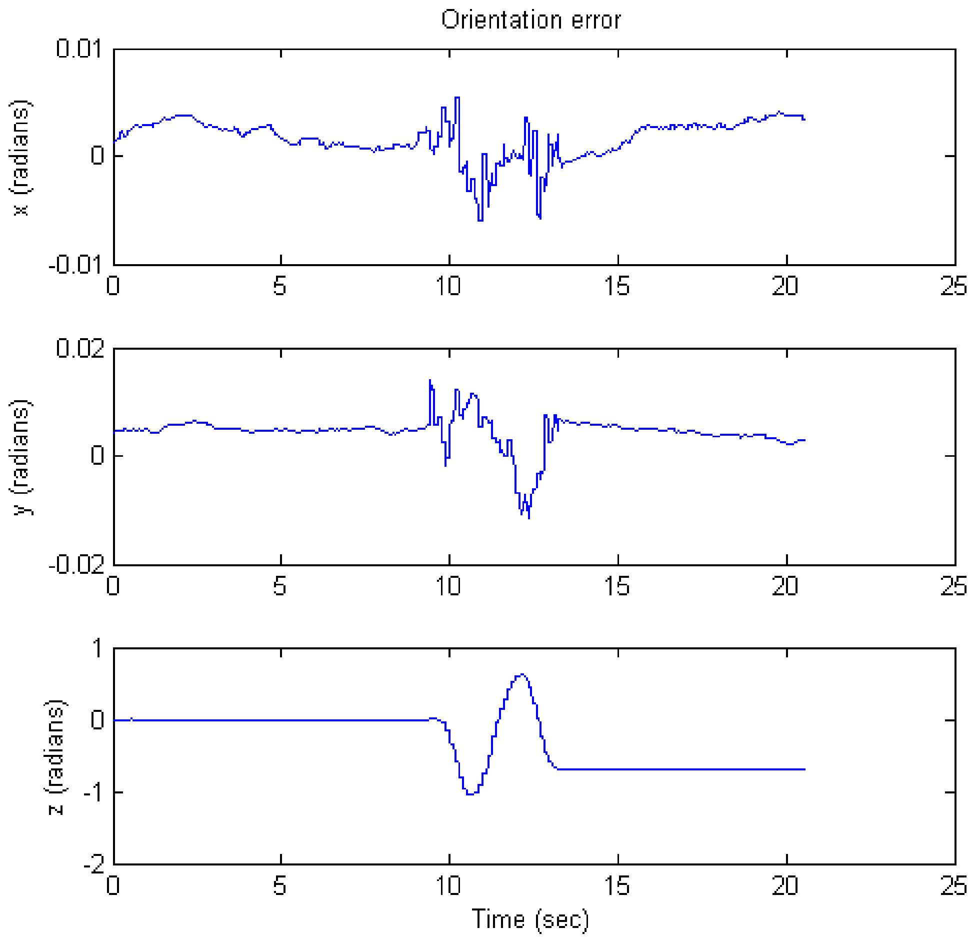


Figure 2.16 Orientation error with disturbances : Experiment 2

Conclusion

This chapter described a PID controller that was used on a quadrotor helicopter to control the orientation of the quadrotor in both simulation and experiment. While the quadrotor is a non-linear system, satisfactory results have nevertheless been achieved for making an angle based controller for the DraganFlyer X-Pro helicopter using the MIDG II sensor for orientation feedback. It is much easier to control the helicopter using the controller compared to open-loop control. Using this angle based control, the DraganFlyer X-Pro can easily be controlled by the pilot, without having to worry about the sensitivity of the controls possibly causing the quadrotor to rotate too much or lose control. The biggest problem is pitching to $\pm\frac{\pi}{2}$ which can be fixed by using quaternions or avoiding that orientation.

The simulation showed that controlling just the angles will allow a pilot to navigate the DraganFlyer X-Pro. Controlling the yaw, pitch, and roll while directly controlling the throttle provides the pilot with a stabilized flight controller. Both the simulation and actual experimentation demonstrated the same feeling that even with an angle controller, the pilot does not need to worry about flipping the quadrotor over, just avoiding the walls or other obstacles.

The experiment conducted on the DraganFlyer X-Pro proves that the wireless feedback system does in fact work. A variety of sensors can be attached to the helicopter and have their signals relayed back to a ground station. This allows for a lighter load on the quadrotor because it does not need a computer to execute the controller. Wireless feedback raises the level of controller complexity so that anything that can be executed on a powerful PC computer is possible, such as a complex non-linear controller.

CHAPTER 3
NON-LINEAR CONTROL

This chapter is organized into eight sections. The chapter starts out with an introduction followed by the motivation for this controller. In Section 3 the system models are described, including the UAV dynamics and UAV and camera kinematics models used for modeling the UAV/camera system. Section 4 covers the development and Lyapunov proof for the non-linear controller that will achieve Globally Uniformly Ultimately Bounded (GUUB) tracking. The following sections cover the simulation system used and the results obtained from the simulation. Finally, the conclusion is followed by idea for future work.

Introduction

Unmanned Aerial Vehicles (UAVs) are well suited for a variety of tasks. The quadrotor unmanned aerial vehicle (UAV) can be used for civilian and military surveillance and inspection tasks by attaching a camera to it. They can go places too dangerous for humans and in places too small for a person [12]. The images may be the objective or they are often used for feedback in the control itself [4]. Unfortunately, the images seen by the camera will always depend on the UAV and its orientation. A way to separate the camera from the UAV would have many advantages, allowing for a freer camera view.

The quadrotor has a six degrees-of-freedom (DOF) rigid body that is positioned by changing the relative speed of the four rotors. These speed differences can produce torques about the roll, pitch, and yaw axes in addition to the thrust produced as the sum of the four rotating blades. Since the helicopter is underactuated, it is only able to translate in one direction, up and down, while rotating about all three axes. The remaining two translational axes depend on the upward force coupled with the orientation of the UAV.

If a camera is to be mounted onto a quadrotor, then only four DOFs of the camera can be controlled, making performing surveillance and inspection tasks challenging. The camera will always depend on the orientation of the UAV. It is possible to use an actuated camera system to cancel out any undesirable rotations of the camera while the quadrotor performs its task. This will allow the camera to focus on its objective and not worry about how the quadrotor achieves its movements. By putting two actuators on the camera, the final position and orientation of the camera becomes a fully actuated six DOF system. With this system, the inputs choose to move and rotate the camera frame no matter what the quadrotor's orientation is, creating an intuitive way of flying the quadrotor by the camera view.

This approach to the control problem was presented in [5] where the UAV and the camera positioning unit are considered to be a single robotic unit. From this perspective a controller can be developed which will simultaneously control both the UAV and the camera positioning unit in a complementary fashion. Here, this control approach is exploited to provide a new perspective for piloting the UAV. This perspective, which shall be referred to as the fly-by-camera perspective, presents a new interface to the pilot. In this proposed approach, the pilot commands motion from the perspective of the on-board camera. It is as though the pilot is riding on the tip of the camera and commanding movement of the camera a la a six DOF flying camera. This is subtly different from the traditional remote control approach wherein the pilot processes the camera view and then commands an aircraft motion to create a desired motion of the camera view.

This chapter will cover a non-linear controller developed using Lyapunov stability methods. A two-DOF camera is mounted on the UAV. This camera combined with a four degree-of-freedom UAV makes a fully actuated six degrees-of-freedom system. Desired velocities are given relative to the camera frame, creating a fly-by-camera interface. When the desired input says go left, what is seen on the camera will go left, regardless of orientation. The camera will use the UAV to move to its left in combination with actuating the camera when necessary.

This controller will use only translational and rotational velocities for feedback. This will satisfy the limitations of the MIDG II sensor [13]. Although the differential GPS (DGPS) system on the MIDG II sensor has the ability to measure position, the velocity of a GPS sensor is much more reliable and accurate and is discussed in Appendix A. Therefore it is interesting to consider a controller using only velocities for the feedback.

For this controller, it is assumed that the quadrotor's angular and translational velocities are measurable with respect to a fixed point (the inertia frame). The angles of the camera with respect to the quadrotor are also needed, easily provided by a servo-controlled camera or encoders. The last measurable quantity is gravity, since small variations in g would show up in the error result. It is also required that the desired velocities be continuous and differentiable. Using these types of sensors in this controller, it will be shown that this system is Globally Uniformly Ultimately Bounded (GUUB) and the simulations will show the ease of flight that a fly-by-camera interface can offer.

Motivation

There are many advantages to this fly-by-camera interface. The first advantage to this system is the ease of choosing a desired trajectory. A human operator can simply look at the camera images and choose where to have the pictures go. This also means visual contact with the UAV is no longer necessary, as long as there is communication with the craft. To see above or to the right of the image, all that is necessary is to tell the controller up or right. Everything from actuating the camera to controlling the rotor speeds is taken care of by the controller for all six DOFs of the camera.

Another advantage to this system is the independence of the camera and the UAV. As seen in Figure 3.1, a typical quadrotor maneuver is demonstrated. A hovering quadrotor seen in Figure 3.1.a will tilt to Figure 3.1.b in order to move forward. Then the quadrotor will tilt back momentarily to come to a

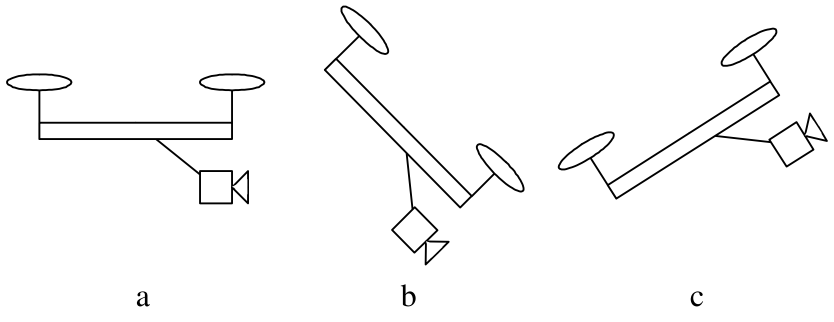


Figure 3.1 A fixed camera mounted on the front of a moving UAV

stop as seen in Figure 3.1.c and then finally back to Figure 3.1.a. The whole time, the motion of the camera is directly tied to the motion of the UAV and the camera has to look up and down when all that is desired is to look straight ahead. It is not possible to simultaneously specify the attitude of the camera and the attitude of the aircraft.

In contrast, the moveable camera depicted in Figure 3.2 maintains the same orientation with minimal deviation regardless of the UAV orientation. The UAV and camera motions must be coordinated in order to keep the camera pointed in a particular direction. The traditional approach has been to have a pilot position the aircraft about a target and have a camera operator position the camera with the sub-task of compensating for motion of the aircraft. The difficulty of coordinating pilot and camera tasks is a big motivation for a 6 DOF camera frame.

System Model

DraganFlyer Quadrotor Overview

Figure 3.4 displays the different effects of certain rotor combinations. The depiction in Figure 3.4.a shows all four rotors spinning at an equal rate which results in an upward force in the z-direction. Since the rotors on the DraganFlyer X-Pro can only spin in one direction, the forces from each rotor and the sum of all four rotors will always be added up in the negative z-direction,

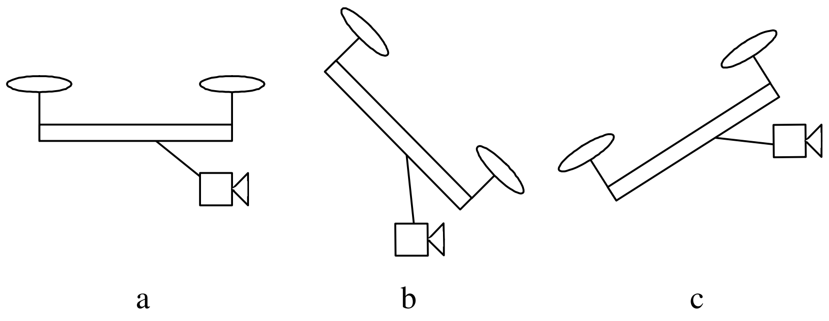


Figure 3.2 An actuated camera mounted on the front of a moving UAV

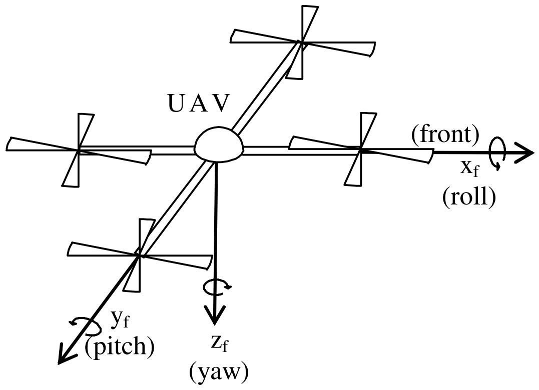


Figure 3.3 Yaw, pitch, and roll definitions.

according to Figure 3.3. If all the rotors spin faster then the craft will rise and if all spin slower then the craft will settle.

The intriguing aspect of a quadrotor is the manner in which the torques, that can be used to move the quadrotor, are generated. The four rotors can be grouped into two sets, group A consisting of the front and back rotors and group B consisting of the left and right rotors. Both rotors in group A spin counter-clockwise while both rotors in group B spin clockwise, shown in Figure 3.4. Pitch will be defined as rotation about the y-axis, roll as rotation about the x-axis and yaw as rotation about the z-axis, as seen in Figure 3.3. To achieve pitch torque, the front and back rotors in group A must spin at different speeds. To pitch clockwise, the front rotor speed is decreased and the rear rotor speed increased while keeping the left and right rotors in group B constant, as depicted in Figure 3.4.b. The front rotor is increased and the back rotor is equally decreased so that the total sum of the four rotor forces remain the same. The same method is used for generating a roll torque in the clockwise direction as seen in Figure 3.4.c. The third body torque is applied using a different method; instead of using the thrusting forces of the rotors as done for roll and pitch, rotating in the yaw direction uses torque

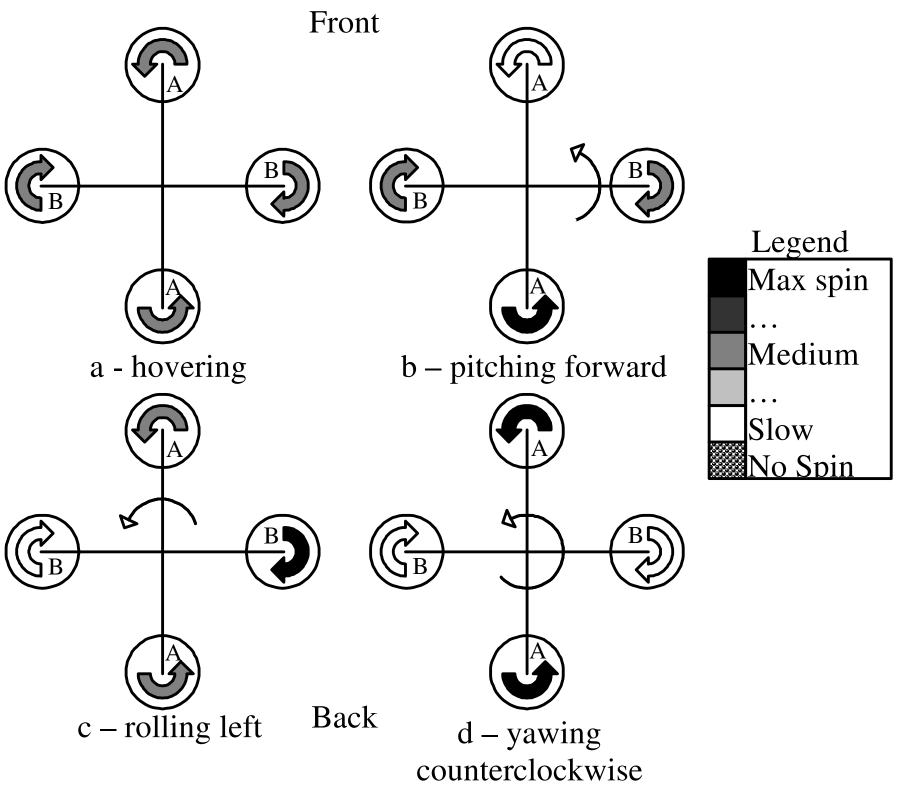


Figure 3.4 Quadrotor method of applying torques to produce motion.

couples. Since group A spins counter-clockwise and group B spins clockwise, the quadrotor creates a clockwise couple and counter-clockwise couple. When all four rotors spin at the same speed, the couples cancel out and there is no yaw rotation. But when group B slows down, and group A speeds up, there will be a clockwise rotation for Figure 3.4.d.

While this explains how the four degrees-of-freedom are controlled, it is still possible for the quadrotor to move in the x- and y-direction with respect to a fixed inertia frame. To move forward in the x-direction, the quadrotor must first pitch clockwise. This will redirect the UAV's upward thrust force in the forward direction, moving it forward. The UAV will then pitch counter-clockwise to stop movement and become level again, as depicted in Figure 3.1. The same is true for left and right. This is the coupling between the translational and rotational velocities.

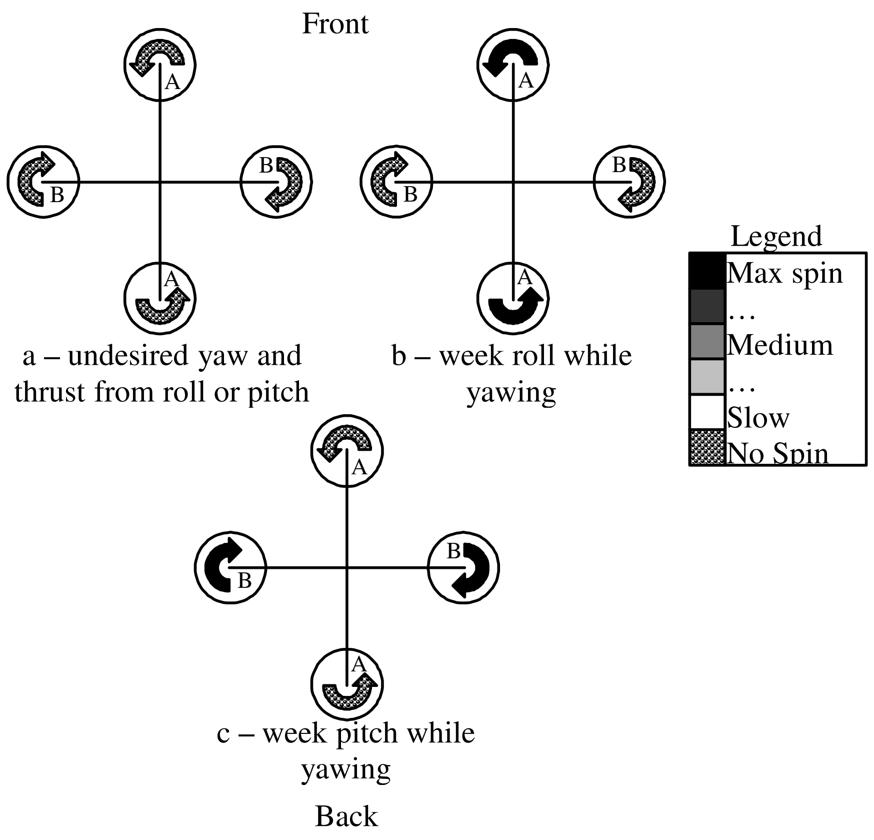

Front
A
B
B
A
a – undesired yaw and thrust from roll or pitch
A
B
B
A
b – week roll while yawing
Legend
Max spin
…
Medium
…
Slow
No Spin
A
B
B
A
c – week pitch while yawing
Back

Figure 3.5 Peculiarities resulting from mono-directional rotor motors and blades.

The DraganFlyer X-Pro is designed for each rotor to spin in one direction only. Because of this restriction, there are certain situations in which all torques cannot be arbitrarily applied. The first example in Figure 3.5.a is when all four rotors are stopped. If roll is the desired torque, one motor cannot be decreased while the opposite is increased as an undesired yaw force will be introduced. This yaw force can be cancelled out with the other two rotors, but then an undesired upward force is generated. If the motors could spin backwards, then the roll torque could be achieved by simply spinning the left rotor and right rotor in opposite directions. It is rare to have all four rotors stopped while flying. The example in Figure 3.5.b is far more likely. When trying to achieve a large yaw torque, two of the motors will shut off, making roll impossible or extremely weak. In order to apply a roll torque, speeding up the left rotor only will start creating a stray yaw torque in addition to an additional upward force. The same is true for pitch in Figure 3.5.c. One method to prevent such a situation is to set a minimum rotor speed. That way the quadrotor can still apply at least a small amount of torque in the roll, pitch, and yaw directions without introducing undesired forces and torques.

