Dynamical Model Learning and Inversion for Aggressive Quadrotor Flight

Alexander E. Spitzer

CMU-RI-TR-22-03

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee

Nathan Michael (chair)	CMU RI
Christopher Atkeson	CMU RI
Changliu Liu	CMU RI
Giuseppe Loianno	NYU

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

December 2021

Abstract

Quadrotor applications have seen a surge recently and many tasks require precise and accurate controls. Flying fast is critical in many applications and the limited onboard power source makes completing tasks quickly even more important. Staying on a desired course while traveling at high speeds and high accelerations is difficult due to complex and stochastic aerodynamic effects, poorly modeled dynamics, and unreliable state estimation. This thesis seeks to design control strategies that enable quadrotors to track aggressive trajectories precisely and accurately in the presence of external disturbances, unmodeled dynamics, and imperfect state estimation. We first introduce a model learning strategy that allows efficient compensation of learned acceleration disturbances using the differential flatness paradigm. Then, we extend our learning approach to the feedback linearization controller and show that feedback linearization is a viable strategy for aggressive quadrotor flight. We also show that learning attitude dynamics models improves attitude control loop performance, which in turn improves position trajectory tracking performance. Finally, we validate our model learning approach in extensive outdoor experiments at high speed, under realistic disturbance conditions, and with imperfect state estimation.

Acknowledgements

First, I would like to thank my advisor, Nathan Michael, for taking a chance on me and guiding and supporting me along this journey despite its many difficulties and setbacks. I would like to thank my committee, Christopher Atkeson, Changliu Liu, and Giuseppe Loianno, for their interest in my work and feedback along the way. Thank you Chris Atkeson, for helping me always mind the big picture and inspiring me to ask challenging questions. Thank you to Wen Sun for serving on my qualifying committee and his interest and excitement about my work. Thank you to Christopher Batten, for encouraging me to pursue a PhD and guiding me through the application process.

To all past and present members of RISLab, thank you for taking me in and providing a source of community within the RI. I want to thank Vishnu Desaraju for guiding me in my first year and trusting me with running his field experiments. Thank you to Curtis Boirum, for his work on the lab hardware that was a key part of the results in this thesis. Thank you to Yves Georgy Daoud and Xuning Yang, for helping me run the outdoor field experiments. Aditya Dhawale, Arjav Desai, Cormac O'Meadhra, Curt, Derek Mitchell, Ellen Cappo, John Yao, Kumar Shaurya Shankar, Kshitij Goel, Lauren Lieu, Logan Ellis, Matt Collins, Micah Corah, Mike Lee, Mosam Dabhi, Moses Bangura, Tabitha Lee, Vibhav Ganesh, Vishnu, Wennie Tabib, Xuning, and Yves, thank you for being memorable labmates and making RISLab a fun and happy place to work.

To Rogerio Bonatti, Cherie Ho, Adithya Murali, Roberto Shu, Anirudh Vemula, Jerry Hsiung, Leo Keselman, Ankit Bhatia, Cara Bloom, Thomas Weng, Jaskaran Grover, Cecilia Morales, Azarakhsh Keipour, Ratnesh Madaan, Adam Harley, Brian Okorn, Ben Newman, John Macdonald, Mark Lee, Allie Del Giorno, Shivam Vats, Shohin Mukherjee, Paloma Sodhi, and many other friends from the RI, sincerely thank you for sharing the journey with me.

To all my soccer friends and the RI D Stars: Kumar Shaurya Shankar, Roberto Shu, Tanmay Shankar, Aditya Dhawale, Ginés Hidalgo, Chris Cunningham, Zach Batts, Dhruv Saxena, Andrew Sywy, Moses Bangura, Jorge Anton, Karun Warrior, Jorge Izar, Jagjeet Singh, Gabriel Vidal Álvarez, Yunsik Ohm, and any other teammate and opponent I've been lucky to share the field with, thank you for providing endless fun, camaraderie, and injuries.

And to all other colleagues and friends I have been fortunate to meet, thank you for making my time in Pittsburgh very special. Thank you Chelsea, for your friendship, positivity, and happiness.

To my friends Andrew, Noah, Einar, and Leo, thank you for the encouragement, support, laughter, and fun trips. Thank you Andrew for all the fun chats and games.

Thank you Xuning, for your tremendous support, care, and unwavering dedication, tirelessly fixing the robot after crashes, and encouraging me during dark times.

Finally, I'd like to thank my parents and sister for their endless love and support. This thesis would not be possible without their dedication and sacrifices.

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

(a) (b) (c)

Figure 1.1: Autonomous aerial vehicle applications where precise and accurate control is critical to ensure a safe and high quality outcome: (a) A hexarotor inspecting an archway using a contact sensor [9], (b) a quadrotor exploring a subterranean environment in near total darkness [84], and (c) a quadrotor autonomously filming an actor while avoiding obstacles and ensuring an aesthetically pleasing shot [11].

Autonomous mobile robots have the potential to revolutionize society by alleviating the burden of menial and dangerous tasks, increasing the speed or efficiency with which tasks are completed, and enabling applications that were previously not possible. Aerial robots in particular have been recently applied to search and rescue operations, inspection [9, 70], cave exploration [84], agriculture [72], and drone cinematography [11], as shown in Figure 1.1.

Underlying nearly every one of these applications is the need for accurate and robust control. Quadrotors have been well represented in the control literature, due to their agility and versatility, and have been a fruitful testbed for nonlinear controllers and trajectory generation strategies [49, 51, 57, 77]. While control theory is able to provide solutions for simulations and ideal scenarios where the environment and dynamical models are known, flying quickly and precisely through unknown environments with real hardware systems is still challenging. Unknown environments come with external disturbances that are difficult to predict and hardware systems add imperfections, delays, and difficult to model dynamics. Inaccurate dynamical models impede the ability of control algorithms to generate accurate control inputs.

