The Pennsylvania State University
The Graduate School
College of Engineering

AN OPEN-LOOP TRAJECTORY IMPLEMENTATION FOR FREE FALL TO POWERED FLIGHT TRANSITION FOR MULTIROTOR AIRCRAFT

A Thesis in
Aerospace Engineering
by
Tyler Rosenberger

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2020

The thesis of Tyler Rosenberger was reviewed and approved by the following:

Jacob W. Langelaan
Professor of Aerospace Engineering
Thesis Advisor

Eric Johnson
Professor of Aerospace Engineering

Puneet Singla
Professor of Aerospace Engineering

Amy Pritchett
Professor of Aerospace Engineering
Head of the Department of Aerospace Engineering

Abstract

Parachute release of rotorcraft presents a relatively unexplored method of vehicle deployment. This release method presents significant challenges in both modelling and control of quadrotor vehicle motion. The largest of these problems involves the occurrence of vortex ring state during high-falling-velocity maneuvers. Without proper control methodology, a falling rotorcraft can become unstable - and in some cases uncontrollable - if vortex ring state is allowed to develop.

This thesis seeks to stabilize and control a falling quadrotor through the use of an intermediate open-loop stage of control before passing full control over to a closed-loop controller. This is accomplished by pre-generating a trajectory maneuver involving a sequence of pitch and body z-axis acceleration commands in an attempt to regain thrust while rapidly descending. These trajectories are generated using a series of heuristically-motivated polynomial splines that seek to mimic the trained behavior of human helicopter pilots.

Optimization of the governing trajectory parameters is then explored through the use of particle swarm optimization in MATLAB. The peak pitch angle and time of maximum thrust are found using the PSO algorithm, and the robustness of the solutions is then examined.

Following MATLAB optimization, the trajectory is implemented within a Gazebo simulation environment through the use of ROS®. The simulation environment and vehicle model are changed to match the flight conditions of a ready-to-fly testbed vehicle, the AVIA Lab-constructed Hoverfly. Multiple trials are conducted for various final conditions to test the effectiveness of the proposed open-loop control methodology.

The presented open-loop control method is shown to allow successful control of a dropped quadrotor vehicle. The designed trajectories are shown to reliably and predictably force the vehicle into an easier-to-control state that allows a traditional closed-loop controller to achieve both a zero-velocity hover and forward-flight condition at the end of the maneuver.

Table of Contents

List of Figures

List of Tables

List of Symbols

$\dot{q}$	First time derivative of quantity 'q'
$\ddot{q}$	Second time derivative of quantity 'q'
$\mathbf{q^B}$	A vector in frame B
$\mathbf{C_A^B}$	Transformation matrix from frame A to frame B
ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle
T	Thrust
$\mathbf{d}$	Drag force
ρ	Air density
ω	Angular velocity
m	Mass
g	Standard gravitational parameter
C_D	Coefficient of drag
k_Q	Air torque coefficient
U	Body frame x velocity
V	Body frame y velocity
W	Body frame z velocity
P	Body frame x-axis angular rate
Q	Body frame y-axis angular rate

R	Body frame z-axis angular rate
I_{xx}	x-x moment of inertia
CV	Coefficient of variation

Acknowledgments

I would like to thank my friends and family for all their support. Their willingness to be sounding boards for my ideas and their interest in my work was immeasurably helpful.

I am also extremely grateful to my advisor, Dr. Jack Langelaan, for his support and guidance throughout the past two years. The freedom he gave me to explore UAS development, flight, and simulation has been an amazing experience.

I would also like Dr. Puneet Singla, Dr. Eric Johnson, and Dr. Amy Pritchett for taking the time to review my thesis.

Lastly, I would like to thank all my labmates in the Air Vehicle Intelligence and Autonomy (AVIA) lab for their technical suggestions and assistance. Everyone's willingness to spend their time explaining complicated concepts and demonstrating procedures has been instrumental in my work.

Chapter 1
Introduction

Parachute release of unmanned aerial vehicles (UAVs) is a relatively new area of interest for aerospace research. In this introductory chapter, the parachute release maneuver discussed throughout this thesis is presented against the backdrop of deployable aircraft as a whole. The challenges facing this novel release method are discussed, with an emphasis on vortex ring state avoidance. Then, the design rationale that guides the rest of the thesis is presented; a semi-heuristic design approach is applied to generating an open-loop trajectory to reduce the solution space and guide trajectory generation. The implemented control architecture which makes the open-loop trajectory portion possible is then briefly covered at a high level. This is then followed by a summary of thesis contributions and a reader's guide for the remainder of this thesis.

1.1 Deployable UAVs

The use of UAVs is a rapidly growing frontier of novel ideas and applications. This can be seen in myriad industries ranging from the military and defense sector to meteorology to fields as seemingly unrelated as healthcare. As UAV use sees such rapid growth, engineers are pushed to explore innovative ways to improve existing uses or even find new uses altogether. This goal has led researchers down numerous development paths including such avenues as energy utilization, vehicle geometry, and control allocation. However, the flight plan has remained largely unchanged; vehicles still usually take off from the ground, fly their missions, and then land. Now, emerging use-cases may benefit from the ability of UAVs to instead "take off" midair before executing their planned mission.

The deployment of UAVs while midair is a rather new field of interest in the Aerospace community. The idea of air-launching aircraft in general is certainly not a novel one; experiments of such maneuvers date back to the early 1900s when biplanes would be

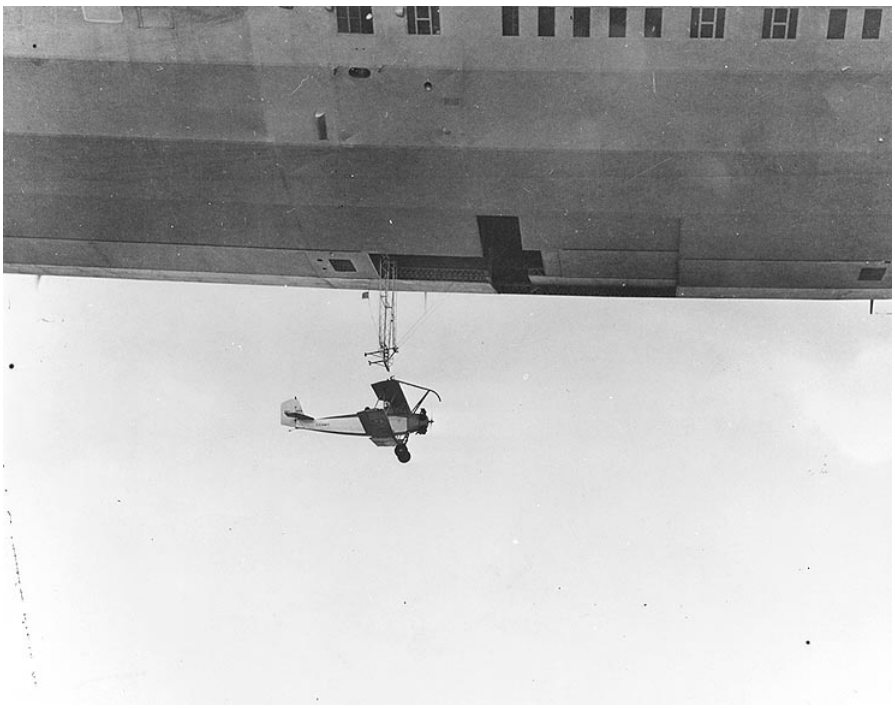

Figure 1.1. USS Akron Releases a Consolidated N2Y

launched from the belly of airships [1] (depicted in Figure 1.1). Research into such deployments then remained quite stagnant for many years, but recent work has shown a renewed interest in air-launching UAVs for both urban and military applications. It was not until 2006 that the first hovering aircraft was air-launched with published flight results [2], and a recent Defense Advanced Research Projects Agency (DARPA) project team air-launched (and air-recovered) their own vehicle [3]. Other work has even paved the way for the use of parafoil UAV systems to allow controllability of falling payloads ejected from aircraft [4] [5].

The deployable UAV application motivating this thesis is Johns Hopkins University Applied Physics Laboratory's Dragonfly project: On June 27, 2019, Dragonfly was named as the finalist in NASA's New Frontiers Program [6] [7]. This vehicle is designed to enter Titan's atmosphere through the use of an aeroshell and parachute system and then release itself from the parachute as depicted in Figure 1.2. After release, the vehicle is to transition to powered flight and fly away and eventually land. This mission highlights the emerging need for UAVs which can be released from a descending parachute.

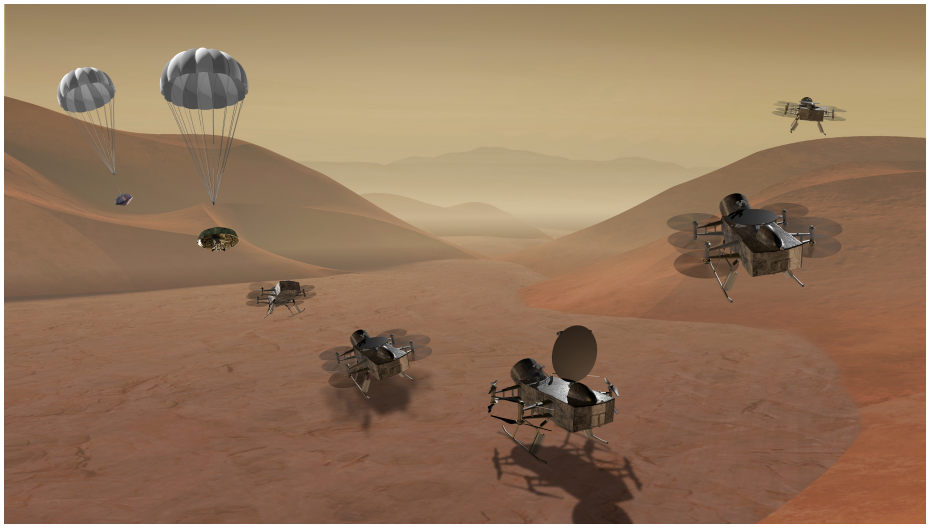

Figure 1.2. Dragonfly Mission Concept [8]

1.2 Challenges of Parachute Release

Parachute release presents a whole host of challenges not present in the standard operations of a UAV. The parachute itself presents a significant risk of vehicle damage. If the rotors are turned on too soon after release (or even before release), they could foreseeably tangle themselves in the parachute itself or the surrounding release apparatus. While such an event would certainly be catastrophic for the mission, this issue can easily be mitigated by preventing the vehicle's rotors from spinning up until separation from the apparatus is ensured. Doing so simply requires introducing some system delay, t_{delay}, which prevents any actuators from functioning until such a time has elapsed. However, this delay introduces several more challenges.

By introducing a delay into the sequence of events, the released vehicle is subject to motion due only to environmental factors without any way to correct itself. The most dangerous factor is the combination of upward airflow (relative to the vehicle's frame of reference) through the rotors combined with gravity constantly accelerating the vehicle downwards. This free-falling state is required due to the release, but it can cause severe problems. Surprisingly, little research has been done on the behavior and control of quadrotor vehicles in extended free fall conditions, as many controllers are designed to control the vehicle well before significant vertical velocities are achieved. However, flight testing videos on YouTube by rctestflight [9] show that rotorcraft UAVs can become unrecoverable if allowed to free fall for too long. The rctestflight video shows severe wobbling and uncontrollability in the worst cases. This suggests that special care

must be taken to ensure that the vehicle's time in free fall does not cause severe control problems when powered flight begins.

In addition to mechanical issues that a free-falling quadrotor may experience, the onset of vortex ring state (VRS) contributes greatly to the lack of controllability observed in [9]. When a rotorcraft's downward velocity approaches the downwash velocity of the rotor, there is the potential for the vehicle to enter VRS. In the words of a NASA Technical Publication [10], "the onset of vortex ring state is associated with the collapse of an orderly structure of the rotor wake into a highly disturbed, irregular, aperiodic flow state". During VRS, the vehicle's rotors begin recirculating a toroidal-shaped airflow around themselves which can result in a severe loss of lift and controllability. This recirculating flow is shown in Figures 1.3 and 1.4. In the worst cases, aircraft descent rates can reach 6000 ft/min, and the aircraft can be subject to highly irregular pitching moments [11]. The thrust distribution along the length of the rotor becomes highly varied, which can cause instability or mechanical damage in the most extreme cases [12]. This is an extremely dangerous state for a falling vehicle to operate within, so it is necessary to consider it. Dealing with this issue requires two considerations: avoidance and correction. The easiest solution for dealing with VRS is to simply avoid it altogether by avoiding flight regimes that allow for it to occur. However, due to the time in free fall required to separate the vehicle from the parachute, this may be impossible, thus requiring the UAV to have a way of removing itself from VRS. Johnson and Turbe briefly describe aerial deployment of a ducted fan aircraft from a larger helicopter [2] [13]. In that case, vortex ring state was unlikely to cause a challenge to the flight control system (high disc loading of the ducted fan means that rotor downwash is high), although at very high descent rates other challenges (e.g. control surface reversal) may occur.

One way to remove the vehicle from VRS is to power out of it. However, this is undesirable as it requires the rotors to produce significantly greater induced air velocity as that produced at hover. For extremely lightweight vehicles, this may be possible, but for many applications, the vehicle will be incapable of this maneuver, or it may cause the vehicle to waste significantly more power. Another correction method is to pitch the nose of the vehicle down to gain forward airspeed until lift is regained [14]. This is supported by researchers, as Brand et al. make note that "a sudden reorientation of the rotor tip path plane effectively establishes a new thrust vector and introduces an acceleration that does not support ring accumulation" [16]. This solution is feasible for vehicles of all sizes and thrust capabilities, so it will be the solution pursued in this thesis.

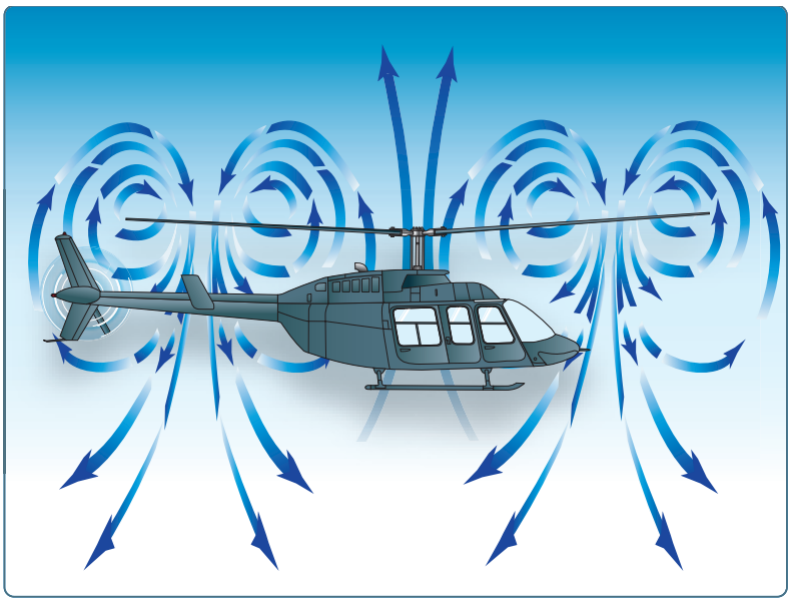

Figure 1.3. VRS Flow Pattern [14]

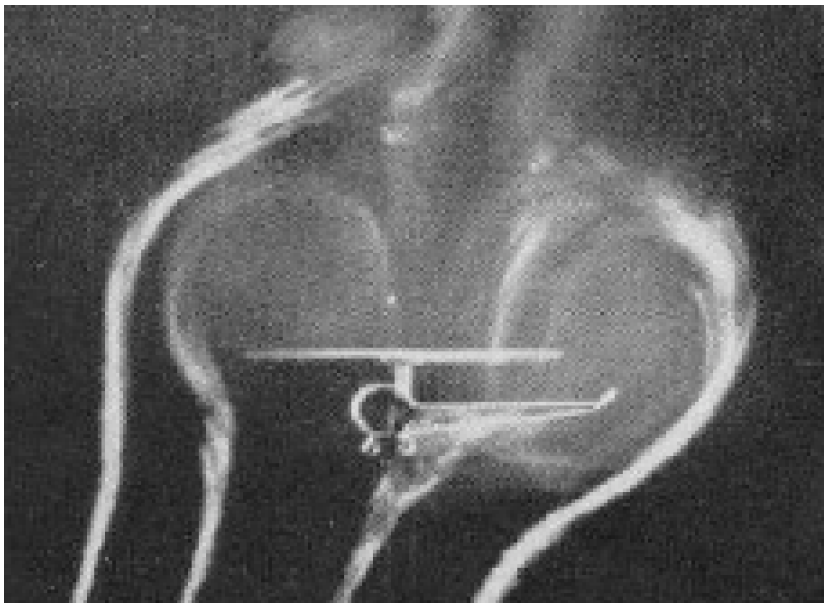

Figure 1.4. VRS Flow Visualization [15]

1.3 A Semi-Heuristic Design Approach

As mentioned previously, one method of removing a descending rotorcraft from VRS is to perform a pitch-down maneuver to increase forward airspeed to regain lift. While intended for the correction of VRS, this maneuver can also be used to prevent VRS. By commanding a falling UAV to first pitch down before attempting to achieve its final

steady state flight condition, we can protect the vehicle against entering VRS. Previous research [17] has investigated a method of optimizing PID controller gains to perform a similar maneuver via closed-loop controller commands. In another VRS avoidance application, researchers have also investigated finding optimal-time descent trajectories while avoiding VRS via actuation in roll [18].

Instead of searching for an optimal control law, another way of ensuring safe operation of the vehicle is to apply the method that will be the focus of this thesis: a semi-heuristic design approach to creating an open-loop trajectory. The "heuristic" part of this method relies on using a tried-and-tested trajectory design as a model for what the solution should look like before trying to find the best way to perform that maneuver. In more specific words, if we start our trajectory design with the goal of always forcing the vehicle into a pitch-down maneuver followed by steady forward flight, we can then optimize the maneuver around the angles, accelerations, and time that define it. In this way, the size of the design space is significantly reduced, and we are using a flight maneuver that has significant validation in the real world.

Implementing this heuristic aspect of design involves forcing the vehicle to behave as if it were actually being piloted in real time instead of using its velocity and acceleration measurements to adjust the maneuver. Such behavior can be accomplished by using open-loop control to react to the parachute release as if it were a learned and rehearsed behavior.

The "semi" part of semi-heuristic refers to the way that we can improve the naive implementation: we can analytically make changes to trajectory parameters to preserve the heuristic trajectory's shape and functionality while gaining improvements in vehicle performance. Additionally, we can introduce some closed-loop feedback control into the system to make it behave more intelligently in a real-time flight situation.

1.4 Control Architecture

To transition to powered flight successfully, a successive series of controllers must be used to make the vehicle behave in the heuristic manner described previously. We need to ensure two things: first, that the vehicle begin its recovery maneuver with the designed open-loop maneuver, and second, that the vehicle eventually end up in a closed-loop state where we can force some final flight condition. This layering of multiple controllers can be achieved by forcing the vehicle to progress through a series of states upon release from its parachute.

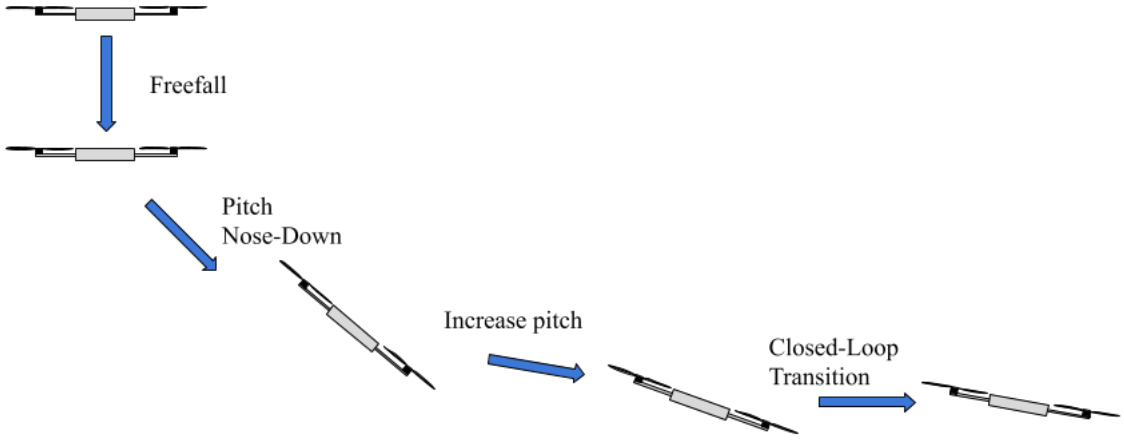

Figure 1.5. Sketch of Open-Loop Pitch-Down Maneuver

During the onset of parachute release, the vehicle's rotors must be kept stationary to prevent tangling or reverse rotation of the props due to the upward airflow. After some delay, the vehicle's semi-heuristically-designed open-loop control commands are sent to the vehicle to initiate and force a pitch-down maneuver. After these commands are expired, the vehicle is assumed to be in an easier-to-control state, and control is passed over to a simple feedback velocity controller. The successive control method is sketched in Figure 1.5. This progression through various stages of control allows for different stages of the release to be handled separately to ensure the vehicle behaves as desired, and it allows us to easily isolate the open-loop portion of the maneuver for optimization purposes. With good enough modelling and optimization to generate the open-loop commands, the vehicle should behave quite predictably.

1.4.1 Open-Loop Control Motivation

As mentioned above in Section 1.2, VRS creates a highly unstable system with high turbulence and sudden changes in thrust dynamics. This makes detection of VRS and subsequent adapted control of the quadrotor difficult if it were to occur. Traditional control of the vehicle's position, velocity, and/or acceleration has the potential to fail in this case, likely commanding more and more throttle until motor saturation occurs. At this point, the vehicle would still be falling and unable to force itself out of the recirculating flow.