As stated, the quadrotor is underactuated, although it is still free to move in all of its six degrees-of-freedom (DOF). An example of quadrotor motion is shown in Figure 3.6. In Figure 3.6.a, the quadrotor is hovering. To move in the x-direction, the quadrotor must pitch clockwise to direct a component of the thrust in the forward direction as seen in Figure 3.6.b. To come to a stop, the quadrotor must pitch back as seen in Figure 3.6.c to bring the quadrotor's velocity down to zero. Once the quadrotor horizontal motion has stopped, it returns to the horizontal state, Figure 3.6.a. This is the coupling between the pitch angle and the x-direction which is used to move in the forward direction. The same coupling is seen between the roll angle and the y-direction. Note that the rotor speeds must increase in Figures 3.6.b and 3.6.c above Figure 3.6.b as the thrust component that counters gravity is reduced.

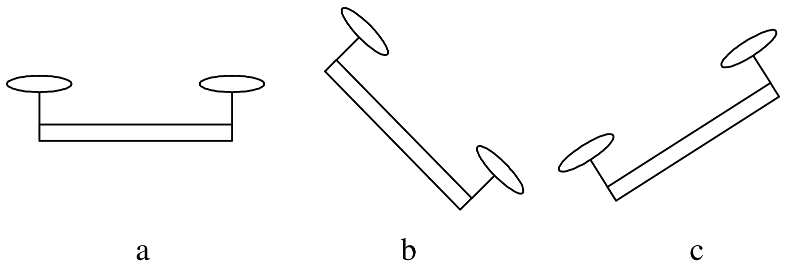


Figure 3.6 Series of motions the quadrotor executes while moving

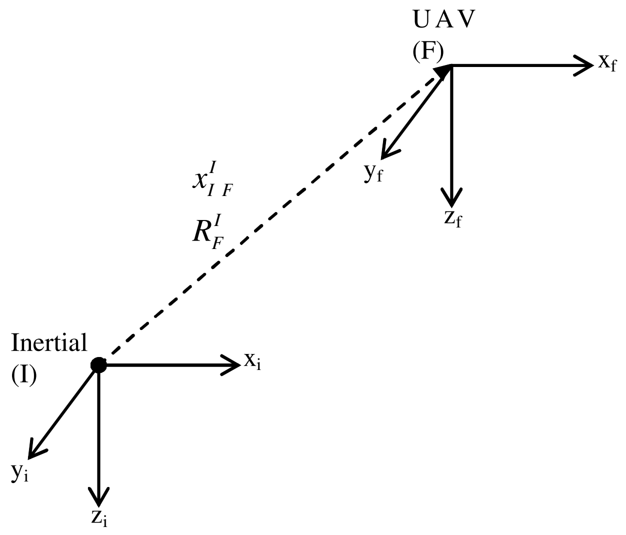


Figure 3.7 The quadrotor helicopter coordinate frames

Quadrotor Dynamics Model

As discussed above, the quadrotor UAV, such as the DraganFlyer X-Pro quadrotor [18], is an inherently underactuated system. While the angular torques are directly actuated, the translational forces are only directly actuated in the z-direction. The forces and torques are expressed as

$$\begin{array}{rcll} F_f^F & = & \begin{bmatrix} 0 & 0 & u_1 \end{bmatrix}^\intercal & \in \mathbb{R}^3 \\ F_t^F & = & \begin{bmatrix} u_2 & u_3 & u_4 \end{bmatrix}^\intercal & \in \mathbb{R}^3 \end{array} \tag{3.1}$$

where $F_f^F(t)$ refers to the UAV translational forces expressed in the UAV frame F and $F_t^F(t)$ are the UAV torques expressed in the UAV frame, as seen in Figure 3.7.

Rigid body dynamics are used for the UAV dynamics because the quadrotor is a rigid body that can thrust and torque freely in space. The four equations to describe the UAV's rigid body dynamics are [3]

$$\dot{x}^I_{IF} = R^I_F v^F_{IF} \tag{3.2}$$

$$m\dot{v}^F_{IF} = -mS\left(\omega^F_{IF}\right)v^F_{IF} + N_1(\cdot) + mgR^F_I e_3 + F^F_f \tag{3.3}$$

$$\dot{R}^I_F = R^I_F S\left(\omega^F_{IF}\right) \tag{3.4}$$

$$M\dot{\omega}^F_{IF} = -S\left(\omega^F_{IF}\right)M\omega^F_{IF} + N_2(\cdot) + F^F_t \tag{3.5}$$

where $\dot{x}_{IF}^{I}(t) \in \mathbb{R}^3$ is the time derivative of the position of the UAV frame with respect to the inertia frame expressed in the inertia frame orientation, $v_{IF}^{F}(t) \in \mathbb{R}^3$ is the translational velocity of the UAV with respect to the inertia frame, M, expressed in the orientation of the UAV frame, $\omega_{IF}^{F}(t) \in \mathbb{R}^3$ is the angular velocity of the UAV, $R_F^I(t) \in SO(3)$ is the rotational matrix that transforms the vectors from the UAV frame, F, to the inertia frame, g is the gravitational constant, $m \in \mathbb{R}$ is the mass of the UAV, and $M \in \mathbb{R}^{3x3}$ is the constant moment of inertia matrix for the UAV. $S(\cdot) \in \mathbb{R}^{3x3}$ represents a skew symmetric defined as [20]

$$S(\omega) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \text{ where } \omega = [\omega_1, \omega_2, \omega_3] \in \mathbb{R}^3. \tag{3.6}$$

Both $N_1\left(R_F^I, v_{IF}^I, t\right) \in \mathbb{R}^3$ and $N_2\left(x_F^I, R_F^I, v_{IF}^I, \omega_{IF}^I, t\right) \in \mathbb{R}^3$ are the bounded unmodeled non-linear terms in the translational and rotational dynamics, respectively. Gravity is shown separately in (3.3) so that it can be analyzed separately from the unmodeled dynamics. Out of the dynamics equations, (3.2) is the easiest to understand. The time derivative $\dot{x}_{IF}^{I}(t)$ is the same as the velocity of the UAV, except for the orientation in which it is expressed. The transformation matrix, $R_F^I(t)$, simply changes the orientation frame, as (1.1) shows. Similarly, (3.4) has to change orientation frames and relate $\omega_{IF}^{F}(t)$ to the time derivative of $\dot{R}_F^I(t)$. Out of the four dynamics equations (3.2)-(3.5), (3.5) is not used in this nonlinear controller. Instead, $\omega_{IF}^{F}(t)$ is calculated directly without modeling the angular dynamics. This is done because the quadrotor applies torques in a very direct way which is like controlling $\omega_{IF}^{F}(t)$ directly, making the normal backstepping process not necessary in this controller. Sometimes a matrix similar to [5]

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} -b & -b & -b & -b \\ 0 & db & 0 & -db \\ db & 0 & -db & 0 \\ k & -k & k & -k \end{bmatrix} \begin{bmatrix} \bar{\omega}_1^2 \\ \bar{\omega}_2^2 \\ \bar{\omega}_3^2 \\ \bar{\omega}_4^2 \end{bmatrix} \tag{3.7}$$

is used where $\bar{\omega}(t)$ are the torques of each rotor on the quadrotor, $d, b, k \in \mathbb{R}^1$ are constant parameters based on the rotor design and placement. Equation

(3.7) describes the relationship between the four rotor torques on the quadrotor and the forces and torques of the quadrotor from (3.1). However, with the DraganFlyer X-Pro (and most quadrotor RC helicopters) this calculation is done internally and the joystick inputs are mapped to $u(t)$ instead of $\omega(t)$.

Quadrotor Kinematic Model

Many of the equations, such as (3.2)-(3.4), will need either $R_F^I(t)$ or $R_I^F(t)$. While (3.4) expresses how to get $R_F^I(t)$, it involves integrating an $SO(3)$ matrix, which will not yield another $SO(3)$ matrix due to numerical integration method errors. However, the integration can be done on the roll, pitch, and yaw angles. A Jacobian will be required in order to satisfy the equation

$$\omega_{IF}^F = J_F \dot{\Theta}_{IF}^F \tag{3.8}$$

which can then be used to solve for

$$\Theta_F^I = \int_0^t J_F^{-1} \omega_{IF}^F dt \tag{3.9}$$

where $\Theta_F^I(t) \in \mathbb{R}^3$ represents the roll, pitch, and yaw angles between the UAV frame and inertia frame. The Jacobian matrix used, $J_F^{-1}\left(\Theta_F^I(t)\right)$ is defined as [2]

$$J_F^{-1} = \begin{bmatrix} 1 & \sin\psi\tan\theta & \cos\psi\tan\theta \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi/\cos\theta & \cos\psi/\cos\theta \end{bmatrix}, \; \Theta_T^I = \begin{bmatrix} \psi \\ \theta \\ \phi \end{bmatrix}. \tag{3.10}$$

As long as the θ term is not near $\pm\frac{\pi}{2}$, the yaw, pitch, roll representation will not reach singularity. To convert between $R_F^I(t)$ and $\Theta_F^I(t)$,

$$R_F^I = \begin{bmatrix} \cos\phi\cos\theta & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \sin\phi\cos\theta & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi \\ -\sin\theta & \cos\theta\sin\psi & \cos\theta\cos\psi \end{bmatrix} \tag{3.11}$$

is used [20].

Table 3.1 Actuations for all six degrees of freedom

Camera Type	Craft Action	Rotations			Translations		
		x_c	y_c	z_c	x_c	y_c	z_c
Tilt-Roll (Front)	UAV	Yaw	0	0	+Thrust	+Roll	-Pitch
	Camera	0	Tilt	Roll	0	-Roll	+Tilt
Pan-Tilt (Bottom)	UAV	0	0	Yaw	-Pitch	+Roll	-Thrust
	Camera	Pan	Tilt	0	+Tilt	-Pan	0

Camera Kinematics

As stated, the quadrotor can thrust in the z-direction, but it cannot thrust in the x- or y-directions. Since the quadrotor helicopter is underactuated in two of its translational velocities, a two actuator camera is added to achieve six DOF control in the camera frame.

There are two ways in which this is done. The first method is to add a tilt-roll camera to the front of the helicopter seen in Figure 3.8. The second method is to add a pan-tilt camera to the bottom of the quadrotor seen in Figure 3.9. With the new camera frame, there are now three rotations and three translations, a total of six DOFs, to actuate. To control any of the DOFs, either the camera must move, the UAV must move, or both. For example, to move in the positive x_c-direction on the tilt-roll configuration in Figure 3.8, the quadrotor thrust must be increased. To move in the positive z_c-direction on the pan-tilt configuration in Figure 3.9, the quadrotor thrust should be decreased. Table 3.1 demonstrates all of the ways the DOFs can be actuated from an initial orientation. Table 3.1 should be read as to translate in the y_c-direction, in Tilt-Roll configuration, the roll of the UAV must increase and the roll of the camera must decrease.

Figure 3.10 shows a 3-link model for a pan tilt roll camera. This model will represent both the camera seen in Figure 3.8 with $\theta_{pan} = 0$ and the camera seen in Figure 3.9 with $\theta_{roll} = 0$. O_0 represents the origin at the base of the camera, known as base B. O_3 is the camera frame denoted C. The Denavit-Hartenberg table for Figure 3.10 is shown in Table 3.2

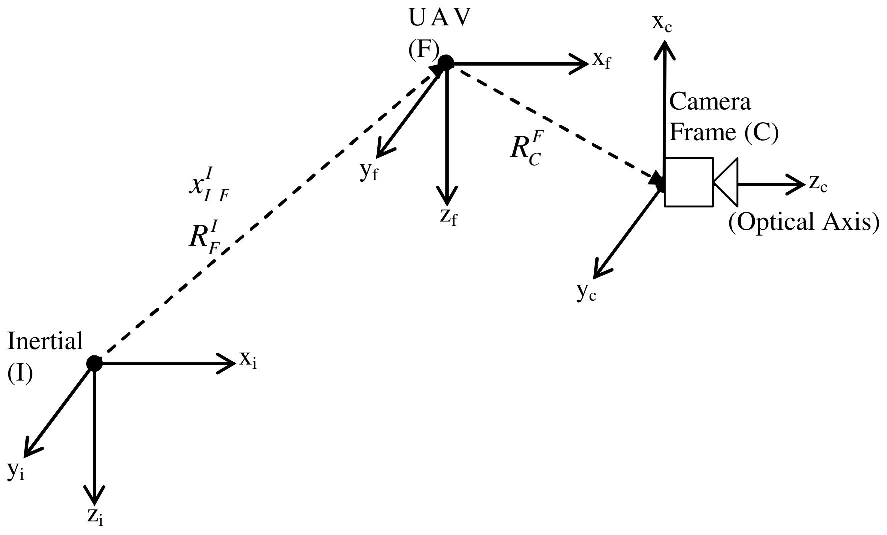


Figure 3.8 The quadrotor helicopter with a tilt-roll in front.

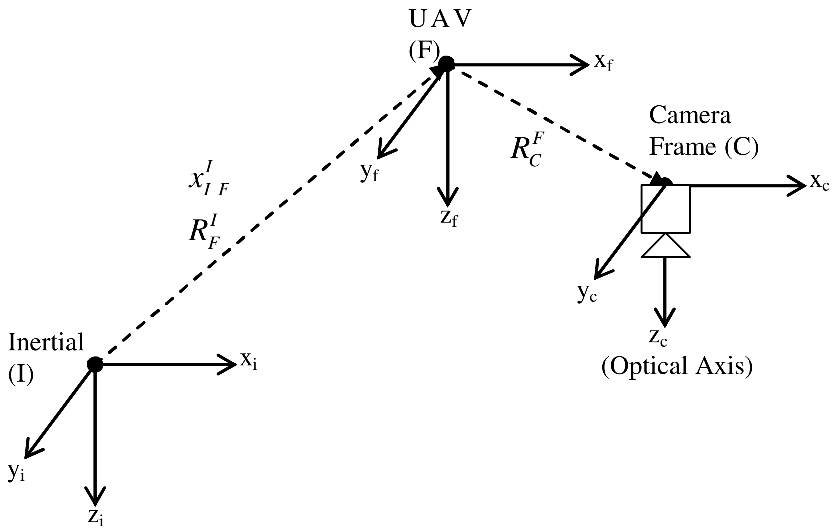


Figure 3.9 The quadrotor with a pan-tilt on the bottom.

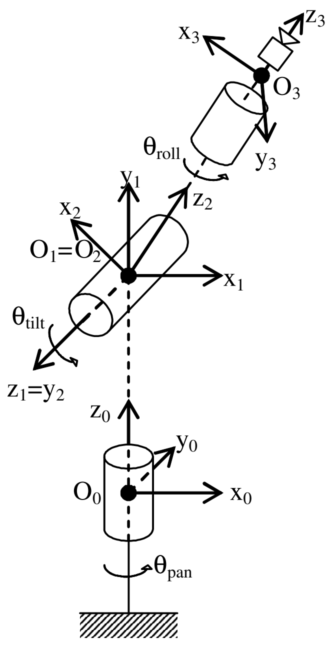


Figure 3.10 Kinematics for pan tilt roll camera

Table 3.2 Denavit-Hartenburg table for 3-link camera

Link	**d (offset)**	**a (length)**	α (twist)	θ (angle)
1	0	0	90°	$\theta_{pan}(t)$
2	0	0	90°	$90°+\theta_{tilt}(t)$
3	0	0	0°	$\theta_{roll}(t)$

The rotation matrix generated by Table 3.2 is

$$R_C^B = \begin{bmatrix} -\cos\theta_p \sin\theta_t \cos\theta_r + \sin\theta_p \sin\theta_r & \cos\theta_p \sin\theta_t \sin\theta_r + \sin\theta_p \cos\theta_r & \cos\theta_p \cos\theta_t \\ -\sin\theta_p \sin\theta_t \cos\theta_r - \cos\theta_p \sin\theta_r & \sin\theta_p \sin\theta_t \sin\theta_r - \cos\theta_p \cos\theta_r & \sin\theta_p \cos\theta_t \\ \cos\theta_t \cos\theta_r & -\cos\theta_t \sin\theta_r & \sin\theta_t \end{bmatrix} \tag{3.12}$$

and the rotation matrix from the camera from to the UAV frame can be expressed as

$$R_C^F = R_B^F R_C^B \tag{3.13}$$

and the angular rate expressed as

$$\omega_{FC}^F = \overbrace{\omega_{FB}^F}^{0} + \omega_{BC}^F. \tag{3.14}$$

$\omega_{FB}^F = 0$ because $R_B^F \in SO(3)$ is a constant rotation matrix describing the orientation in which the camera's base is mounted onto the UAV.

To get the Jacobian matrix, use

$$J_C = \begin{bmatrix} z_0 & z_1 & z_2 \end{bmatrix} \in \mathbb{R}^{3\times 3} \tag{3.15}$$

where $z_0(t)$, $z_1(t)$, and $z_2(t) \in \mathbb{R}^3$ are the z-axes from Figure 3.10 [20]. Using Table 3.2 to get the z-axes, (3.15) becomes

$$J_C = \begin{bmatrix} 0 & \sin\theta_p & \cos\theta_p \cos\theta_t \\ 0 & -\cos\theta_p & \sin\theta_p \cos\theta_t \\ 1 & 0 & \sin\theta_t \end{bmatrix}. \tag{3.16}$$

Tilt-Roll Camera on the front of the UAV

For the first camera method seen in Figure 3.8, the rotation matrix between the UAV frame and camera base frame is generated by inverting the signs on the y and z axes, arriving at

$$R_B^F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{3.17}$$

Substituting (3.17) and (3.12) into (3.13) while setting $\theta_p = 0$ yields

$$R_C^F = \begin{bmatrix} -\sin\theta_t \cos\theta_r & \sin\theta_t \sin\theta_r & \cos\theta_t \\ \sin\theta_r & \cos\theta_r & 0 \\ -\cos\theta_t \cos\theta_r & \cos\theta_t \sin\theta_r & -\sin\theta_t \end{bmatrix}. \tag{3.18}$$

Since only two of the angles vary, the Jacobian for this method is expressed as

$$J_{C(front)} = \begin{bmatrix} 0 & \cos\theta_t \\ 1 & 0 \\ 0 & -\sin\theta_t \end{bmatrix} \tag{3.19}$$

and finally

$$\omega_{FC}^F = J_{C(front)}\dot{\theta}_{C(front)},\ \dot{\theta}_{C(front)} = \begin{bmatrix} \dot{\theta}_t & \dot{\theta}_r \end{bmatrix}^\intercal \in \mathbb{R}^2 \tag{3.20}$$

which facilitates the calculation of the angles of the camera when mounted in front.

Pan-Tilt Camera on the Bottom of the UAV

For the second camera method seen in Figure 3.9, the rotation between the UAV frame and base frame of the camera is generated by switching the x and z axes and inverting the y to arrive at

$$R_B^F = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}. \tag{3.21}$$

Substituting (3.21) and (3.12) into (3.13) while setting $\theta_r = 0$ yields

$$R_C^F = \begin{bmatrix} \cos\theta_t & 0 & \sin\theta_t \\ \sin\theta_p \sin\theta_t & \cos\theta_p & -\sin\theta_p \cos\theta_t \\ -\cos\theta_p \sin\theta_t & \sin\theta_p & \cos\theta_p \cos\theta_t \end{bmatrix}. \tag{3.22}$$

Since only two of the angles vary, the Jacobian for this method is expressed as

$$J_{C(bottom)} = \begin{bmatrix} 1 & 0 \\ 0 & \cos\theta_p \\ 0 & \sin\theta_p \end{bmatrix} \tag{3.23}$$

and finally

$$\omega_{FC}^{F} = J_{C(bottom)}\dot{\theta}_{C(bottom)},\ \dot{\theta}_{C(bottom)} = \begin{bmatrix} \dot{\theta}_p & \dot{\theta}_t \end{bmatrix}^{\intercal} \in \mathbb{R}^2 \tag{3.24}$$

which facilitates the calculation of the angles of the camera when mounted on the bottom.

Control Method

Error Formulation

Translational Velocity Error

It has been proposed to make a non-linear controller that will control all six DOFs of the camera frame. The control inputs will be the six desired translational and angular velocities and will only use velocities as feedback in the control loop. This is a significant detail because it is difficult to get reliable and accurate position values even with GPS. Although the GPS velocities are more accurate, it may be possible to get velocity information from accelerometers. Positional information based on accelerometers, however, would be terribly unreliable.