Modeling systems beforehand can remove some of this uncertainty, but such procedures are complex, time intensive, and cannot account for changes to the vehicle and environment that happen online.

Furthermore, in many quadrotor applications, speed is important. In addition to improving performance on time-critical tasks such as search and rescue, flying faster can also open entirely new domains, as the distance that can be traveled without replacing batteries will be increased. Autonomous algorithms must be able to capture the full potential of quadrotors by flying at high speeds and at high acceleration. This in turn, increases the complexity of the dynamics encountered, such as aerodynamics due to strong headwind, making the control problem more difficult.

This thesis focuses on improving quadrotor controls while operating at the limits of vehicle performance in unknown environments and with unreliable state estimation.

Initially, advanced quadrotor controllers were developed with the aid of motion capture systems such as Vicon, MotionAnalysis [63], and OptiTrack[1]. Such systems provide an accurate, high rate state estimate that allows for high gain control. As vehicles are employed outdoors in conditions where obtaining an accurate state estimate is difficult, it becomes increasingly important to develop control algorithms that enable accurate flight without requiring accurate state estimation.

We define accuracy as the closeness between the actual vehicle position $p(t) \colon \mathbb{R} \mapsto \mathbb{R}^3$ and a prespecified reference trajectory $p_{\text{ref}}(t) \colon \mathbb{R} \mapsto \mathbb{R}^3$ for all time $t > 0$. We define accurate flight as flight for which $||p(t) - p_{\text{ref}}(t)||$ is minimized. Aggressive flight is defined as the vehicle following a reference trajectory p_{ref} for which the magnitude of the velocity $||\dot{p}_{\text{ref}}(t)||$ and the magnitude of the acceleration $||\ddot{p}_{\text{ref}}(t)||$ are large for a significant set of times t.

The goal of this thesis is to enable accurate and aggressive flight in the presence of complex nonlinear unmodeled dynamics, unknown external disturbances, and degraded state estimation.

1.1 What is Forward Model-based Control?

Forward model-based controllers convert predictive power into control performance. Forward models are functions that can predict a system's evolution *forwards* in time as a function of the system's state x and provided control inputs u.

$$\dot{x} = f(x, u) \tag{1.1}$$

The model-based controller's task is then to choose control inputs that best satisfy some prescribed objective after evolving the system according to the system model. If the model is accurate, then the actual system's evolution will be close to that predicted by the model and the desired outcome will be achieved. The objective may be described in a number of ways but a very common choice is a cost function combined with a given trajectory or goal state.

[1] https://optitrack.com/applications/robotics

The task of choosing the control inputs for a given model and objective can, at least in principle, be solved by forward simulating the system according to the model for some time for all possible sequences of control inputs, evaluating the resulting trajectory of each control input choice, and choosing the "best" outcome, or a "good-enough" outcome. Of course, the task of trying *every valid sequence of control inputs* is impossible to solve explicitly for all but the most simple discrete systems. Fortunately, there are a number of compromises that can be made to compute the solution in a reasonable time.

For example, restricting the problem to smooth systems with a smooth, real-valued cost function allows for gradient-based algorithms to descend down the cost surface to a local minimum. Assuming linear dynamics and a quadratic cost function allows the use of the large set of tools in linear control theory, e.g. the Linear Quadratic Regulator (LQR). Simplified dynamics models will never match the real system exactly, but the speed of computation they afford can often make up for it and allow the controller to run at a high rate or on compute constrained systems. In particular, the Model Predictive Control (MPC) paradigm can be used in real-time, to continuously correct for error arising due to an inaccurate model or noise, using state feedback.

In general, higher fidelity models and more complicated objectives require more computational resources to *explicitly* forward simulate the system. However, controllers can *implicitly* predict the future in a number of ways. For example, due to the simplified linear dynamics, the optimal LQR controller gain can be computed in closed-form during the design phase, and can then be applied in real time without looking ahead. Dynamic inversion is a technique where a controller *inverts* some assumed system dynamics to convert the system into a linear one, whose evolution is known.

This thesis focuses on these implicit simulation methods, and attempts to improve control performance by inverting a more accurate model of the quadrotor system. However, it should be noted that the endeavor of learning a more accurate model is useful for any model-based control scheme, as long as the improved model can be readily used by the control system.

There are other control methods not based on forward models. Inverse models can be built that directly help compute the control inputs that are "best" for a given system state. *Model-free* methods in general use policies, or functions, that output the correct control inputs. These can be learned using the reinforcement learning paradigm, for example. For this thesis however, we focus on forward model-based control and consider these other methods, while interesting and potentially promising for the future, as out of scope.

1.2 How is Tracking Error Handled?

In a perfect world, systems can be controlled with zero tracking error. Real systems can exhibit error due to model inaccuracies, system stochasticity, and anything else in the environment not considered by the controller. For this discussion, assume there exists some desired trajectory of states $x_{\text{ref}}(t)$ and consider the following computation of the

control input u, which is decomposed into a term dependent on tracking error, $x - x_{\text{ref}}$, and a term only dependent on the reference x_{ref}.

$$u = -K(x - x_{\text{ref}}) + u_{\text{ff}}(x_{\text{ref}}) \tag{1.2}$$

We refer to the first term as the *feedback* controls, since it depends on the true state x, and the second term as the *feedforward* controls. If the cost we want to minimize is

$$J_{\text{total}} = \int_0^\infty J(x - x_{\text{ref}})dt, \tag{1.3}$$

then we notice that to achieve minimal cost, we want to minimize the contribution of the feedback controller. In short, because a feedback controller is driven by error, we want to use as little of it as possible.