To combat this, the vehicle's velocity can be commanded via open-loop inputs to the

commanded thrust and pitch angle. This is an implementation similar to model-based predictive control (MBPC), which has seen use in a variety of fields including UAVs more recently [19]. In contrast to MBPC, the commands are to be pre-planned and not changed in real time. The inputs can be measured and adjusted, though their effects on the desired dynamics are not due to the very short time span of the planned maneuver and the high likelihood of instability. Instead, this implementation is more similar to open-loop intermittent feedback control [20], where periodic measurements of the vehicle's state are used to generate open-loop commands over a relatively short time-horizon. So long as the system model adequately describes the relationship between the inputs and outputs and the time interval of control is not too long, open-loop control can produce close-to-desired motion. Though the optimal control design in the open-loop case is subject to the initial conditions, the optimal closed-loop control law is not [21]. This suggests that varying initial conditions of the vehicle's state should be investigated when developing the open-loop control method.

1.4.2 Sensor Usage

One last consideration that should be addressed is the availability of sensors for determining the state of the vehicle. While state information is less important during open-loop control than it would be in closed-loop control, care must be taken to ensure that the control architecture is capable of operating within the limitations of the vehicle.

Since this project is motivated by the Dragonfly project, the decision was made to design the system to not use actual position data and instead treat the Hoverfly as if it were flying in a GPS-denied environment during the recovery portion of its mission. Fortunately, the Pixhawk used for navigation in the implementation portion of this thesis is capable of estimating velocities and orientations by integrating data from its inertial measurement units (IMUs) and gyroscopes.

These sensing limitations require that the open-loop trajectory be designed to only define commands for the vehicle's orientation and body-accelerations since any other measurements at the onset of the maneuver would be subject to integration errors [22]. By keeping the recovery maneuver short and open-loop, we can also avoid the need for sensor augmentation, which usually involves complicated vision-based systems [23] [24].

1.5 Contributions

The main contribution of this thesis is the definition of a transition to powered flight trajectory that can be followed in open loop and development of a particle-swarm-based approach that optimizes this open-loop trajectory.

The shape of the trajectory is based on well-known piloting strategies for recovery from stall (in fixed-wing aircraft) and vortex ring state (in rotary wing aircraft). It consists of a combination of a pitch maneuver and aircraft body-z acceleration (push the nose down, increase thrust, and pull the nose up to recover from the dive); both are defined as sequences of commands that are parameterized using a spline. Particle swarm optimization is used to determine spline parameters that minimize a cost function based on safety of flight. Here the cost function seeks to minimize the time spent in a condition where vortex ring state is likely to occur while bringing the aircraft to a steady state (i.e. constant velocity) flight condition. The result is a "heuristic/optimal" trajectory: heuristic because the shape is defined based on known piloting strategies, optimal because the parameters defining the shape are optimized to maximize a reward.

Robustness of this approach is evaluated by applying the optimized control sequence to recovery cases that have varying initial conditions. Software in the loop simulations using the Gazebo environment are used for further validation. These results show that a safe transition to powered flight is possible for a parachute-dropped multi rotor vehicle.

1.6 Reader's Guide

The subsequent sections of this thesis are organized as follows:

Chapter 2 contains a thorough description of the vehicle and system dynamics involved in creating both the open-loop and closed-loop control structures necessary for recovering a quadrotor vehicle. This includes quadrotor dynamics considered in both 6-DOF and 3-DOF as well as the controller design considerations in both the open-loop and closed-loop portions of flight for completeness. This section also includes a simplified model of VRS and an overview of how the vehicle testbed was designed and its properties ascertained.

Chapter 3 covers the way in which the open-loop (with respect to velocity) trajectory was defined and optimized. This includes the way that vehicle properties were implemented in MATLAB and the use of PSO in optimizing trajectory parameters. The optimization results are presented for several final goal vehicle states. This section concludes with an evaluation of the generated trajectory's robustness to changes in the

vehicle's initial conditions.

Chapter 4 presents an implementation of the controller in Gazebo for SITL validation. This covers the properties of the simulation environment, vehicle communications, and trajectory implementation. After implementation, performance metrics are analyzed to assess the validity of semi-heuristic design.

Chapter 5 concludes the thesis with a summary of contributions and recommendations for future work.

Chapter 2
Defining the Release Trajectory Problem

This chapter covers the mathematics defining the motion and control of the Hoverfly during the release maneuver. First, rotorcraft dynamics are covered for the 6 degree-of-freedom and reduced 3 degree-of-freedom system used in the simulation and analysis of the problem solution. Next, the design process for the Hoverfly vehicle is covered along with the methods used to determine the vehicle parameters and tuned flight control unit (FCU) parameters for the purposes of developing an adequate system model. Then, the two stages of the release maneuver are described and defined, first by considering the open-loop heuristic-driven portion followed by the subsequent closed-loop velocity control method.

2.1 Quadrotor Dynamics

Rotorcraft have numerous physical properties which affect vehicle dynamics, and they experience many different forces and moments during flight. Accurate modelling of the problem at hand requires a thorough understanding of all of these effects. In addition to the physical properties and dynamics, numerous frames of reference exist to describe vehicle motion, each with their own benefits and drawbacks. Derivations of 6DOF flight vehicle equations of motion are given in many textbooks [25]. For completeness, this is summarized in this section.

One simplification made throughout model development is the assumption that the vehicle's propellers can instantaneously change their rotational speeds. While this is certainly not true, small rotorcraft are capable of quickly changing each rotor's rotational speed. The non-ideal dynamics are later addressed in Section 3.1.2. The vehicle models

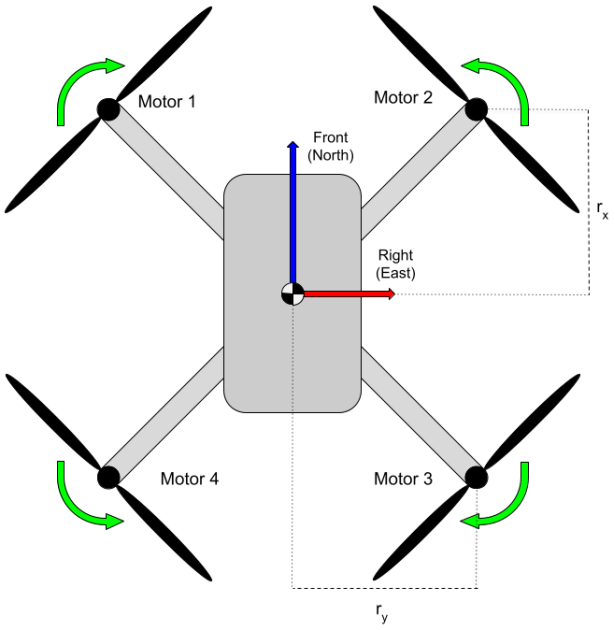

Figure 2.1. Quadcopter motor numbers and locations

presented in this section neglect these effects for simplicity.

Before any analysis or equations can be presented, it is necessary to establish consistent ways of describing the vehicle. Figure 2.1 shows how the motors are labeled 1-4, each with identical distances, r_x and r_y, describing the front-back and left-right distance from the vehicle's center of mass (COM) respectively. Building on the vehicle diagram, we can go on to describe actuation characteristics in different coordinate frames with varying degrees of freedom.

The coordinate frames considered in this analysis are the local North-East-Down (NED) and body forward-right-down (FRD) frames. These frames of reference are typical convention for use on atmospheric aircraft. In both of these frames, full 6-DOF models exist to fully describe the motion. Through various assumptions and constraints, the equations can be reduced to 3-DOF models. Both 6-DOF and 3-DOF models are presented for various uses throughout this thesis.

2.1.1 Coordinate Frames

The local NED frames and body FRD frame are useful frames of reference for describing the motion of an aircraft. As shown in Figure 2.2, NED/FRD frames are left-handed coordinate frames with the local frame having a fixed origin and the body-carried frame

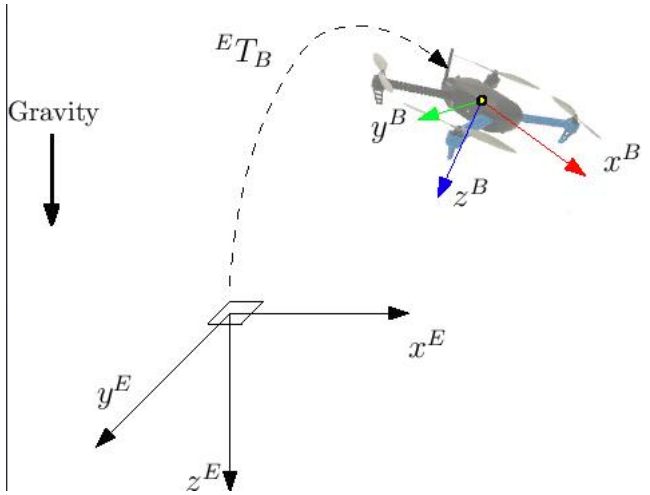

Figure 2.2. Local NED and Body-Carried FRD Frames [26]

remaining aligned with the vehicle COM. Both frames have their place for describing different aspects of the vehicle's position and motion. The local NED frame is useful for describing where the vehicle is relative to some initial origin location, frequently located at the vehicle's takeoff location. In contrast, the body FRD frame is most useful for calculating how different external forces and actuator inputs affect the velocity, orientation, and acceleration of the vehicle.

The conventional way to relate between these frames is through a 3-2-1 Euler angle rotation defined by roll, ϕ; pitch, θ; and yaw, ψ, with the associated rotation matrix given by:

$$\mathbf{R}_{\mathbf{B}}^{\mathbf{I}} = \begin{bmatrix} \cos\theta\cos\phi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{2.1}$$

We can use a combination of both the local and body frames to express vehicle dynamics in a meaningful way.

2.1.2 6-DOF Dynamics

In the real world, the vehicle exists with six degrees of freedom - x, y, and z translational directions and their corresponding axis rotations. These translational and rotational degrees of freedom are adequate to fully describe vehicle motion. To build up a working set of equations, first consider the body forces and moments imparted on the vehicle in the body frame. The rotors are fixed to point in the body-z direction, and the drag force

acts in any direction based on airspeed. Therefore, the body forces are given by

$$\mathbf{f}^B = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \tag{2.2}$$

with drag calculated using

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = -\frac{1}{2}\rho C_D |\mathbf{v}| \mathbf{v} \tag{2.3}$$

where T_i is the thrust force from the ith motor, $\mathbf{d}$ is the body drag, and $\mathbf{v}$ is the body-frame velocity vector.

For this analysis, the body moments will only be the direct result of motor actuation. The body x and y moments are imparted by the thrust force of each rotor, and the yawing moment is caused by the aerodynamically induced torque differences of the rotors. This relation is calculated as:

$$\mathbf{M}^B = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} r_y & -r_y & -r_y & r_y \\ r_x & r_x & -r_x & -r_x \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} + k_Q \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{2.4}$$

where ω_i is the angular rate of the ith rotor. Using the forces and moments, we can then calculate acceleration and angular acceleration in the body frame. The body acceleration calculation has contributions from body forces, gravitational acceleration, and cross-terms due to rotation:

$$\mathbf{a}^B = \begin{bmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} + g \begin{bmatrix} -\sin\theta \\ \sin\phi\cos\theta \\ \cos\phi\cos\theta \end{bmatrix} + \begin{bmatrix} RV - QW \\ PW - RU \\ QU - PV \end{bmatrix} \tag{2.5}$$

Similarly, the body angular accelerations have contributions from the current rotational states as well as the applied moments. We can clean up the calculation a little bit by

assuming symmetry about the $x-z$ body plane to arrive at:

$$\begin{bmatrix}\dot{P}\\ \dot{Q}\\ \dot{R}\end{bmatrix} = \begin{matrix}\frac{1}{I_{xx}I_{yy}-I_{xz}^2}\left(I_{xz}(I_{xx}-I_{yy}+I_{zz})PQ-(I_{zz}^2-I_{zz}I_{yy}+I_{xz}^2)QR+I_{zz}L+I_{xz}N\right)\\ \frac{1}{I_{yy}}\left((I_{zz}-I_x x)RP-I_{xz}(P^2-R^2)+M\right)\\ \frac{1}{I_{xx}I_{zz}-I_{xz}^2}\left((I_{xx}^2-I_{xx}I_{yy}+I_{xz}^2)PQ-I_{xz}(I_{xx}-I_{yy}+I_{zz})QR+I_{xz}L+I_{xx}N\right)\end{matrix} \tag{2.6}$$

Lastly, we can relate the body-frame dynamics to the local NED frame dynamics through the Euler Angle Kinematics equation

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\theta & -\sin\phi\\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta}\end{bmatrix}\begin{bmatrix}P\\ Q\\ R\end{bmatrix} \tag{2.7}$$

and a transformation from body-frame velocities to local-frame velocities:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \mathbf{R}_B^I\begin{bmatrix}U\\ V\\ W\end{bmatrix} \tag{2.8}$$

With all of these equations, we can calculate the future state of a quadrotor given the current state information and environmental properties as well as any prescribed commands to the vehicle's rotors. From the 6-DOF equations of motion, we see that quadrotor dynamics involves a highly coupled system with many components, and a reduced-order model may prove useful.

2.1.3 3-DOF Dynamics

Instead of the more-complicated equations presented in Section 2.1.2, consider a simplified system where the vehicle is only able to move in the North and Down directions and rotate around its y-axis (pitch). In Figure 2.3, the quadrotor is depicted in 2 dimensions. Motors 1 and 2 are combined to be the "front motors," and motors 3 and 4 combined to be the "rear motors." Making this dimensionality reduction can also be thought of as making assumptions to simplify the 6-DOF EOMs already presented. By forcing the vehicle into the x-z plane, we are effectively enforcing constraints on the Y, ϕ, and ψ

degrees of freedom such that

$$\begin{aligned} &Y = y = 0 \quad \dot{Y} = \dot{y} = 0 \quad \ddot{Y} = \ddot{y} = 0 \\ &\phi = \psi = 0 \quad \dot{\phi} = P = \dot{\psi} = R = 0 \quad \ddot{\phi} = \dot{P} = \ddot{\psi} = \dot{R} = 0 \end{aligned} \tag{2.9}$$

Under these assumptions, Equations (2.2)-(2.8) can be reduced to:

$$\begin{bmatrix} f_x \\ f_z \end{bmatrix} = \begin{bmatrix} -d_x \\ T_f + T_r - d_z \end{bmatrix} \tag{2.10}$$

$$\mathbf{M}^B = m = (T_f - T_r) r_x \tag{2.11}$$

$$\begin{bmatrix} \dot{U} \\ \dot{W} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} f_x \\ f_z \end{bmatrix} + g \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} + \begin{bmatrix} -QW \\ QU \end{bmatrix} \tag{2.12}$$

$$\dot{\theta} = Q \tag{2.13}$$

$$\dot{Q} = \frac{m}{I_{yy}} \tag{2.14}$$

$$\begin{bmatrix} \dot{X} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} U \\ W \end{bmatrix} \tag{2.15}$$

The above assumptions are made to plan a trajectory only in the $x - z$ plane. Removing movement in the other directions removes most complicated couplings of motion. However, we would then be operating under the assumption that roll, yaw, and y-translational motion can be controlled by the vehicle's FCU with minimal impact on the capabilities to follow the planar trajectory. This assumption appears to be valid, with justification provided in Section 4.3.

2.2 A Simplified Model for Vortex Ring State

Since one of the primary motivating factors behind this work is to avoid VRS, a thorough understanding of how it can occur is necessary. Developing an understanding of this requires a brief dive into rotor aerodynamics as many of the factors influencing VRS are related to the airflow velocities around the rotors. The first quantity needed is the induced velocity of the rotor in a stable hover. This is given by:

$$v_h = \sqrt{\frac{T_{\text{hover}}}{2\rho A}} \tag{2.16}$$

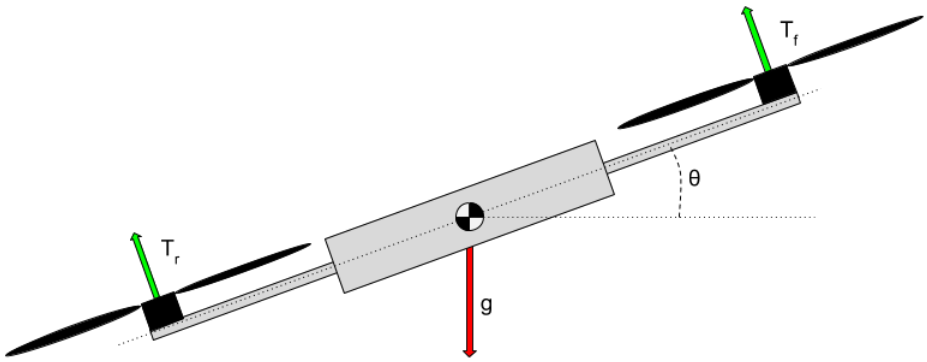

Figure 2.3. Reduced-Order Quadrotor Diagram

where T_{hover} is the thrust produced by the rotor at hover, ρ is the air density, and A is the disk area of the rotor. This value provides a reference velocity for comparing the vertical descent velocity and rotor inflow [10].

Via momentum theory, the hover induced velocity can be used to calculate the induced velocity of a moving rotor in various flight orientations as

$$v = \frac{v_h^2}{\sqrt{U^2 + (v - W)^2}} \tag{2.17}$$

where U and W are the body-x and body-z velocities respectively [27]. This tells us that sufficiently large values of U can be used to divert the dangerous turbulent wake away from the rotor.

However, a typical way of characterizing VRS occurrence is the ratio of W and v_h. The rotor instability characteristics that are the hallmarks of VRS occur for varying ratios of these values, with the onset of instability occurring anywhere in the range $0.5 < \frac{W}{v_h} < 1.5$ [10]. These boundaries can be observed in Figure 2.4 along the vertical axis.

One last consideration involving VRS is the thrust loss associated with operating near VRS conditions. Betzina explored thrust loss in non-axial flows by measuring the change in thrust for various values of $\frac{V_z}{v_h}$ and $\frac{V_x}{v_h}$. These datapoints were used to create the thrust contour shown in Figure 2.5. These contours allow us to create the

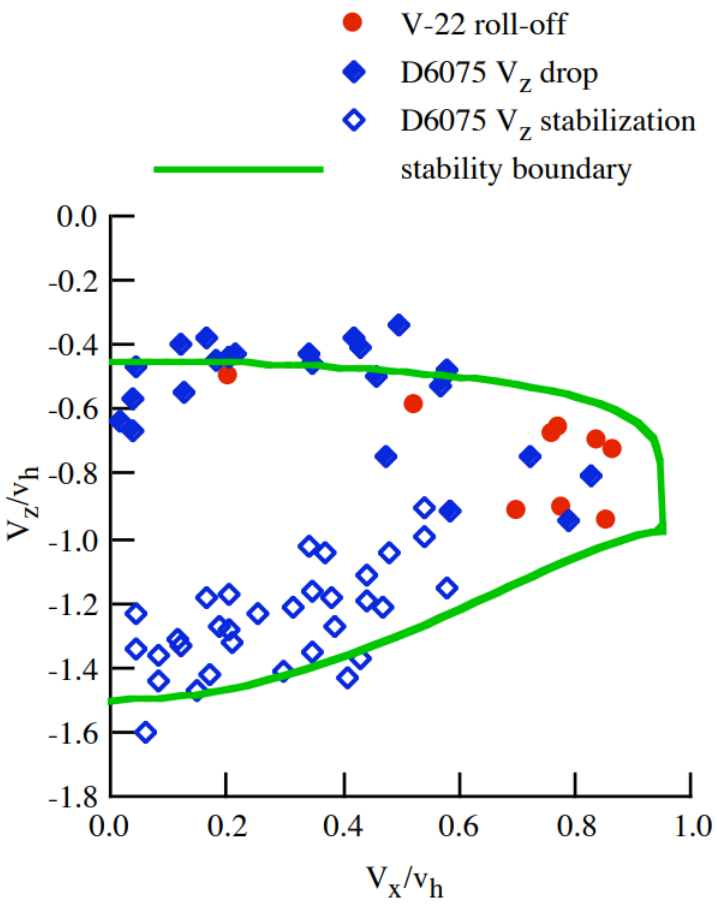

Figure 2.4. VRS Boundaries with Test Data [10] [28] [29] [30] [31]

functional approximation for simulating thrust loss during rapid descent maneuvers given in Equation (2.18), where "Clamp" is the clamping function that clips its argument to be between 0 and 1.

$$T_{\text{actual}} = T_{\text{ideal}}\text{Clamp}\left(\begin{cases} 1 - 0.3\frac{W}{v_h} + 0.3\frac{U}{1.6v_h} & \text{if } 0 < W < v_h \\ 0.4 + 0.3\frac{W}{v_h} + 0.3\frac{U}{1.6v_h} & \text{if } W > v_h \end{cases}\right) \tag{2.18}$$

This simple linear model approximates the contours on from Betzina's data well, and it is further supported by Shetty and Selig, who researched the thrust changes affecting small-scale rotors operating in VRS [33]. Their paper shows the same similar linear decrease in thrust down to a minimum of 70% followed by a subsequent linear increase, though they did not examine the changes in thrust due to non-axial flow effects. Because of the agreement between the numerous datapoints, the thrust model presented in Equation (2.18) is the one implemented to provide real-time thrust calculations for use in this thesis.

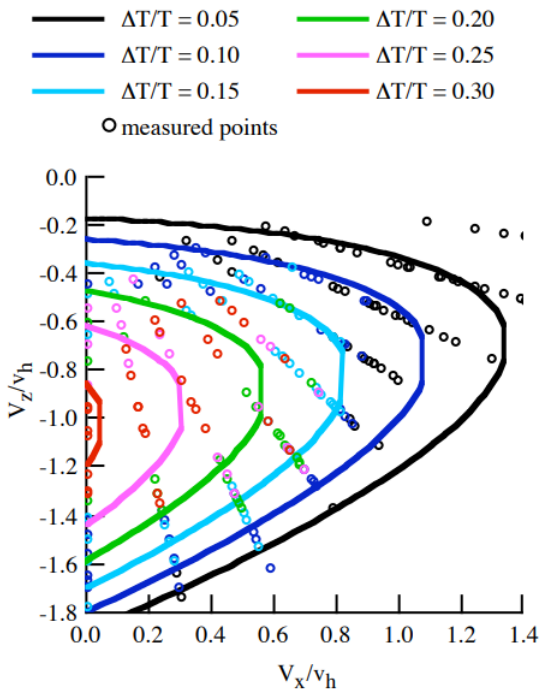

Figure 2.5. VRS Thrust Losses with Test Data [10] [32]

2.3 Vehicle Design

The quadrotor used for analysis of the problem was the PSU Air Vehicle Intelligence and Autonomy (AVIA) Lab-built Hoverfly UAV. The design process and properties of the vehicle are covered in this section to establish a thorough understanding of how and why the vehicle functions as it does. Adequately understanding vehicle properties and behavior is crucial for developing simulation models.