The control system development begins by defining the velocity error between the desired camera velocity, $v_{ICd}^{C} \in \mathbb{R}^3$, and the actual velocities in the camera frame,

$$e_v \triangleq v_{IC}^{C} - v_{ICd}^{C}. \tag{3.25}$$

Using the velocity error, an auxiliary signal is defined as

$$r_v \triangleq e_v + R_F^C \delta, \tag{3.26}$$

where δ will be used to couple the velocity error with the angular rates. This coupling term mathematically says "When there is velocity error, rotate the UAV in order to speed up in that direction and correct for that error." The δ term is a gain that represents which directions the UAV can thrust. The quadrotor can only thrust in the z-direction, so δ is in the form of $[0, 0, \delta_3]^\intercal$. Since δ is expressed in the UAV frame and $e_v(t)$ is in the camera frame, a change of frame, $R_F^C(t)$, is multiplied by δ. Substituting (3.25) into (3.26) and expanding $v_{IC}^C(t)$ produces

$$r_v = \overbrace{v_{FC}^C + v_{IF}^C}^{v_{IC}^C} - v_{ICd}^C + R_F^C \delta. \tag{3.27}$$

Since the camera is fixed to the UAV, there is zero velocity between the camera and UAV and $v_{FC}^C = 0$. Since $v_{IF}^F(t)$ is measured with respect to the UAV frame, a transformation to camera frame is made to yield

$$r_v = R_F^C v_{IF}^F - v_{ICd}^C + R_F^C \delta. \tag{3.28}$$

The time derivative of (3.28) is

$$\dot{r}_v = \dot{R}_F^C v_{IF}^F + R_F^C \dot{v}_{IF}^F - \dot{v}_{ICd}^C + \dot{R}_F^C \delta. \tag{3.29}$$

Similar to (3.4), $\dot{R}_F^C(t)$ can be expressed as

$$\begin{aligned} \dot{R}_F^C &= R_F^C S\left(\omega_{CF}^F\right) \\ &= -R_F^C S\left(\omega_{FC}^F\right). \end{aligned} \tag{3.30}$$

Substituting (3.3) for $\dot{v}_{IF}^F(t)$ and (3.30) for $\dot{R}_F^C(t)$ into (3.29) yields

$$\begin{aligned} \dot{r}_v &= -R_F^C S \underbrace{\left(\omega_{FC}^F + \omega_{IF}^F\right)}_{\omega_{IC}^F} v_{IF}^F + \frac{1}{m} R_F^C N_1(\cdot) + g R_I^C e_3 \\ &\quad + \frac{1}{m} R_F^C F_f^F - \dot{v}_{ICd}^C - R_F^C S\left(\omega_{FC}^F\right)\delta. \end{aligned} \tag{3.31}$$

By adding and subtracting $R_F^C(t) S\left(\omega_{IF}^F(t)\right)\delta$, (3.31) will become

$$\begin{aligned} \dot{r}_v &= -R_F^C S\left(\omega_{IC}^F\right) v_{IF}^F + \frac{1}{m} R_F^C N_1(\cdot) + g R_I^C e_3 + \frac{1}{m} R_F^C F_f^F \\ &\quad - \dot{v}_{ICd}^C - R_F^C S\left(\omega_{IC}^F\right)\delta + R_F^C S\left(\omega_{IF}^F\right)\delta. \end{aligned} \tag{3.32}$$

Some skew-symmetric properties include [20]

$$\begin{aligned} S\left(\omega_{IC}^{F}\right) &= S\left(R_{C}^{F}\omega_{IC}^{C}\right) \\ &\triangleq R_{C}^{F}S\left(\omega_{IC}^{C}\right)R_{F}^{C}, \end{aligned} \tag{3.33}$$

and

$$S\left(a\right)b = a\times b \tag{3.34}$$

which follows that

$$S\left(a\right)b = -S\left(b\right)a = -b\times a. \tag{3.35}$$

Using (3.33) in (3.32) and adding and subtracting $S\left(\omega_{IC}^{C}\left(t\right)\right)v_{ICd}^{C}\left(t\right)$ to the result to obtain

$$\begin{aligned} \dot{r}_v &= -\overbrace{R_{F}^{C}R_{C}^{F}}^{I^3}S\left(\omega_{IC}^{C}\right)\left(R_{F}^{C}v_{IF}^{F}+R_{F}^{C}\delta-v_{ICd}^{C}\right) \\ &\quad+\frac{1}{m}R_{F}^{C}N_1\left(\cdot\right)+gR_{I}^{C}e_3+\frac{1}{m}R_{F}^{C}F_{f}^{F}-\dot{v}_{ICd}^{C} \\ &\quad+R_{F}^{C}S\left(\omega_{IF}^{F}\right)\delta-S\left(\omega_{IC}^{C}\right)v_{ICd}^{C}. \end{aligned} \tag{3.36}$$

The non-linear term is redefined as

$$N_{11} = \frac{1}{m}R_{F}^{C}N_1\left(\cdot\right) \tag{3.37}$$

and the gravity term as

$$G_{11} = gR_{I}^{C}e_3,\ e_3 = \left[0,0,1\right]^{\intercal}. \tag{3.38}$$

Using (3.35) on $S\left(\omega_{IF}^{F}\left(t\right)\right)\delta$ in (3.36) yields

$$\begin{aligned} \dot{r}_v &= -S\left(\omega_{IC}^{C}\right)r_v+N_{11}+G_{11} \\ &\quad-\left[S\left(\omega_{IC}^{C}\right)v_{ICd}^{C}+\dot{v}_{ICd}^{C}\right]+\left[\frac{1}{m}R_{F}^{C}F_{f}^{F}-R_{F}^{C}S\left(\delta\right)\omega_{IF}^{F}\right]. \end{aligned} \tag{3.39}$$

Substituting $F_{f}^{F}\left(t\right)$ with (3.1) and $S\left(\delta\right)$ with (3.6) in the last term of (3.39) produces

$$\frac{1}{m}R_{F}^{C}F_{f}^{F}-R_{F}^{C}S\left(\delta\right)\omega_{IF}^{F} = R_{F}^{C}\left(\frac{1}{m}\begin{bmatrix}0\\0\\u_1\end{bmatrix}-\begin{bmatrix}0 & -\delta_3 & \delta_2\\ \delta_3 & 0 & -\delta_1\\ -\delta_2 & \delta_1 & 0\end{bmatrix}\omega_{IF}^{F}\right) \tag{3.40}$$

which makes (3.39)

$$\begin{aligned}\dot{r}_v &= -S\left(\omega_{IC}^C\right) r_v + N_{11} + G_{11} - \left[S\left(\omega_{IC}^C\right) v_{ICd}^C + \dot{v}_{ICd}^C\right] \\ &\quad + R_F^C \begin{bmatrix} 0 & 0 & \delta_3 & -\delta_2 \\ 0 & -\delta_3 & 0 & \delta_1 \\ \frac{1}{m} & \delta_2 & -\delta_1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ \\ \omega_{IF}^F \end{bmatrix}. \end{aligned} \tag{3.41}$$

Angular Velocity Error

The auxiliary signal definition in (3.28) and the subsequent open-loop system (3.41) captures the behavior of the translational velocity error. A similar error term for the rotational velocity is now motivated. The rotational velocity error is defined as

$$e_\omega \triangleq \omega_{IC}^C - \omega_{ICd}^C \tag{3.42}$$

where $\omega_{IC}^C(t)$ is the rotational velocities of the camera and $\omega_{ICd}^C(t)$ is the desired rotational velocity. Expanding $\omega_{IC}^C(t)$ produces

$$e_\omega = R_F^C\left(\omega_{IF}^F + \omega_{FC}^F\right) - \omega_{ICd}^C. \tag{3.43}$$

Using (3.20) or (3.24), (3.43) can be rewritten as

$$e_\omega = R_F^C \omega_{IF}^F + R_F^C J_C \dot{\theta}_C - \omega_{ICd}^C. \tag{3.44}$$

Open-Loop Error Dynamics

In preparation for the control design, it is useful to combine $\dot{r}_v(t)$ from (3.41) and $e_\omega(t)$ from (3.44) into a single vector to obtain the open-loop error system

$$\begin{aligned}\begin{bmatrix} \dot{r}_v \\ e_\omega \end{bmatrix} &= \begin{bmatrix} -S\left(\omega_{IC}^C\right) r_v \\ O_{3x1} \end{bmatrix} + \begin{bmatrix} -S\left(\omega_{IC}^C\right) v_{ICd}^C - \dot{v}_{ICd}^C \\ -\omega_{ICd}^C \end{bmatrix} + \begin{bmatrix} N_{11} + G_{11} \\ O_{3x1} \end{bmatrix} \\ &\quad + \underbrace{\begin{bmatrix} R_F^C & O_{3x3} \\ O_{3x3} & R_F^C \end{bmatrix} \begin{bmatrix} 0 & 0 & \delta_3 & -\delta_2 & \\ 0 & -\delta_3 & 0 & \delta_1 & O_{3x2} \\ \frac{1}{m} & \delta_2 & -\delta_1 & 0 & \\ 0 & 1 & 0 & 0 & \\ 0 & 0 & 1 & 0 & J_C \\ 0 & 0 & 0 & 1 & \end{bmatrix}}_{\triangleq \bar{B} \in \mathbb{R}^{6x6}} \underbrace{\begin{bmatrix} u_1 \\ \omega_{IF}^F \\ \dot{\theta}_C \end{bmatrix}}_{\triangleq \bar{U} \in \mathbb{R}^6} \end{aligned} \tag{3.45}$$

$\bar{U}(t)$ contains the six control inputs available to the controller; however is it multiplied by the $\bar{B}(t)$ matrix. Instead of calculating $\bar{U}(t)$ directly, a new control signal $U(t)$ is defined as

$$U = \bar{B}\bar{U} \tag{3.46}$$

where

$$U \triangleq \begin{bmatrix} U_1 \in \mathbb{R}^3 \\ U_2 \in \mathbb{R}^3 \end{bmatrix} \in \mathbb{R}^6. \tag{3.47}$$

With this new $U(t)$ vector, (3.45) becomes

$$\begin{aligned} \begin{bmatrix} \dot{r}_v \\ e_\omega \end{bmatrix} &= \begin{bmatrix} -S\left(\omega_{IC}^C\right) r_v \\ O_{3x1} \end{bmatrix} + \begin{bmatrix} -S\left(\omega_{IC}^C\right) v_{ICd}^C - \dot{v}_{ICd}^C \\ -\omega_{ICd}^C \end{bmatrix} \\ &\quad + \begin{bmatrix} N_{11} + G_{11} \\ O_{3x1} \end{bmatrix} + \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \end{aligned} \tag{3.48}$$

in which $U_1(t)$ and $U_2(t)$ can be designed to control $r_v(t)$ and $e_\omega(t)$.

Control Design and Stability Analysis

The first control input $U_1(t)$ is designed using the non-negative scalar Lyapunov candidate, $V_1(t)$, formed from the auxiliary signal defined in (3.28) as

$$V_1 = \frac{1}{2} r_v^\intercal r_v. \tag{3.49}$$

$V_1(t)$ can be used to find a control input that will guarantee $r_v(t)$ is bounded as long as the time derivative is also negative semi-definite. The time derivative of (3.49) is found to be

$$\dot{V}_1 = r_v^\intercal \dot{r}_v. \tag{3.50}$$

Substituting (3.48) into (3.50) yields

$$\dot{V}_1 = r_v^\intercal \left(U_1 - S\left(\omega_{IC}^C\right) v_{ICd}^C - \dot{v}_{ICd}^C + N_{11} + G_{11} - S\left(\omega_{IC}^C\right) r_v\right). \tag{3.51}$$

The design of $U_1(t)$ will cancel out as many terms as possible. Leaving only $\dot{v}_{ICd}^C(t)$ and $S\left(\omega_{IC}^C(t)\right) r_v(t)$ untouched,

$$U_1 = -k_r e_v - r_v \frac{\zeta_1^2\left(\left\|v_{IF}^F\right\|\right)}{\epsilon_1} + S\left(\omega_{IC}^C\right) v_{ICd}^C - G_{11}. \tag{3.52}$$

$\zeta_1(\cdot)$ is a non-decreasing function used to compensate for the unmodeled non-linear terms. This can include terms such as air resistance, which is a function of velocity. In order to guarantee $\zeta_1(\cdot)$ can compensate for $N_{11}(\cdot)$, it must satisfy the inequality

$$\alpha \left\| v_{IF}^{F}(t) \right\| + \beta \leq \zeta_1\left(\left\| v_{IF}^{F}(t) \right\|\right) \geq \left\| N_{11}(\cdot) \right\| \leq 0. \tag{3.53}$$

To design $U_2(t)$, a non-negative scalar Lyapunov candidate is defined as

$$V_2 = \frac{1}{2} e_\omega^\intercal e_\omega. \tag{3.54}$$

The time derivative of (3.54) is found to be

$$\dot{V}_2 = e_\omega^\intercal \dot{e}_\omega. \tag{3.55}$$

The time derivative of $e_\omega(t)$ can be obtained from (3.48) as

$$\dot{e}_\omega = \dot{U}_2 - \dot{\omega}_{ICd}^{C}. \tag{3.56}$$

Substituting (3.56) into (3.55) for e_ω yields

$$\dot{V}_2 = e_\omega^\intercal \left(\dot{U}_2 - \dot{\omega}_{ICd}^{C} \right). \tag{3.57}$$

In order to force $\dot{V}_2(t)$ to be negative, $\dot{U}_2(t)$ should cancel out $\dot{\omega}_{ICd}^{C}(t)$ and add an extra term to guarantee $\dot{V}_2(t)$ is negative, making

$$\dot{U}_2 = \dot{\omega}_{ICd}^{C} - k_i e_\omega. \tag{3.58}$$

Integrating (3.58) becomes

$$U_2 = -k_i \int e_\omega + \omega_{ICd}^{C}. \tag{3.59}$$

Theorem 1 *The control laws of (3.52) and (3.59) guarantee that the translational velocity error (3.25) and angular velocity error (3.42) are Globally Uniformly Ultimately Bounded (GUUB) in the sense that*

$$\left\| e_v \right\| \leq \alpha_{11} e^{-\alpha_{12}} + \alpha_{13} \tag{3.60}$$

$$\left\| e_\omega \right\| \leq \alpha_{21} e^{-\alpha_{22}} + \alpha_{23} \tag{3.61}$$

where $\alpha_{11}, \alpha_{12}, \alpha_{13}, \alpha_{21}, \alpha_{22}, \alpha_{23} \in \mathbb{R}^+$.

Proof:

To show that $e_v(t)$ and $e_\omega(t)$ are bounded, a Lyapunov function is used. A Lyapunov function based on $r_v(t)$ and $e_\omega(t)$ is used instead of one based on $e_v(t)$ and $e_\omega(t)$. The Lyapunov candidate is defined by the addition of $V_1(t)$ in (3.49) and $V_2(t)$ in (3.54) to obtain

$$V = \frac{1}{2} r_v^\intercal r_v + \frac{1}{2} e_\omega^\intercal e_\omega \tag{3.62}$$

which is positive definite. The next step is showing that the derivative has two parts: a negative definite part and a positive part that will be bound by an arbitrarily small constant. Taking the derivative of 3.62 yields

$$\dot{V} = r_v^\intercal \dot{r}_v + e_\omega^\intercal \dot{e}_\omega. \tag{3.63}$$

Substituting $\dot{r}_v$ from (3.48) and $\dot{e}_\omega$ from (3.56) into (3.63) yields

$$\begin{aligned} \dot{V} &= r_v^\intercal \left(U_1 - \dot{v}_{ICd}^C + N_{11} + G_{11} - S\left(\omega_{IC}^C\right) r_v \right. \\ &\quad \left. - S\left(\omega_{IC}^C\right) v_{ICd}^C\right) + e_\omega^\intercal \left(\dot{U}_2 - \dot{\omega}_{ICd}^C\right). \end{aligned} \tag{3.64}$$

Substituting U_1 from (3.52) and $\dot{U}_2$ from (3.58) into (3.64) results in

$$\begin{aligned} \dot{V} &= r_v^\intercal \left(-k_r r_v + k_r R_F^C \delta - r_v \frac{\zeta_1^2(\cdot)}{\epsilon_1} - \dot{v}_{ICd}^C + N_{11} - S\left(\omega_{IC}^C\right) r_v \right. \\ &\quad \left. + \underbrace{\left(G_{11} - G_{11}\right) + \left(S\left(\omega_{IC}^C\right) v_{ICd}^C - S\left(\omega_{IC}^C\right) v_{ICd}^C\right)}_{0}\right) \\ &\quad + e_\omega^\intercal \left(-k_i e_\omega + \underbrace{\dot{\omega}_{ICd}^C - \dot{\omega}_{ICd}^C}_{0}\right). \end{aligned} \tag{3.65}$$

To show that the Lyapunov function derivative is negative definite, using vector norms and using (3.53) to replace $N_{11}(\cdot)$ with $\zeta_1(\cdot)$ yields

$$\begin{aligned} \dot{V} &\leq -k_r \left\|r_v\right\|^2 + k_r \left\|\delta\right\| \left\|r_v\right\| - \left\|r_v\right\|^2 \frac{\zeta_1^2(\cdot)}{\epsilon_1} + \left\|\dot{v}_{ICd}^C\right\| \left\|r_v\right\| \\ &\quad + \left\|\zeta_1(\cdot)\right\| \left\|r_v\right\| - \overbrace{r_v^\intercal S\left(\omega_{IC}^C\right) r_v}^{0} - k_i \left\|e_\omega\right\|^2. \end{aligned} \tag{3.66}$$

With $\dot{v}_{ICd}^{C}(t)$ being the time derivative of the desired velocity, it was stated earlier that $v_{ICd}^{C}(t)$ is continuous, so the derivative exists and is bound. $\dot{v}_{ICd}^{C}(t)$ is bound by

$$\left\|\dot{v}_{ICd}^{C}\right\| \leq \beta_1(t), \; \beta_1(t) \in \mathbb{R}^{+}. \tag{3.67}$$

Rewriting (3.66) using (3.67) yields

$$\begin{aligned} \dot{V} \;\; \leq & \;\; -k_r \|r_v\|^2 - k_i \|e_\omega\|^2 + \|r_v\| \left(\beta_1(t) + k_r \|\delta\|\right) \\ & + \|r_v\| \|\zeta_1\| \left(1 - \|r_v\| \frac{\|\zeta_1(\cdot)\|}{\epsilon_1}\right). \end{aligned} \tag{3.68}$$

While the first two terms of the Lyapunov derivative in (3.68) are negative definite, the rest are not. The last term of (3.68) can be bound such that

$$\epsilon_1 \geq \|r_v\| \left\|\zeta_1\left(v_{IF}^{F}\right)\right\| \left(1 - \|r_v\| \frac{\|\zeta(\cdot)\|}{\epsilon_1}\right) \tag{3.69}$$

because it is in the form of non-linear damping according to Lemma A.10 in [21]. ϵ_1 was first introduced in (3.52) and is completely adjustable in the control input. The remaining term of (3.68) can be bound such that

$$\epsilon_2 \geq \sup_{\forall t} \left(\beta_1(t)\right) + k_r \|\delta\| \tag{3.70}$$

because all of these terms are constants. Since ϵ_2 is multiplied by $r_v(t)$, it will have to be bound also. To do this, it can be said that

$$0 \leq \left(\sqrt{\lambda_1}\|r_v\| - \frac{1}{\sqrt{\lambda_1}}\epsilon_2\right)^2, \;\; \lambda_1 \in \mathbb{R}^{+}. \tag{3.71}$$

This can be expanded to say

$$\|r_v\| \epsilon_2 \leq \frac{1}{2}\left(\lambda_1 \|r_v\|^2 + \frac{1}{\lambda_1}\epsilon_2^2\right). \tag{3.72}$$

This finally bounds (3.68) to make

$$\dot{V} \leq -\left(k_r - \frac{\lambda_1}{2}\right)\|r_v\|^2 - k_i \|e_\omega\|^2 + \epsilon_1 + \frac{\epsilon_2^2}{2\lambda_1}. \tag{3.73}$$

In order to keep the $\|r_v\|^2$ term negative definite, it follows that

$$k_r \geq \frac{\lambda_1}{2}. \tag{3.74}$$

A constant λ_2 is defined as

$$\lambda_2 = \max\left(k_r - \frac{\lambda_1}{2}, k_i\right). \tag{3.75}$$

A larger upper bound can be placed on (3.73) using λ_2 to get

$$\dot{V} \leq -\lambda_2\left(\|r_v\|^2 + \|e_\omega\|^2\right) + \epsilon_1 + \frac{\epsilon_2^2}{2\lambda_1}. \tag{3.76}$$

Since λ_1 can be arbitrarily picked, by choosing a larger k_r gain, λ_1 in (3.74) is forced to be bigger and can make the ϵ_2 term in (3.76) arbitrarily small. The same is true for ϵ_1 since it is also arbitrarily picked. Solving for the differential equation in (3.73) finally arrives at

$$V \leq V(0)\, e^{-(2\lambda_2 t)} + \frac{2\epsilon_1\lambda_1 + \epsilon_2^2}{4\lambda_1\lambda_2}, \tag{3.77}$$

which is in the form of (3.60) and (3.61). λ_2 in (3.77) affects both the rate at which the exponential term approaches zero and the value of the bounding constant. Since λ_2 is merely the maximum from (3.75), by increasing the gains the bound can be made arbitrarily small and the Lyapunov function can approach zero arbitrarily fast.