In practice, executing identical sequences of control inputs on a real system results in a different sequence of states, even for seemingly deterministic systems such as quadrotors. This is due to the inherent stochasticity of the combined system and environment and is a large reason why feedback controls are ubiquitous and essential. Since the x_{ref} is deterministic and static, any controller for a stochastic system ought to consider the true state x to correct for deviations along the trajectory. Furthermore, the feedforward controls $u_{\text{ff}}(x_{\text{ref}})$ may not result in the desired state sequence x_{ref} when applied even on a deterministic system. This is usually a consequence of using an imperfect dynamics model to compute u_{ff}, an infeasible x_{ref}, or both.

So in summary, a feedback controller has two roles: (1) correct for any deviations arising from system stochasticity, or variance, and (2) correct for errors arising from the feedforward controls, or bias. The distinction between the two types of errors can be blurred, since some behavior may be possible but infeasible to predict (e.g. complex fluid disturbances). Practically, errors in the first category (variance) should be thought of as unpredictable and errors in the second category (bias) should be thought of as predictable or repeatable. Now we can see that handling bias errors using feedback controls is suboptimal if one wishes to minimize tracking error. Thus, the conclusion is that achieving *minimal* tracking error, requires a control designer to handle all repeatable errors in the feedforward controls, and leave the non-repeatable errors to be handled by the feedback controller.

There is a second reason one may wish to lower the influence of the feedback controller: an unreliable state estimate. Since the feedback controller's output depends on the state, any errors, noise, or jumps in the output of the state estimator will be propagated to the controls and to the system. This suggests a design optimization problem considering (1) system stochasticity or variance, (2) dynamics model accuracy, and (3) state estimator accuracy, for the purposes of minimizing tracking error. Improving any one of the above three considerations, that is, reducing system variance, improving feedforward control input accuracy (through a more accurate dynamics model), or improving state estimate accuracy, is a drawback-free way to improve system tracking error. This thesis focuses on the second method: improving feedforward control input accuracy by learning a more accurate dynamics model.

Figure 1.2: An example high speed trajectory executed outdoors on a quadrotor using our proposed methods.

1.3 Challenges

Achieving low tracking error with aerial robots flying at high speed is difficult due to the following challenges.

Challenge 1: Real-world vehicle exhibit complex nonlinear dynamics that are difficult to model and these dynamics only increase in complexity at higher speed.

Challenge 2: External disturbances, such as wind and other aerodynamic effects, are unknown and may be stochastic and time-varying.

Challenge 3: State estimation during aggressive flight is difficult, which leads to noisy and discontinuous state estimates.

1.4 Contributions

We tackle the problem of external disturbances and unknown dynamics by learning a dynamical system model using regression techniques from machine learning. We first combine learned models with the differential flatness paradigm to fully invert the modeled disturbance and recover optimal control performance despite an unknown model. We then extend this approach to the feedback linearization controller for the quadrotor. We show that control input delay modeling can be used to increase the robustness of the feedback linearization controller. We also show feedback linearization's superiority in responding to jumps in the reference or state estimate, thus presenting an advantage over traditional, or feedforward linearization controllers, when using vision-based state estimation. We then explore model learning for the attitude dynamics of the quadrotor and experimentally compare filtering and modeling approaches in the inner loop.

A summary of the contributions is below.

1. A natural framework for the analysis of attitude control feedback controllers.

2. Invert learned dynamical models to improve trajectory tracking performance in the presence of state and input-dependent disturbances for the traditional cascaded controller.
3. Invert learned dynamical models in the presence of state and input-dependent disturbances for the feedback linearization controller.
4. A quadrotor feedback linearization controller that is more robust to control input delay and has reduced order auxiliary dynamics.
5. A model learning strategy for the quadrotor inner loop that is competitive with filtering-based approaches.
6. Validation of the proposed model learning method outdoors with high speed flight by showing a decrease in tracking error. Figure 1.2 shows an example trajectory.

1.5 Outline

An outline of this document is provided below.

In Chapter 2, we review relevant concepts for aggressive and accurate quadrotor control, and regression-based model learning strategies.

In Chapter 3, we use learned dynamical models to improve control performance in the presence of unmodeled dynamics and external disturbances.

In Chapter 4, we extend the work described in chapter 3 to improve the robustness of the feedback linearization controller in the presence of unmodeled dynamics and external disturbances.

In Chapter 5, we explore model learning strategies to improve performance of the quadrotor inner loop.

In Chapter 6, we validate the model learning strategy at high-speed in extensive outdoor experiments.

In Chapter 7, we provide a summary and outline possible avenues for further research.

CHAPTER 2
Background

In this chapter, we review the background material that will be necessary to pursue the goal of accurate trajectory following using a quadrotor. First, we provide the dynamic models relevant for quadrotors and examine the typical quadrotor controller architecture, along with the concept of differential flatness. We also provide an intuitive framework for analyzing quadrotor attitude controllers using rotational error functions. Then, we frame existing quadrotor controllers in the feedforward linearization formulation. Finally, we give an overview of regression strategies that are used later in this thesis such as incremental linear regression, random Fourier features, and Gaussian Processes.

Table 2.1 summarizes the common symbols used in this thesis.