The Hoverfly was designed to approximate the flight dynamics of the Dragonfly, a coaxial octocopter to be flown on Saturn's moon Titan. Hoverfly started out with the design goal of being usable for repeated drop-testing to validate controller design decisions. As such, a much smaller-scale vehicle was the obvious candidate due to the ease of getting it to an ample release altitude. It is significantly more practical to lift smaller vehicles to suitable altitude for drop testing. Furthermore, smaller vehicles typically benefit from faster motor response. This characteristic was desirable to lessen the effects of inherent system delay caused by slow-responding motors.

With the aforementioned design goals in mind, Hoverfly started out as a DYS-320 racing quad. The vehicle was then significantly modified to better-suit the mission needs: a Pixhawk 4 was substituted as the primary autopilot board, an Odroid XU-4

Figure 2.6. Hoverfly Final Flight Configuration

was attached to the underside to handle onboard high-level autonomy functions, and 3D-printed components were made to assist with landing and housing of the parachute release mechanism. Hoverfly's final configuration is shown in Figure 2.6. Despite the numerous modifications, the flight characteristics of the base drone were not modified too much as to make achieving the mission unfeasible.

Even though the characteristics of the vehicle were not significantly altered through the modification process, it was still crucial that vehicle parameters be found/calculated for improved modelling accuracy. Additionally, the repeated modification process required numerous updates to the Pixhawk's default controller parameter list in addition to the control gains used in the state machine running onboard the Odroid.

2.3.1 Added Vehicle Components

As mentioned previously, the DYS-320 quad was outfitted with a Pixhawk 4, an Odroid XU-4, a 3D-printed landing gear, and a constructed release mechanism. All of these modifications make it better-suited for carrying out its test operations.

The included flight control system was changed out for a Pixhawk 4 to make use of the vast number of tunable quadrotor parameters in the open-source Px4 flight stack. The software provides considerable customizability for making small changes to things such

as motor mixing and flight mode control. Additionally, Pixhawks provide a large number of built-in sensors for precise attitude determination that would prove useful in various testing environments. This sensor suite includes 2 accelerometer/gyro combinations, a magnetometer, and a barometric pressure sensor [34]. These sensors combine to produce a complete package that is excellent for adequate state estimation without the use of GPS, so it satisfies the mission design objectives.

While the Pixhawk 4 has a whole host of sensors, and even some processing capabilities, the processing power is inadequate for the mission implementation. To remedy this, the Odroid XU-4 was selected as the companion computer to be mounted to the vehicle and connected to the Pixhawk via serial cable running MAVROS. This small, powerful computer was configured to run with a Linux environment to deal with the high-level autonomy aspects of the mission. Its onboard storage is sufficiently large to store all of the necessary trajectory parameters. These can then be used by an onboard state machine, along with FCU data read from the Pixhawk, to generate vehicle commands/waypoints in real time. It additionally allows for additional control layers to be built on top of the Pixhawk's built in attitude control system. By connecting to a ground computer via Wi-Fi, the Odroid is able to host a server for Robot Operating System (ROS), where programs such as the state machine and vehicle controller wrapper/interface can be run.

Lastly, physical modifications to the vehicle were crucial for allowing it to carry out takeoff and release missions. Unsurprisingly, taking off in a grass field is not the general use-case for a racing quad. However, such a location was an ideal testing area to prevent serious vehicle damage in the event of unavoidable system failures. The vehicle's included landing gears were inadequate for testing in such a location, so a new landing gear was fashioned to keep the blades free of obstruction in addition to protecting the Odroid mounted on the vehicle's underbelly. The other physical modification to the vehicle was the drop apparatus. It consists of a 3D-printed housing with a servo that pulls a pin from a shackle when actuated. This simple construction allows a simple radio signal to pull the pin and release the vehicle from whatever dropping implement being used.

2.3.2 Determining Vehicle Parameters

Several vehicle properties needed to be determined to successfully model the vehicle dynamics in both simulation and trajectory planning. These properties include the vehicle mass, moments of inertia, total thrust, and motor time constant.

The first (and easiest) property to be found was the vehicle mass using a simple calibrated scale. It was found that the fully equipped vehicle has a mass of 1050 grams. In

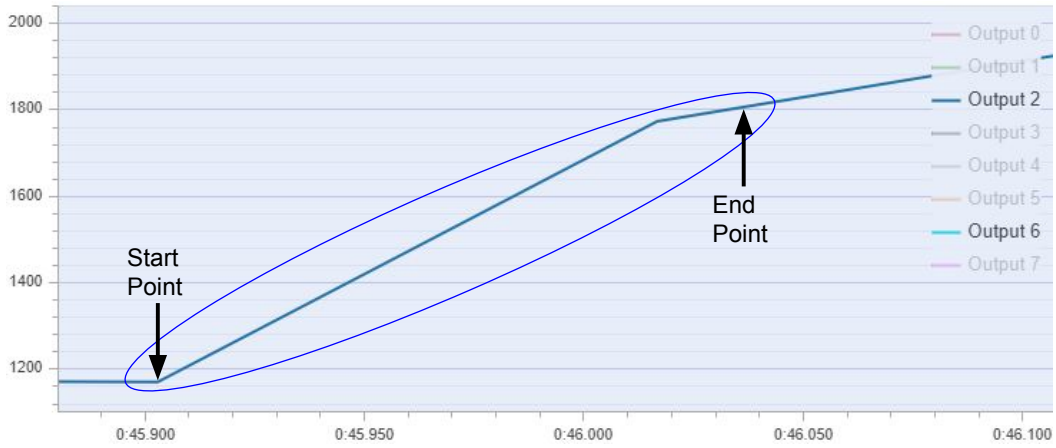

Figure 2.7. PWM Output vs Time Test

addition to weighing the fully laden vehicle, the craft was disassembled so that individual parts could be weighed. Having separate masses for all of the components would allow the moments of inertia to be calculated. To do this, each section of the Hoverfly was then measured, and the center of mass was calculated. With the center of mass found, the parallel axis theorem:

$$I = I_{cm} + md^2 \tag{2.19}$$

was used to sum all of the moments of inertia.

The next properties to be found were those pertaining to motor/rotor performance: the total thrust and time constant of each motor. Rather than connecting the motor to a force sensor, the motor's thrust was tested in flight by commanding full throttle, measuring a_z, and using Newton's second law to calculate the exerted force since the vehicle mass is known. A 1-second snippet of one such test is shown in Figure 2.8. By filtering the raw acceleration data during peak thrust, the vehicle was found to be accelerating at a rate of $-14.4m \cdot s^{-2}$. This results in a calculated total thrust $T_{total} = 15N$.

To find the time constant of the motors, the response time for the speed to increase by 63% needed to be measured. To do this, actuator output logs were examined and averaged by finding the time at the onset of speed increase and the point at which the speed was 63% higher than idle and taking the difference. One such log is depicted in Figure 2.7. From a series of tests, the average time constant for each of the motors was found to be 0.13 sec, which seems to agree well with typical time constants for such a small motor.

The last property to be found for modelling Hoverfly was the reference area for drag

Figure 2.8. Raw Acceleration Data from Maximum Thrust Testing

calculations. For model simplicity, the vehicle was assumed to have the same area in all directions (i.e. approximately spherical). To calculate this reference area, a top, front, and side profile area were all found and averaged, resulting in a value of $S = 0.13\ m^2$. The drag coefficient, C_D, was assumed to be approximately 0.5 to match that of the spherical assumption. For brevity, all of the previously discussed vehicle properties are collected in Table 2.1.

Table 2.1. Hoverfly Vehicle Properties

Property	Value
m	$1.05\ kg$
I_{xx}	$0.028\ kg \cdot m^2$
I_{yy}	$0.045\ kg \cdot m^2$
I_{zz}	$0.053\ kg \cdot m^2$
T_{max}	$15\ N$
τ_{motor}	$0.13s$
S	$0.13\ m^2$
C_D	≈ 0.5

2.3.3 Tuning FCU Parameters

With the vehicle's physical properties determined, the next step in modelling the dynamics is to tune and determine the parameters of the flight control software. The Pixhawk code has a plethora of options allowing for massive customizability. For this use-case however, only a relative handful of them required tuning and inputting into simulation models. These important parameters included those related to the controller gains, integrator limits, and acceleration limits. All of these gains were tuned heuristically

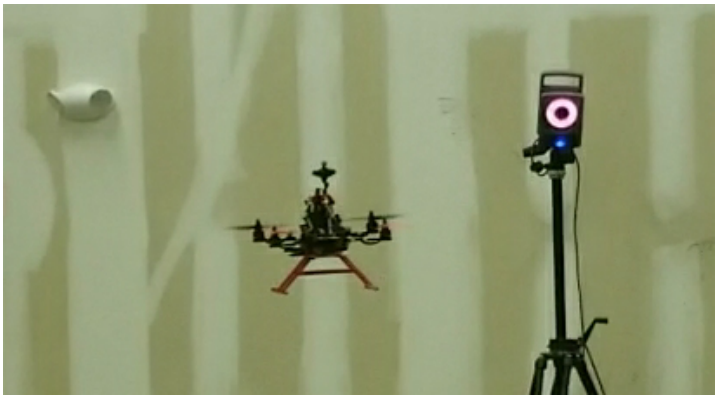

Figure 2.9. Tuning Hoverfly's FCU with Motion Capture

during manually controlled indoor flights in a motion capture studio. The process for tuning each parameter involved taking off and achieving a stable hover using the Pixhawk's built-in position hold functionality. Once in a stable hover, the vehicle was commanded to roll or pitch. The movement was assessed after landing, and the gains were modified until instability occurred and then adjusted, as described in the commonly used Ziegler-Nichols tuning method [35]. After all gains were tuned individually, more complex motions were flown to verify stability. Flying indoors with the motion capture technology allowed for a more-controlled environment where the effects of small modifications to the gains could be observed more easily. These parameters are given in Table 2.2. Once

Table 2.2. Hoverfly FCU Parameters

Parameter	**Value**
$K_{P,roll}$	6
$K_{P,pitch}$	6
$K_{P,rollrate}$	5
$K_{P,pitchrate}$	5
$K_{I,rollrate}$	0.05
$K_{I,pitchrate}$	0.05
Integrator $\text{Limit}_{\text{roll}}$	0.3
Integrator $\text{Limit}_{\text{pitch}}$	0.3

determined, these parameters could be easily saved, read, and exported to the simulation environment.

2.4 Open-Loop Trajectory Control

During the period immediately following vehicle release, an open loop control architecture was selected to propel the vehicle into a flight condition closer to the desired steady-state conditions. By "open-loop" in this case, we are referring to the fact that the controller does not use measurements of the vehicle's velocity to change its commands. Instead, the controller uses feedback only on the pitch, pitch rate, and body-z acceleration of the vehicle. This choice was made for two reasons. First, open-loop control prevents a closed-loop controller from making unexpected changes to the desired vehicle behavior that could upset the recoverability of the vehicle. Additionally, the use of a pre-established pitch profile ensures that the vehicle will mimic the desired pitch-down maneuver. Therefore, the trajectory need only be defined in terms of pitch, pitch rate, and body-z acceleration with respect to time.

The transition to powered flight (TPF) maneuver was modelled after the pitch-down technique taught to helicopter pilots for the recovery from VRS. The FAA has recommended such a maneuver for helicopter pilots for a long time [36]. At their recommendation, a suitable recovery maneuver includes pitching the rotorcraft nose-down to increase forward airspeed, with recovery being complete when "the aircraft passes through effective translational lift and a normal climb is established." This specific maneuver was chosen due to its simplicity and long history of reliable rotorcraft recoveries.

Since the recommended maneuver only involves pitching down and no mention of roll or yaw movement, a planar trajectory (in the x-z plane) can be used to adequately define the vehicle's flight path. This simplification of the trajectory space allows the open-loop trajectory to be found using only the 3-DOF system models. This is similar to the VRS analysis in [18], except this architecture operates in the X-Z plane (with pitch) instead of the Y-Z plane (with roll). The resultant command components consist of a profile on pitch and body-z acceleration only. These 2 profiles, when considered together, result in a pitch-down maneuver similar to that depicted in Figure 1.5, with progressively increasing and then decreasing throttle once upward movement is achieved. Such a trajectory nearly mimics the maneuver used by real-world pilots to pull full-sized aircraft out of vortex ring state, as desired.

2.4.1 Open-Loop Trajectory Generation

As mentioned above, the heuristically-motivated profile consists of commands involving both pitch and body-z acceleration terms. These profiles had to be procedurally generated

using several inputs to allow for optimization to be applied: the initial falling state of the vehicle, $\mathbf{x}_0$; a total trajectory time, t_{total}; a peak pitch angle, θ_{Peak}; final pitch angle, θ_f; peak acceleration, a_{Peak}; final acceleration, a_f; and times of peak input t_{Peak_1} and t_{Peak_2}. These parameters were chosen to define the trajectory because both the peak pitch angle and peak acceleration provide meaningful measurements of the magnitude of the control action, while the time terms can be used to describe the speed with which the vehicle is expected to maneuver.

In order to create the pitch profile (θ_{des}), a quintic spline was selected for function generation. A quintic spline was selected to allow for constraints on the angular jerk ($\theta^{(3)}$) and angular yank ($\theta^{(4)}$) of the pitch angle. Since there is a second-order relationship between rotor thrust and pitch angle, rapidly changing thrust commands could result at the spline point if the higher order derivatives were not considered. By enforcing $\theta^{(3)} = 0$ and $\theta^{(4)} = 0$ at the peak pitch angle, excessively sharp increases in motor commands are prevented, thereby improving the achievability of the trajectory. We are effectively creating a cubic spline of $\dot{Q}$ to guarantee smoothness of the angular acceleration function. This methodology is supported by work to minimize jerk for the purposes of more-accurate trajectory tracking [37]. With function order selected, a quintic spline can be calculated using a series of 3 points and their associated constraints:

$$t_0 = 0, \qquad t_1 = t_{p1}, \qquad t_2 = t_{total} \tag{2.20}$$

$$\begin{bmatrix} \theta \\ Q \\ \dot{Q} \end{bmatrix}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} \theta \\ Q \\ \dot{Q} \end{bmatrix}_1 = \begin{bmatrix} \theta_{\text{Peak}} \\ 0 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} \theta \\ Q \\ \dot{Q} \end{bmatrix}_2 = \begin{bmatrix} \theta_f \\ 0 \\ 0 \end{bmatrix} \tag{2.21}$$

An example pitch profile generated using the above-described spline procedure is shown in Figure 2.10. As desired, we see a smooth pitch-down command to achieve the desired forward airspeed. This is then followed by a return to near-zero pitch to achieve a non-zero final forward flight condition. At a glance, this profile generation method appears to have the desired smoothness and shape for accurate tracking.

With a pitch profile created, the next problem is creating a smooth profile for the vehicle's body-z acceleration ($\dot{W}_{\text{des}}$). Since acceleration and motor thrust are of the same order, there is no need to go beyond a cubic spline to ensure smoothness of the thrust commands. Whereas the pitch profile used two segments, the $\dot{V}_{\text{des}}$ profile consists of 3 sections: a cubic spline from a_0 to a_p, a constant max-acceleration command between t_{p1} and t_{p2}, and then a cubic spline from a_p to a_f. The shape of this acceleration function was

designed to smoothly ramp the acceleration up to some peak value where the acceleration would then be commanded to stay at that maximum magnitude to slow the vehicle's vertical velocity. After most of the slowdown had occurred, the vehicle's acceleration then smoothly climbs back to 0 vertical acceleration so that the vehicle ends the maneuver in steady-state flight. The same peak time, t_{p1}, was used for both the maximum pitch angle and the onset of maximum acceleration since we want to wait to expend maximum energy until the vehicle is pointed furthest from the vertical velocity vector. This serves to maximize the forward velocity gained from the maximum thrust output, again mimicking the behavior of trained pilots.

$$t_0 = 0, \qquad t_1 = t_{p1}, \qquad t_2 = t_{p2}, \qquad t_3 = t_{total} \tag{2.22}$$

$$a_0 = a_0, \qquad a_1 = a_2 = a_p, \qquad a_3 = a_f \tag{2.23}$$

An example pitch profile generated using the above-described spline procedure is shown in Figure 2.11. This acceleration profile corresponds to the same trajectory that was used to generate Figure 2.10. The acceleration steadily increases until peak pitch is achieved at 2 seconds, remains constant at the maximum value until 2.5 seconds, and then smoothly increases and levels off at the end of the maneuver. Relating this to the FAA-recommended maneuver, the vehicle's acceleration increases since the vehicle would be nose-down, and acceleration is maintained until the vehicle begins to climb. At which point, the maneuver is complete.

2.4.2 Vehicle Commands from Open-Loop Waypoints

With pitch and acceleration waypoints created, the vehicle model uses proportional-integral (PI) control on θ_{des} and Q_{des} in addition to $\dot{V}_{des}$ commands to generate actual thrust commands. The pitch control architecture is shown in block diagram form in Figure 2.12. Written out, Equations (2.24) and (2.25) are first used to calculate an appropriate pitch rate command. Then Equations (2.26) and (2.27) are used to calculate a desired pitch rate.

$$\Delta\theta = \theta_{des} - \theta_{measured} \tag{2.24}$$

$$Q_{cmd} = Q_{des} + K_{\theta,P}\Delta\theta + K_{\theta,I}\int_{t_0}^{t} \Delta\theta dt \tag{2.25}$$

$$\Delta Q = Q_{cmd} - Q_{measured} \tag{2.26}$$

$$\dot{Q}_{cmd} = K_{Q,P}\Delta Q + K_{Q,I}\int_{t_0}^{t} \Delta Q dt \tag{2.27}$$

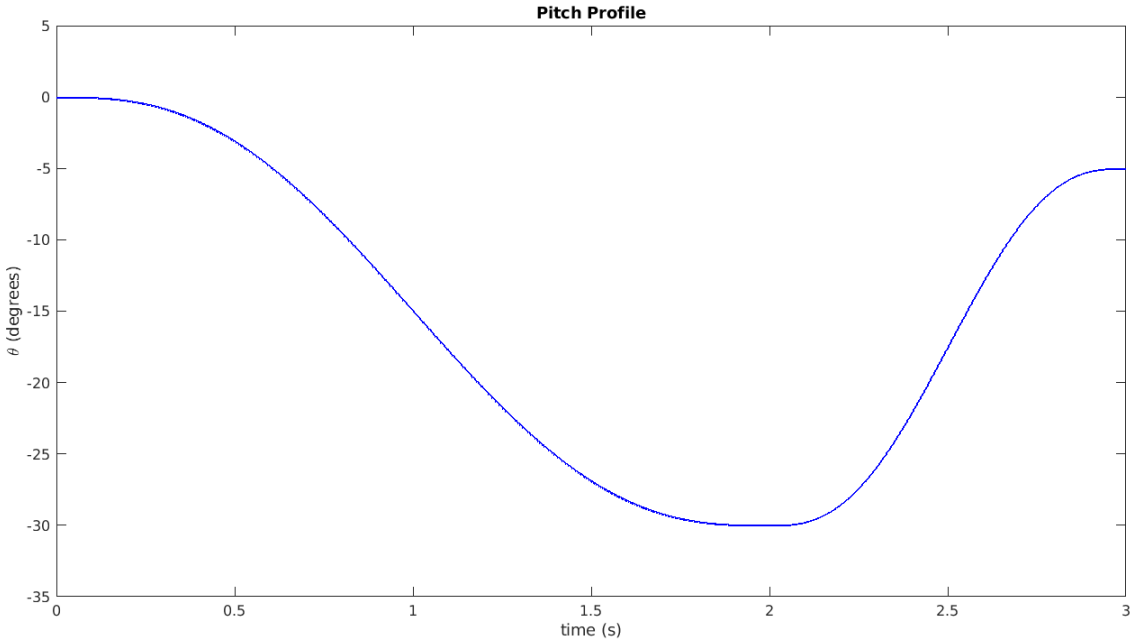

Figure 2.10. Pitch profile w/ $t_{total} = 3$ sec, $t_{p1} = 2$ sec, $\theta_0 = 0°$, $\theta_{\text{Peak}} = -30°$, and $\theta_f = -5°$

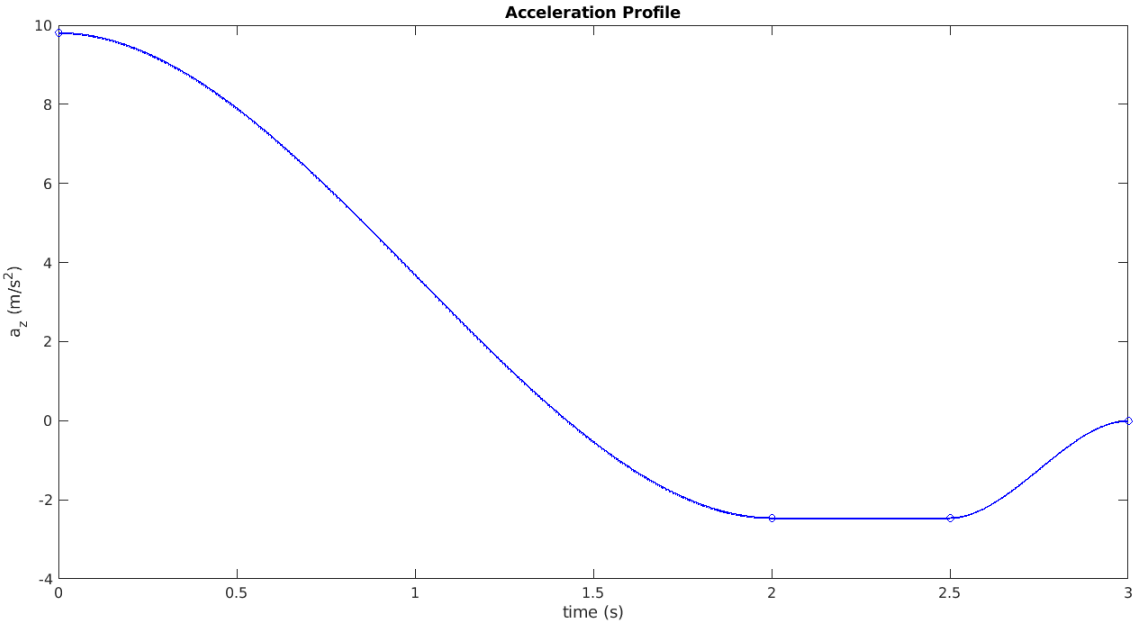

Figure 2.11. Accel profile w/ $t_{total} = 3$ sec, $t_{p1} = 2$ sec, $t_{p2} = 2.5$ sec, $a_0 = g$, $a_p = -0.25g$, and $a_f = 0g$

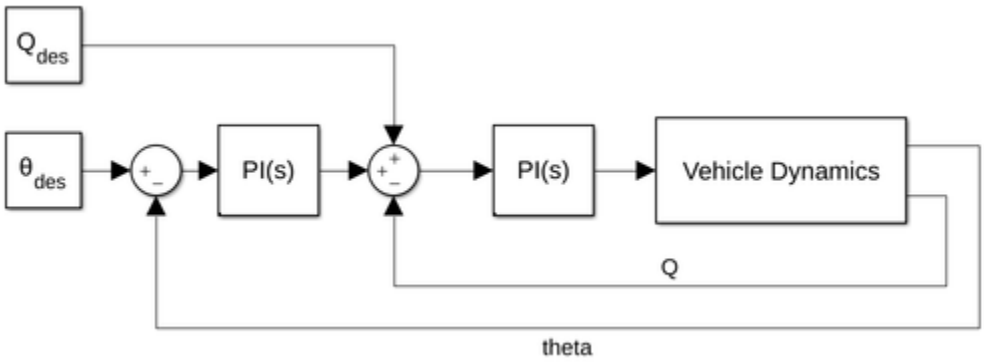

Figure 2.12. Pitch Control Block Diagram

This control system architecture was chosen to emulate the flight control code on the Pixhawk autopilot so that simulated trajectories could be ported over to the real Hoverfly for flight testing.