Remark 3.1 *It has been shown that $\|e_\omega(t)\|$ is GUUB according to the Lyapunov proof for $V(t)$, as stated in Theorem 1. The proof also shows that $\|r_v(t)\|$ is GUUB. According to (3.26), $r_v(t)$ is $e_v(t)$ plus a constant multiplied by an $SO(3)$ rotation matrix, making $e_v(t)$ GUUB. Both desired velocities, $v_{ICd}^C(t)$ and $\omega_{ICd}^C(t)$, are bound by design, therefore $v_{IC}^C(t)$ and $\omega_{IC}^C(t)$ are bound. The details of the signal chasing are covered in Appendix B.*

Simulation

Quadrotor Model Simulation Parameters

In order to have an accurate simulation of the DraganFlyer X-Pro, certain parameters have to be measured. The mass and inertia matrix are needed for

Table 3.3 DraganFlyer X-Pro Parameters

Parameter	**Value**	**Units**
Mass	2.041	kg
Additional Weight (battery and sensors)	.68	kg
Max Thrust	35.586	N
Max *Roll Torque*	4.067	Nm
Max *Pitch Torque*	4.067	Nm
Max *Yaw Torque*	2.034	Nm

the dynamics equations and the actual maximum thrust and torques produced by the quadrotor are needed to set realistic limits on the simulation.

To measure the mass, the helicopter was weighted using a spring scale. The inertia matrix is never needed because (3.5) is not utilized in this simulation. To measure the total force the helicopter can produce, a spring scale is used to measure the amount of force one rotor can create when spinning at maximum speed. This quantity multiplied by four will yield the full thrust ability. In addition, the distance from the rotor to the center of the UAV will yield the total roll and pitch torque that can be generated with one rotor spinning at its maximum velocity while the opposite rotor stopped, as in Figure 3.4.b. A similar measurement was made when two opposite rotors were spinning at maximum speed and the other two rotors were not spinning, as in Figure 3.5.b, to estimate the maximum yaw torque. All of these measurements are displayed in Table 3.3.

Simulation Setup

To simulate this controller, there are three main tasks that must be implemented, shown in Figure 3.11. The first block is the helicopter dynamics that must be simulated, based on (3.2)-(3.4). Then the control input must be formulated based on sensor readings that come from the dynamics equations. The control input will then be fed back into the dynamics equations to close the loop. The third portion of the simulation is displaying the position and orientation of the quadrotor in a 3D-simulation program called FlightGear.

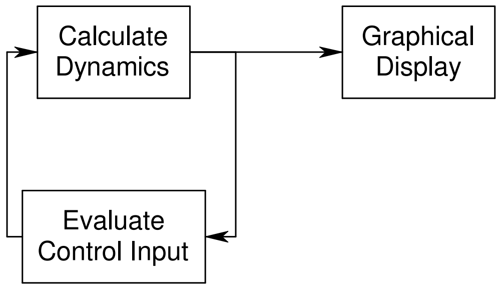


Figure 3.11 Simulation diagram

For calculating the dynamics, equations (3.2)-(3.4) are rewritten as

$$\begin{aligned}
\dot{x}^I_{IF} &= R^I_F v^F_{IF} && (3.78)\\
\dot{v}^F_{IF} &= -S\left(\omega^F_{IF}\right) v^F_{IF} + gR^F_I e_3 + \frac{N_1(\cdot) + F^F_f}{m} && (3.79)\\
\dot{\Theta}^F_{IF} &= J^{-1}_F \omega^F_{IF} && (3.80)\\
\dot{\Theta}^C_{FC} &= \dot{\Theta}^C_{FC}. && (3.81)
\end{aligned}$$

Equation (3.2) remains unchanged and (3.3) is divided by m to solve for $\dot{v}^F_{IF}(t)$. Equation (3.4) is replaced with the roll, pitch, yaw version of the equation from (3.8) instead of rotation matrices. Using the yaw, pitch, roll representation results in a singularity at $\theta_{pitch}(t) = \pm\frac{\pi}{2}$. The solution to this problem is to avoid $\pm\frac{\pi}{2}$. The last equation is used to calculate the angles between the camera and UAV frame. The equations are then integrated on both sides yielding

$$\begin{aligned}
x^I_{IF} &= \int \left(R^I_F v^F_{IF}\right) && (3.82)\\
v^F_{IF} &= \int \left(-S\left(\omega^F_{IF}\right) v^F_{IF} + gR^F_I e_3 + \frac{N_1(\cdot) + F^F_f}{m}\right) && (3.83)\\
\Theta^I_F &= \int \left(J_F\left(\Theta^I_F\right) \omega^F_{IF}\right) && (3.84)\\
\Theta^F_C &= \int \dot{\Theta}^C_{FC}. && (3.85)
\end{aligned}$$

All three control inputs are utilized in these equations, $F^F_f(t)$, $\omega^F_{IF}(t)$, and $\dot{\Theta}^C_{FC}(t)$. The numerical integration method used is an Adams Integrator at a 1000 Hz update frequency.

The software platform used is QNX Real-Time Operating system [16] running a QMotor program [17] written in C++. The entire program consists of seven parts outlined in Figure 3.12. The program starts by initializing all variables in “Start”. Then in “Calculate Dynamics”, (3.82)-(3.85) are utilized to find $x^I_{IF}(t)$, $v^F_{IF}(t)$, $\Theta^I_F(t)$, and $\Theta^F_C(t)$. In “Evaluate Control Input”, the program uses the trajectories for $v^C_{ICd}(t)$ and $\omega^C_{ICd}(t)$ from “Read in joystick values and generate trajectories” to evaluate (3.52) and (3.59). “Saturate Control Inputs Based on UAV Parameters” uses the parameters from Table 3.3 to make sure the control inputs do not exceed realistic values, and if they do, saturate the value to the maximum limit. A minimum rotor speed is also utilized to prevent the quadrotor coupling singularities, due to the motors only spinning in one direction. In addition to preventing the singularities, the coupling effects are also added to the control inputs. “Update System States” is where the all positions and velocities in the inertia, UAV, and camera frames are computed for use in other calculations in the next iteration. The resulting position and velocity are sent to FlightGear by “Send UDP Packets” and received and show on the screen by “Virtual Simulation.” The code for the QNX program can be found at http://www.ece.clemson.edu/crb/research/uav/sixdof.zip. In the simulate.cpp file, the “Send UDP Packets” is accomplished by

d_flight_UDP->PackitSend($Latitude, Longitude, Altitute, Roll, Pitch, Yaw$);

where all numbers are in radians except *altitude* which is in meters and d_flight_UDP is of type FlightUDP. A UDP is send to a PC on a specific port defined in the constructor in FlightUDP.cpp as

d_UDP_client=new UDPClient(“192.168.1.100”,4444,*d_timeout*);

where the UDP packet is sent to the FlightGear IP address, 192.168.1.100 in this case, on port 4444 and *d_timeout* is a struct of type *timeval* set to 50ms. The UDP packet is of type FGNetFDM defined by net_fdm.hxx [1]. The only use for the second computer is the FlightGear simulation, because a simulation such as FlightGear requires| almost all of a computer’s CPU, video card, and input/output system [1]. To execute the FlightGear simulator at GSP airport, the following command is used

```
fgfs.exe –fg-root=”C:\Program Files\FlightGear\data”
–fg-scenery=”C:\Program Files\FlightGear\data\Scenery;
C:\Program Files\FlightGear\scenery” –airport-id=KGSP
–aircraft=draganfly –control=joystick –enable-game-mode
```

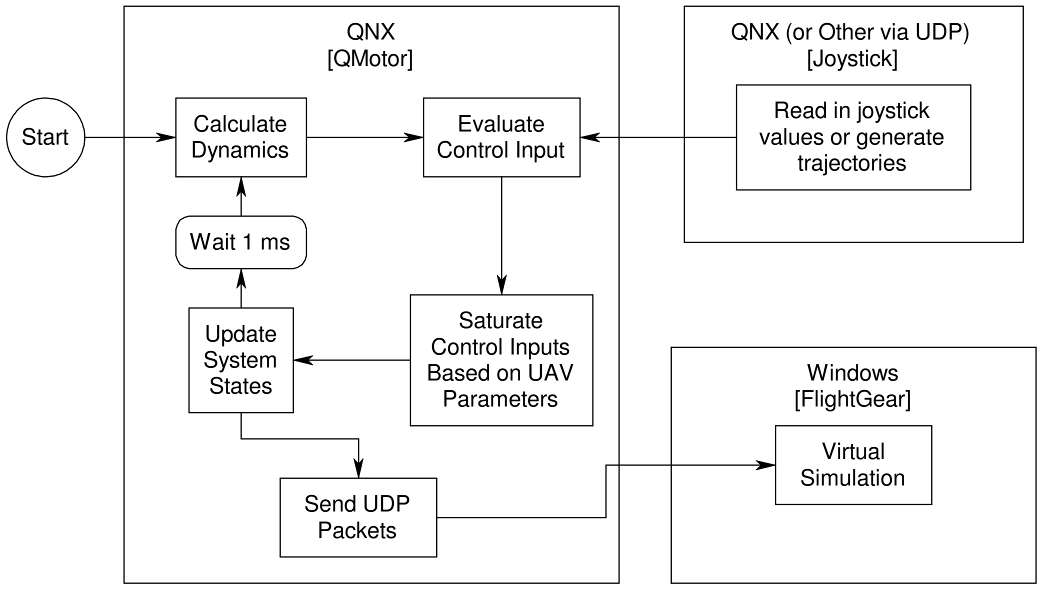


Figure 3.12 Two Computer Simulation Flow

–disable-splash-screen –disable-random-objects –disable-sound

–disable-ai-models –disable-clouds –fog-disable

–geometry=1680x1050 –timeofday=noon –fdm=external

–native-fdm=socket,in,20,192.168.1.1,4444,udp

where 192.168.1.1 is the IP of the QNX PC sending the UDP packets on port 4444. The rest of the options load the other setting such as environment, location, aircraft model, and resolution.

Input

The desired trajectories can be generated by any means as long as $v_{ICd}^{C}(t)$ and $\omega_{ICd}^{C}(t)$ are continuous and differentiable. The method used for this experiment used a Logitech Extreme 3D Pro joystick [19], seen in Figure 3.13, to create four inputs labeled as x, y, twist and throttle. The x, y, and twist on the joystick are used to generate either $v_{ICd}^{C}(t)$ or $\omega_{ICd}^{C}(t)$. The throttle is not utilized since out of the four inputs, only x, y, and twist can be easily controlled with one hand. Throttle requires removing the hand from the joystick

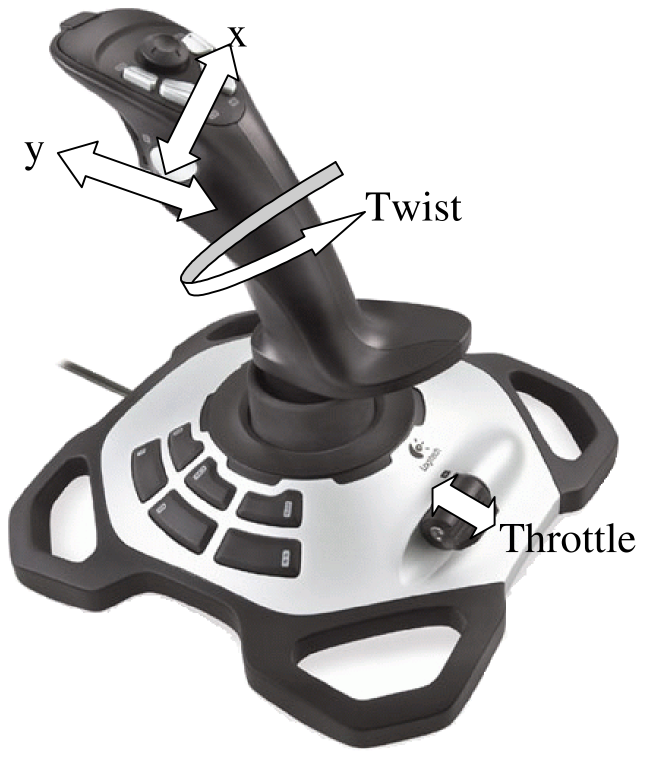


Figure 3.13 Logitech Wingman Extreme 3D Pro joystick

or using a second hand. One method of using this joystick is to only input three of the desired velocities at one time. Depending on whether Button 1 is pressed, the joystick's three axes will control the 3 translational velocities, $v^C_{ICd}(t)$, or the three angular velocities, $\omega^C_{ICd}(t)$. However this prevents all six degrees from being controlled at once. A second method uses two Wingman joysticks to allow the user to control three desired velocities with each hand, for a total of all six desired velocities.

FlightGear

FlightGear is an open-source flight simulator [1]. It can run on many computer platforms including Linux and Windows. FlightGear offers a versatile package with multiple input-output (I/O) systems and an interface for importing custom helicopter models. FlightGear is used to display the position and orientation and show the output in a 3D virtual world using the DraganFlyer X-Pro helicopter model [18].

FlightGear has as least three Flight Dynamics Models (FDM) built in. The user can use any of these FDMs on a real airplane model. However, since this paper uses its own dynamics equations, it was decided to use a custom model using the dynamic equations (3.2)-(3.5) instead of FlightGear's dynamics. FlightGear's I/O system allows it to receive UDP packets of FDM calculations in real-time. This network FDM (netfdm) allows any computer on the network to perform the dynamics calculations, such as (3.2) through (3.5). This provides a live fully visual representation of what is going on inside the QMotor simulations.

There are two useful viewpoints that can be shown from FlightGear. The most important of these is the camera view. Since this is a fly-by-camera interface, it is useful to be able to see the camera view for simulation. FlightGear's first person view looks out the quadrotor's x-axis, as seen in Figure 3.8. To accomplish this, a transformation is made so that the camera frame's z-axis is the x-axis seen in the simulator using the rotation matrix

$$R_{C(fonrt)}^{simulation} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.86}$$

that goes from the camera frame to the FlightGear simulation frame. If the camera were mounted on the bottom of the helicopter as seen in Figure 3.9, the transformation will be from the camera's z-axis to the z-axis on the helicopter. This transformation is needed because FlightGear is attempting to display the UAV and this transformation allows it to display the camera correctly, allowing for a complete camera view in the simulator. The other important camera view is the helicopter itself. A third person mode on FlightGear will show the quadrotor helicopter and the angles as which it rotates as it moves around. QMotor has the option of showing either the camera frame for first person mode or the UAV frame for third person mode to demonstrate what the helicopter is actually doing.

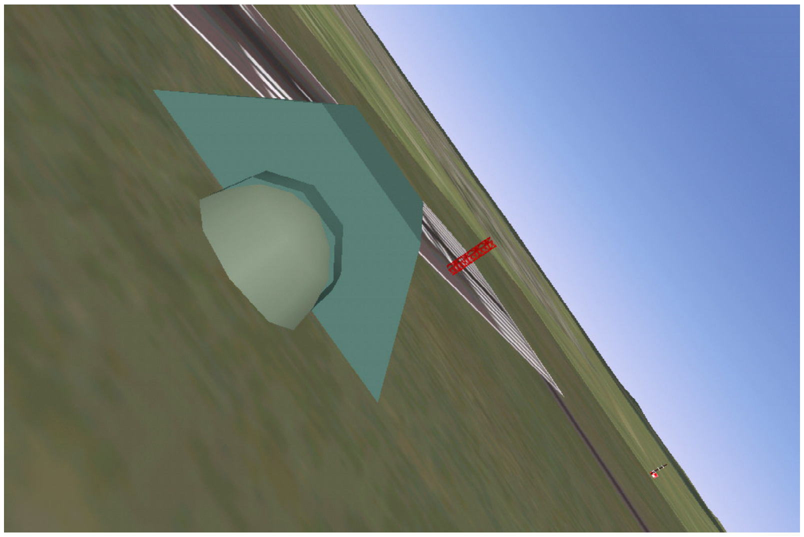


Figure 3.14 First person view of quadrotor while moving left

FlightGear is used to show two points of view while the simulation is running: the UAV view and the camera view. The UAV view will show the position and orientation of the UAV helicopter. Likewise, the camera frame shows the orientation and position of the camera. It is important to show the camera for the simulation since the experiment is the fly by camera interface. In Figures 3.14 and 3.15, the UAV is oriented so that it can move to the left. Figure 3.16 shows what the camera will show in the same situation. The actual level of the ground can be seen in Figure 3.15 and is approximately the same in Figure 3.16, thus demonstrating the independence of the UAV and camera frame.

Observations and Results

The very first simulations were a complete failure. This is because the original U_1 control input was different from (3.52). Instead, U_1 was defined as

$$U_1 = -k_r r_v - r_v \frac{\zeta_1^2 \left(\left\| v_{IF}^F \right\| \right)}{\epsilon_1} + S\left(\omega_{IC}^C\right) v_{ICd}^C \tag{3.87}$$

with no $G_{11}(t)$ term and the $e_v(t)$ replaced by $r_v(t)$. The lack of a $G_{11}(t)$ term was because it was originally intended that gravity would be included

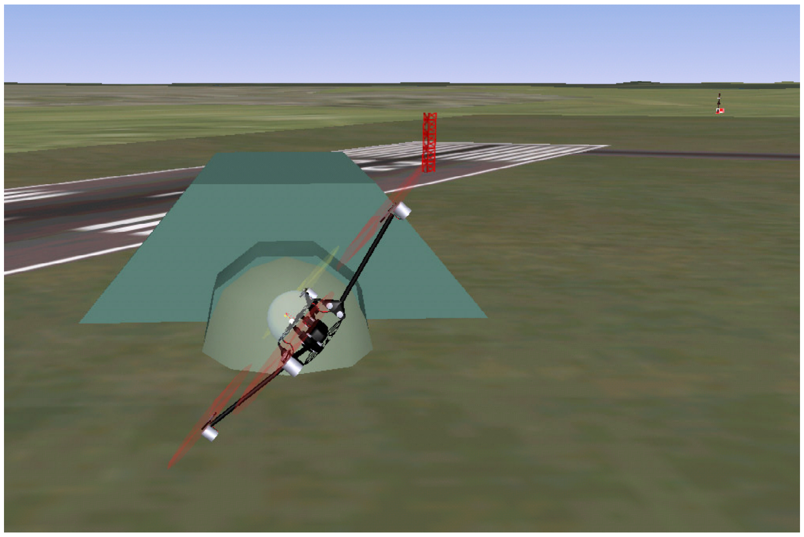


Figure 3.15 External view of quadrotor while moving left

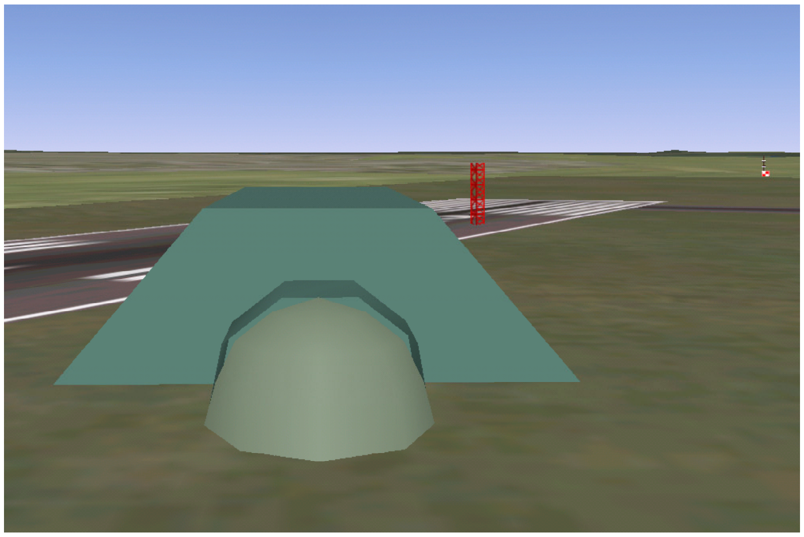


Figure 3.16 Camera view while moving left

in the $N_{11}(t)$ term. This was extremely difficult to overcome because according to Table 3.3 the DraganFlyer X-Pro uses approximately 75% of it's total thrust just to overcome gravity. Using $r_v(t)$ instead of $e_v(t)$ also proved to be unusable. Using (3.26), (3.87) can be rewritten as

$$U_1 = -k_r e_v - k_r R_F^C \delta - r_v \frac{\zeta_1^2\left(\left\| v_{IF}^F \right\|\right)}{\epsilon_1} + S\left(\omega_{IC}^C\right) v_{ICd}^C. \tag{3.88}$$

The $k_r R_F^C \delta$ term was adding a constant to the control input, and overpowering the rest of the system making it unusable. This is where the solution of using (3.52) came from. (3.87) satisfies its own Lyapunov proof and the velocities were GUUB, however they never came near to zero and the bound could not be made arbitrarily small on $e_v(t)$.