2.1 Modeling

2.1.1 Note on Reference Frames

Most of the equations written in this thesis and derivatives computed hold true in any fixed reference frame, as long as all quantities used are expressed in the same frame. The two most commonly used frames are the world frame $\mathcal{W}$ and the body frame $\mathcal{B}$. The body frame $\mathcal{B}$ is assumed to be rigidly attached to the center of mass of the vehicle and its axes are aligned with the natural vehicle axes; notably for quadrotors, the body z-axis is aligned to the direction of thrust of all rotors. The axes of the body frame $\mathcal{B}$ are x, y, and z. A vector $n \in \mathbb{R}^3$ expressed in the body frame $n_\mathcal{B}$ can be related to the same vector expressed in the world frame $n_\mathcal{W}$ using the rotation matrix $R_\mathcal{W} = \begin{bmatrix} x_\mathcal{W} & y_\mathcal{W} & z_\mathcal{W} \end{bmatrix}$.

$$n_\mathcal{W} = R_\mathcal{W} n_\mathcal{B} \tag{2.1}$$

Although the body frame $\mathcal{B}$ is attached to the body, it is assumed to be an *inertial* frame for the purposes of expressing physical quantities and their derivatives.

When an expression contains quantities expressed in two different frames, the subscripts $\mathcal{W}$ and $\mathcal{B}$ will be used to denote the frame for all relevant quantities.

2.1.2 Dynamics

We model the quadrotor as a rigid body in SE(3), with the state consisting of position $p \in \mathbb{R}^3$, velocity $v \in \mathbb{R}^3$, orientation $R = \begin{bmatrix} x & y & z \end{bmatrix} \in \mathrm{SO}(3)$, and angular velocity

Table 2.1: Summary of mathematical notation

$\dot{x}$	total time derivative of x
$[v_1]_\times$	cross product matrix M s.t. $Mv_2 = v_1 \times v_2$
I_3	3 by 3 identity matrix
e_1, e_2, e_3	3D unit vectors
ψ	Euler angle around the z-axis
θ	Euler angle around the y-axis
ϕ	Euler angle around the x-axis
$\boldsymbol{x}$	aggregate state vector
$\boldsymbol{u}$	aggregate control vector
p	position
v	velocity, $\dot{p}$
a	acceleration, $\dot{v}$
j	jerk, $\dot{a}$
s	snap, $\dot{j}$
$\mathcal{W}$	inertial world frame
$\mathcal{B}$	inertial frame attached to the body
R	orientation of the body in SO(3)
x	body x-axis, Re_1
y	body y-axis, Re_2
z	body z-axis, Re_3
ω	body angular velocity
u	body z acceleration control input
α	body angular acceleration control input
τ	body torque
m	mass
g	gravity vector (points down)
I	inertia matrix

$\omega \in \mathbb{R}^3$, and the control consisting of thrust along the body z-axis $u \in \mathbb{R}^+$ and angular acceleration $\alpha \in \mathbb{R}^3$. Typically, the position p and velocity v are expressed in the fixed frame, while the angular velocity ω and angular acceleration α are expressed in the body frame. The state and control inputs are shown below.

$$\boldsymbol{x} = \begin{bmatrix} p \\ v \\ R \\ \omega \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} u \\ \alpha \end{bmatrix} \tag{2.2}$$

Let the state space be described by

$$\mathcal{X} = \mathbb{R}^3 \times \mathbb{R}^3 \times \mathrm{SO}(3) \times \mathbb{R}^3 \tag{2.3}$$

and the control space by

$$\mathcal{U} = \mathbb{R}^+ \times \mathbb{R}^3. \tag{2.4}$$

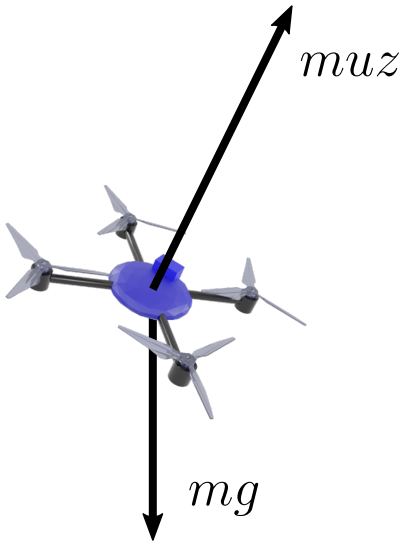


Figure 2.1: The forces acting on the quadrotor are visualized: the gravity force mg points straight down and the force due to the motor thrusts muz points in the direction of the body z-axis.

Fig. 2.1 depicts the two forces considered in the standard quadrotor model: gravity and motor thrust. From this, the acceleration $\dot{v} = a$ can be computed as $uz + g$.

The time derivative of the rotation matrix expressed in the world frame $\dot{R}_{\mathcal{W}}$ as a function of the angular velocity expressed in the body frame $\omega_{\mathcal{B}}$ can be computed by calculating the time derivatives of the body axes x, y, and z, which are rotating at an angular velocity of ω.