2.5 Closed-Loop Trajectory Control

Once Hoverfly finishes executing its open-loop pitching trajectory, it will be in a much more stable flight condition than it was immediately after release. Following the pre-planned profile, control is immediately switched over to a velocity-profile-following controller. Understanding how this portion of the recovery maneuver functions is important for better-understanding how the vehicle states at the end of the open-loop maneuver affect the overall controllability of the system as a whole.

Similar to the open-loop trajectory portion, the closed-loop trajectory is also calculated using the 3-DOF system models. This velocity-based profile is calculated to provide the vehicle a series of velocity and acceleration waypoints that it can follow to achieve the desired steady-state flight condition. This allows roughly identical open-loop pitch-down maneuvers to be used for various final flight conditions. Using a closed loop on velocity allows Hoverfly to deal with perturbations from wind and potential modelling error, thus assuring the convergence to the goal flight path. An additional noteworthy constraint on the pathing and tracking problem is that raw position data cannot be used in order to most-accurately emulate a GPS-less environment such as Titan, where the full-scale Dragonfly vehicle is designed to fly.

2.5.1 Closed-Loop Trajectory Generation

The velocity profile generation makes use of the real-time vehicle states at the onset of the path in addition to the pre-planned final flight states. At this stage, the trajectory is described in the local NED frame rather than the body FRD frame to facilitate safer flight testing when it comes time to perform outdoor validation. In order to generate a smooth velocity profile in both the North and Down directions, the initial and final velocities and accelerations were the only input variables needed. Instead of selecting a profile time, the endpoint time, t_f is calculated using vehicle acceleration limits:

$$t_f = max \left\{ \frac{\dot{X}_f - \dot{X}_0}{limit_{\ddot{X}}}, \frac{\dot{Z}_f - \dot{Z}_0}{limit_{\ddot{Z}}} \right\} \tag{2.28}$$

By dynamically calculating the profile time, we can prevent the planner from generating an unrealizable trajectory due to the unchangeable vehicle dynamical properties. A fourth-order spline (cubic with respect to acceleration) is then generated using the initial and final FRD velocities and accelerations with $t_0 = 0$ and t_f as calculated from Equation (2.28).

2.5.2 Vehicle Commands from Velocity Waypoints

The controller used to generate vehicle commands during the velocity-controlled stage of the mission is a PI controller of velocity with a feedforward acceleration term. Similar to the rationale for the open-loop control selection, this architecture mimics the FCU architecture in the Pixhawk software's mission flight mode and is depicted in Figure 2.13 and written out in Equations (2.29) and (2.30).

$$\Delta \mathbf{v} = \mathbf{v}_{des} - \mathbf{v}_{measured} \tag{2.29}$$

$$\mathbf{a}_{cmd} = \mathbf{a}_{des} + K_{\mathbf{v},P} \Delta \mathbf{v} + K_{\mathbf{v},I} \int_{t_0}^{t} \Delta \mathbf{v} dt \tag{2.30}$$

2.6 Summary

At the upper-most level, Chapter 2 outlined the underlying vehicle dynamics of the problem. Then, the aerodynamic issues were described. Vehicle design was covered for

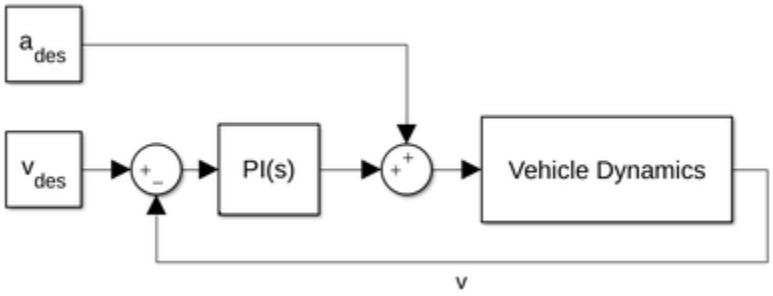

Figure 2.13. Velocity Control Block Diagram

the purposes of design justification and gathering a thorough list of vehicle properties. Lastly, trajectory generation methods were presented along with the way the vehicle would interpret commands from the desired trajectories. More detailed descriptions are as follows:

First, the dynamical equations to describe rotorcraft motion were presented in Section 2.1. This included an overview of the local-NED and body-FRD coordinate frames used throughout this thesis in addition to the 6-DOF equations of motion. Following the explanation of the 6-DOF equations of motion, 3-DOF equations of motion were presented for simplified analysis in preparation for simpler trajectory generation.

Next, VRS dynamics were presented in Section 2.2. This included an overview of aerodynamic effects and relationships that may lead to the onset of VRS, with included equations to relate the body-z velocity to induced hover velocity for the purpose of predicting VRS. The boundaries and properties presented in this section are useful for identifying dangerous regions of the flight envelope.

The process of designing the Hoverfly was then presented in Section 2.3. Included was an overview of the motivating factors behind numerous design decisions with respect to both software and hardware. The initial vehicle selection and the added components are described along with a calculation of their physical properties for use in dynamical models. Then, the onboard controller parameters within the Pixhawk FCU were tuned, with the tuning process and results given. The design considerations presented here strongly affect the parameters used by the system models to find an optimal open-loop trajectory.

Sections 2.4 and 2.5 finally outlined the way in which the open-loop and closed-loop portions of the release maneuver were designed respectively. Beginning with the way in which the open-loop commands were generated from input parameters, the way in which the desired behavior would be transformed into usable vehicle commands was

described. Then, the resultant closed-loop path to be followed was described. This closed-loop portion of the maneuver is developed to allow the vehicle to reliably achieve many different final flight conditions.

Chapter 3
Solving for a Trajectory

In this chapter, the methods used to create a suitable system model in MATLAB are discussed. The ways in which vehicle dynamics are simplified and then simulated are discussed which is then followed by a description of the numerical integration method to track the vehicle states. Then, particle swarm optimization is discussed, and its application to this problem is described. The outputs from the particle swarm is presented with a discussion of the solution's robustness to changes in the vehicle initial conditions.

3.1 Modelling in MATLAB

The initial simulation and trajectory planning were done in MATLAB for rapid testing and development. To simplify computational complexity, the 3DOF model of the vehicle was created in the software. To capture all of the vehicle dynamics, functions were needed to calculate control inputs, calculate applied force dynamics, and numerically integrate the vehicle states. The MATLAB simulation was run with a timestep of $\Delta t = 0.02s$ and 4th-Order Runge-Kutta integration methods.

3.1.1 Controller Simulation

There is little difference between the 2 controllers outlined previously. Both the pitch profile and velocity waypoints were calculated using the methods outlined in Section 2.4 and Section 2.5 respectively at the appropriate times during the simulated time-history. The PI control architectures were applied in discrete time to calculate the desired acceleration and pitch rate at each timestep which was then fed into the force calculations.

3.1.2 Simulated Force Dynamics

With the MATLAB simulation using the 3DOF model, the vehicle thrust dynamics were modeled with the front and rear motor combinations as depicted in Figure 2.3. Using $\dot{Q}_{cmd}$ and $a_{z,\text{cmd}}$ as inputs, the simulated motor mixer solves the linear system

$$\begin{bmatrix} a_{z,\text{cmd}} \\ \dot{Q}_{cmd} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ r_x & -r_x \end{bmatrix} \begin{bmatrix} T_{f,\text{cmd}} \\ T_{r,\text{cmd}} \end{bmatrix} \tag{3.1}$$

and then deals with potential saturation in desired thrust. Motor saturation is dealt with by clipping the maximum thrust value to the maximum allowable and then scaling the lesser value by the same percentage. This 2-stage saturation management prevents both the front and rear rotors from saturating at the same time, which would cause the vehicle to lose pitch control.

The desaturated thrust commands are then applied to the motor model with the same time constants as found in Table 2.1. The relationship between the commanded thrust and the thrust response (in the Laplace domain) is given by

$$\frac{T}{T_{\text{cmd}}} = \frac{1}{\tau s + 1} \tag{3.2}$$

The thrust output for both the front and rear motor is then used in the integration of vehicle dynamics.

In addition to the forces imparted by the front and rear motors, the MATLAB simulation also included the drag effects from Earth's atmosphere and gravity. To model this, a reference density was required. This was chosen to be standard sea level conditions (SSL) because Hoverfly was intended to be flown at low altitudes (<400 ft above ground level). This resulted in a constant simulation air density of $\rho = 1.225kg \cdot m^{-3}$. Also due to the low, roughly constant altitude, Earth's gravity was fixed to be $g = 9.81m \cdot s^{-2}$.

3.1.3 Updating States

The vehicle simulation made use of 6 states to keep track of the 3DOF vehicle properties:

$$\mathbf{x} = \begin{bmatrix} X & Z & \theta & \dot{X} & \dot{Z} & \dot{\theta} \end{bmatrix}^T \tag{3.3}$$

The derivatives of these 6 states could be found using the current states and the current time-steps prescribed inputs. These derivatives were then numerically integrated by one

timestep using a 4-th Order Runge-Kutta method. These states were then fed back into the simulated controller where new inputs were calculated, and this process was repeated for the entire duration of the simulation.

3.2 Modelling the Release Maneuver

As per the mission design, Hoverfly is to be released from a parachute and wait a small amount of time before allowing the rotors to spin in order to prevent entanglement with the chute. This amount of time was chosen to be 1 second, so the simulation remains "stuck" in a loop with no actuation for the first second. After this time, the open-loop pitch and acceleration commands are calculated using the state information at the transition time and the trajectory parameters θ_{Peak} and t_{Peak_2}. For the time between $t = 1$ sec and $t = t_{total}$, the simulated vehicle follows the open-loop pitch and acceleration commands generated by the heuristic-driven spline. Next, the simulation calculates a velocity-decreasing trajectory with the goal to achieve the desired final steady-state flight condition using the current state information and $\mathbf{v}_{\text{des}}$. This set of velocity and acceleration waypoints are used by the simulated controller to actuate the vehicle until the end of the simulation. Once the simulation time runs out of new velocity waypoints, the final velocity waypoint is repeatedly used until the end of the simulation, similar to a constant step command.

3.3 Selecting Trajectory Parameters with PSO

Selecting the parameters from which the open-loop profiles are generated requires some form of optimization algorithm. Optimizations algorithms are frequently used to find the best solution to a problem when the equations describing the system are too complicated to be analytically solved practically. Particle swarm optimization (PSO) is one such algorithm, and it has been a popular choice in optimization problems since it was first proposed in 1995 [38], particularly in aerospace applications. It attempts to iteratively improve candidate solutions with respect to some cost function, which is used to measure the quality of a solution.

For application in this problem, the PSO algorithm was adapted from [39]. Each particle was constructed with 2 component dimensions such that for each particle, $\mathbf{p_i}$, is defined as

$$\mathbf{p_i} = \begin{bmatrix} \theta_{\text{Peak}} & t_{\text{Peak}_2} \end{bmatrix}^T \tag{3.4}$$

The other inputs to the trajectory generation procedure were held constant. Since the trajectory was designed to be for recovery from free fall, the initial acceleration was set to be $a_0 = 1g$. For the maximum acceleration during recovery, a value of $a_{\text{Peak}} = -0.45g$ was chosen to provide a 5% margin from maximum thrust to allow for adequate pitch actuation. For the final acceleration, steady-level flight was the goal, so $a_f = 0$. For initial testing, the vehicle was assumed to be falling with no initial pitch angle or pitch rate, $\theta_0 = 0$ and $Q_0 = 0$. The last input parameter for the trajectory was the initial peak pitch angle time, t_{Peak_1}. Since it was desirable for the vehicle to pitch forward as quickly as possible, this time would be governed by both the peak pitch rate and peak pitch acceleration. Both of these values were vehicle-specific parameters.

Due to the properties of Hoverfly, the $\dot{Q}_{\max}$ was the limiter on the peak time. To find the peak time, t_{Peak_1}, $\ddot{Q} = \theta^{(3)}$ was analyzed to find the maximums of $\dot{Q}$. Using the boundary conditions that $\theta_{t_{\text{Peak}_1}} = \theta_{\text{Peak}}$ and $Q_{t_{\text{peak}_1}} = 0$, it was determined that

$$t_{\text{Peak}_1} = \sqrt{\frac{5.773\theta_{\text{Peak}}}{\dot{Q}_{\max}}} \tag{3.5}$$

Therefore, the peak pitch time would be a function of the peak pitch angle. Using this relationship ensures that the vehicle pitches nose-down as quickly as possible (within the vehicle parameter limits) to the peak angle, as desired.

The particles are first randomly distributed around the solution space using uniform distributions, $\text{var} \sim U(\text{var}_{\min}, \text{var}_{\max})$ where "var" represents any of the individual variables making up the particle. Each component is also initialized with a velocity sampled from $U(-|\text{var}_{\max} - \text{var}_{\min}|, |\text{var}_{\max} - \text{var}_{\min}|)$. Since no iterations have elapsed at this point, each particle records its best position as its initial position. Then, the cost function is evaluated for each particle to measure its quality. The particle with the best resultant cost is then recorded as the globally current best.

At each successive iteration, $k+1$, each particle's velocity is updated by first sampling $r_p, r_g \sim U(0,1)$ and then applying Equation (3.6).

$$\mathbf{v_{i,k+1}} = \omega \mathbf{v_{i,k}} + \phi_p r_p(\mathbf{b_{i,k}} - \mathbf{p_{i,k}}) + \phi_g r_g(\mathbf{g_k} - \mathbf{p_{i,k}}) \tag{3.6}$$

where ω, ϕ_p, and ϕ_g are appropriately selected weights affecting the convergence of the algorithm while $\mathbf{b_{i,k}}$ and $\mathbf{g_k}$ are the particle's all-time best and global best positions at iteration k respectively. Each particle's velocity is added to its position in the update

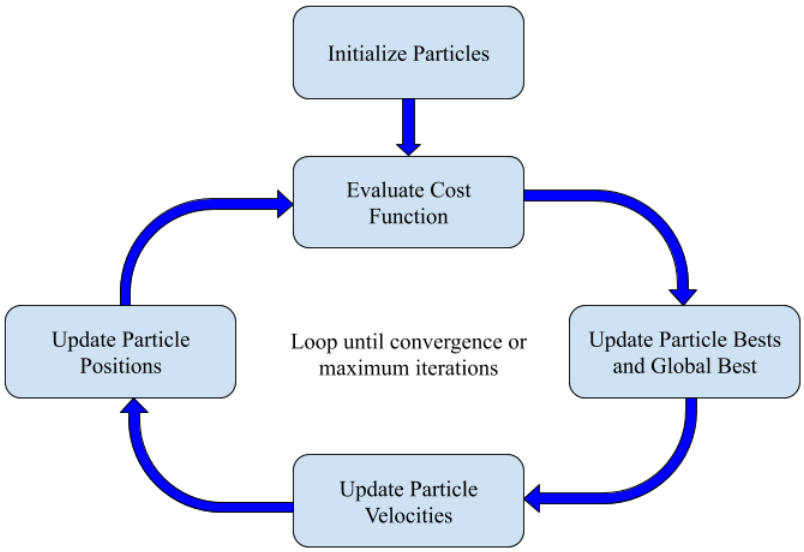

Figure 3.1. PSO Flowchart

step:

$$\mathbf{x_{i,k+1}} = \mathbf{x_{i,k}} + \mathbf{v_{i,k+1}} \tag{3.7}$$

The cost function is again evaluated, and the best positions are updated. This procedure is repeated until either convergence of the cost function or a maximum number of iterations have been completed. A flowchart of this process is shown in Figure 3.1. Once the algorithm exits, the best trajectory parameters that were found are then stored for use in the open-loop trajectory generation.

3.3.1 PSO Search Space Limits

In order to achieve adequate convergence behavior, it is imperative to place boundaries on the particle parameters. From heuristic knowledge of vortex-ring-state maneuvers, the pitch angle limits were selected to ignore the low-angle solutions of $[-20°, 0°]$ while also preventing the vehicle from exceeding its safe pitch orientation of $-60°$. From numerous preliminary trials (with few particles), t_{Peak_2} was converging to values between 3.5 and 4 seconds. To provide adequate margin, this region was expanded to include all time values between 2.5 and 5 seconds.

$$\theta_{\text{Peak}} \in [-60°, -20°] \quad t_{\text{Peak}_2} \in [2.5 \text{ s}, 5 \text{ s}] \tag{3.8}$$

3.3.2 Cost Function Selection

Proper selection of a cost function is crucial for good performance of optimization algorithms. Since the cost function is how the algorithm decides which solutions are better than others, it must be ensured that the function weighs different performance parameters appropriately. The parameters that were chosen to comprise the cost function are: the total thrust output of the Hoverfly, the altitude loss during the drop, a penalty for thrust commands during high-VRS-risk conditions, and the integral of the square of the body-z velocity.

Each portion of the cost function was chosen for different reasons. The minimum altitude loss portion was incorporated to seek trajectories that resulted in the least altitude loss during the entire maneuver. This is an important consideration for the trajectory (aside from stability) so that maneuvers are feasible at lower altitudes.

Similarly, the thrust integral was incorporated to minimize the total energy used by the vehicle during its recovery so that the vehicle would choose to minimize energy expenditure during recovery. To augment this thrust integral, a penalty was added which greatly added cost for rotor actuating during high-U maneuvers. From Section 2.2, we know that the VRS stability boundary occurs when $0.5v_h < W < 1.5v_h$, so thrust during this region was penalized differently by applying a conditional function to the thrust:

$$\tau(t) = \begin{cases} 0 & \text{if } 0.5v_h < W < 1.5v_h \\ T(t) & \text{otherwise} \end{cases} \tag{3.9}$$

The last component of the cost function, the integrated square of the body-z velocity, $\int_0^{t_{\text{final}}} W^2(t)dt$, was incorporated to both keep the vehicle away from VRS conditions and penalize deviations away from non-zero vertical velocity. Since the Hoverfly's induced velocity is $\approx 9.77m/s$, higher values of W mean that the Hoverfly is closer to VRS conditions. Also, higher magnitudes of W in the negative direction correspond with wasted energy, so squaring W minimizes both of these undesirable conditions and prefers trajectories which tend to $W = 0$ and stay there.

The total amount of thrust was found by numerically integrating the time-history of the thrust output, and the thrust penalty term was found by integrating $\tau(t)$. The altitude loss was found by subtracting the minimum altitude from the altitude at the start of the maneuver:

$$h_{\text{loss}} = h_{\text{start}} - h_{\text{min}} \tag{3.10}$$

The last element of the cost function to be minimized is the integral of the squared body-z

velocity, W. The cost function was then constructed with variable weight parameters for each of the 4 terms

$$J = \omega_L h_{\text{loss}} + \omega_{\text{T}} \int_0^{t_{\text{final}}} T(t)dt + \omega_\tau \int_0^{t_{\text{final}}} \tau(t)dt + \omega_{\text{W}} \int_0^{t_{\text{final}}} W^2(t)dt \tag{3.11}$$

where ω_L, ω_{T}, ω_τ, and ω_{W} correspond to the weighting of the minimum altitude, the thrust action, the thrust penalty in VRS conditions, and the integral of W^2 respectively. Changing these weights affects how the cost function prioritizes each of the different measurements of the solution quality to produce a single scalar output. A common problem in optimization problems is appropriately choosing the values for all of these weight parameters. The weights were determined through both intuition and trial-and-error.

The most important parameter for vehicle performance is to avoid over-actuation during high-W maneuvers. Such thrust commands would be essentially wasted energy due to the thrust loss, so the highest penalty was placed on thrust commands occurring at that time. Next, the total thrust and W penalization were weighed intermediately. Lastly, the altitude loss was the least-significant performance parameter since stability (and hence VRS avoidance) should be prioritized over slight changes in altitude loss, though we still want the optimizer to attempt to minimize altitude loss, all other things being equal. With qualitative descriptions for how the weights were related, many trials were conducted using various initial conditions and weights to see how different weights affected the spaghetti plots of the North-Down paths and their associated costs. Following these trials, the parameters in Table 3.1 were recorded and used for subsequent analysis and design.