In simulation when the UAV is just hovering, it will stand still with no errors using (3.52) and (3.59). To examine how the control reacts, a simple experiment is conducted by commanding the camera to move left and right. All of the experiments are conducted with $\delta = 100$, $k_r = 1$, and no k_i because there is no angular error in the simulation. To look at how the velocities and positions of the camera react, the first experiment will show the UAV going left then right using the first camera mounting method highlighted in Figure 3.8. The desired velocity graph seen in Figure 3.17 is used on the y-coordinate of the camera frame only. All values remain zero for the first 15 seconds of the experiment and are not shown. The desired x- and z-velocities remain zero and $\omega_{ICd}^C = 0$. The actual velocities of the camera frame in Figure 3.20 approach the desired velocities in a few seconds. It can also be seen that the x-velocity changes when the camera moves left and right. This is because gravity is in the x-direction. Sudden changes cause the quadrotor to lift or fall a little, however, the controller does respond and stabilizes the x velocity to zero.

In this first experiment when the camera is commanded to go left and right, the UAV will tilt left and right along the camera z-axis, as seen in Figure 3.18. The other two axes not shown remain zero for the entire simulation. The graph shows the UAV tilting over .5 radians (29 degrees). The controller is suppose

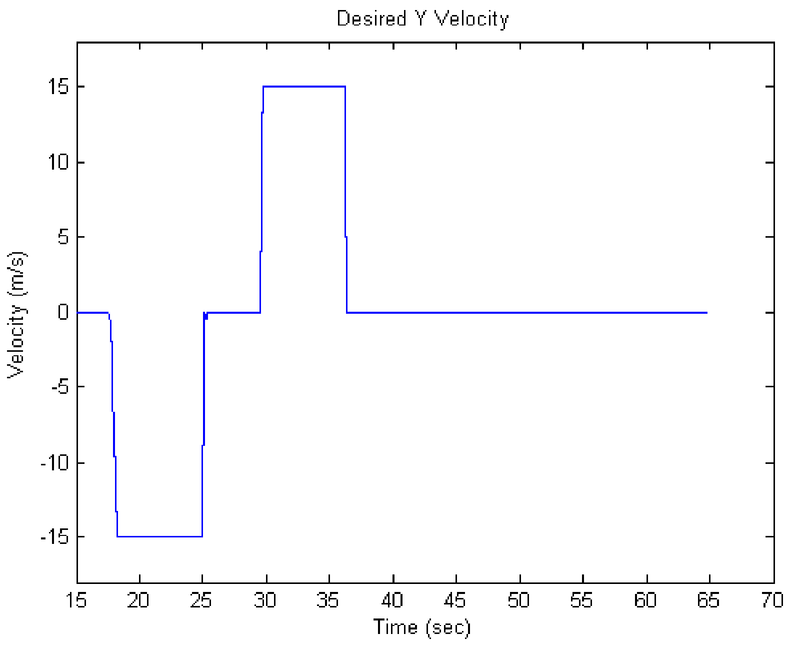


Figure 3.17 Desired velocities of Camera Frame (v_{ICd}^{C}): Experiment 1

to apply a counter rotation to the camera frame in an attempt to minimize the amount of rotation felt in the camera frame. As seen in Figure 3.19, the camera frame's rotation is much less than that of the UAV. There is still a certain amount of rotation error in the camera frame. However, the error is not due to an error in $\omega_{IC}^{C}(t)$. In the simulation, $e_{\omega}(t)$ is zero since the simulation model controls the angular rates directly. The error is caused by the design of the controller. The error is in the rotation of the camera, however the feedback and inputs are in terms of angular velocities. Because the exact angles of the UAV and camera are not known by the controller, there is a drifting error in the camera rotation. The camera angles are essentially a result of integrating the control inputs, thus creating this drift error over time. The drift error in the translational velocities is less detrimental because the position error is not focused on.

The camera is tilting a little more in the opposite direction than needed. When the UAV turns left .38 radians, the camera is actually turning right .46 radians, resulting in the final .08 radian rotation. This gives viewers and observers a counterintuitive feeling because they are familiar with there being no correction.

The error is relatively small and is easy to compensate for by telling the camera to tilt a little in the opposite direction of the error. The total error after about a minute of flight time can be seen in Figures 3.18 and 3.19 when the UAV angle returns to zero and the camera angle settles around .15 radians (8.6°). The feel from the camera is a lot smoother then if there was no correction.

If the drift error seen in Figure 3.19 is to be fixed, then the controller has to be changed. Theorem 1 does not state that the angles are bounded. In order to control the angles, an angle sensor will need to be added to the controller. With angle feedback, the angle can be controlled and the error can eventually be bounded.

In this particular experiment, the desired camera angle was zero throughout

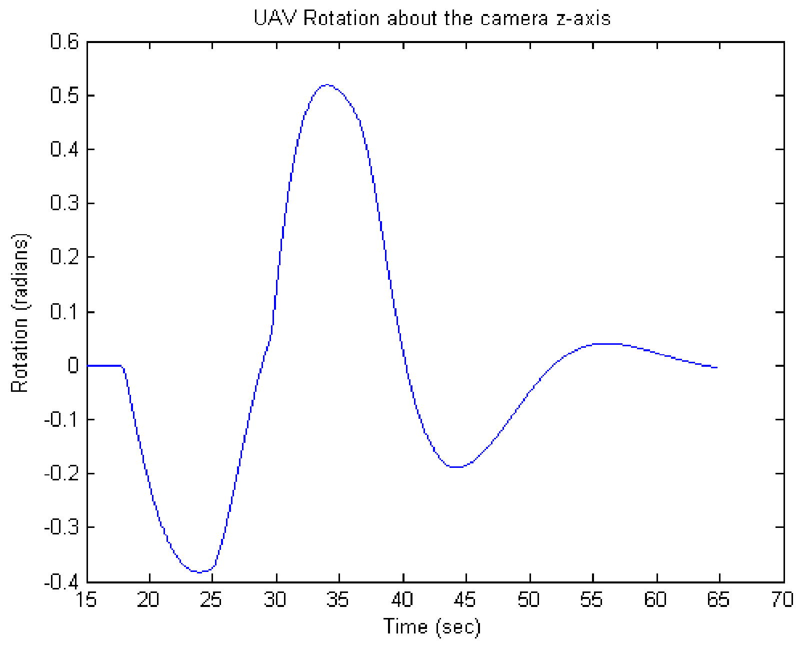


Figure 3.18 Angles of UAV while moving left and right (Θ_I^F)

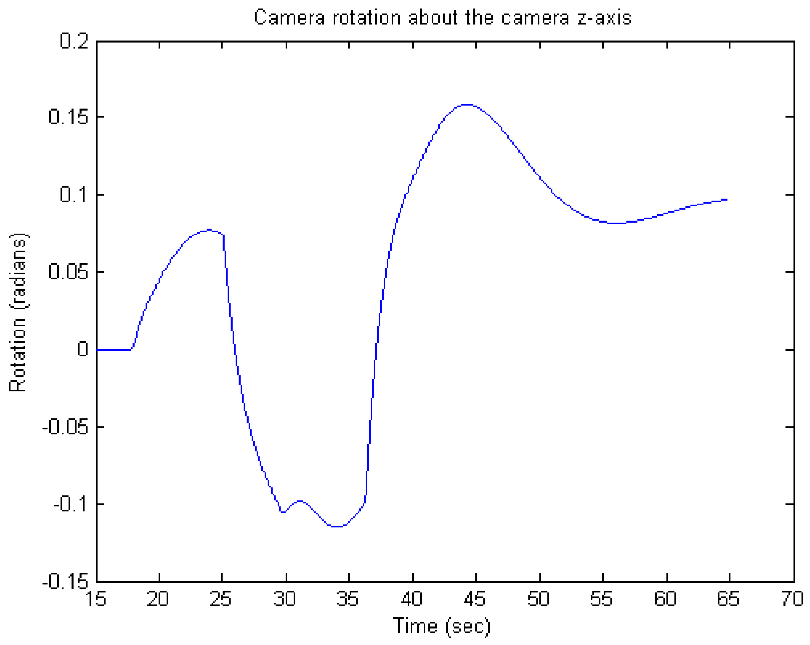


Figure 3.19 Camera error while moving left and right (Θ_I^C)

the experiment, even though the actual angle is not zero as seen in Figure 3.19. Because of this, the velocities expressed in the camera frame and the inertia frame are in fact different. It is interesting to compare these velocities to evaluate how significant this error is. In Figure 3.20, the camera y velocity can be seen to be in the approximate shape of the desired velocity in Figure 3.17. There is an apparent flat spot around 25 second and 35 seconds. While the z-velocity remains zero the entire experiment, the x-velocity does not. This is because the x-velocity is along the same axis as gravity in the Figure 3.8 configuration. When the quadrotor rolls left and right, rotation about the x-axis is disturbed and the controller must correct for this. As stated before, if there were no errors in the rotation, then $v_{IC}^{C} = v_{IC}^{C(0)}$, where $C\,(0)$ is the camera frame at time 0. Comparing Figure 3.20 to Figure 3.21, there are small differences, however the errors seen in Figure 3.19 do not make a huge difference in the velocities. The last velocity to look at is the UAV velocity in the UAV frame seen in Figure 3.22. It is a much smoother curve compared to the rest. This is how the quadrotor actually reacts: in smooth glides left and right.

Another interesting graph to look at is the path the UAV and camera actually travel in. Since the camera is closely mounted onto the UAV, it is safe to say their positions are approximately the same. Figure 3.23 shows the y-position going left then coming back right. Since the velocity to the right seen in Figure 3.21 is faster and longer then to the left, the UAV ends up going right more, before settling close to zero. The z-position is off by about 5 meters in the end. This is all part of the drift error that will occur in the position when only the velocity is bounded, as stated in Theorem 1.

Examining the control inputs for the first experiment shows that the F_f^F control input u_1, seen in Figure 3.24, only reaches its saturation point once, at around 30 seconds. Due to the controller's nature, where there are sharp changes in the velocities, there are sharp changes in the thrusts, seen especially at 25, 30, and 36 seconds. As for the other control inputs, only the x-axis of

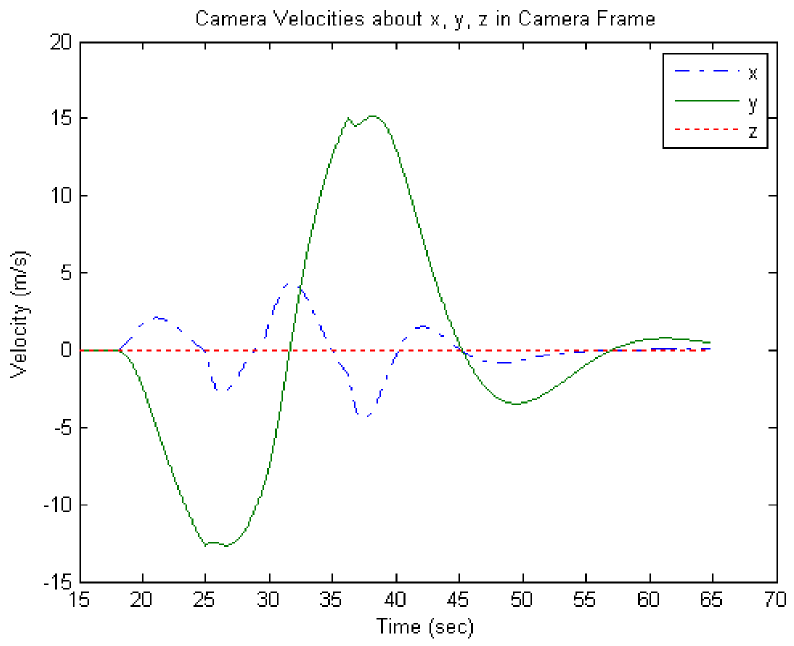


Figure 3.20 Velocities of Camera Frame (v_{IC}^{C}): Experiment 1

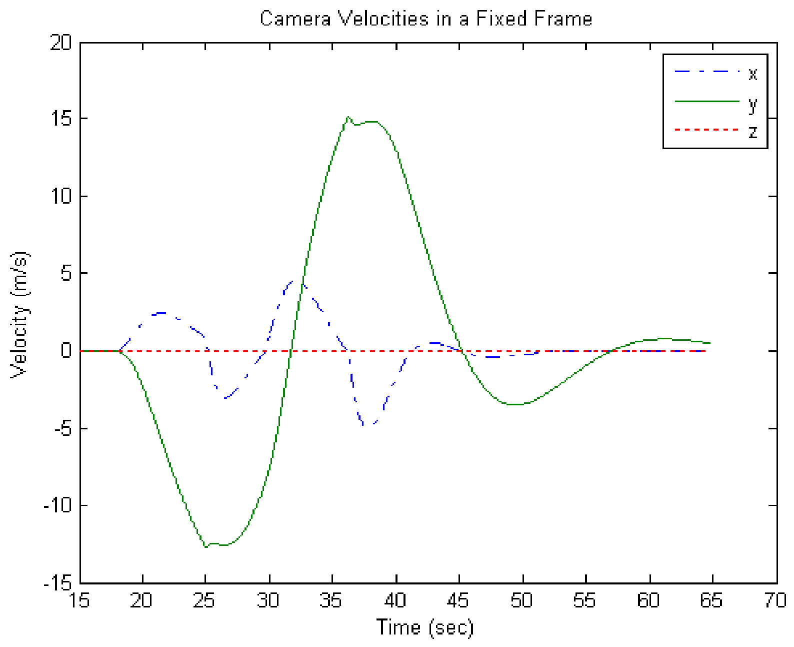


Figure 3.21 Velocities of Camera in a Fixed Inertia Frame ($v_{IC}^{C(0)}$): Experiment 1

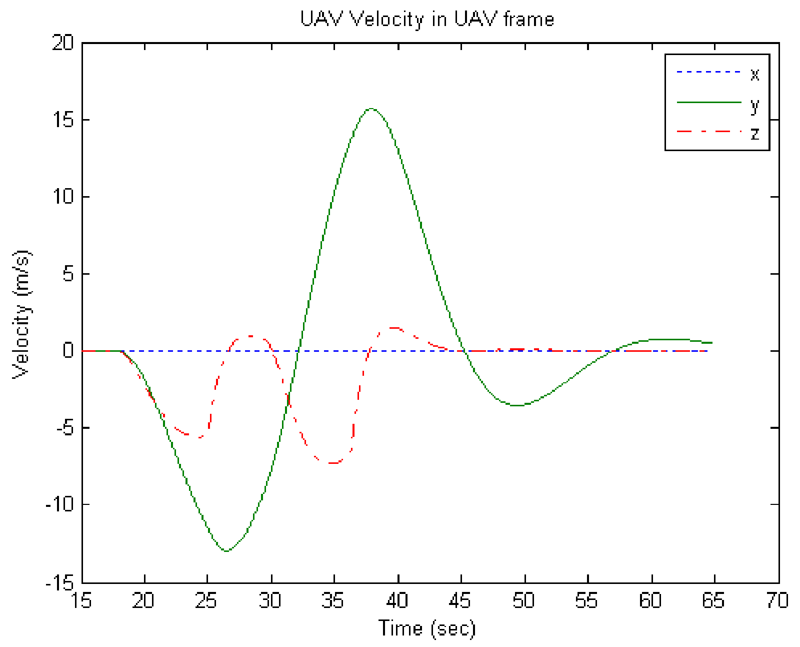


Figure 3.22 Velocities of UAV in UAV Frame (v_{IF}^F): Experiment 1

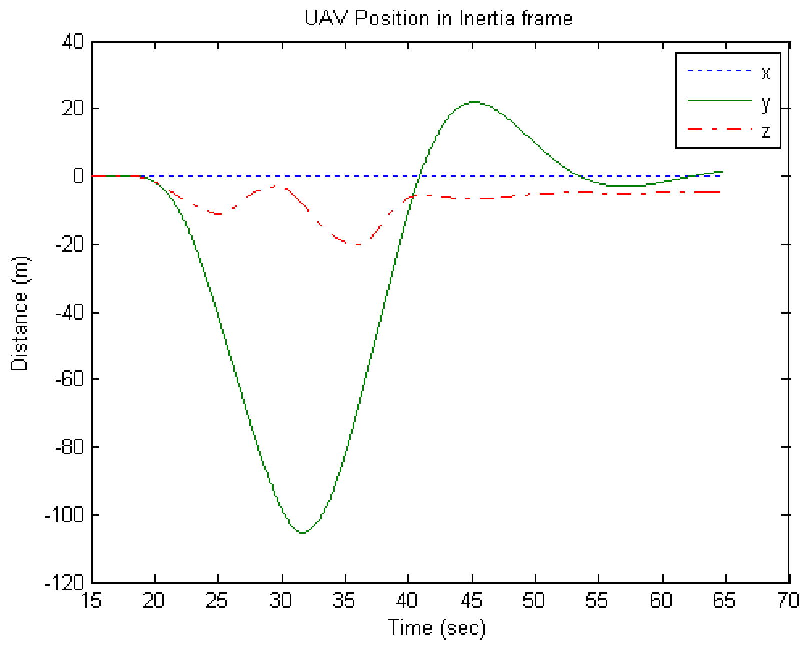


Figure 3.23 Position of UAV in Inertia Frame ($x_{IF}^I \approx x_{IC}^I$): Experiment 1

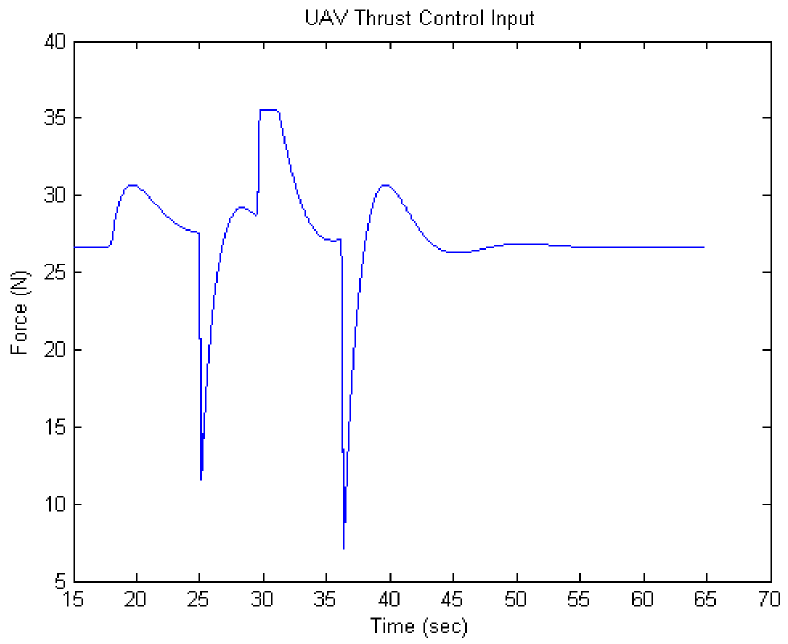


Figure 3.24 UAV trust (F_f^F): Experiment 1

ω_{IF}^F and the roll angle of $\dot{\theta}_C$ do not remain zero, as shown in Figure 3.25. It can be seen that these two curves are near opposites of each other, as would be expected given the desired inputs and the θ_C.