$$\begin{aligned}
\dot{R}_{\mathcal{W}} = \begin{bmatrix} \dot{x}_{\mathcal{W}} & \dot{y}_{\mathcal{W}} & \dot{z}_{\mathcal{W}} \end{bmatrix} &= \begin{bmatrix} \omega_{\mathcal{W}} \times x_{\mathcal{W}} & \omega_{\mathcal{W}} \times y_{\mathcal{W}} & \omega_{\mathcal{W}} \times z_{\mathcal{W}} \end{bmatrix} && \text{(2.5a)} \\
&= \begin{bmatrix} (R_{\mathcal{W}}\omega_{\mathcal{B}}) \times x_{\mathcal{W}} & (R_{\mathcal{W}}\omega_{\mathcal{B}}) \times y_{\mathcal{W}} & (R_{\mathcal{W}}\omega_{\mathcal{B}}) \times z_{\mathcal{W}} \end{bmatrix} && \text{(2.5b)} \\
&= R_{\mathcal{W}} \begin{bmatrix} \omega_{\mathcal{B}} \times (R_{\mathcal{W}}^{\top} x_{\mathcal{W}}) & \omega_{\mathcal{B}} \times (R_{\mathcal{W}}^{\top} y_{\mathcal{W}}) & \omega_{\mathcal{B}} \times (R_{\mathcal{W}}^{\top} z_{\mathcal{W}}) \end{bmatrix} && \text{(2.5c)} \\
&= R_{\mathcal{W}} \begin{bmatrix} \omega_{\mathcal{B}} \times e_1 & \omega_{\mathcal{B}} \times e_2 & \omega_{\mathcal{B}} \times e_3 \end{bmatrix} && \text{(2.5d)} \\
&= R_{\mathcal{W}} [\omega_{\mathcal{B}}]_{\times} && \text{(2.5e)}
\end{aligned}$$

Thus, the dynamics are given by

$$\dot{\boldsymbol{x}} = \begin{bmatrix} v \\ uz + g \\ [\omega]_{\times} R \\ \alpha \end{bmatrix}. \tag{2.6}$$

2.1.3 Motor Modeling

In Section 2.1, we modeled the quadrotor with angular acceleration as a control input. However, quadrotors in practice use Electronic Speed Controllers (ESCs) to control rotor speed. ESCs with basic functionality take as input a throttle level from 0 to 100, and

apply power to the motor proportional to the input. The drawback to this simple scheme is that as the voltage of the input power source drops, typically from a battery, the rotational speed of the motor will drop as well. Thus, the rotational speed of the motor is a function of the input voltage as well as the input throttle. To maintain a consistent rotational speed for a given input, certain ESCs feature "closed-loop RPM mode", which allows a higher level controller to send a desired motor rotation speed (RPM) as a setpoint to the motor controller.

In order to precisely control the quadrotor using a model-based paradigm, the onboard controller needs to know how commands sent to the ESCs affect the motion of the vehicle. Typically, an empirical relationship is estimated between motor speed and the thrust produced for a particular motor and propeller combination. The rotor force model used by the hardware in this thesis is a quadratic and calibrated beforehand using a load cell.

$$f_{\text{motor}}(r) = ar^2 + br + c \tag{2.7}$$

Additionally, the relationship between the torque a rotor produces around the shaft axis and the thrust is assumed to be linear.

$$\tau_{\text{motor}}(r) = c_\tau f_{\text{motor}}(r) \tag{2.8}$$

Then, rotor forces can be converted into a force and torque acting on the center of mass of the quadrotor using knowledge of the vehicle geometry. Since typically, all rotors are aligned in the direction of the vehicle's z-axis, the force produced is a vector aligned with the body z-axis with magnitude equal to the sum of all of the motor forces. The direction of the torque exerted on the vehicle from each motor is opposite the direction of rotation of the rotor shaft. Given the position of each rotor expressed in the body frame p_i, the total torque produced is the sum of each of the torques produced by the motors, $p_i \times (f_i e_3) + (-1)^i c_\tau f_i e_3$.

$$\begin{bmatrix} mu \\ \tau \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 1 \\ p_1 \times e_3 + c_\tau e_3 & \cdots & p_n \times e_3 - c_\tau e_3 \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} = Mf \tag{2.9}$$

Strictly speaking, the term quadrotor refers to vehicles with four rotors, while multirotor refers to vehicles more generally with any number of rotors, typically an even number ≥ 4. While we use the term quadrotor throughout, all of the control methods in this thesis can be applied to hexarotors, octocopters, and most other multirotors by recomputing M from (2.9) to reflect the geometry of the vehicle. If the rotors are not aligned with the vehicle's z-axis, which is out of this thesis' scope, the vehicle is often fully actuated and additional complexity is introduced [10].

2.2 Differential Flatness

Differential flatness, first introduced by Fliess, Lévine, Martin, and Rouchon [26], is a property of a dynamical system that permits analytic expressions relating the system's

state and control inputs to derivatives of a certain *flat output*. For a simple example, consider a wheel of radius r rolling along a 1-dimensional line. The states of the system are the wheel's position p and angular velocity ω, while the control input is the torque applied to the wheel τ. The dynamics of the wheel are given by

$$\dot{p} = \omega r \tag{2.10}$$

$$\dot{\omega} = \tau \tag{2.11}$$

The position p can be seen as a flat output, since the remaining state, angular velocity, can be expressed as $\omega = \dot{p}/r$ and the control input, torque, can be expressed as $\tau = \ddot{p}/r$. This is very useful for the purpose of controlling the wheel along a smooth position trajectory. We can obtain the necessary control input τ by using the second derivative of the desired position trajectory.

Van Nieuwstadt and Murray [88] showed how the concept of differential flatness can be used to generate control inputs and tracking control laws that follow a given trajectory for *differentially flat* systems. A large variety of practically useful system models are differentially flat, such as many configurations of singular and coupled bodies [67], and in particular, the quadrotor model described by (2.6).

Differential flatness of the quadrotor has been shown in many works but perhaps most popularly in [61], which uses the Euler ZXY convention. Faessler, Franchi, and Scaramuzza [23], which uses the Euler ZYX convention, extends the differential flatness property of quadrotors to include linear rotor drag, and corrects the yaw handling equations from [61]. Both [61] and [23] show differential flatness component-wise.

Before providing a vector-based derivation of differential flatness for the quadrotor model (2.6), we first review different conventions for the definition of yaw ψ.