Table 3.1. Cost Function Weights

Weight Parameter	Value
ω_{L}	0.5
ω_{T}	1
ω_τ	4
ω_{W}	1

3.3.3 Optimization Results

The PSO procedure was performed for two final flight conditions to allow for more thorough results in the simulation environment, as outlined in Chapter 4. For all simulation runs, the PSO search parameters were identical. These values are summarized

in Table 3.2. These PSO parameters were not subject to meta-optimization but were instead adjusted for many trials until reliable convergence. These could certainly be further fine-tuned, so optimizing these weights and parameters may be an avenue for future work.

Table 3.2. PSO Parameters

Description	Symbol	Value
Swarm Size	S	200
Inertia Weight	ω	0.3
Self Acceleration	ϕ_p	0.1
Social Acceleration	ϕ_g	0.15
Convergence Threshold	ϵ	10^{-9}
Maximum Iterations	$N_{\max}$	50

With these parameters set, simulations were first run with the final flight condition being $\mathbf{v}_{\text{des}}^{\text{NED}} = \vec{0}$. With this goal condition, the optimized trajectory parameters were found to be $\theta_{\text{Peak}} = -46.9°$ and $t_{\text{Peak}_2} = 3.66$s. Next, simulations were conducted for a final flight condition of $\mathbf{v}_{\text{des}}^{\text{NED}} = \begin{bmatrix} 5.0 & 0 & 0 \end{bmatrix}^T$, corresponding to a steady forward flight condition. In this setup, the trajectory parameters were found to be $\theta_{\text{Peak}} = -47.8°$ and $t_{\text{Peak}_2} = 2.93$s.

Table 3.3. Trajectory Parameter Summary

Parameter	Static Hover	Forward Flight
θ_{Peak}	$-46.9°$	$-47.8°$
t_{Peak_2}	3.66 s	2.93 s

In both of the optimally generated profiles (shown in Figures 3.2 and 3.3), we see the vehicle pitch nose-down quickly with the acceleration moving toward the peak value. As the vehicle re-levels itself, the acceleration stays at a maximum until moving back up to zero at the end of the maneuver. This allowed the open-loop trajectory to remove the vehicle from VRS and pass the control over to the closed-loop controller more quickly to allow for it to handle the constant-velocity final condition. Based on these optimization trials, we can see that the pitch angle and peak time vary little with respect to the final flight condition chosen.

3.4 Robustness of Solutions to Altered Initial Conditions

While the trajectory found by PSO is optimal with respect to the initial vehicle conditions given to it, we cannot be sure that the trajectory is good for perturbed initial conditions.

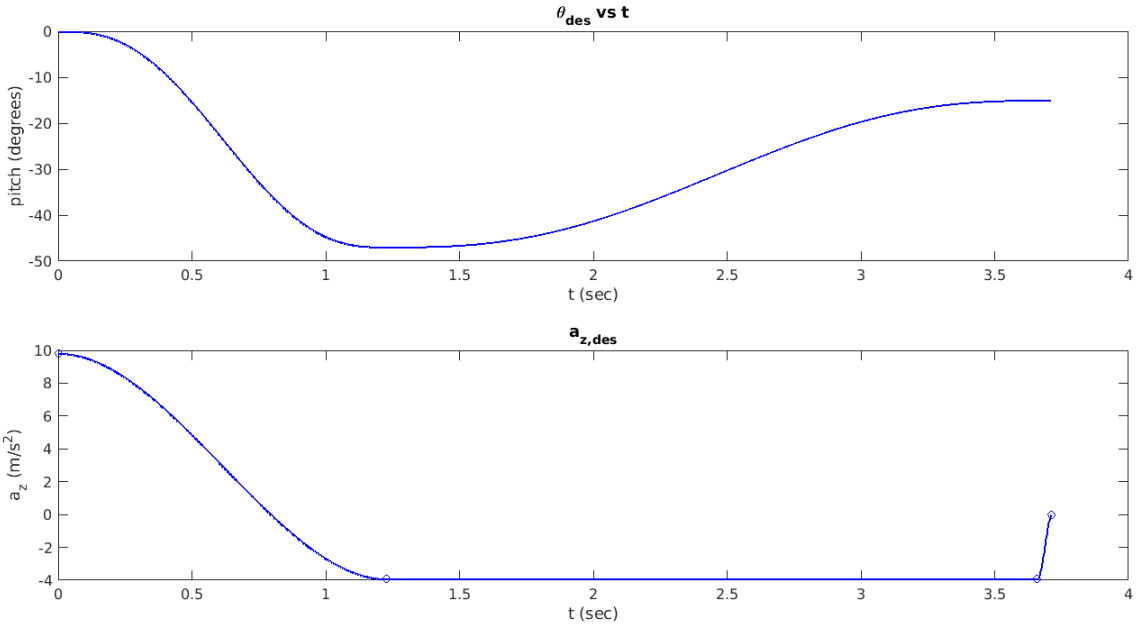

Figure 3.2. Static Goal Optimal Trajectories

In fact, it is almost certain that slight perturbations to θ_0 and Q_0 would somewhat alter the open-loop trajectory that is most optimal. To check the sensitivity of the vehicle's stability and the trajectory parameters to initial conditions, trials were conducted by systematically sampling $\theta_0 \in [-15, 15]^\circ$ and $Q_0 \in [-15, 15]^\circ s^{-1}$.

Each sampled data point was applied to the PSO algorithm, and the optimal trajectory parameters were stored for the static goal condition ($\mathbf{v}_{\text{des}}^{\text{NED}} = \vec{0}$). (Note: the robustness of the forward-flight goal optimization is given in Appendix B.) Each trajectory was then evaluated and plotted to examine for vehicle instabilities. The spaghetti plots of these trajectories were then compared against those obtained without optimizing the trajectory parameters, as seen in Figure 3.4. Additionally, the time histories of down velocities (W) as well as θ and Q were examined via spaghetti plots to check for instabilities, seen in Figures 3.5 through 3.7. Then, contours of each of the trajectory parameters were created and examined to check for irregularities or sharp changes in certain regions of the space as shown in Figures 3.8 and 3.9. Lastly, simulation of initial conditions was carried out where the trajectories were made using only the parameters obtained from the case where $\theta_0 = 0^\circ$ and $Q_0 = 0^\circ s^{-1}$ to see how much different the optimal result was from the naive implementation.

From Figure 3.4, we see that the vehicle does not deviate substantially from the group in any of the trials. Furthermore, the vehicle is able to achieve steady-level flight in every single case within the simulated timeframe (10 seconds in this case). However,

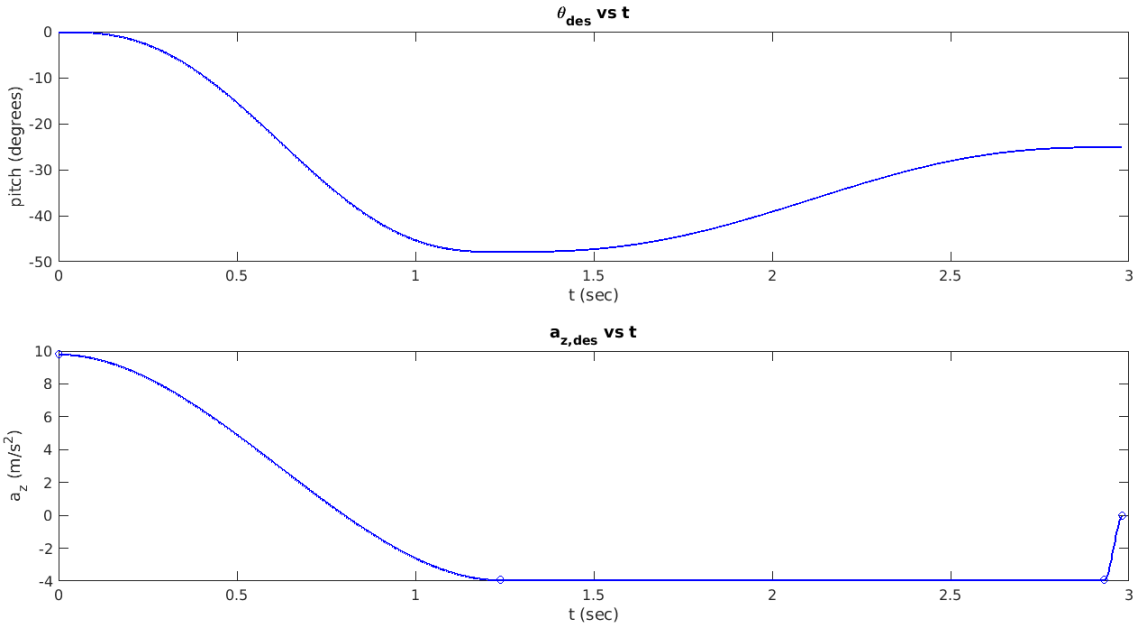

Figure 3.3. Forward Goal Optimal Trajectories

the optimized parameters result in a significantly larger spread of the final altitude of the Hoverfly and significantly less forward (North) movement in most of the trials. The forward travel distance is of little concern since we are attempting only to arrest the vehicle's descent. This suggests that optimizing around each set of initial conditions may not be necessary to produce safe and predictable results.

Figures 3.5 through 3.7 tell a similar story. The optimized parameters generate more varied trajectory shapes given different initial conditions as expected. Qualitatively speaking, however, the plots are extremely similar in both shape and bounds for the time histories of each of the states. It may seem naive to simply use the trajectory determined from zero initial conditions when the vehicle has non-zero initial conditions in practice, but the plots show that this approach still results in stable response. Not accounting for varied initial conditions simply results in the vehicle's states taking a longer time to settle, but this worst-case scenario still satisfies the mission requirements.

From Figures 3.8 and 3.9, we can see that there are some locations in the initial condition space where small perturbations to the initial conditions significantly affects the optimal trajectory parameters; these are depicted as sudden changes in color within a similar mesh region. However, these deviations in the trajectory parameters cause little deviation in the cost function's evaluation. This suggests that numerous combinations of parameters can be used to generate similar trajectories performance-wise, further validating the robustness of the heuristically-driven trajectory design.

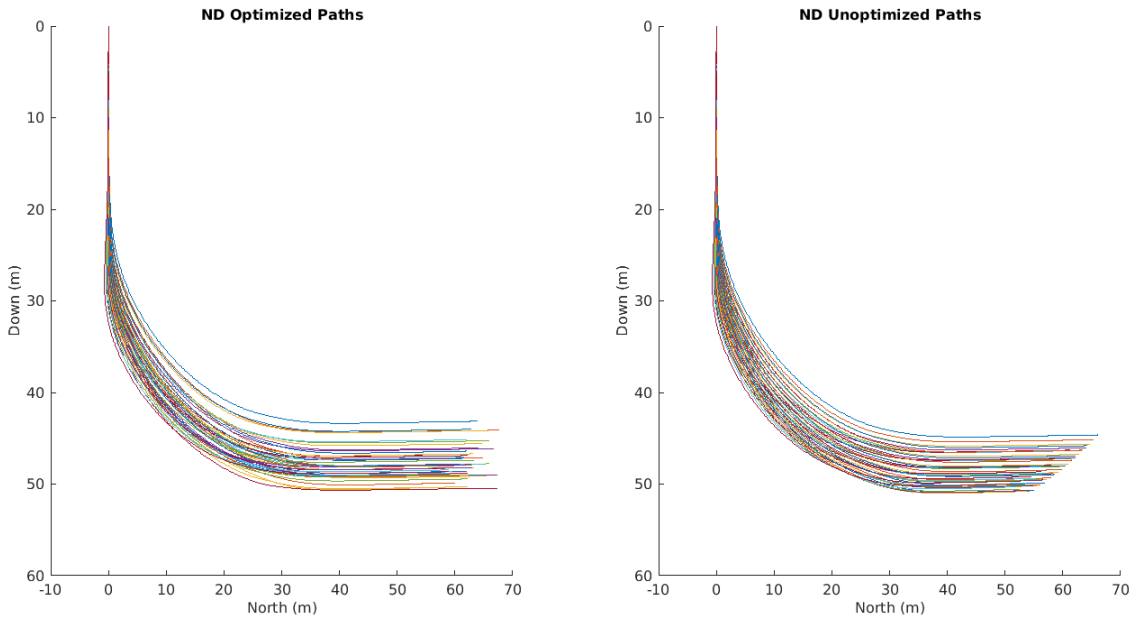

Figure 3.4. ND Path Plots with and without Optimization

One small side note here is that both optimized parameters change significantly more with changes in the initial pitch rate than they do for changes in the initial pitch angle. This suggests that the vehicle's rotational inertia plays a larger role in the cost of a trajectory's execution than the initial attitude of the vehicle. This makes intuitive sense, as the cost function weighs the thrust action most heavily.

By comparing all of the plots describing the motion in both the individually optimized and non-optimized trajectories, we can see that some performance characteristics are improved. However, we also see that the trajectory generated by assuming zero initial conditions still resulted in suitable (while not always optimal) vehicle behavior. Lastly, the smoothness of the cost function (Figure 3.10) to non-zero initial conditions suggests robustness of the solution. These plots appear to be very similar in both shape and magnitude, with initial nose-down orientations and angular velocities corresponding to lower costs. This is expected as starting with the vehicle in a nose-down orientation would reduce the necessary control action to orient the vehicle with an even larger pitch angle. Further analysis also reveals that optimization around each initial condition provides an improvement of at most 2% in vehicle performance, with much smaller improvements typical.

Because the improvements are so marginal, we can confidently say that the heuristic-based trajectory and subsequent control structure are robust to changes in initial conditions. Instead of requiring some large lookup table of parameters to be decided with

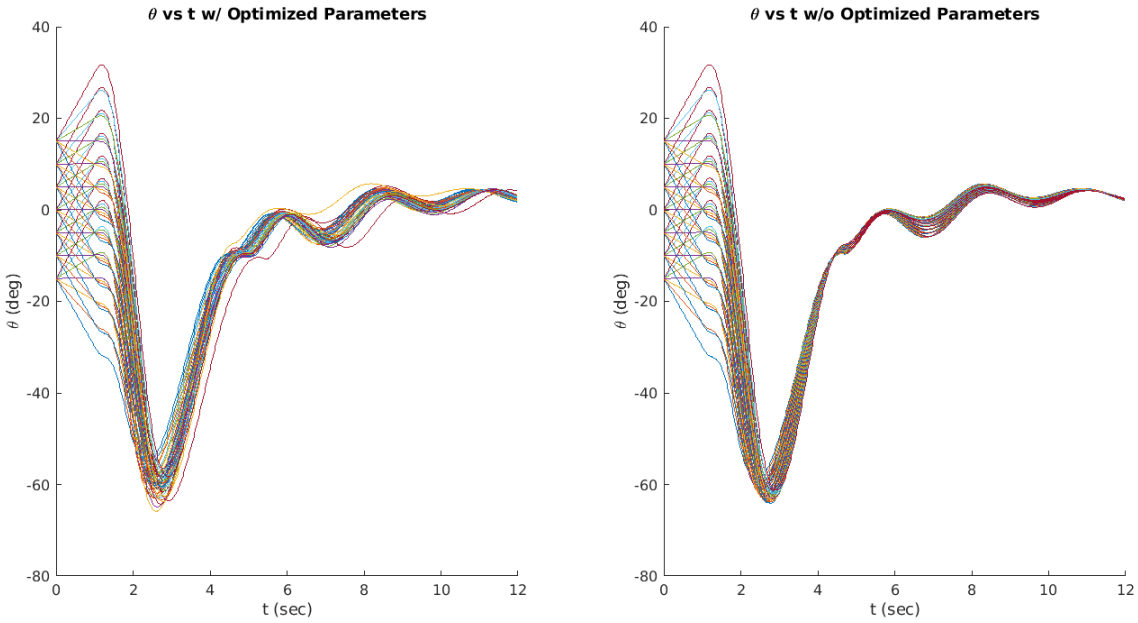

Figure 3.5. Theta Plots with and without Optimization

each initial condition, the vehicle can instead have its trajectory generated from a static set of trajectory parameters and its initial conditions.