The exact same results are seen in a second experiment when the desired velocities tell the camera to go forward and backward instead of left and right, since the quadrotor is completely symmetrical in this way.

In a third experiment, the camera is commanded to go up and down to examine the effects that a limited thrust has. ω_{ICd}^C is kept at zero while v_{ICd}^C is in the shape of Figure 3.26. Since the quadrotor is only translating in the up and down directions, there will be no torques or change of angle. The actual velocity achieved by the camera can be seen in Figure 3.27. Since the thrust is limited to 35.586 N in Table 3.3, the quadrotor only significantly reaches its limit at 7 seconds and again at 40 seconds, seen in Figure 3.28. By comparing the acceleration of the UAV in Figure 3.27, it can be seen that the acceleration up is much slower then the acceleration down due to this

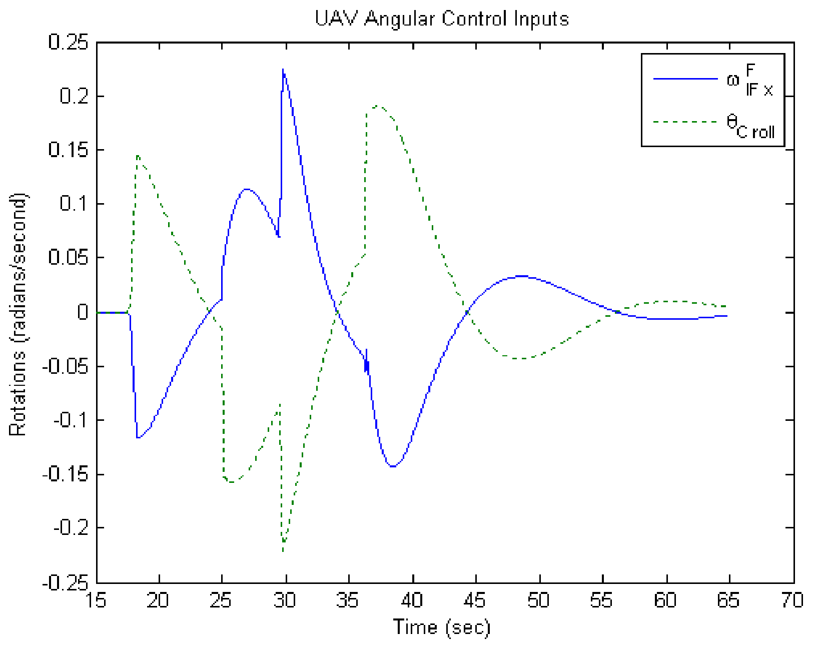


Figure 3.25 UAV Angular rates (ω_{IF}^{F} and θ_C): Experiement 1

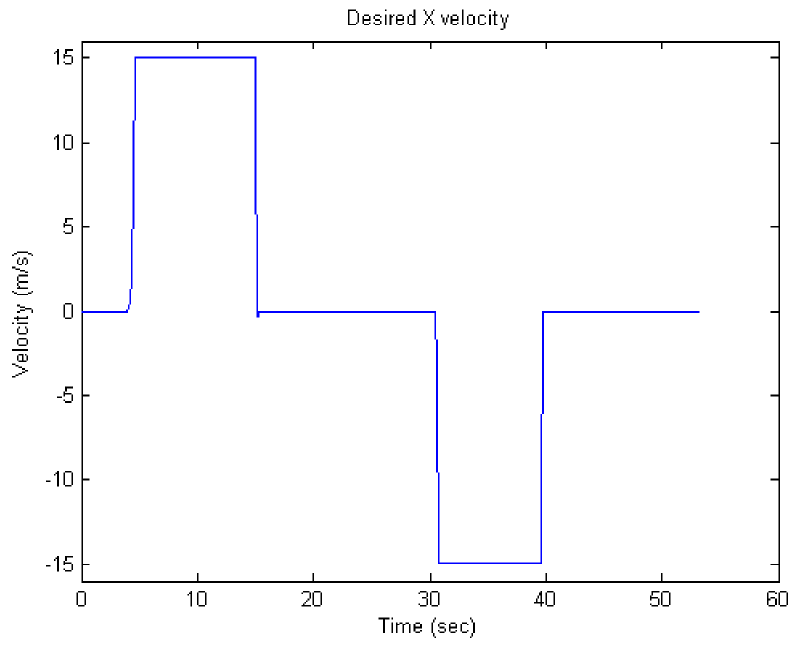


Figure 3.26 Desired velocities of Camera Frame (v_{ICd}^{C}): Experiment 3

limitation. The saturation points reached at 15 seconds and 31 seconds are there to stop the rotors from spinning and losing all roll, pitch, yaw control.

In the last experiment, a smoother desired velocity is used to tell the quadrotor to move right then stop, as seen in Figure 3.29. Looking at the camera velocities, it can be seen that there is a small overshoot at around 15 seconds, and then the velocity settles near the desired 15 m/s. The same event occurs at around 50 seconds as the quadrotor comes to a stop. Similar to experiment 1, the UAV drops in the camera's x-direction. The drop in the x-direction is smaller then that observed in experiment 1. In addition, a small angular error can be observed. However this error is less then .1 radians for all time, smaller than that from experiment 1. With less sudden changes in the desired velocity, there is less error in the angles of rotation and less of a drop in the x-direction. The thrust control input F_f^F in Figure 3.30 varies considerably less than experiment 1 did. The velocity error shown in Figure 3.31 shows most of the error in the y-direction and is in the same shape as u_1 seen

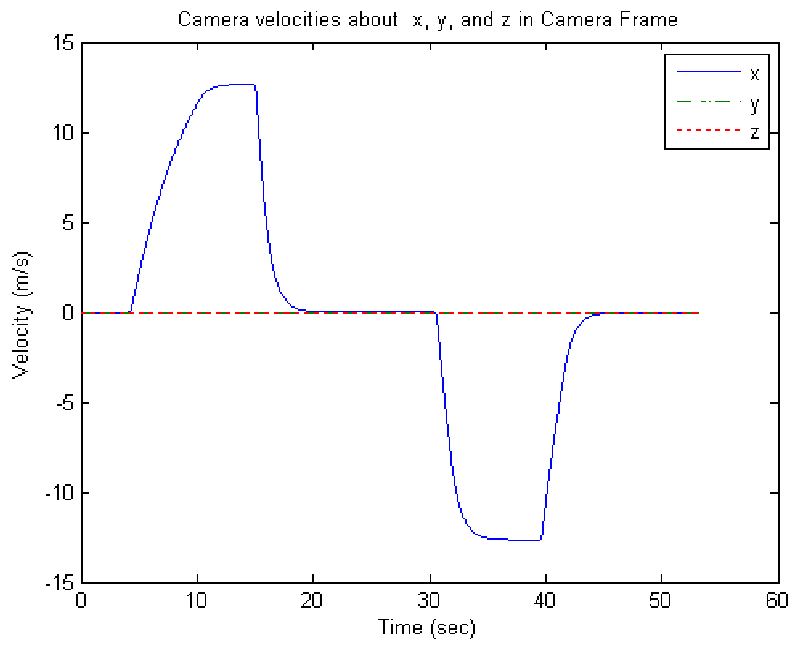


Figure 3.27 Velocities of Camera Frame (v_{IC}^{C}): Experiment 3

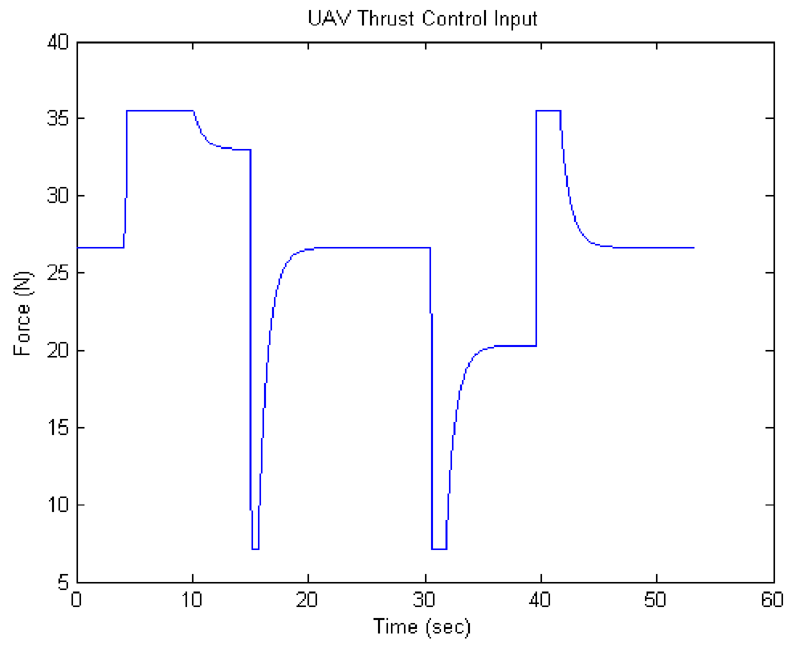


Figure 3.28 UAV trust (F_f^F): Experiment 3

in Figure 3.30. All of these results are expected for this kind of experiment.

All of these experiments show that the controller is GUUB in both v_{IC}^{C} and ω_{IC}^{C}. One of the difficulties in a velocity controller like this is that the positions and angles are not bounded and can only be kept small by making the bounds on the velocities close to zero. By using (3.52) instead of (3.87), it is possible to have a useful velocity GUUB controller.

Future Work

There are a number of improvements that can be made on this controller for the future. One improvement would be to add a k_{vi} integral gain into equation (3.52) to make

$$U_1 = -k_r e_v - r_v \frac{\zeta_1^2 \left(\left\| v_{IF}^{F} \right\| \right)}{\epsilon_1} + S\left(\omega_{IC}^{C}\right) v_{ICd}^{C} - k_{vi} e_i - G_{11} \tag{3.89}$$

where

$$e_i \triangleq R_C^I \int v_{IC}^{C} - v_{ICd}^{C}. \tag{3.90}$$

This will help compensate for constant errors and is especially good for correcting for uncertainties in gravity. This relaxes the restriction that gravity to be known. It has been observed that small uncertainties in gravity will cause the velocities to drift to an offset instead of going to zero, as would be expected. Another improvement to (3.52) could be to use a predictor for $\dot{v}_{ICd}^{C}(t)$. By adding a $\|r_v\| \left\| \tilde{v}_{ICd}^{C} \right\|$ term to U_1 to get

$$U_1 = -k_r e_v - r_v \frac{\zeta_1^2 \left(\left\| v_{IF}^{F} \right\| \right)}{\epsilon_1} + S\left(\omega_{IC}^{C}\right) v_{ICd}^{C} - \|r_v\| \left\| \tilde{v}_{ICd}^{C} \right\| - G_{11}, \tag{3.91}$$

there will be no need for a $\beta_1(t)$ term in (3.70). ϵ_2 will become

$$\epsilon_2 = k_r \|\delta\| \tag{3.92}$$

and there would be a smaller constant that does not vary with $\dot{v}_{ICd}^{C}(t)$.

Another change that should be added is the frame in which the gains are added. As seen in (3.90), there is a change of frame back to the UAV frame. This is important because the $k_i e_i$ term will be used to remove slowly varying

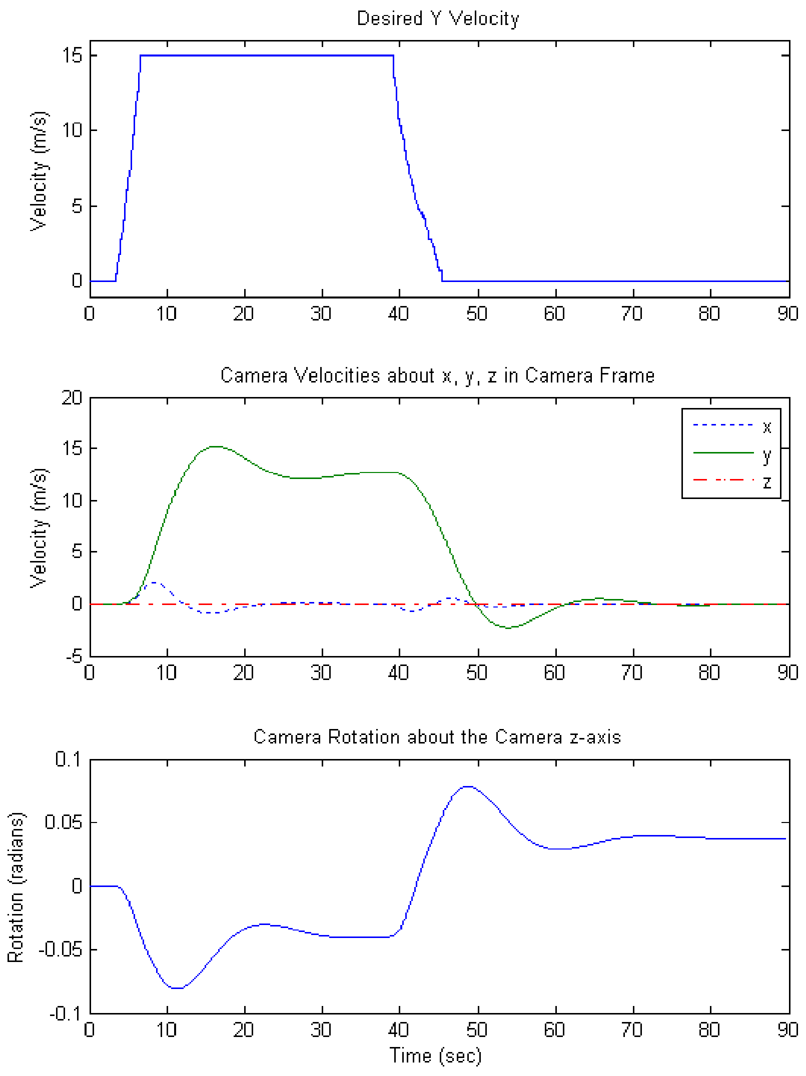


Figure 3.29 Graphs of v_{ICd}^{C}, v_{IC}^{C}, and θ_{I}^{C}: Experiment 4

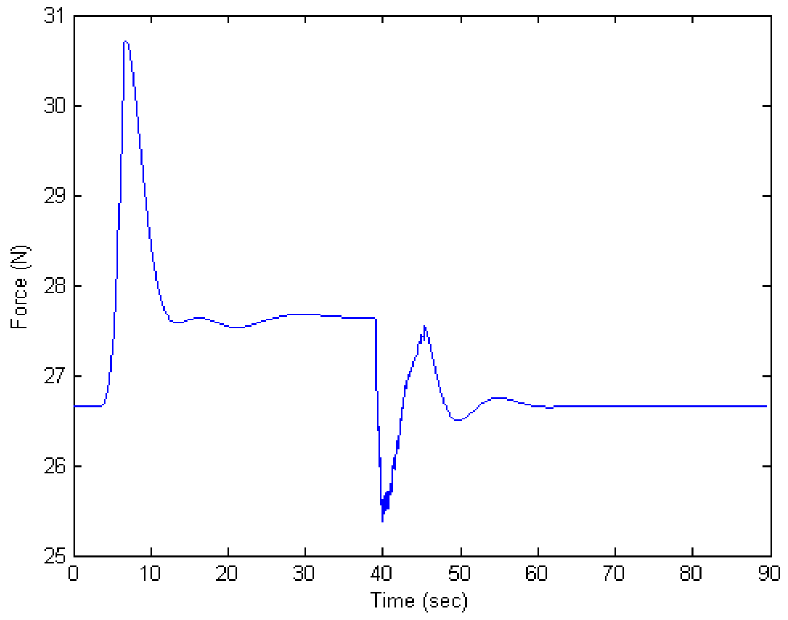


Figure 3.30 UAV Thrust (F_f^F): Experiment 4

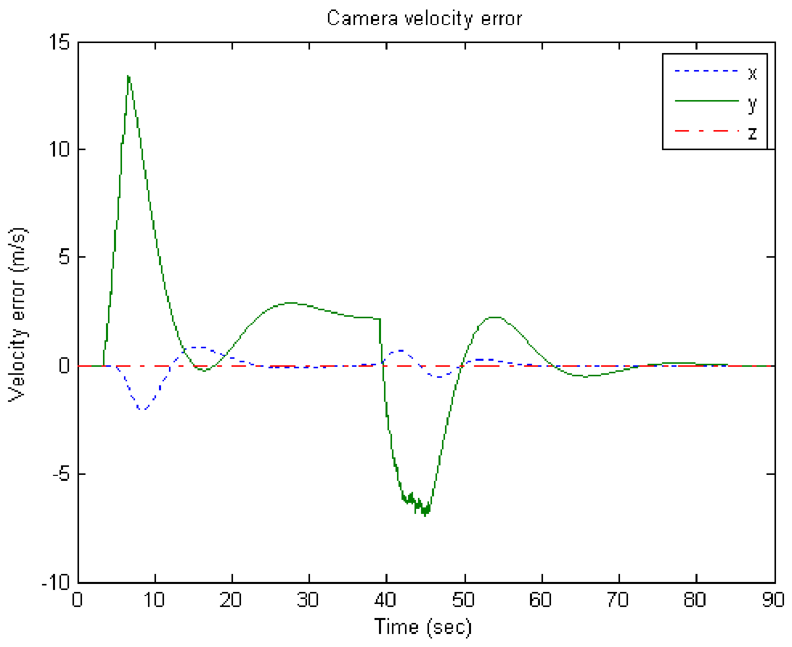


Figure 3.31 Velocity Error (e_v): Experiment 4

errors. Errors such as wind and gravity will never vary as the UAV turns or the camera turns, it is detrimental to make the term with respect to any from other then an inertia frame. Another frame that should be changed is the $k_r e_v$ term. This term is currently in terms of the camera frame with a single constant k_r. This forces all three axes to react the same, when in fact, only the UAV's x- and y-axes behave the same. A solution for this is the change (3.52) into

$$U_1 = -diag\left(R_C^F e_v\right) k_r - r_v \frac{\zeta_1^2\left(\left\|v_{IF}^F\right\|\right)}{\epsilon_1} + S\left(\omega_{IC}^C\right) v_{ICd}^C - G_{11}, \;\; k_r \in \mathbb{R}^3 \quad (3.93)$$

allowing for three gains for each of the three axes of the UAV. $diag(\cdot)$ puts the values of a vector on the diagonal in a square matrix. The same can be done for (3.59) changing to

$$U_2 = -diag\left(R_C^F \int e_\omega\right) k_i + \omega_{ICd}^C. \quad (3.94)$$

This second solution allows for a gain to control the yaw angle differently from the roll and pitch angle, since the yaw torque is weaker then the other two. While these changes make only small improvements in the performance, none of them would change the main concept of this controller. A change that would significantly change the controller is using position and orientation in the feedback loop. While this will definitely improve the performance of the controller, it will no longer be a velocity-only controller and the choices of sensors and their accuracies will be more limited.

Conclusion

This chapter shows the development of a non-linear controller for use on a quadrotor helicopter and a two DOF camera to successfully create a fully actuated fly-by-camera interface. The controller is shown to be Globally Uniform Ultimately Bounded (GUUB) using only velocity information for feedback. The fly-by-camera system allows the camera and UAV to be controlled by the user in an intuitive manner. Simulation verifies that the controller works based on the rigid body model and that the fly-by-camera system is easy for anyone to fly.

APPENDICES

Appendix A
Sensors

MIDG II Interface

To get information from the MIDG II sensor, the sensor had to be connected to a computer through a converter. To communicate with the MIDG II sensor, two tasks must be completed: i) create a hardware link from the MIDG II connector to a PC computer's DB-9 connector, ii) convert the MIDG II RS-422 protocol to the PC computer's RS-232 protocol, and iii) implement software to parse the data packets and store the information in a convenient data structure.. Tasks (i) and (ii) requires combining a variety of devices to complete the data link while task (iii) requires adapting the software provided by Microbotics Inc. to work in both Windows and QNX. Once these tasks are completed, QNX programs will have an easy and reliable interface for retrieving sensor data.