2.2.1 Yaw Definitions

The "yaw" is typically defined as the rotation around the z-axis for a particular sequence of elementary rotations around the three axes x, y, and z. We will consider two Tait-Bryan (also commonly referred to as Euler angles) sequences of intrinsic rotations: ZYX and ZXY.

ZYX refers to first rotating about the z-axis (yaw), then rotating about the new body y-axis (pitch), and finally rotating around the new body x-axis (roll). ZXY rotates around z first, then rotates around the new x-axis, and finally the new y-axis. Although ZYX is more common in the literature [55], ZXY has also been used [61], and often times the convention is not explicitly mentioned, which leads to confusion.

We will now show that for a given orientation R, *ZYX yaw is equal to the angle formed by projecting the x-axis onto the horizontal plane and e_1 and ZXY yaw is equal to the angle formed by projecting the y-axis into the horizontal plane and e_2.*

To see this intuitively, take the case of ZYX. The first rotation rotates both the x and y axes an angle of ψ with respect to the original x and y axes, i.e. e_1 and e_2. The

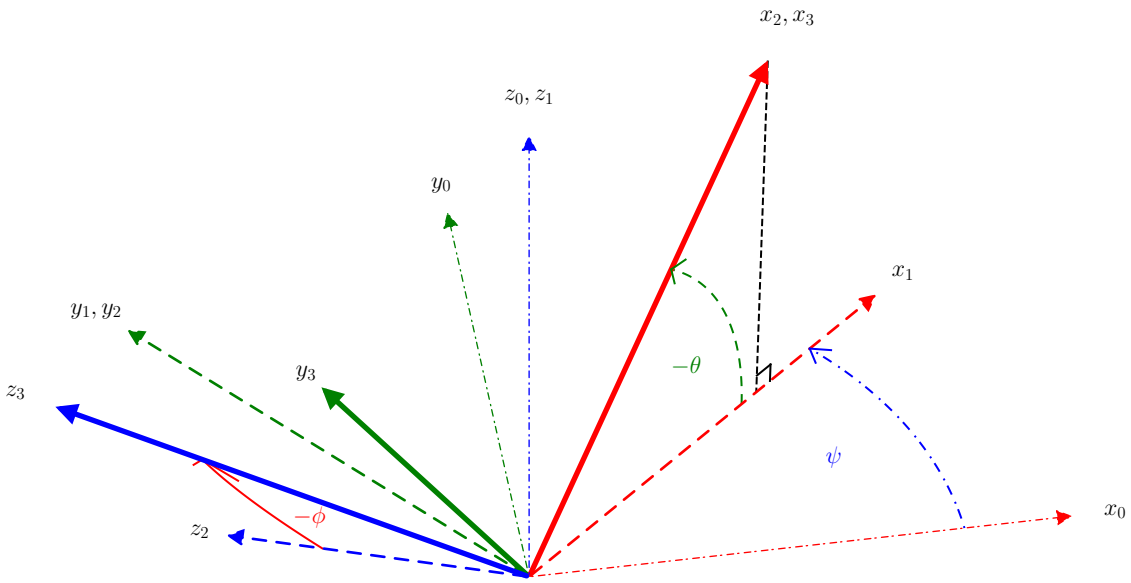


Figure 2.2: A visualization of the Euler ZYX convention. First, the body is rotated around the z-axis z_0 by an angle of ψ, then around the new y-axis y_1 by an angle of θ, and finally around the new x-axis x_2 by an angle of ϕ. Since the last rotation around x_2 does not change the x-axis, the yaw ψ can be defined as the angle between the projection of the body x-axis x_3 onto the horizontal plane and the world x-axis x_0.

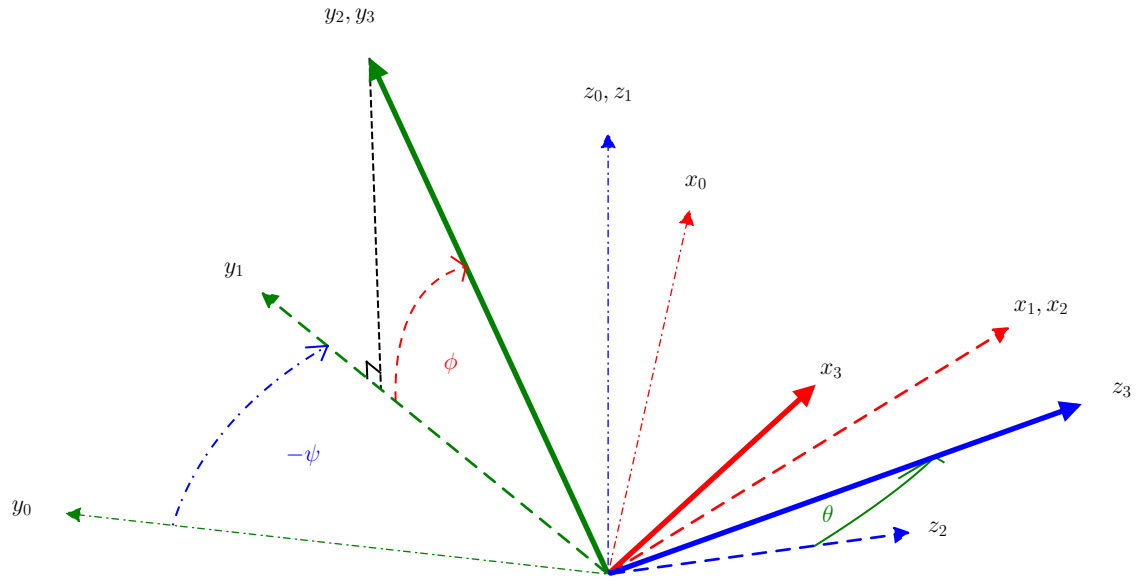


Figure 2.3: A visualization of the Euler ZXY convention. First, the body is rotated around the z-axis z_0 by an angle of ψ, then around the new x-axis x_1 by an angle of ϕ, and finally around the new y-axis y_2 by an angle of θ. Since the last rotation around y_2 does not change the y-axis, the yaw ψ can be defined as the angle between the projection of the body y-axis y_3 onto the horizontal plane and the world y-axis y_0.