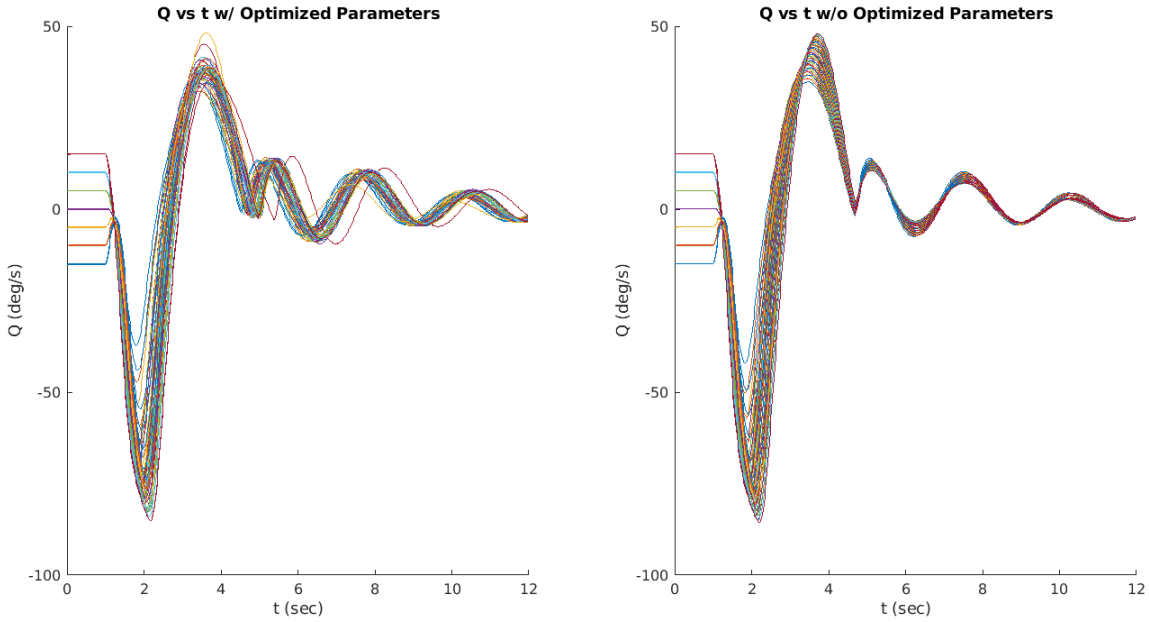

Figure 3.6. Q Plots with and without Optimization

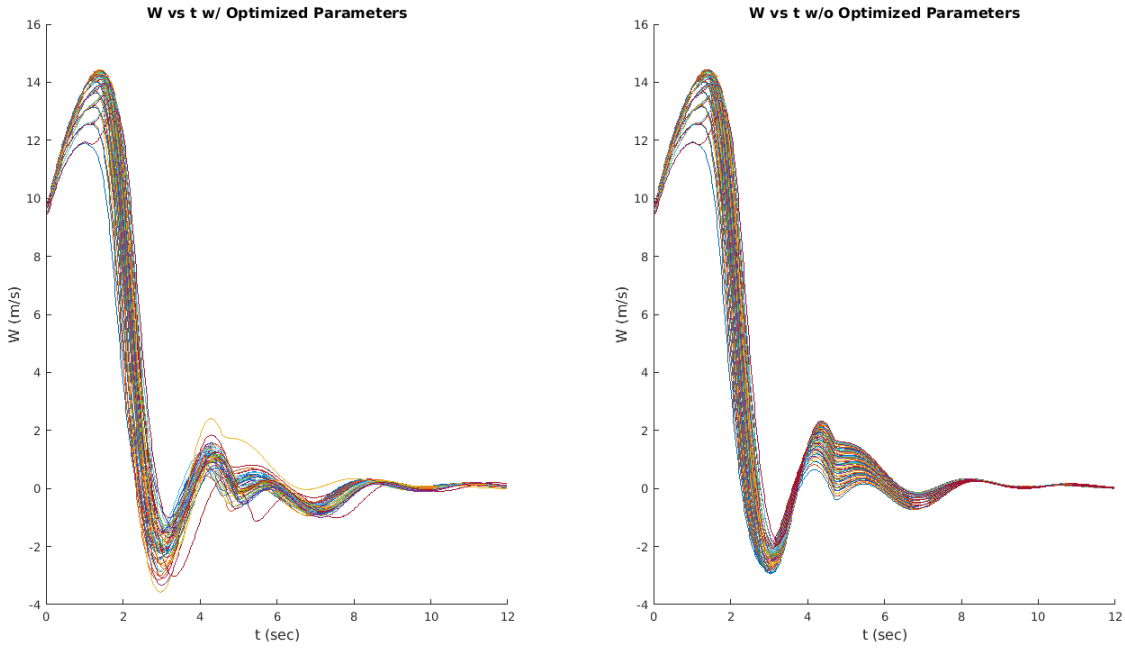

Figure 3.7. Down Velocity Plots with and without Optimization

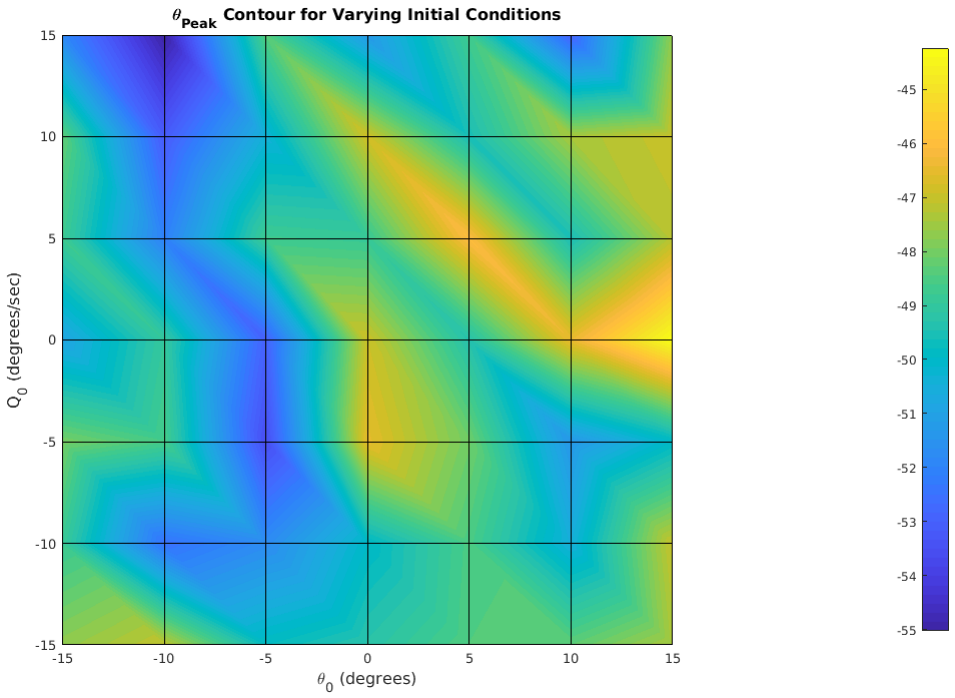

Figure 3.8. Optimal Peak Theta Parameter Contour

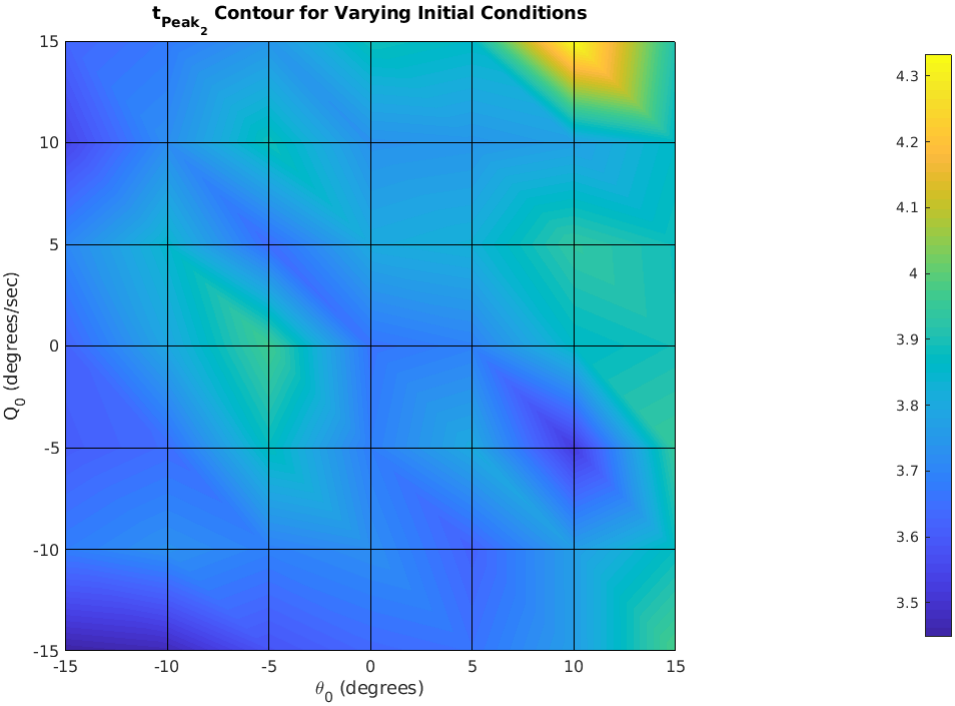

Figure 3.9. Optimal Peak Time Contour

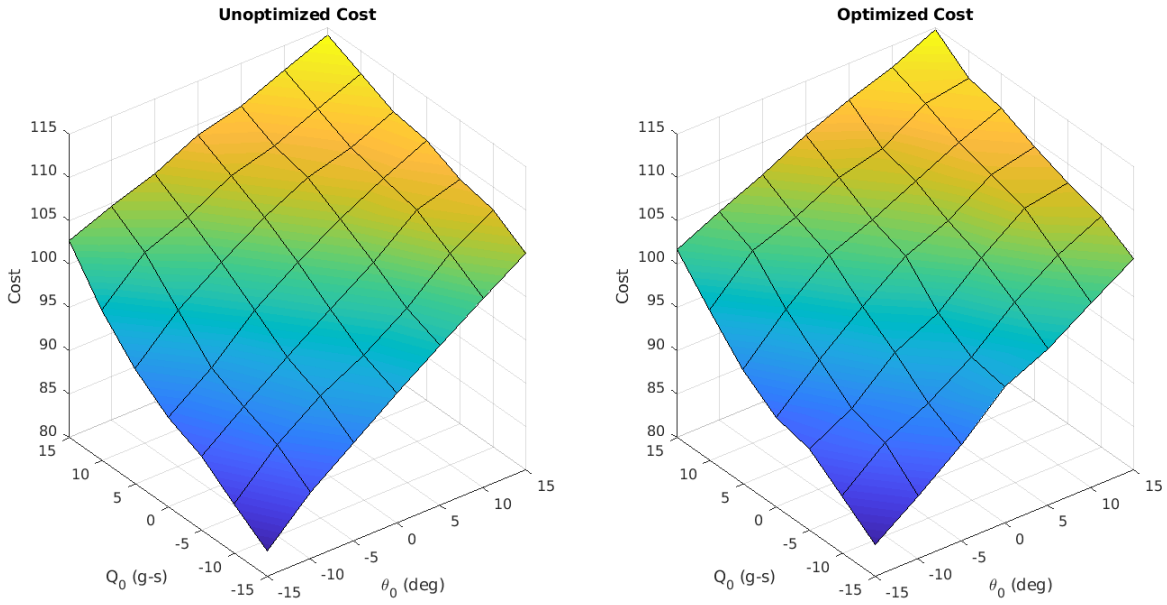

Figure 3.10. Comparison of Cost Functions

Chapter 4
Simulation Implementation and Results

This chapter is focused on the software-in-the-loop implementation of the trajectory described and found in Chapters 2 and 3. This discussion begins with an overview of how the Robot Operating System works and how it is useful for the purpose of testing trajectories in both software-in-the-loop and real-world flight tests. Then, the tweaks that needed to be made to the vehicle controller to allow for onboard control are described. Next, the Gazebo SITL environment is described, and the results from testing within it are presented. Results analysis here includes the accuracy by which the vehicle tracks the planned trajectory and how well the assumption of 3-DOF motion are satisfied.

4.1 Trajectory Implementation

The defined trajectory needed to be implemented in a way that the code could be used in both simulation and real-world flight testing of Hoverfly. This meant that code needed to be as modular as possible. Furthermore, the onboard flight code needed to be able to manage multiple mission scenarios with the ability for the user to easily update trajectory parameters and controller gains as post-implementation tuning would inevitably need to occur. With all of these goals in mind, the flight code was developed in the Robot Operating System (ROS) as a series of nodes that would control various aspects of vehicle functionality: a way to capture user input, a supervising state machine, and the onboard vehicle controller itself. All of these features will be discussed in the following subsections.

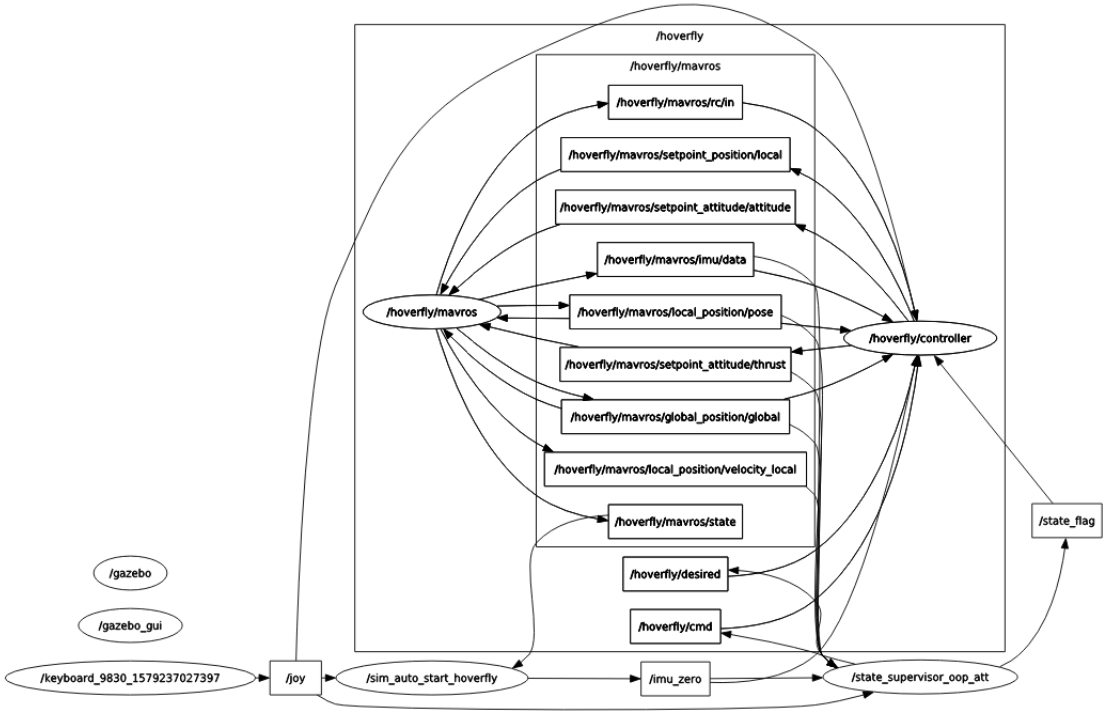

Figure 4.1. ROS Communication Graph During Gazebo Simulation

4.1.1 The Robot Operating System and MAVROS

ROS® composes the backbone of much of the AVIA lab's flight code architecture. It is "a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior" [40]. ROS provides a wrapper for Python and C++ code that allows different *nodes* to communicate with each other via a network of message *publishers* and *subscribers*. This architecture is highly useful for allowing the modular parts of the flight code to communicate with one another to share variable values. This message passing architecture is shown graphically in Figure 4.1. The arrows show how a large number of nodes can pass variables between each other over a series of *topics* to facilitate modularity.

Upon inspection of the network, one can see the large number of topics related to /hoverfly/mavros/... These topics all fall under an important MAVROS node, which serves as a communication driver for sharing ROS topics with MAVLink-based autopilots, such as the Pixhawk. The combination of MAVROS with the AVIA Lab-written nodes allow the Hoverfly to make use of modular flight code components to fly and execute its various tests.

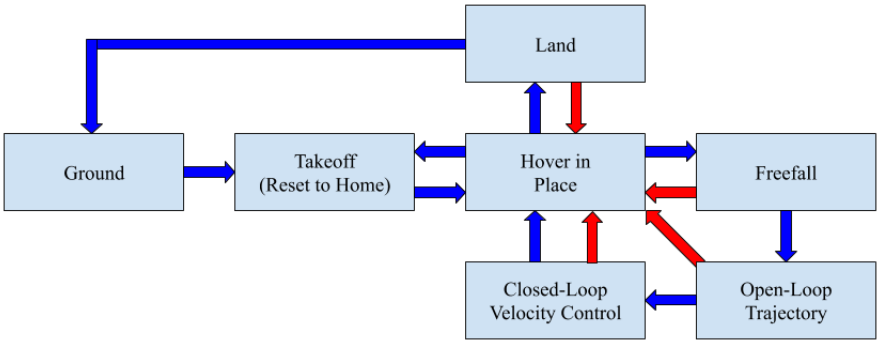

Figure 4.2. State Machine Transitions

4.1.2 Capturing User Input

As shown in the lower left corner of Figure 4.1, a node exists for the purpose of allowing user input into the system. The node makes use of the "Joy" topic to read in keystrokes from the user and convert them to meaningful state-transition messages. This allows for state transitions to be controlled or overridden for safe test operations during pre-testing and drop-testing maneuvers.

Table 4.1. Allowable Keyboard Inputs

Keystroke	Vehicle Command
C	Calibrate Origin
T	Takeoff
H	Force Hover
F	Enter Descent Trajectory (Fall)
L	Land

4.1.3 The State Machine

The state supervisor, or state machine, manages the current operational state of the Hoverfly. This is useful for restricting how the vehicle is allowed to behave at any given time - in effect creating safeguards against unplanned behavior. The state machine structure is shown in Figure 4.2. During normal operation, the vehicle transitions between the states following the blue path arrows. The red paths signify abort conditions where the vehicle is immediately commanded to hover in place until the issue is resolved.

A typical flight of the Hoverfly involves first transitioning from the Ground to the Takeoff state, followed by hovering in place once the prescribed takeoff altitude is reached.

Next the vehicle is commanded via user input into the free-falling state. Once a threshold velocity is detected by the vehicle sensors, the Hoverfly transitions to Open-Loop state. There, the state machine looks up the most appropriate trajectory parameters using real-time information and calculates command waypoints for the open-loop trajectory. These waypoints are sent to the vehicle controller. After the open-loop trajectory time has elapsed, the vehicle is then put into the Closed-Loop state. The state machine once then calculates the best set of velocity waypoints to reach the desired steady-state flight condition and sends these waypoints to the vehicle controller. Once the goal flight condition has been reached, the vehicle transitions to once again hover in its current position. This transition occurs when $|\mathbf{v} - \mathbf{v}_{\text{goal}}| < 0.25$ for 3 consecutive seconds. Once this transition is made, the vehicle hovers in place until the operator tells the Hoverfly to either Land or return to the Takeoff position for another dropping test.

4.1.4 Onboard Vehicle Control

The Vehicle Control Node is responsible for taking the waypoints from the state machine and creating appropriate attitude and thrust commands to be sent to the Pixhawk for motor mixing. With each timestep, the vehicle's current NED/FRD velocity, NED/FRD acceleration, and attitude quaternion are read from the Pixhawk via a MAVROS connection. The control node then selects the appropriate controller based on the current state (as dictated by the state machine). If the vehicle is not following a trajectory, control defaults to a position PID controller that keeps the vehicle stationary for safety.

4.1.4.1 Open-Loop Control

If the vehicle is in the open-loop state, control is calculated using the $\dot{W}_{\text{cmd}}$ and θ_{cmd} waypoints. The attitude command to be sent to the vehicle FCU is then simply defined as:

$$\begin{bmatrix}\phi_{\text{cmd}} & \theta_{\text{cmd}} & \psi_{\text{cmd}}\end{bmatrix}^T = \begin{bmatrix}0 & \theta_{\text{cmd}} & 0\end{bmatrix}^T \tag{4.1}$$

and thrust is found by passing $\dot{W}_{\text{cmd}}$ to a PI controller that calculates throttle percentage. The commanded throttle and attitude vector are sent to the Pixhawk code where the internal motor mixers are allowed to calculate the proper throttle allocation to each of the 4 rotors.

4.1.4.2 Closed-Loop Control

If, instead, the vehicle is in the closed-loop state, the controller's behavior is a little more interesting. In this state, the controller receives NED acceleration and velocity waypoints for use in its PI control with feedforward as described in Equations 2.29 and 2.30. $\mathbf{a}_{\text{cmd}}$ is then used to calculate thrust and attitude commands as given in [41]:

$$T_{\text{total}} = \frac{-m}{\cos\phi\cos\theta}\left(a_{\text{cmd},Z} - g\right) \tag{4.2}$$

$$\begin{bmatrix}\sin\theta_{\text{cmd}} \\ \sin\phi_{\text{cmd}}\end{bmatrix} = \frac{-m}{T_{\text{total}}}\begin{bmatrix}\cos\phi\cos\psi & \sin\psi \\ \cos\phi\sin\psi & -\cos\psi\end{bmatrix}^{-1}\begin{bmatrix}a_{\text{cmd},X} \\ a_{\text{cmd},Y}\end{bmatrix} \tag{4.3}$$

The calculated thrust is then used by the throttle PI controller, and the throttle and attitude vectors are finally sent to the Pixhawk for motor mixing.

4.1.5 Communicating with the Vehicle FCU

The Px4 code handles the mapping from a series of angle and thrust commands to the actual PWM outputs sent to each of the Hoverfly's 4 motors. Because of this, every command needs to be quickly relayed from the Odroid controller to the FCU. Additionally, data needs to be read from the Pixhawk's sensor measurements to be used by the Odroid for trajectory planning and command updating. However, special care must be taken to ensure that both the Odroid's code and the Px4 code use the same conventions for describing motion.

The Px4 flight stack uses the East-North-Up (ENU) coordinate system by convention and accepts attitude commands in the form of quaternions. Since the Odroid operates on an NED/FRD Euler-angle-based control structure, vehicle commands need to be converted so that both systems are speaking the same language so to speak. The conversion is not particularly difficult; NED Euler Angles are first converted to ENU Euler angles, and then those angles are used to calculate the ENU quaternion representing the Hoverfly's orientation:

$$\begin{bmatrix}\phi \\ \theta \\ \psi\end{bmatrix}_{\text{ENU}} = \begin{bmatrix}\phi_{\text{NED}} \\ -\theta_{\text{NED}} \\ -\psi_{\text{NED}}\end{bmatrix} \tag{4.4}$$

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \end{bmatrix} \tag{4.5}$$

Equations 4.4 and 4.5 can also be used in reverse to read Hoverfly's attitude information from the Pixhawk for use onboard the Odroid.

4.2 Validation in Gazebo SITL

Before flight testing the MATLAB-generated trajectory, testing the waypoints and pathing within in a 6-DOF SITL simulation is a much safer intermediate step. Gazebo is ideal for this purpose because of its easy interfacing with the SITL version of both the Pixhawk flight stack and ROS. To set up the environment, vehicle properties and FCU parameters were inputted, and a version of the Odroid code was run on the same computer. The entirety of the flight mission was followed exactly as it would be outdoors with data collected via ROSBags for post-test data analysis and verification. Included in the following subsections is a thorough description of the simulation environment, an explanation of how the MATLAB trajectory was implemented in ROS, and a description of how simulation results will be analyzed.

4.2.1 The Simulation Environment

The simulation environment consists of a combination of Gazebo ROS and Pixhawk's SITL flight stack. The combination of these two software packages allows the Hoverfly to be flown identically to how it would be outside since all of the same interfaces, measurements, and parameters exist within this simulation space.

Gazebo is a popular simulation software with roboticists for testing and visualizing machine behavior. The version used for flight testing of Hoverfly was Gazebo ROS, which has a nice wrapper that can export data from the environment to the ROS variable node space. Using Gazebo allows things such as drag and vehicle mechanical properties to be used in simulations without having to deal with the minutia of modelling them oneself. Instead, Gazebo takes care of the many physical properties of the vehicle and its surrounding environment. It additionally provides the origin and coordinate frames for reference and location measurement. Most importantly of all, the Gazebo GUI allows the pilot to view what is happening to the vehicle model in real time since it uses 3D

vehicle models to show real-time position and orientation.

Figure 4.3 depicts the hoverfly hovering a short distance above the local origin. Clearly, the vehicle model does not have the appearance of Hoverfly. The 3D model resembled a 3DR Iris Quad, but all of the vehicle properties that were measured and described in Section 2.3.2 are shared by the model. This was done by using all of the physically measured performance parameters to calculate the corresponding simulation parameters.

Special care was needed to ensure that the vehicles shared the same thrust and rotor dynamics. Parameters such as hover throttle percentage, maximum thrust, and motor time constants had to be transformed into motor constants, moment constants, and rotor drag coefficients. A summary of the most important parameter calculations is presented in Appendix A. Once this was done, the vehicle behaved very closely in simulation compared to actual flight tests of the Hoverfly.

Beyond vehicle properties, atmospheric conditions were needed to make use of the simulated drag and thrust models. Standard sea level (SSL) atmospheric properties were used in the environment, with values shown in Table 4.2. Windspeed was set to zero since it was not used to develop the trajectories in Chapter 3. However, the potential effects of non-zero wind are explained in Section 5.2.2 for consideration in future work.

Table 4.2. Conditions in the Simulation Environment

Property	**Value**
P	101.3 kPa
ρ	1.225 $kg \cdot m^{-1}$
T	288.2 K

In addition to the Gazebo environment, a simulated FCU is required to treat the simulated vehicle in the same way that the real Hoverfly would be flown. Fortunately, the Px4 flight stack includes a version specifically for running SITL simulations with Gazebo. The Gazebo SITL version of the software can be edited to accept all of the FCU parameters that were tuned on the real vehicle, as was described in Section 2.3.3. All communications with the FCU behave identically to those on the physical vehicle.

4.3 6-DOF Tracking Results

For analysis of within the 6-DOF simulation, several methods of measurement are required to analyze stability, verify assumption validity, and compare the intelligently-designed trajectories to those resulting from simple velocity control lacking any pre-planning. In Section 3.3.2, the components of the cost function were described. These components

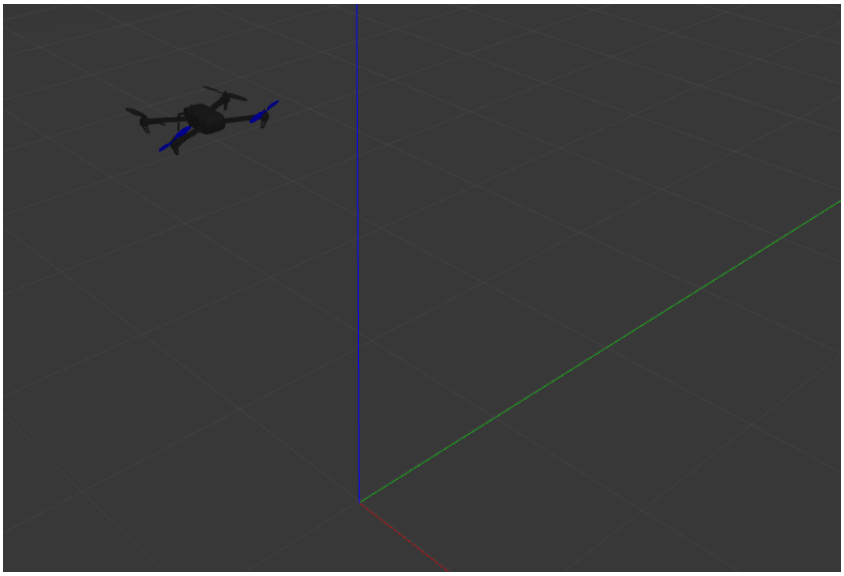

Figure 4.3. The Hoverfly Model Flying in a Gazebo Simulation

included the altitude loss during the maneuver, the total thrust action of the rotors, a penalty on thrust action during VRS conditions, and a cost for non-zero values of W.