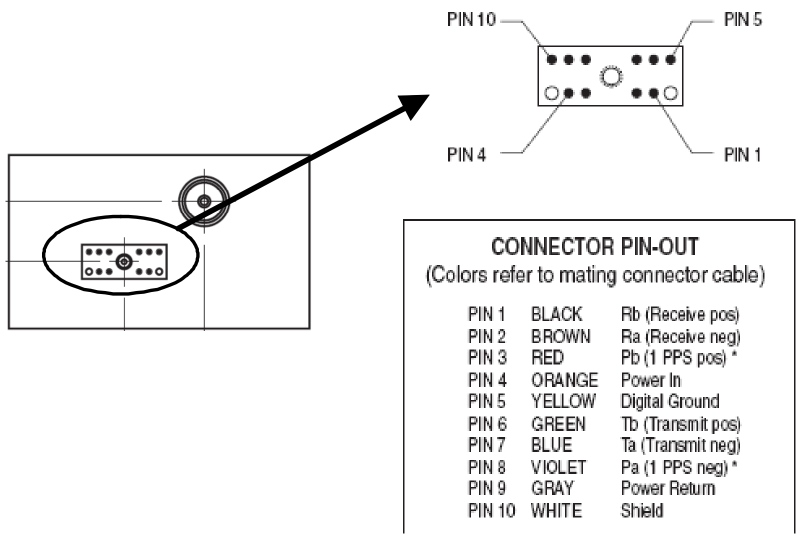
Connecting the MIDG II to a PC has two obstacles: i) the connecting the connector in Figure A.1 to a DB-9 on a PC and ii) converting the signal from RS-422 to RS-232. To power the MIDG II, pins 4 and 9 of the MIDG II connector in Figure A.1 are connected to the on-board Lithium Polymer battery on the UAV through an RCA plug. A standard RJ-45 connector is used to connect to the MIDG II sensor connector, using the pin layout displayed in Table A.1 and labeled as "MIDG II to (A) RJ-45". Then an RJ-45 to DB-9 connector labeled as "(A) to (B)" is used to get the signals into a DB-9 connector using the wiring shown in the second and third column of Table A.1. Then there is a third connector labeled as "(B) to (C)" is used to get the signals back to the RJ-45 connector used on the Serial Converter RS422 to RS232 (SLC22232) board from Microbotics Inc [13]. This converter will change the RS-422 signals into the RS-232 voltage and DB-9 connector which can there be plugged in on a PC computer. There is a USB plug that must be plugged into a computer to power the converter. This completes the first path shown in Figure A.2 using the “Wired Method”. All pathways are bi-directional allowing for configuration data to be sent back to the MIDG II sensor, although a majority of the data is sensor information from the MIDG II to the PC computer.

A second method for connecting the sensor a PC involves the X-Tend Serial Wireless transceivers [14] seen in Figure A.2 using the “Wireless Method”. Using the “(A) to (B)” connector discussed above, the MIDG II can be plugged into an X-Tend transceiver using the RS-422 protocol.. The X-Tend transceiver will wirelessly communicated with another transceiver that can then connect to a computer using RS-232, removing the need for an external RS-422 to RS-232 converter. This is the method used for actual UAV experiments, providing a range of up to 40 miles [14].

To use the data from the MIDG II, a client/server program is written for the MIDG II using the software provided by Microbotics to parse the data packets into a single data structure that includes GPS position, orientation, time, and everything the sensor transmits. For Windows, the data must be saved from the serial port to a file, and then the file can be parsed using the Microbotics program [13]. In QNX, a server program, MIDGServer, runs in the background and receives serial data, then parses the data packets and stores them in a single shared memory location as a data structure. Then a client program running in QMotor will read the shared mem-

Table A.1 Wiring table for the MIDG II connectionst

MIDG II Signal	**(A) RJ-45**	**(B) DB-9**	**(C) RJ-45**
Pb (Not used)	1	NC	NC
NC	2	NC	NC
Rb	3	8	8
Ground	4	5	5
Ra	5	2	2
Ta	6	3	3
Pa (Not used)	7	NC	NC
Tb	8	7	7



CONNECTOR PIN-OUT
(Colors refer to mating connector cable)

PIN 1	BLACK	Rb (Receive pos)
PIN 2	BROWN	Ra (Receive neg)
PIN 3	RED	Pb (1 PPS pos) *
PIN 4	ORANGE	Power In
PIN 5	YELLOW	Digital Ground
PIN 6	GREEN	Tb (Transmit pos)
PIN 7	BLUE	Ta (Transmit neg)
PIN 8	VIOLET	Pa (1 PPS neg) *
PIN 9	GRAY	Power Return
PIN 10	WHITE	Shield

Figure A.1 MIDG II RS-422 Connector [13]

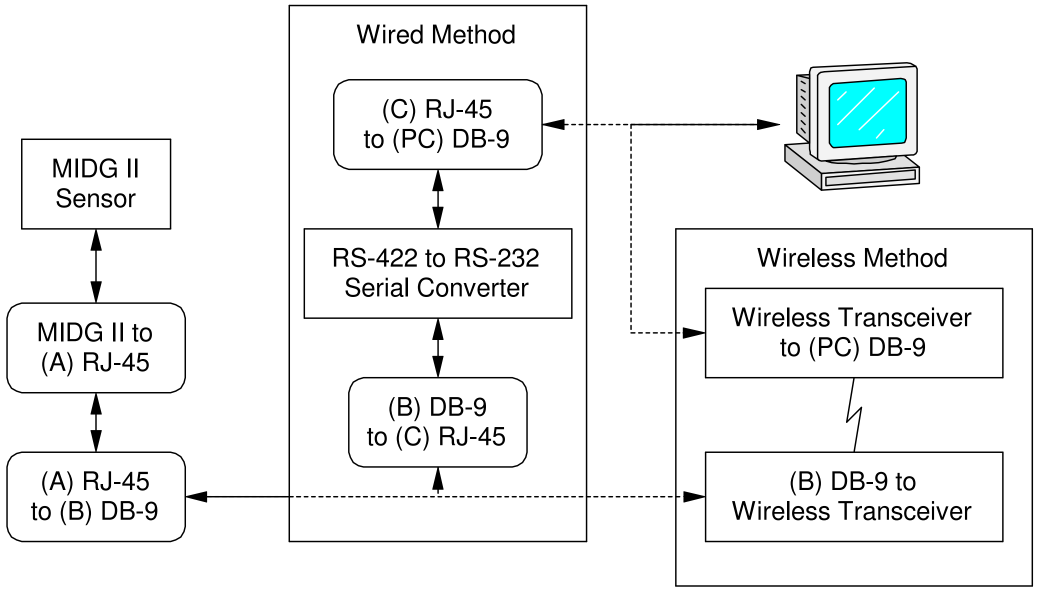


Figure A.2 A connection diagram for a Wireless and Wired Method of connecting the MIDG II sensor

ory location to get feedback information such as orientation. The two programs are designed to run simultaneously without error and can be found at http://www.ece.clemson.edu/crb/research/uav/MIDGServer.zip. In a C++ program, the MIDG II client is initiated by the command

```
d_client = newMIDGClient("/dev/midg0");
```

which initiates the client with the shared memory location "/dev/midg0." To access the sensor data structure of type mtMIDG2State, the method

```
d_client->getM2();
```

is used to access the different pieces of sensor information, defined in "mMIDG2.h." This is the only code needed on the client side to get all of the information from the server.

Magnetometer Background

The MIDG II sensor is a position and orientation measurement system suitable for many UAV applications because of its small size and number of integrated sensors. Part of the MIDG II sensor is a magnetometer used for orientation. Since Most UAVs are gas powered, there is no known electromagnetic interference surrounding the vehicle. This is not the case for electric helicopters. A strong electromagnetic interference has been observed throughout the DraganFlyer X-Pro and all other electric helicopters. While companies such as Rotomotion position the magnetometer far from the magnetic interference, on the DraganFlyer X-Pro there is no safe location for the magnetometer, making the magnetometers unusable. A solution has been found that allows for the use of magnetometer readings during times of no interference.

A typical robotic arm can use encoders to measure angles of individual links and then calculate the end effector's orientation. Having end effector orientation is almost always an important part for any control problem. However, in the case of a flying UAV, there are no links attached to the helicopter.

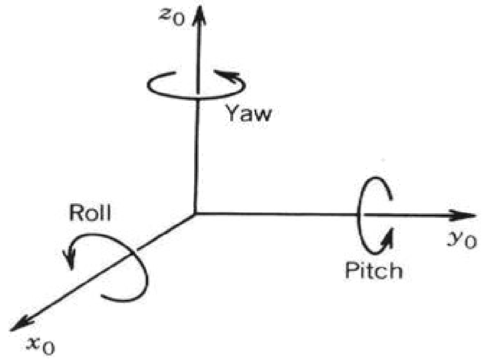


Figure A.3 Yaw, Pitch, and Roll angles

A sensor has to be able to measure the orientation of the helicopter without making ground contact. An inclinometer uses a level to determine the roll and pitch angles, but it is sensitive to vibration and cannot measure yaw. Gyroscopes (gyros) can measure angular rate which can then be used to derive the angle. A sensor like the gyro can measure its own orientation and is very useful in the UAV control problem.

The original spinning mechanical gyro uses conservation of momentum to create gyroscopic forces which maintain a level orientation. Some of the downfalls to mechanical gyros are their bulky size and fragility. The X-UFO by SilverLit successfully uses a mechanical gyro for stabilization of the quad-rotor helicopter. Unfortunately, the fragile wires of the gyro often break.

Micro-Electro-Mechanical Sensor (MEMS) gyros are beginning to manifest themselves as an orientation sensor in many UAVs. MEMS gyros contain vibrating masses that generate a force when they rotate due to Coriolis Forces. By measuring these movements, angular rates can be determined. Some of the advantages of MEMS gyros are their miniature size and durability. There are many navigation sensors that use MEMS gyros for determining orientation, including the MIDG II sensor.

MEMS Gyros can measure angular rates in the yaw, pitch, and roll di-

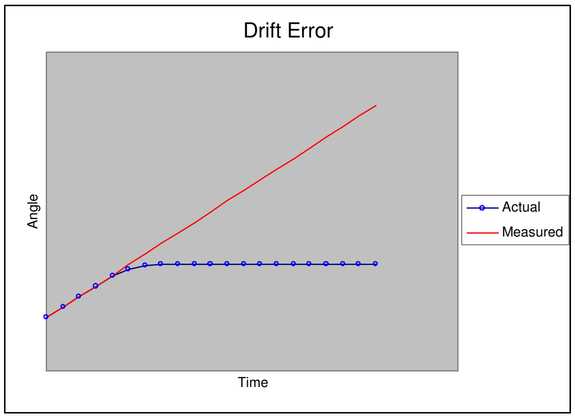


Figure A.4 A simulated example of drift error over time.

rections, as shown in Figure A.3. MEMS Gyros cannot directly measure the orientation angles. Only inclinometers can directly measure orientation, and they can only measure pitch and roll. To get the orientation angles from the gyroscopes, the angular rates must be integrated to get angles as

$$\theta = J^{-1} \int \omega - \omega_{bias}, \tag{A.1}$$

where $\theta(t) \in \mathbb{R}^3$ is the calculated orientation of the sensor with respect to an inertia fixed frame, $\omega\ (t) \in \mathbb{R}^3$ is the measured angular rate of the sensor with respect to an inertia frame, $\omega_{bias}(t) \in \mathbb{R}^3$ is the angular rate bias that slowly varies over time, and $J^{-1}(t)$ is the Jacobian matrix. All sensors have errors in their measurements. In the case of a MEMS gyro, an error will increase over time, as seen in Figure A.4. The source of this error can be a bias in the sensor, noise, or quantization error. In any event, a small error can grow over time. A method of removing this error must be used to achieve accurate readings, or else the sensor becomes ineffective in determining the angle. One possible way of doing this is to calculate a slowly varying $\omega_{bias}(t)$ to correct for the drift.

After mounting the MIDG II onto the DraganFlyer X-Pro seen in Figure A.5, the actual MEMS gyro data itself is heavily noisy, as seen in Figure A.6 and A.7. Figure A.6 shows MIDG II gyro readings while the sensor remains

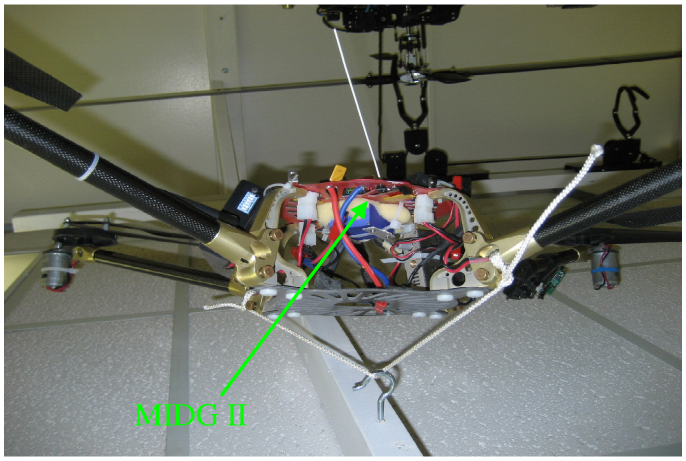


Figure A.5 The MIDG II mounted on the DraganFlyer X-Pro UAV

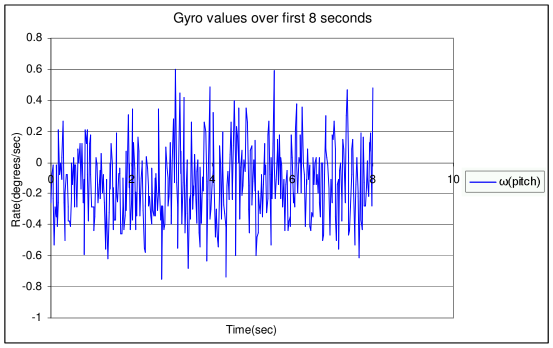


Figure A.6 Gyroscope data for 8 seconds before the quadrotor is powered.

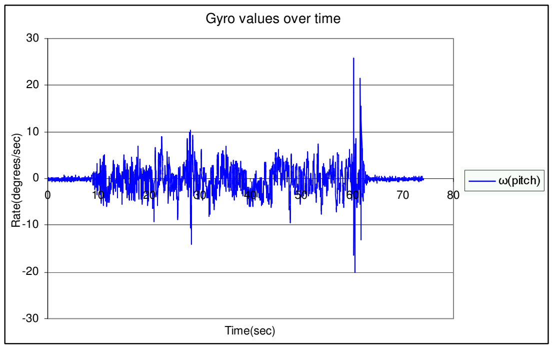


Figure A.7 Gyroscope data for a experiment while quadrotor vibrate and the angles are held constant.

still. In Figure A.7 from 10 to 60 seconds, the MIDG II gyro shows a large change in the rates due to vibration from a DraganFlyer X-Pro helicopter while the angle remained approximately still. This noise requires heavy filtering to finally get a usable angle. The internal hardware of the MIDG II sensor uses a Kalman filter with a secondary measurement method to remove the drift error.

For the roll and pitch angles, the accelerometers act as the secondary method of measurements. When the MIDG II sensor remains still, the accelerometer readings will point one gravity (g) in the downward z-direction according to the earth's inertia frame. The difference in the sensor angle and the known inertia angle will generate $\theta_{acc} \in \mathbb{R}^2$ which is compared with θ_i

(A.1) to generate the ω_{bias} for the roll and pitch angles to correct for drift.

The yaw angle is independent from gravity and cannot use the gravity vector to correct for drift, so an additional sensor must be used for correcting the yaw drift. The secondary yaw sensor used in the MIDG II is a 3-axis magnetometer. Throughout the earth surface a magnetic field can be measured. A north seeking compass is a primitive tool that can detect this magnetic field. A 3-axis magnetometer actually measures this magnetic force (in milligauss) in the x-, y-, and z-directions. These three pieces of information are magnitudes and allow the sensor to know the 3-D magnetic vector. What is desired, however, is the yaw orientation. By projecting the vector on the x-y plane, the yaw angle can be calculated relative to the North Pole. This assumes the magnetic field in the area is approximately the constant (north seeking). This assumption holds true as long as the sensor is not near ferrites or magnetic fields. Three scalars can be used to determine up to two orientations and one magnitude, as shown in Figure A.8. A point, P, contains two angles (θ and ϕ) and one magnitude (ρ). Since there are two angles in the 3-D magnetic vector ($\overline{OP}$), it can potentially be used to measure a second orientation angle. However, this is not done since the accelerometers take care of pitch and roll. The 3-D magnetic vector cannot correct for a third orientation because the third orientation rotates along the axis and cannot be measured. The accelerometers follow the same reasoning and do correct for two orientations: pitch and roll. With the accelerometers and magnetometers combined, three orientations are measured. However, if the magnetic field and gravitational field line up, there would be a singularity and one orientation cannot be measured. This does not happen unless the magnetic field is no longer north seeking.

Both secondary measurement methods mentioned allow drift to be removed and give a reference point to all orientations. For pitch and roll, zero degrees is orthogonal to gravity. For yaw, zero degrees is north, assuming the magnetic field in the area is due to the earth's magnetic field. The secondary measurements are combined with the gyroscope data by Kalman filtering the

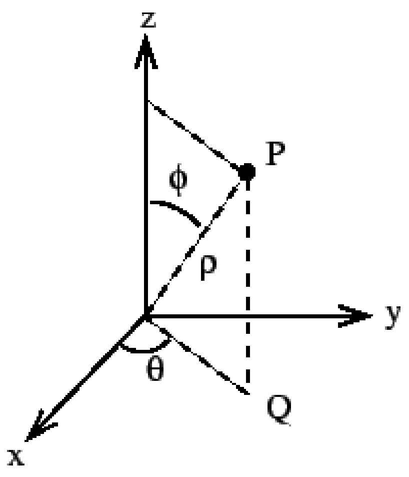


Figure A.8 Cartesian and Cylindrical coordinates

two together over time. This allows for the accelerometer vector to change a little (by shaking the sensor around) without drastically changing the values of pitch and roll, and allowing it to re-correct the values in seconds. In the yaw case, a shift in the magnetic field causes the yaw heading to change in only a few seconds. This is the cause for a great error in the yaw angle.

Magnetometer Problem

One of the applications for the MIDG II sensor is UAV application. Many UAVs use gas power and will have no magnetic interference for the MIDG II sensor. The DraganFlyer X-Pro is an electric helicopter with permanent magnet motors and currents ranging from 30 to 70 amps (total current for all four motors). The permanent magnets can be spaced 20 inches away from the sensor, but the current is all over the DraganFlyer X-Pro and cannot be avoided.

Figure A.9 shows a 3-D plot of magnetic field vectors shown as points in x, y, and z from an experiment using the MIDG II on the DraganFlyer X-Pro. The shift between the different ellipses shows the effects of interference in the magnetic vectors. The magnetic field starts in ellipse a before the motors are turned on. Ellipses b and c show an immediate shift in the direction of the magnetic vector readings after turning on the helicopter. The magnetometer readings vary throughout ellipses b and c depending on how much current flows through the motors. The shift from ellipse a to b or c is across the x- and y-axes, an error of up to 180°. Such a large error is detrimental to the operation of the sensor.

A magnetic source is not the only cause for disturbance in the earth's magnetic field. The existence of a ferrous material alone can cause a shift in the magnetic field, as shown in Figure A.10. Buildings, roads, and bridges commonly contain iron beams and rods for structural reinforcement. Rotomotion uses a magnetometer for orientation data and cannot go near buildings for this reason. This demonstrates one of the many possible errors observed when

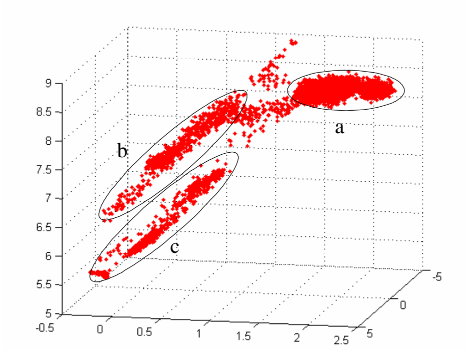


Figure A.9 Magnetometer vector values

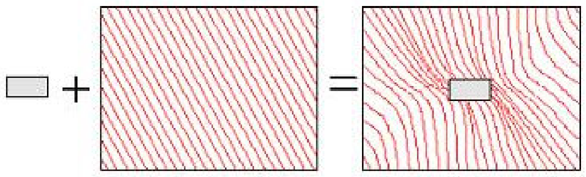


Figure A.10 Ferrous object disturbance in uniform magnetic field

using magnetometers.