second rotation rotates around the new y-axis, which, while lifting the x-axis out of the horizontal plane, preserves the angle between the *projection* of the x-axis onto the horizontal plane and e_1[1]. Thus after two rotations, the body x and body y axes, when projected onto the horizontal plane, both make an angle of ψ with respect to e_1 and e_2 respectively. Finally, the third rotation rotates around the body x-axis, which leaves the x-axis unchanged. Now, while the y-axis' projection onto the horizontal plane, for $\phi \neq 0$, no longer makes an angle of ψ with e_2, the x-axis' projection still does. This process is visualized in Fig. 2.2.

The same argument with the axes flipped shows that for ZXY, the yaw ψ is described by the angle between the y-axis' projection onto the horizontal plane and e_2. This is visualized in Fig. 2.3, which is essentially a mirror image of Fig. 2.2.

Below we denote sin and cos by s and c for brevity and show the above mathematically.

$$R_{ZYX} = R_z(\psi)R_y(\theta)R_x(\phi) \tag{2.12a}$$

$$= \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} \tag{2.12b}$$

$$= \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta & s\theta s\phi & s\theta c\phi \\ 0 & c\phi & -s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{2.12c}$$

$$= \begin{bmatrix} c\theta c\psi & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ c\theta s\psi & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{2.12d}$$

From (2.12d), note that for $c\theta \neq 0$, $x_{ZYX} = R_{ZYX}e_1$ makes an angle of ψ with e_1 when projected onto the horizontal plane (zero out the z-component) and thus

$$\tan(\psi_{ZYX}) = \frac{x^\top e_2}{x^\top e_1}. \tag{2.13}$$

Repeating this for the Euler ZXY convention,

$$R_{ZXY} = R_z(\psi)R_x(\phi)R_y(\theta) \tag{2.14a}$$

$$= \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \tag{2.14b}$$

$$= \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta & 0 & s\theta \\ s\phi s\theta & c\phi & -s\phi c\theta \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \tag{2.14c}$$

$$= \begin{bmatrix} c\phi c\theta - s\psi s\phi s\theta & -s\psi c\phi & c\psi s\theta + s\psi s\phi c\theta \\ s\psi c\theta + c\psi s\phi s\theta & c\psi c\phi & s\psi s\theta - c\psi s\phi c\theta \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \tag{2.14d}$$

[1]When the rotation around the y-axis, or pitch, is 90°, the yaw is undefined.

From (2.14d), note that for $c\phi \neq 0$, $y_{ZXY} = R_{ZXY}e_2$ makes an angle of ψ with e_2 when projected onto the horizontal plane (zero out the z-component) and thus

$$\tan(\psi_{ZXY}) = -\frac{y^\top e_1}{y^\top e_2}. \tag{2.15}$$

We define the following vectors that lie in the horizontal plane and represent the body x and body y axes after rotation around the z-axis by ψ.

$$x_C = \begin{pmatrix}\cos(\psi) & \sin(\psi) & 0\end{pmatrix}^\top \tag{2.16}$$

$$y_C = \begin{pmatrix}-\sin(\psi) & \cos(\psi) & 0\end{pmatrix}^\top \tag{2.17}$$

In the remainder of this thesis, we assume that yaw ψ is defined using the Euler ZYX convention.

2.2.2 Flat Output to States and Control Inputs

We show that the model described by (2.6) is differentially flat with the outputs of position p and yaw ψ, as defined by the Euler ZYX convention[2]. To do this, we need to show that the states and control inputs are functions of the flat outputs and their higher order derivatives.

Let $R = \begin{bmatrix}x & y & z\end{bmatrix} \in \mathrm{SO}(3)$. Note x, y, and z are all of unit length and pair-wise orthogonal. Let a (acceleration), j (jerk), and s (snap), all in $\mathbb{R}^3$, represent the second, third, and fourth derivative of the vehicle position.

The objective is, given p, v, a, j, s, ψ, $\dot{\psi}$, and $\ddot{\psi}$, compute R, ω, u, and α.

Let the flat output and its derivatives be represented by

$$\mathcal{Z} = \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathcal{S}^1 \times \mathbb{R} \times \mathbb{R}. \tag{2.18}$$

The next three parts will implement a transformation

$$\mathcal{T}^{-1} \colon \mathcal{Z} \mapsto \mathcal{X} \times \mathcal{U}, \tag{2.19}$$

which can also be written as (omitting the trivial mapping between position and velocity)

$$R, \omega, u, \alpha = \mathcal{T}^{-1}(a, j, s, \psi, \dot{\psi}, \ddot{\psi}). \tag{2.20}$$

Note that while $\mathcal{X}$ and $\mathcal{U}$ together only have 16 degrees of freedom, $\mathcal{Z}$ has 18 degrees of freedom due to the inclusion of the jerk and snap of the position along the z-axis.

Before computing the mapping $\mathcal{T}^{-1}$, we first find the forward mapping $\mathcal{T}$.

$$a, j, s, \psi, \dot{\psi}, \ddot{\psi} = \mathcal{T}(R, \omega, u, \alpha) \tag{2.21}$$

[2]The computation using the Euler ZXY convention is similar but is omitted here for brevity.