Trials were first conducted to show that a falling vehicle at significant velocity can become unrecoverable. By simulating several drops of the Hoverfly model and then waiting the requisite one second until the controller turned on, a closed-loop velocity controller was unable to arrest the descent of the falling vehicle model. In each trial, the rotors were not able to produce sufficient thrust once W became too large. The vehicle quickly saturated its motors before striking the ground in every drop from 100 meters altitude.

To analyze the effectiveness of the proposed heuristic controller, 10 trials were conducted dropping the vehicle with a goal condition of static hover, and 10 trials were conducted with a commanded forward velocity. These trials all resulted in a controlled and stable vehicle every time, suggesting that the open-loop control methodology is effective. The results of these trials are summarized in the following subsections.

4.3.1 Transition to Zero Velocity

The primary testing condition in this analysis was the transition of the vehicle from free-fall to a stable hover in-place with no forward, sideways, or downwards velocity as the goal. During each trial, the vehicle was flown to 100m altitude and then released. The controller first followed the pitch and acceleration commands and then commanded

$\mathbf{v} = 0$ in each trial.

The heuristically-designed controller performed well, with stability and eventual achievement of the goal condition in every flight. A summary of these results is presented in Table 4.3.

Table 4.3. Summary of Static Hover Goal Results

Result	**Value**
Mean Alt. Loss (m)	43.51
Std-dev (m)	2.12
CV (%)	4.9%
Mean Time w/ $W > \frac{v_h}{2}$ (s)	1.58
Std-dev (s)	0.58
CV (%)	36.4%
Mean Maneuver Time (s)	10.33
Std-dev (s)	0.35
CV (%)	3.4%

Based on the data in the table, we see that using the open-loop control results in an average altitude loss of 43.51 meters. Furthermore, the control method quickly reduces W to outside of the at-risk boundary of $\frac{v_h}{2}$ in 1.58 seconds. The total achievement of the goal condition also took an average of 10.33 seconds. Each of these results, especially the altitude loss, are close to those predicted by the MATLAB simulations.

Another important takeaway from Table 4.3 is the low coefficients of variation (CV) attached to the altitude loss and maneuver time. These low variability measures or 4.9% and 3.4% tell us that the recovery maneuver has high reliability and predictability. However, the high deviation in the "at risk" region is something that requires closer examination.

In addition to summary data, the spaghetti plots of the flight variables lend themselves well to further-validating the controller. The first plot worth examining is the spaghetti plot of the flight paths as North-Down profiles in Figure 4.4. The dots along each trajectory represent the point of minimum altitude for each trial. We see that each of the trials results in a stable vehicle recovery with little spread. However, we notice that the vehicle begins descending faster around 15m north in each trial. This corresponds to the transition point between the two controllers.

The plots of W vs t in Figure 4.5 also show how the vehicle reduces its body-z velocity quite quickly by following the pre-planned maneuver. The shaded region on the plot represents the VRS region established in Section 2.2. The trials all show the vehicle quickly exiting the instability region, but a few of the tests show the vehicle

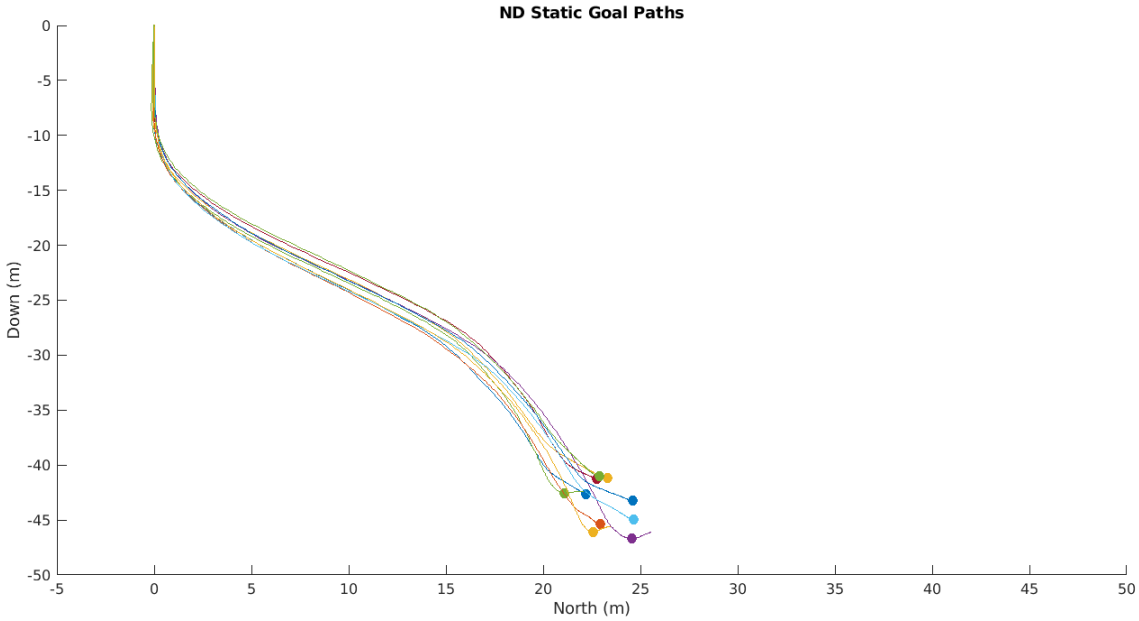

Figure 4.4. North-Down Paths for All Trials with Static Goal Condition

briefly coming back and reentering the region. These trials likely contributed to the high variation in the VRS region times. All trials show the Hoverfly model eventually approaching its final static condition. This shows us that the vehicle successfully reduces W in all cases to ensure stability and recoverability of the system, though the vehicle re-approaches the instability region when the controller begins pitching the nose back up to slow the Hoverfly to a stable hover.

The last plot worth observing for control-method validation is the time-history of the vehicle's pitch angle. In Figure 4.6, we see that the vehicle closely-follows the prescribed pitch profile when controlled by the open-loop controller, albeit with the unavoidable process delay. Afterwards, there is some variation in θ, though all of the trials resulted in the same general trend of a small oscillation followed by the pitch steadily moving towards zero as the velocity is scaled down by the closed-loop velocity controller. This small jerk in theta can be explained by the controller waiting for new waypoints to be calculated as the open-loop controller hands over control to the closed-loop controller. This is an issue that should be further addressed in the future. Furthermore, we see that the initial condition validation of $\theta \in [-15, 15]$ was somewhat excessive (at least as far as simulation application is concerned) because θ did not deviate beyond $\pm 2^\circ$ before the start of the maneuver.

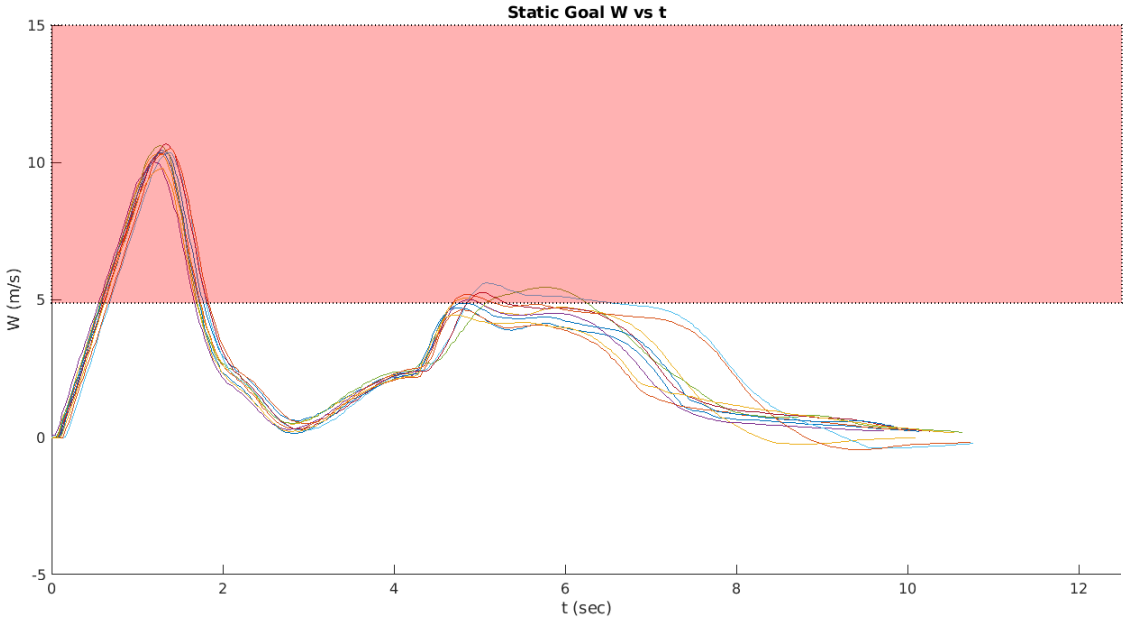

Figure 4.5. W vs t for All Trials with Static Goal Condition

4.3.2 Transition to Forward Flight

The second testing condition in this analysis was the transition of the vehicle from free-fall to a forward flight condition of $v_{\text{des}}^{\text{NED}} = \begin{bmatrix} 5.0 & 0 & 0 \end{bmatrix}^T$. During each trial, the vehicle was once again flown to 100m altitude and then released. The open-loop waypoints were adjusted from the static trial to those obtained from the forward-flight optimization covered in the second half of Section 3.3.3. The results of the 10 simulation trials are summarized in Table 4.4

Table 4.4. Summary of Forward Flight Goal Results

Result	Value
Mean Alt. Loss (m)	44.37
Std-dev (m)	3.88
CV (%)	8.8%
Mean Time w/ $W > \frac{v_h}{2}$ (s)	1.54
Std-dev (s)	0.46
CV (%)	29.6%
Mean Maneuver Time (s)	11.08
Std-dev (s)	1.04
CV (%)	9.4%

Based on the data in the table, we see that using the open-loop control results in an average altitude loss of 44.37 meters, a VRS-risk escape time of 1.54 seconds, and

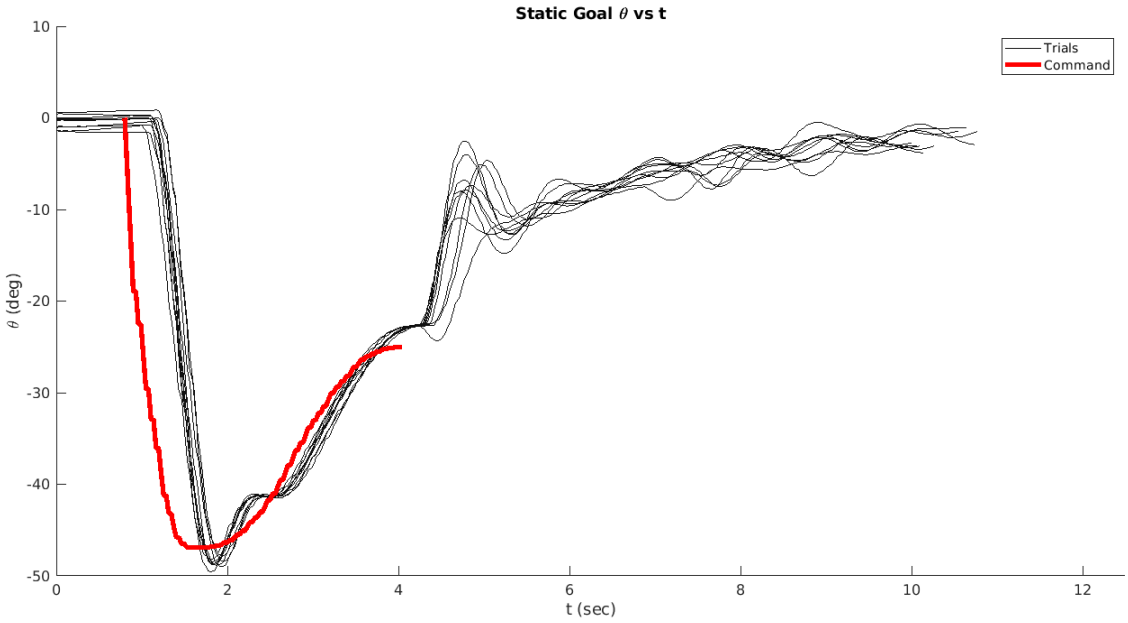

Figure 4.6. θ vs t for all Trials with Static Goal Condition

an average time to goal achievement of 11.08 seconds. While the time spent in the VRS-boundary is about the same as that when attempting to achieve a static hover, the altitude loss and time to goal achievement see increases of 2.0% and 7.3% respectively in the forward flight condition. This suggests that much of the maneuver time is dictated by the transition to the closed-loop controller. Further final flight conditions could be investigated to see if any show significant benefit.

Beyond the summary results, we can again look at the North-Down paths, W vs t plot, and θ vs t plot in Figures 4.7, 4.8, and 4.9 respectively. Obviously, the North-Down trials show significantly more forward movement than those from the static goal condition. The W vs t plots show the trajectories all moving away from the instability region more quickly, as forward flight corresponds to a negative value for W. However, the re-entry into the instability region also occurs sooner. The θ vs t plots also show the same initial oscillation in pitch following the conclusion of the open-loop maneuver portion. However, the pitch seems to remain more constant than in static goal condition.

4.3.3 Validation of 3-DOF Simplification

One last item to address with regards to this analysis is to examine the deviation from the planar trajectory design assumptions. Between the 3-DOF assumptions discussed in Section 2.4 and the subsequent optimization discussed in Chapter 3, it is necessary to

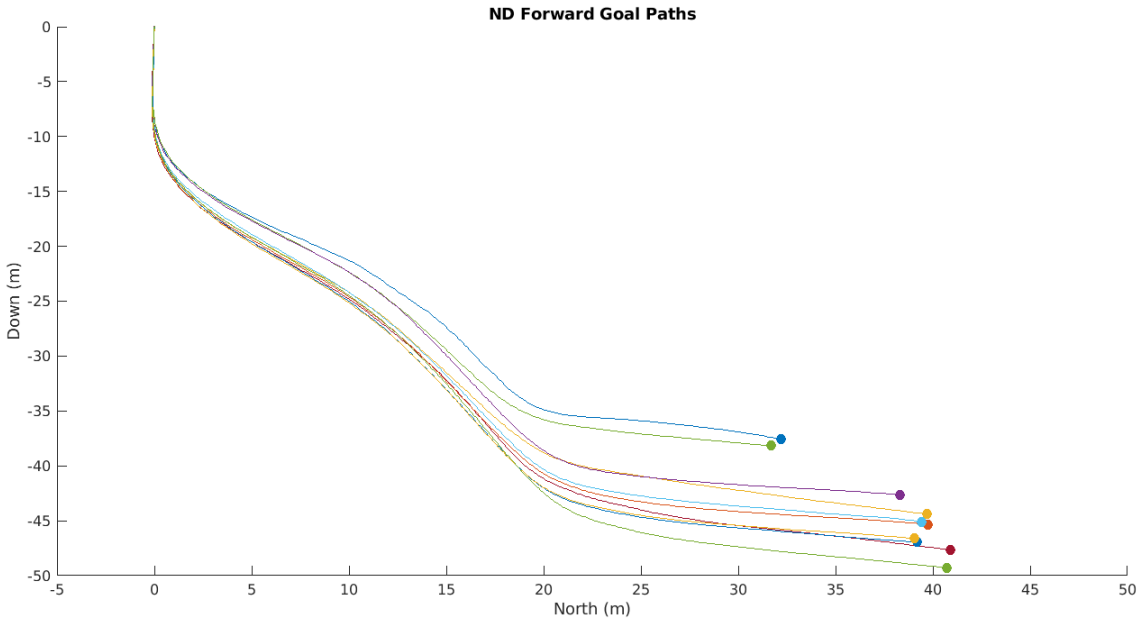

Figure 4.7. North-Down Paths for All Trials with Forward Goal Condition

show that these reductions were reasonable. To do so, we can examine the East-direction movement of the vehicle throughout the course of each trial.

Figure 4.10 shows that the lateral deviations in the vehicle movement were bounded within $E \in (-2.5, 2.5)$. This deviation accounts for less than 10% of the movement in either the North or Down directions. Because of this, it would seem fair to design around the assumption that the vehicle would stay close enough to the X-Z plane for a 3-DOF trajectory to be valid. One note to make here is that the Forward Goal condition plots show oscillations that share shape and amplitude in the Y-direction. This is likely due to coupling of the Euler angles with respect to the body dynamics. However, the small dynamics of the movement relative to the other directions suggest that it is a relatively insignificant effect for short timeframes.

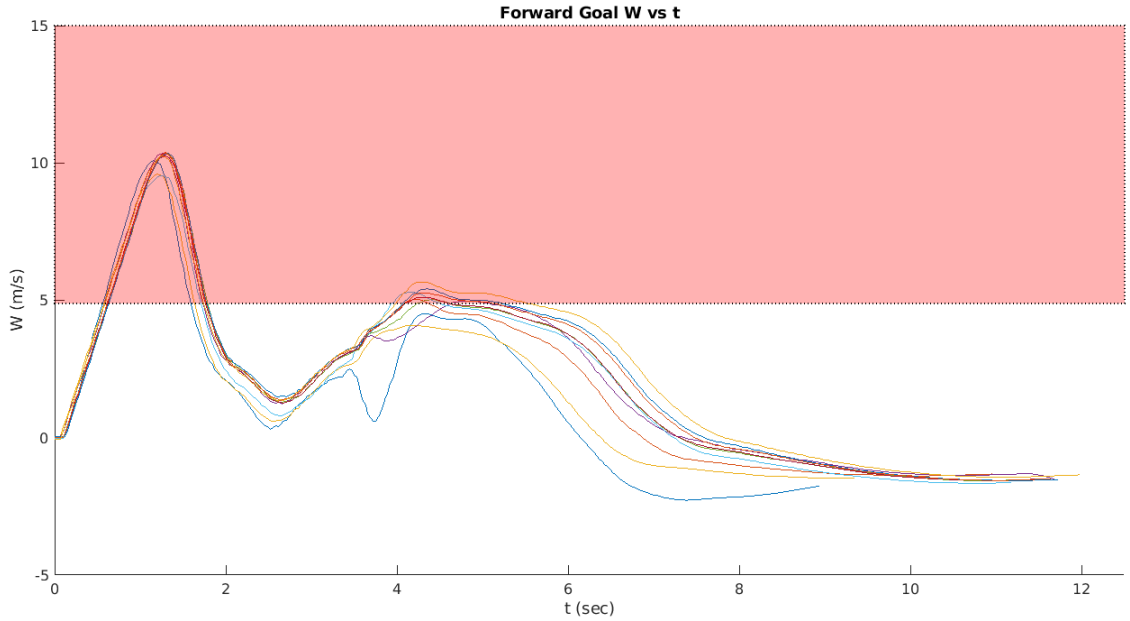

Figure 4.8. W vs t for All Trials with Forward Goal Condition

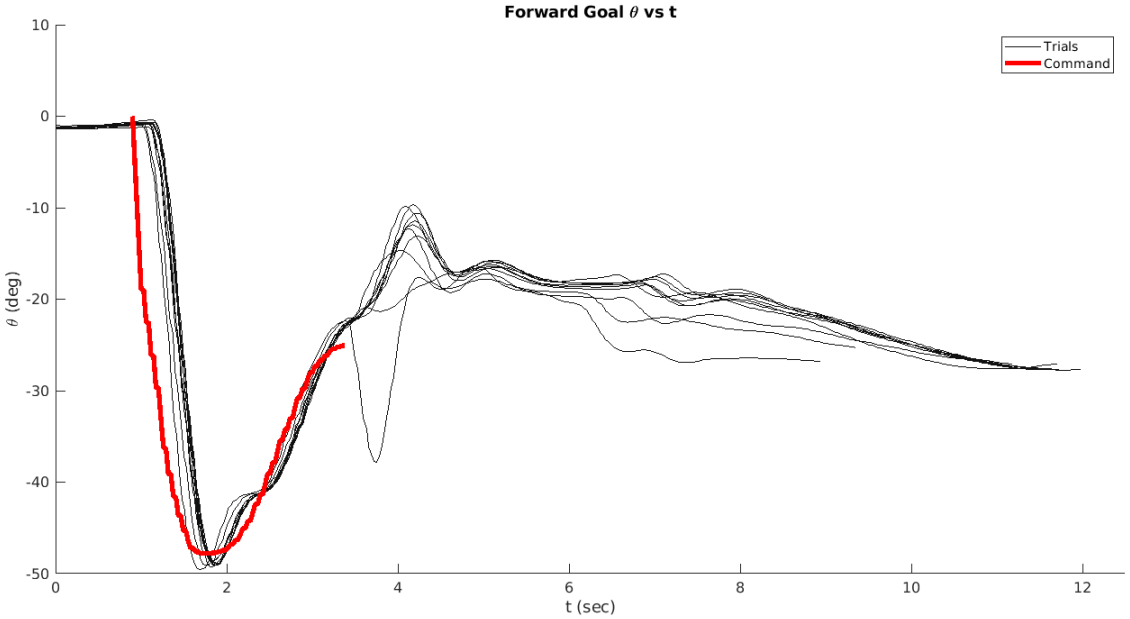

Figure 4.9. θ vs t for all Trials with Forward Goal Condition

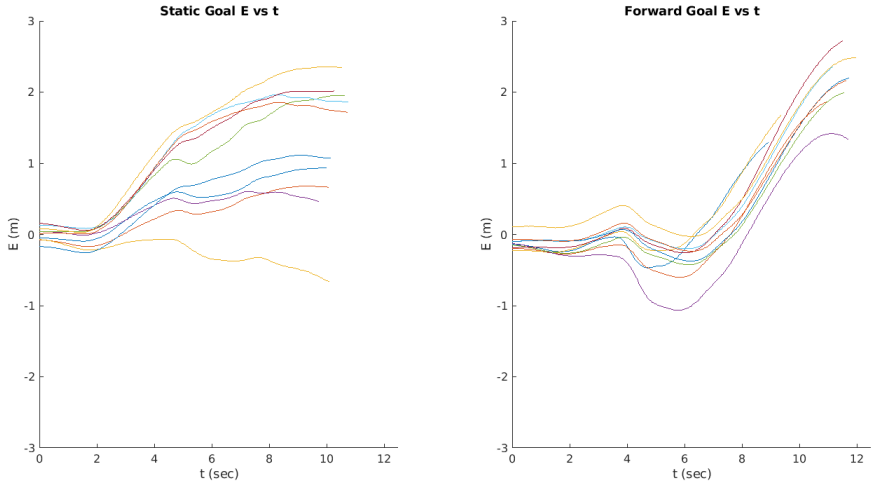

Figure 4.10. E vs t Comparison

Chapter 5
Conclusion

The challenges of controlling a falling vehicle around and out of vortex ring state make the recovery of a parachute-released quadrotor a challenging problem to address. Whereas past research into this problem has sought to optimize a closed-loop controller [17], the work presented in this thesis sought to approach the problem from a different direction: pre-plan an open-loop trajectory via model-based control design and then use that trajectory to transition the vehicle to an easier-to-control regime of flight conditions.