Magnetometer Solution

The MIDG II sensor uses MEMS gyros for determining the orientations. If a gyro is used by itself, there will always be a drift error due to a non-zero bias and noise. The magnetometers are used to correct drift error over time by monitoring the earth's magnetic field for all time and continuously correcting yaw $\omega_{bias}(t)$.

The MIDG II sensor has three paths for the modes of operations: IMU, VG, and INS. IMU path (Inertia Measurement Unit) is not used because it only supplies raw unfiltered gyro and accelerometer information. INS (Inertia Navigation System) path requires GPS and cannot be used inside of a building where a majority of the experiments occur. The VG path (Vertical Gyro) uses the gyros, accelerometers and magnetometers to generate yaw, pitch, and roll values. The VG path has 5 sequential modes of operation: VG Initialize, VG Fast, VG Med, VG Slow, and VG SE. Each mode is more accurate than the last. VG SE mode can continue to INS path if GPS is engaged, or return to VG Med mode if the gyro rates saturate (the sensor rotates too fast). To read more about the different sensor modes, please see the MIDG II information sheet "Operating Modes" [13].

Once in the VG SE mode, the MIDG II has a good estimation of the bias in the system. The solution to the magnetometer problem is to correct for yaw for only a limited period of time and not use the magnetometers in the VG SE mode. If the magnetometers were simply not used in VG SE mode, then

(A.1) can be used with a constant value of ω_{bias} for the yaw angle and any drift error would only be due to variations in the yaw ω_{bias} term. A modified firmware is used to disable magnetometer readings in VG SE mode only.

It is important to have an accurate value of ω_{bias} for yaw before entering VG SE mode despite any magnetic interference. Even with the magnetometers turned off in VG SE mode, if ω_{bias} has a corrupted value there will be a large drift over time due to the incorrect ω_{bias}. To alleviate this problem, the sensor remains motionless and the helicopter motors remain off until the sensor enters VG SE mode. The only magnetic interferences that remain are fluctuations in the magnetic fields due to ferrous materials, such as buildings. As long as the magnetic field does not vary and does not line up with the gravitational field, this solution works. There will still be a small drift error from the gyros, but over a 15 minute time frame, it should be unnoticeable.

Upon implementing the new firmware, the magnetic interference due to magnets and magnetic currents has no effect on the yaw reading at all. Microbotics Inc, maker of the MIDG II sensor stated that without the magnetometer readings, the sensor can expect up to 5 degrees of error per minute. After a 10 minute experiment, there was an actual error of about 3 degrees. Even if there was an error of 5 degrees a minute, it could be corrected with knowledge of the sensor's velocity. As the helicopter loses yaw, it would slowly move forward in the wrong direction, and this can be detected and compensated for. It is concluded that even a small 5 degrees per minute error would be acceptable, given the 15 minute fly time of the DraganFlyer X-Pro.

GPS

The Differential GPS sensor is investigated to observer how it reacts under normal conditions. In this experiment, the GPS sensor is positioned on top of the Flour Daniel building and is held stationary while gathering data for 1 hour and 45 minutes. All the GPS data from the MIDG II sensor is recorded during this time. For this experiment, WAAS signals are not received. It has been

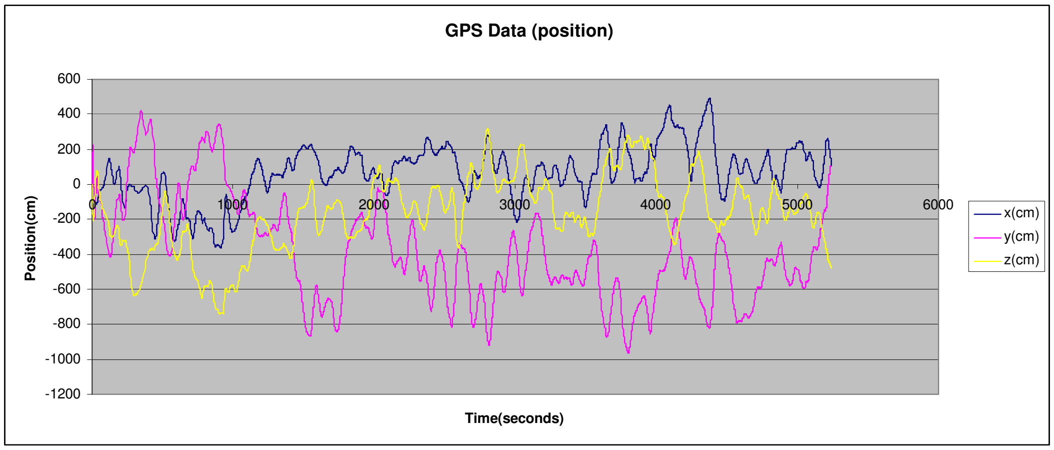


Figure A.11 GPS position values

difficult to receive WAAS signals during 2006 due to a change in the satellites, according to Rotomotion. The ENU (east, north, up) GPS coordinates are use where (0, 0, 0) is the starting point.

Figure A.11 shows the variation of the GPS positions over time. It can be seen that the x varies 5 meters east and about 3 meters west, the y varies 4 meters north and 9 meters south, and the z varies 3 meters up and 7 meters down. Figure A.12 shows the corresponding GPS velocities, which have been observed to be more accurate. A direct integration of the GPS velocities yields the positions seen in Figure A.13. The positions resulting from the integration vary at a considerably slower rate, and only achieves an error of 2 meters in the x- and y-directions by the end of the experiment. As expected, the error in the z direction is larger with a 12 meter error by the end of the experiment.

The GPS values and accuracies are not the only attributes to consider. The MIDG II DGPS sensor can achieve a GPS update every 200 milliseconds. However, it does not always achieve this update rate. Figure A.14 shows a histogram of the time between updates. A majority of the updates are around 1 second, with 2077 of the samples exactly 1.0 seconds apart. The most disturbing part of this is the GPS updates that took too long, with 44

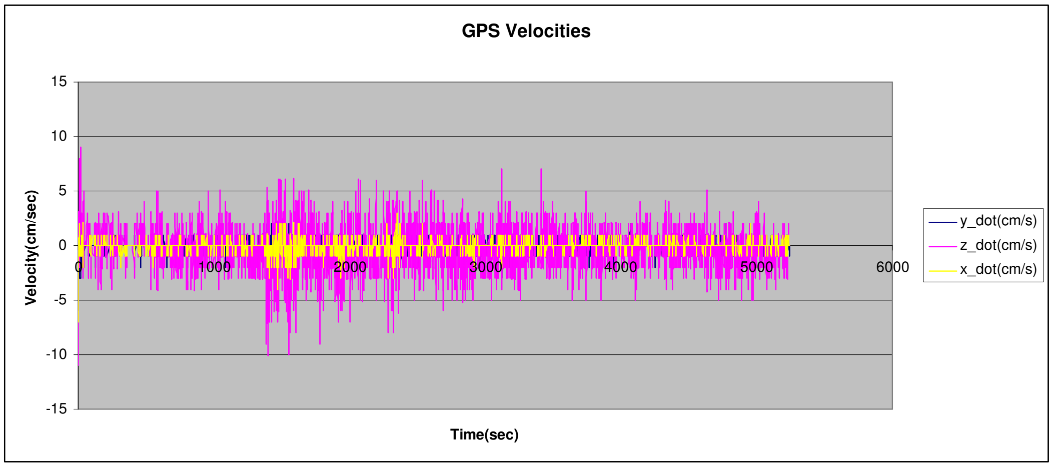


Figure A.12 GPS velocity values

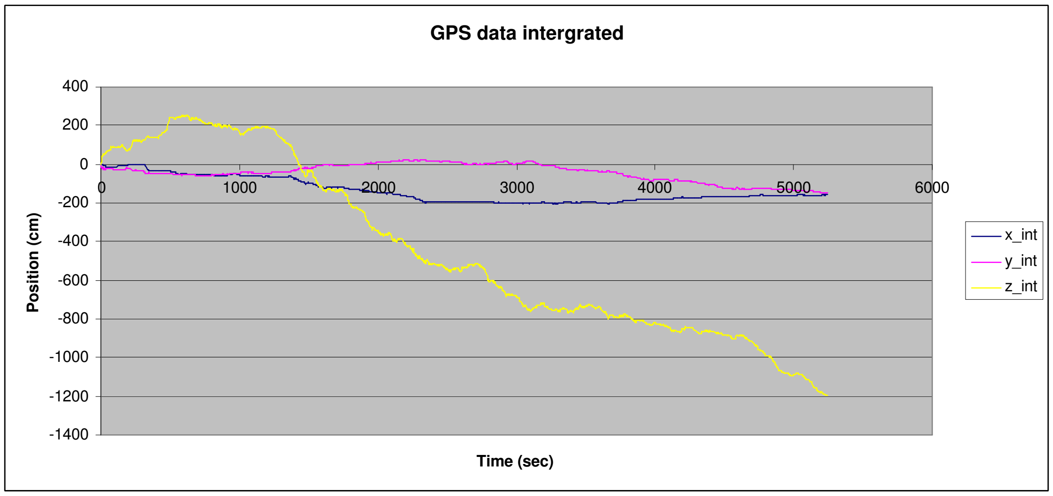


Figure A.13 GPS velocities integrated to yield position

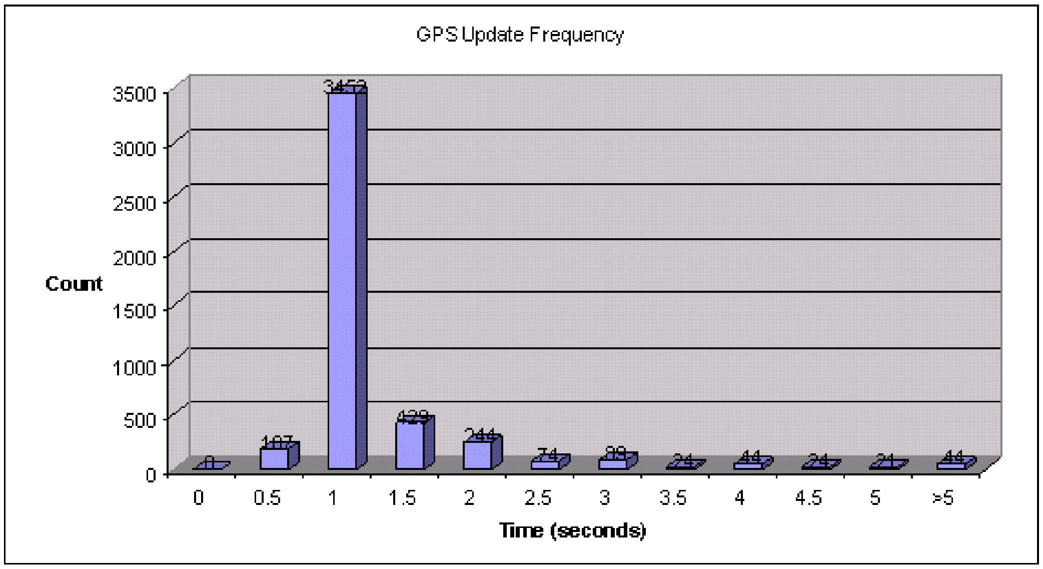


Figure A.14 Times between GPS updates

updates taking over 5 seconds. One update took over half a minute. Being on top of a building, it is unlikely that the satellite connections were lost and Figure A.15 shows a healthy number of satellites at all times. There are always at least 6 satellites connected, and usually 9. Until this problem can be resolved, it would be very difficult to reliably use the MIDG II GPS sensor. If the sensor is fixed or replaced, the integrated GPS velocities and positions can be filtered together, with a heavier emphasis on the velocities, to achieve an accurate position.

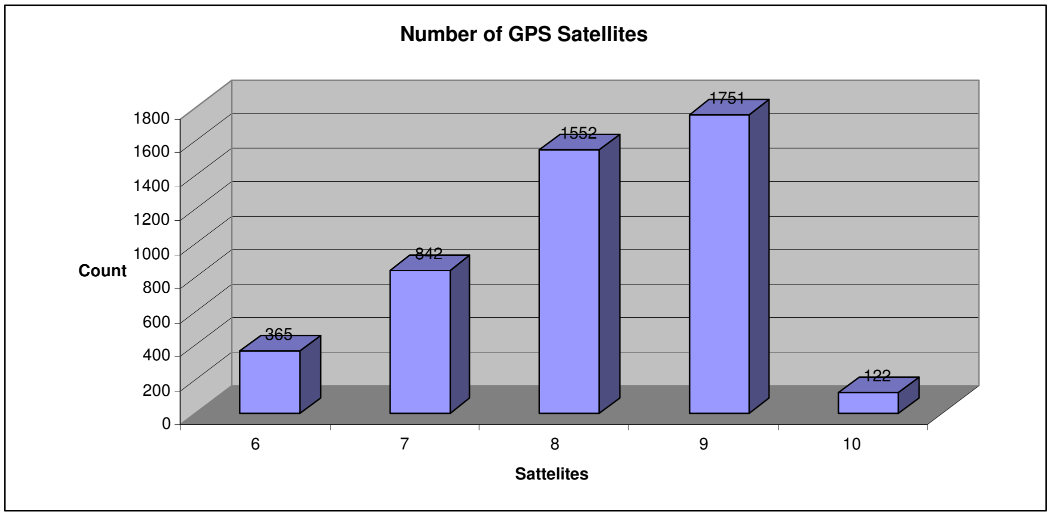


Figure A.15 Number of satellites receiving GPS signals from.

Appendix B
Signal Chasing for Theorem 1

According to Theorem 1 and its subsequent stability analysis from (3.62) to (3.77), $V(t)$ is bounded according to (3.77). Since the conditions in (3.71) and (3.74) are guaranteed to be satisfied by some arbitrary variables that always exist, it ensures that $e_\omega(t)$ and $r_v(t)$ are bounded. The desired trajectories, $v^C_{ICd}(t)$ and $\omega^C_{ICd}(t)$, are assumed to be bounded. The transformation matrices $R(\Theta)$ and $J_F^{-1}(\Theta)$ are bounded under the assumption that $\theta_p \neq \pm\frac{\pi}{2}$, based on the definition of (3.10) and (3.11) which applies to all rotation matrices. Now it can be said that $e_v(t)$ in (3.26) is bounded, resulting in $v^C_{IC}(t)$ in (3.25) and $\omega^C_{IC}(t)$ in (3.42) being bounded. Thus $v^F_{IF}(t) \in \mathcal{L}_\infty$ by (3.28) which results in $\dot{x}^I_{IF}(t) \in \mathcal{L}_\infty$ by (3.2). Since $J_C(t) \in \mathcal{L}_\infty$ by (3.19)/(3.23), all elements of $\bar{B}(t)$ in (3.45) are bounded. $\zeta\left(\left\|v^F_{IF}(t)\right\|\right)$ can be shown to be upper and lower bounded by the inequality in (3.53). $N_1(\cdot)$ and g are assumed to be bounded, yielding $N_{11}(\cdot)$ and $G_{11}(\cdot) \in \mathcal{L}_\infty$ by (3.37) and (3.38). Now $U_1(t)$ and $U_2(t)$ can be shown to be bounded in (3.52) and the lower half of (3.48). Owing to $\bar{B}(t)$ and $U(t) \in \mathcal{L}_\infty$, $\bar{U}(t)$ is bounded by (3.46), yielding $u_1(t)$, $\omega^F_{IF}(t)$, and $\dot{\theta}_C(t)$ are bounded. Thus $\omega^F_{FC}(t)$ is bounded by (3.43). Owing to $\omega^F_{FC}(t) \in \mathcal{L}_\infty$, $S\left(\omega^F_{FC}(t)\right)$ is also bounded by (3.6), resulting in $\dot{R}^C_F(t)$ being bounded by (3.30). Based on (3.10), $J_F(t)$ can be found to be

$$J_F = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\psi) & \sin(\psi)\cos(\theta) \\ 0 & -\sin(\psi) & \cos(\psi)\cos(\theta) \end{bmatrix}, \tag{B.1}$$

which is bounded. Since $\omega^F_{IF}(t) \in \mathcal{L}_\infty$, $\dot{\Theta}^F_{IF}(t)$ is bounded by (3.8). $\left\|\dot{v}^C_{ICd}(t)\right\|$ is assumed to be upper bounded by $\beta_1(t)$ expressed in (3.67). Owing to $U(t) \in \mathcal{L}_\infty$, $\dot{r}_v(t)$ is bounded by (3.41) and $\dot{v}^F_{IF}(t) \in \mathcal{L}_\infty$ by modeling equation (3.3). Therefore, we can conclude that all signals are bounded in the velocity control of fly-by-camera interface.

BIBLIOGRAPHY

[1] FlightGear, http://www.flightgear.org/.

[2] T. I. Fossen, *Marine Control Systems : Guidance, Navigation, and Control of Ships, Rigs, and Underwater Vehicles*, Marine Cybernetics, 2002.

[3] T. Hamel, R. Mahony, R. Lozano, and J. Ostrowski, "Dynamic Modelling and Configuration Stabilization for an X-4 Flyer," *Proceedings of the IFAC World Congress*, Barcelona, Spain, July 2002.

[4] V. Chitrakaran, D. Dawson, H. Kannan, and M. Feemster, "Vision-Based Tracking for Unmanned Aerial Vehicles." Technical Report CU/CRB/2/27/06/#1, College of Engineering and Science Control and Robotics, Clemson University, Feb. 2006.

[5] V. Chitrakaran, D. Dawson, J. Chen, and M. Feemster, "Vision Assisted Autonomous Landing of an Unmanned Aerial Vehicle," Proceedings of the IEEE Conf. on Decision and Control, Seville, Spain, pp. 1465-1470, December, 2005.

[6] P. Pounds, R. Mahony, J. Gresham, P. Corke, and J. Roberts, "Towards Dynamically-Favourable Quad-Rotor Aerial Robots," *Proc. of the 2004 Australasian Conf. on Robotics and Automation*, Canberra, Australia, Dec. 2004.

[7] G. Hoffmann, D. Rajnarayan, S. Waslander, D. Dostal, J. Jang, and C. Tomlin, "The Stanford Testbed of Autonomous Rotorcraft for Multi Agent Control (STARMAC)", *Proceedings of the 23rd Digital Avionics Systems Conference*, Salt Lake City, Utah, pp. 12.E.4-12.E.10, November, 2004.

[8] J. Jang and C. Tomlin, "Longitudinal Stability Augmentation System Design for the DragonFly UAV using a Single GPS Receiver," Proceedings of the AIAA Guidance, Navigation, and Control Conference, Austin, Texas, AIAA Paper Number 2003-5592, August 2003.

[9] A. Tayebi and S. McGilvray, "Attitude Stabilization of a VTOL Quadrotor Aircraft", *IEEE Transactions on Control Systems Technology*, pp. 562-571, May 2006.

[10] T. Hamel and R. Mahony, "Attitude estimation on SO(3) based on direct inertial measurements", *Proc. of the 2006 IEEE Int. Conf. on Robotics and Automation*, Orlando, Florida, pp. 2170-2175, May 2006.

[11] P. McKerrow, "Modelling the Draganflyer Four-Rotor Helicopter", *Proc. of the 2004 IEEE Int. Conf. on Robotics and Automation*, pp. 3596-3601, New Orleans, April 2004.

[12] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor", *Proc. of the 2004 IEEE Int. Conf. on Robotics and Automation*, New Orleans, Louisiana, pp. 4393-4398, April 2004.

[13] MIDG II INS/GPS Sensor, http://microboticsinc.com/ins_gps.php.

[14] XTend RS-232/RS-485 RF Modem, http://www.maxstream.net/products /xtend/rf-modem-rs232.php.

[15] Procerus Technologies: Vision-Centric, http://www.procerusuav.com /cameraControl.php.

[16] QNX Software Systems, http://www.qnx.com/.

[17] QMotor Real-Time Control Environment, http://www.ece.clemson.edu /crb/research/realtimesoftware/qmotor/index.htm.

[18] DraganFlyer X-Pro, http://www.rctoys.com/draganflyerxpro.php.

[19] Logitech Extreme 3D Pro Joystick, http://www.logitech.com/.

[20] M. W. Spong and M. Vidyasagar, *Robot Dynamics and Control*, John Wiley and Sons, Inc: New York, NY, 1989.

[21] M. De Queiroz, D. Dawson, S. Nagarkatti, and F. Zhang, *Lyapunov-Based Control of Mechanical Systems,* Birkhäuser, Boston, MA, 2000.