The means to develop adequate system models were presented along with an optimization method for generating optimal trajectories. This was then followed by validating the heuristically-designed trajectories in software-in-the-loop simulation. Such a design strategy showed controllability within otherwise dangerous flight conditions. The Hoverfly vehicle model was able to safely hover or fly away in all trials, which shows us that the proposed open-loop control portion of the release maneuver is reliable and effective. This chapter serves to summarize the results and contributions presented throughout this thesis as well as discuss avenues for future work that would build-on and further-validate the results and analysis presented here.

5.1 Summary of Contributions

5.1.1 Trajectory Design, Analysis, and Optimization

A technique to generate model-based open-loop pitch and body-acceleration trajectory profiles was developed for use on quadrotor vehicles. Such trajectories, motivated by the desire to force a pitch-down maneuver, were shown to reliably allow a falling rotorcraft to escape vortex ring state conditions given appropriate parameter selection. The selection of such parameters was shown to be possible through the use of particle swarm optimization.

These optimum trajectories were flown in simulations to further validate their potential usefulness in real-world parachute release of a quadrotor vehicle.

5.1.2 Controller Implementation

The means to effectively implement an open-loop VRS recovery maneuver were presented. Such an implementation involves a progression through various layers of control, building up from rotors off to stable closed-loop controllability through the use of multiple state transitions. Organizing the control structure in this way ensures that the open-loop trajectory profile can do the "heavy lifting" of removing the vehicle from VRS reliably. This progression was shown to be effective for the repeatable SITL testing of the Hoverfly and can likely be extended to other releasable rotorcraft.

5.2 Recommendations for Future Work

The results of this thesis suggest that the recovery maneuver presented provides substantial improvement in the recoverability of a quadrotor from free fall. However, numerous aspects of the trajectory problem could be further tested or fleshed out to improve their usefulness or better-ensure the validity of the results in real-world applications. The largest of these areas for future work include flight testing on hardware, adding wind effects to the model, and dealing with non-vertical dropping configurations.

5.2.1 Hardware Implementation

An obvious extension of the simulation results presented here is to test the release maneuver on the hardware setup described in Section 2.3. Simulating parachute release in Gazebo is good for examining vehicle behavior and response due to the most significant forces and environmental factors. However, several small changes to the maneuver were made to make the simulation work, and simulation does not fully capture the system dynamics.

To simplify the simulation implementation, the largest change to the parachute release was to not actually release the vehicle from a stationary/slowly falling body. Instead, the drone was flown to a prescribed altitude, held in a stable hover, and then dropped by commanding zero rotation of the rotors. While small perturbations to the initial release angle and angular rate were examined in Section 3.4, the extent of these effects is not fully known without conducting physical testing. Additionally, small changes to the

vehicle's release velocity could not be examined in this way. One last potential change caused by a drop from hover is that the vehicle's rotors would already be spinning at the moment of release. This may slightly alter the response/behavior of the falling vehicle.

The simulation environment also does not capture every little part of the vehicle dynamics simply due to its construction. The Gazebo simulation lacks well-fleshed-out turbulence models, exact effects of rotor geometry, or perfect motor response models. Since VRS is significantly affected by turbulence around the rotor disk, more-accurate modelling of turbulence could influence the response or recoverability of the falling vehicle. Additionally, the simplifications made in modelling the rotors could introduce error in the onset/avoidance of VRS, as it is highly dependent on things such as blade twist, rotor planform, and number of blades [10].

For all of the reasons mentioned above, testing the proposed release maneuver on a physical vehicle is a crucial next step in validating the proposed trajectory solution.

5.2.2 Wind Effects

Throughout this thesis, the effects of environmental wind were neglected in favor of preliminary design of a controller in a best-case scenario. However, this prevents the controller from using wind measurements at the time of release to generate a better trajectory, and this prevents vehicle response to the wind from being explored. The response of the vehicle is simply something that just needs testing and validation. However, Equation (2.17) shows us that a significant windspeed could reduce the induced velocity without requiring much attitude actuation on the part of the vehicle. These effects are something that could be explored either through further simulation with wind considerations, or through flight testing on days when there is significant wind with the aid of some wind probe.

5.2.3 Non-Vertical Release

The only parachute release condition that has been examined here is that of a quadrotor falling from a static point with no initial lateral or vertical velocity at the time of its release. To say this is an unlikely use-case in practice is an understatement. In addition to the perturbations due to release angle offsets, the effects of adding lateral or vertical velocity components at the onset of the maneuver should be investigated. Once again, Equation (2.17) tells us that both of these velocities would affect the induced velocity of the rotors and hence its performance.

In addition to perturbation analysis, exploring non-static release conditions opens the door for more elaborate release maneuvers. Instead of parachute release, quadrotors could instead be released from a moving airplane, for instance. Such a release would have substantial differences in initial conditions from those explored in this thesis, but it shares enough similarities that the heuristically-driven design process presented here could prove useful. Expanding the realm of potential release maneuvers is an important step in expanding what is possible with UAVs.

Appendix A
Simulation Variables for Capturing Vehicle Dynamics

As covered in Chapter 4, the Px4 Gazebo SITL vehicle properties needed to be matched to those of the physical Hoverfly to better-represent the vehicle dynamics. However, many of the simple vehicle properties like $T_{\max}$, C_D, and S are not directly used within the Gazebo simulation environment. Instead, these properties are calculated using simulate motor models and drag models. In this Appendix, the parameters needed to best-approximate vehicle dynamics are derived.

Table A.1. Simulation Variables

Property	Value
$\Omega_{\max}$	$2750\frac{rad}{s}$
K_T	$4.152 \cdot 10^{-3}$
K_{thrust}	$4.959 \cdot 10^{-7}$

A.1 Maximum Rotational Rate

The first parameter that needed determining for the rotor model was the maximum rotational velocity of the rotors. This parameter is used by the simulator to capture the relationship between motor and battery properties. The value is calculated as:

$$\Omega_{\max} = K_v \cdot V_{\max} \cdot \eta \cdot \frac{2\pi}{60}$$

The Hoverfly has $2300K_v$ motors with the 3S LiPo batteries producing a maximum voltage of $12.6V$. Assuming an efficiency of $\approx 90\%$, we arrive at an acceptable estimate of $\Omega_{\max} = 2750\frac{rad}{s}$.

A.2 Motor Moment Constant

The next constant to calculate is the motor's moment constant (sometimes called torque constant) which relates the motor's applied torque to the applied current. While this is not a particularly meaningful need to know for the needs of this thesis, it was still calculated to improve model accuracy. The moment constant is given by:

$$K_T = \frac{60}{2\pi \cdot K_v}$$

With $K_v = 2300$, $K_T = 4.152 \cdot 10^{-3}$

A.3 Motor Thrust Constant

The motor thrust constant is used by the Gazebo environment to relate the rotational rate of the motor to the thrust produced by the attached rotor. Gazebo relates rotational velocity to the produced force as:

$$K_{\text{thrust}} = \frac{T_{\max}}{\Omega_{\max}^2}$$

. With each rotor producing $T_{\max} = 3.75N$ and $\Omega_{\max} = 2750\frac{rad}{s}$, $K_{\text{thrust}} = 4.959 \cdot 10^{-7}$

Appendix B
Optimizing Alternative Flight Conditions

The optimization procedure outlined in Chapter 3 can be used for a variety of flight conditions. While the robustness of the optimization is presented in Section 3.4 for the static goal condition, the optimization results for the forward flight condition are presented here for completeness. We see that the parameter contours are quite similar to those found during the static goal optimization. Furthermore, we see that while optimization of the forward paths reduces the spread and North-direction travel of the vehicle, unoptimized parameters result in very similar path shapes without the real-time computational costs associated with real-time parameter calculation.

Figure B.1 shows the optimal peak pitch angles for the forward flight optimization. Similarly, Figure B.2 shows the optimal t_{Peak_2} found for the forward flight condition. Lastly, Figure B.3 shows the simulated paths in the North-Down coordinate frame for the forward flight condition.

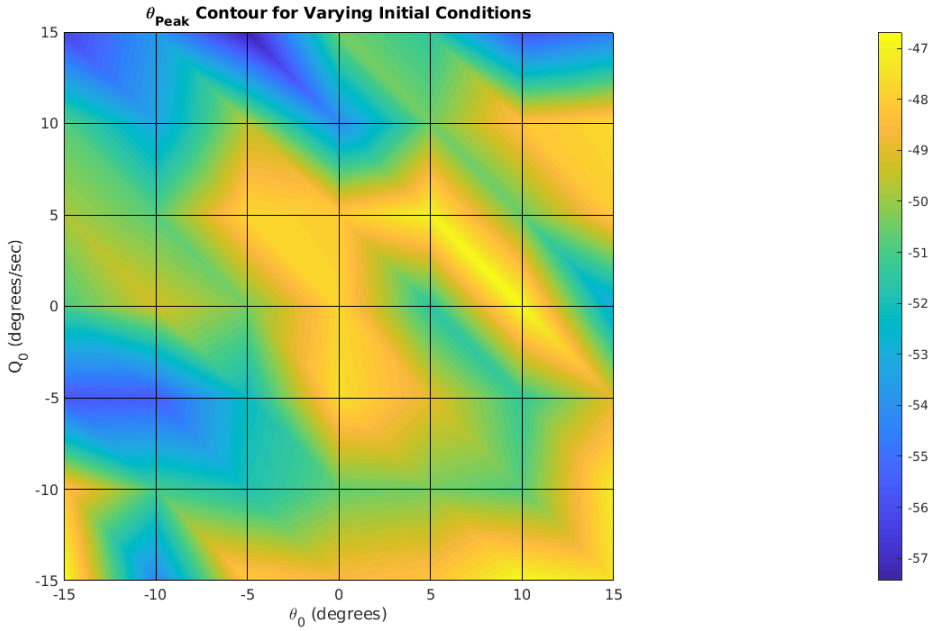

Figure B.1. Peak θ for Forward Flight

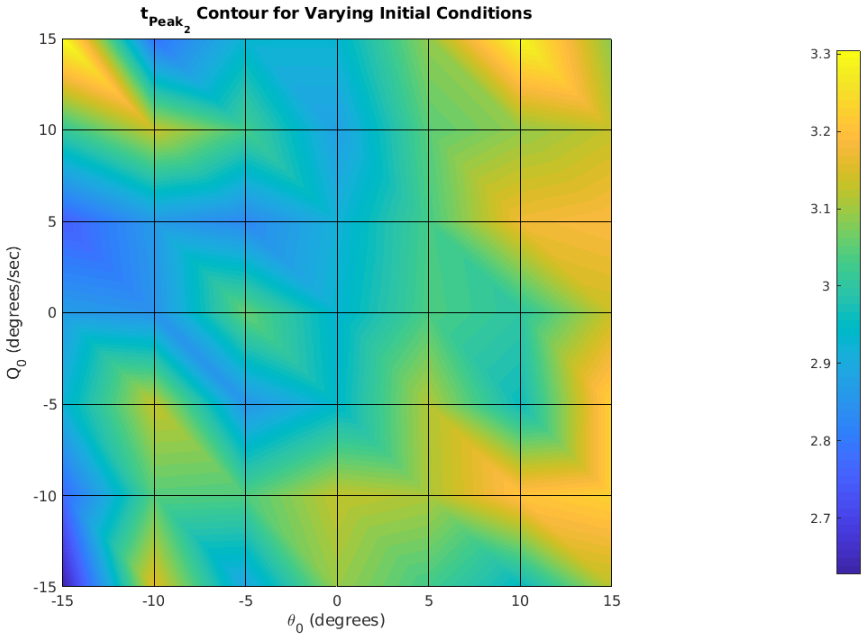

Figure B.2. Peak Time for Forward Flight

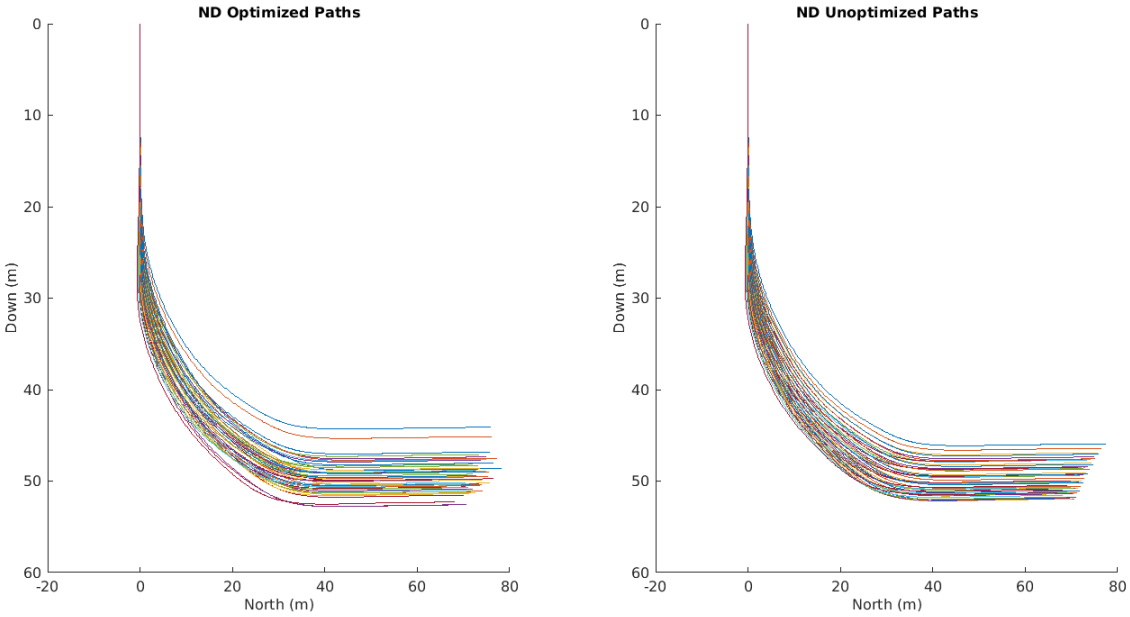

Figure B.3. ND Paths for Forward Flight

Bibliography

[1] "Airships: HMA 23 Class," https://www.airshipsonline.com/airships/hma23/index.html.

[2] Johnson, E. N. and M. A. Turbe (2006) "Modeling, Control, and Flight Testing of a Small-Ducted Fan Aircraft," *Journal of Guidance, Control, and Dynamics*, **29**(4), p. 769–779.

[3] (2020), "Gremlins Program Completes First Flight Test for X-61A Vehicle," https://www.darpa.mil/news-events/2020-01-17.

[4] Slegers, N. and M. Costello (2005) "Model Predictive Control of A Parafoil and Payload System," *Journal of Guidance, Control, and Dynamics*, **28**(4), p. 816–821.

[5] Ward, M. and M. Costello (2013) "Adaptive Glide Slope Control for Parafoil and Payload Aircraft," *Journal of Guidance, Control, and Dynamics*, **36**(4), p. 1019–1034.

[6] Lorenz, R., E. Turtle, J. Barnes, M. Trainer, D. Adams, K. Hibbard, C. Sheldon, K. Zacny, P. Peplowski, D. Lawrence, M. Ravine, T. McGee, K. Sotzen, S. MacKenzie, J. Langelaan, S. Schmitz, L. Wolfarth, and P. Bedini (2018) "Dragonfly: A rotorcraft lander concept for scientific exploration at titan," *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, **34**(3), pp. 374–387.

[7] Brown, D. W. (2019), "NASA Announces New Dragonfly Drone Mission to Explore Titan," https://www.nytimes.com/2019/06/27/science/nasa-titan-dragonfly-caesar.html.

[8] Northon, K. (2017), "NASA Invests in Concept Development for Missions to Comet, Saturn Moon," https://www.nasa.gov/press-release/nasa-invests-in-concept-development-for-missions-to-comet-saturn-moon-titan.

[9] Riley, D., "High Altitude H-Quad Free Fall - RCTESTFLIGHT," https://www.youtube.com/watch?v=nA8_6Oa3_3c.

[10] Johnson, W. (2005) *Model for Vortex Ring State Influence on Rotorcraft Flight Dynamics*, *Tech. rep.*, NASA Ames Research Center.

[11] Padfield, G. D. (1996) *Helicopter flight dynamics: the theory and application of flying qualities and simulation modeling*, Blackwell Science.

[12] Azuma, A. and A. Obata (1968) "Induced flow variation of the helicopter rotor operating in the vortex ring state." *Journal of Aircraft*, **5**(4), pp. 381–386, https://doi.org/10.2514/3.43954.

[13] (2005) "University testbeds demonstrate major advances in autonomous rotorcraft control software," *Vertiflite*, **51**(3), pp. 22–27.

[14] (2019) *Chapter 11: Helicopter Emergencies and Hazards*, United States Department of Transportation, Federal Aviation Administration, Airman Testing Branch, pp. 11–9–11–10.

[15] Drees, I. J. M. and I. W. Hendal (1951) "Airflow Patterns in the Neighbourhood of Helicopter Rotors," .

[16] Brand, A., M. Dreier, R. Kisor, and T. Wood (2011) "The Nature of Vortex Ring State," *Journal of the American Helicopter Society*, **56**(2), p. 22001–2200114.

[17] Opazo, T. and J. W. Langelaan (2020) "Longitudinal control of transition to powered flight for a parachute-dropped multirotor," *AIAA Scitech 2020 Forum*.

[18] Talaeizadeh, A., D. Antunes, H. Pishkenari, and A. Alasty (2020) "Optimal-time quadcopter descent trajectories avoiding the vortex ring and autorotation states," *Mechatronics*, **68**, p. 102362.

[19] Slegers, N., J. Kyle, and M. Costello (2006) "Nonlinear Model Predictive Control Technique for Unmanned Air Vehicles," *Journal of Guidance, Control, and Dynamics*, **29**(5), p. 1179–1188.

[20] Ronco, E., T. Arsan, and P. J. Gawthrop (1999) "Open-loop intermittent feedback control: practical continuous-time GPC," *IEE Proceedings - Control Theory and Applications*, **146**(5), pp. 426–434.

[21] Leitmann, G. (1974) "A Note on Optimal Open-Loop and Closed-Loop Control," *Journal of Dynamic Systems, Measurement, and Control*, **96**(3), pp. 360–362.

[22] Siciliano, B. and O. Khatib (2008) *IMU Packages*, p. 484.

[23] Chowdhary, G., E. Johnson, D. Magree, A. Wu, and A. Shein (2013) "GPS-denied indoor and outdoor monocular vision aided navigation and control of unmanned aircraft," *Journal of Field Robotics*, **30**(3), pp. 415–438.

[24] Langelaan, J. W. (2007) "State Estimation for Autonomous Flight in Cluttered Environments," *Journal of Guidance, Control, and Dynamics*, **30**(5), pp. 1414–1426, https://doi.org/10.2514/1.27770.

[25] Stevens, B. L., F. L. Lewis, and E. N. Johnson (2016) *Aircraft control and simulation: dynamics, controls design, and autonomous systems*, Wiley.

[26] "External Position Estimation," https://dev.px4.io/v1.9.0/en/ros/external_position_estimation.html.

[27] Glauert, H. (1926) *The analysis of experimental results in the windmill brake and vortex ring states of an airscrew*, HMSO.

[28] Jimenez, J., A. Desopper, A. Taghizad, and L. Binet "Induced velocity model in steep descent and vortex-ring state prediction," in *27th European Rotorcraft Forum*.

[29] Jimenez, J., A. Taghizad, and L. Binet "Helicopter Flight Tests in Steep Descents: Vortex-Ring State Analysis and Induced Velocity Models Improvement," in *CEAS-TRA3 Conference Royal Aeronautical Society*.

[30] Taghizad, A., L. Binet, J. Jimenez, and D. Heuze "Experimental and Theoretical Investigations to Develop a Model of Rotor Aerodynamics Adapted to Steep Descents," in *American Helicopter Society 58th Annual Forum*.

[31] Kisor, R., R. Blyth, A. Brand, and T. MacDonald "V-22 Low Speed/High Rate of Descent (HROD) Test Results," in *American Helicopter Society 60th Annual Forum*.

[32] Betzina, M. D. (2001) "Tiltrotor Descent Aerodynamics: A Small-Scale Experimental Investigation of Vortex Ring State," in *American Helicopter Society 57th Annual Forum*.

[33] Shetty, O. and M. Selig *Small-Scale Propellers Operating in the Vortex Ring State*.

[34] "Pixhawk 4," https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4.html.

[35] Ziegler, J. G. and N. B. Nichols (1942) "Optimum Settings for Automatic Controllers," *Transactions of the ASME*, **64**, p. 759–768.

[36] U.S. Department of Transportation, F. A. A. (1978) *Vortex Ring State (Settling with Power)*, vol. 61-13B of *Advisory Circular*.

[37] Yu, J., Z. Cai, and Y. Wang (2014) "Minimum jerk trajectory generation of a quadrotor based on the differential flatness," in *Proceedings of 2014 IEEE Chinese Guidance, Navigation and Control Conference*, pp. 832–837.

[38] Kennedy, J. and R. Eberhart (1995) "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948 vol.4.

[39] Clerc, M. "Standard Particle Swarm Optimisation," 15 pages.

[40] "About ROS," `https://www.ros.org/about-ros/`.

[41] Lee, D., A. Franchi, H. I. Son, C. Ha, H. H. Bülthoff, and P. R. Giordano (2013) "Semiautonomous Haptic Teleoperation Control Architecture of Multiple Unmanned Aerial Vehicles," *IEEE/ASME Transactions on Mechatronics*, **18**(4), pp. 1334–1345.