For ease of reference in later sections, we introduce the notation $\mathcal{T}_\xi(R, \omega, u, \alpha)$, representing the computation of an arbitrary quantity ξ using a subset of the input (R, ω, u, α).

First we compute $\dot{z}$, $\ddot{z}$. $\dot{x}$, and $\ddot{x}$.

$$\mathcal{T}_{\dot{z}}(z, \omega) := \dot{z} = \omega \times z \tag{2.22}$$
$$\mathcal{T}_{\ddot{z}}(z, \omega, \alpha) := \ddot{z} = \alpha \times z + \omega \times \dot{z} \tag{2.23}$$
$$\mathcal{T}_{\dot{x}}(x, \omega) := \dot{x} = \omega \times x \tag{2.24}$$
$$\mathcal{T}_{\ddot{x}}(x, \omega, \alpha) := \ddot{x} = \alpha \times x + \omega \times \dot{x} \tag{2.25}$$

From the dynamics (2.6), we can compute a, j, and s.

$$\mathcal{T}_a(z, u) := a = uz + g \tag{2.26}$$
$$\mathcal{T}_j(z, u, \dot{u}, \omega) := j = \dot{u}z + u\dot{z} \tag{2.27}$$
$$\mathcal{T}_s(z, u, \dot{u}, \ddot{u}, \omega, \alpha) := s = \ddot{u}z + 2\dot{u}\dot{z} + u\ddot{z} \tag{2.28}$$

Let $x_1 = x^\top e_1$ and $x_2 = x^\top e_2$. From (2.13), we can compute ψ.

$$\mathcal{T}_\psi(x) := \psi = \text{atan2}(x_2, x_1) \tag{2.29}$$

Differentiating (2.29) provides $\dot{\psi}$ and $\ddot{\psi}$.

$$\mathcal{T}_{\dot{\psi}}(x, \omega) := \dot{\psi} = \frac{-x_2\dot{x}_1 + x_1\dot{x}_2}{x_1^2 + x_2^2} \tag{2.30}$$
$$\mathcal{T}_{\ddot{\psi}}(x, \omega, \alpha) := \ddot{\psi} = \frac{-x_2\ddot{x}_1 + x_1\ddot{x}_2 - 2\dot{\psi}(x_1\dot{x}_1 + x_2\dot{x}_2)}{x_1^2 + x_2^2} \tag{2.31}$$

2.2.2.1 Computing Orientation and Thrust

From (2.26), we can compute u and z.

$$\mathcal{T}_u^{-1}(a) := u = ||a - g|| \tag{2.32}$$
$$\mathcal{T}_z^{-1}(a) := z = \frac{a - g}{||a - g||} \tag{2.33}$$

The body x-axis is computed as the normalized cross product between y_C and z. This ensures that x is perpendicular to z and also parallel to x_C when projected onto the horizontal plane (i.e. satisfying the yaw constraint).

$$x = \frac{y_C \times z}{||y_C \times z||} \tag{2.34}$$

The y-axis is computed from the x and z-axes to satisfy orthonormality using the right-hand rule.

$$y = z \times x \tag{2.35}$$

Eqs. (2.33)–(2.35) define the following inverse mapping for the orientation R.

$$\mathcal{T}_R^{-1}(a, \psi) := R = \begin{bmatrix} x & y & z \end{bmatrix} \tag{2.36}$$

2.2.2.2 Computing Angular Velocity

Before we compute the angular velocity ω as a function of the flat states, we compute ω as a function of $\dot{z}$ and $\dot{x}$ using the scalar and vector triple products[3], in effect inverting (2.22) and (2.24).

$$\begin{aligned} z \times \dot{z} &= z \times (\omega \times z) \\ &= (z^\top z)\omega - (\omega^\top z)z \\ &= \omega - (\omega^\top z)z \end{aligned} \tag{2.37}$$

and analogously for $\dot{x}$,

$$x \times \dot{x} = \omega - (\omega^\top x)x \tag{2.38}$$

Since (2.37) computes ω less its z-component, and (2.38) computes ω less its x-component, we can add the z-component of (2.38) to (2.37) to get a full expression for ω.

$$\begin{aligned} \omega &= z \times \dot{z} + \left((x \times \dot{x})^\top z\right) z \\ &= z \times \dot{z} + \left((z \times x)^\top \dot{x}\right) z \\ &= z \times \dot{z} + (y^\top \dot{x})z \end{aligned} \tag{2.39}$$

(2.39) computes the angular velocity ω from the orientation and the first time derivatives of the body x and body z axes.

We can compute $\dot{z}$ from (2.27) directly and $\dot{u}$ by projecting (2.27) along z.

$$\dot{z} = \frac{1}{u}\left(j - \dot{u}z\right) \tag{2.40}$$

$$\dot{u} = j^\top z \tag{2.41}$$

To compute $\dot{x}$, differentiate (2.34).

$$\dot{x} = \frac{I - xx^\top}{||y_C \times z||}\left(-\dot{\psi}x_C \times z + y_C \times \dot{z}\right) \tag{2.42}$$

Now, combine (2.42) with (2.40) and (2.39) to get the inverse mapping for ω.

$$\mathcal{T}_\omega^{-1}(u, a, j, \psi, \dot{\psi}) := \omega = \frac{1}{u}z \times j + \frac{1}{||y_C \times z||}\left(\dot{\psi}x_C^\top x + \frac{1}{u}(j^\top x)(y_C^\top z)\right) z \tag{2.43a}$$

(2.43a) can also be expressed without x or x_C, as

$$\mathcal{T}_\omega^{-1}(u, a, j, \psi, \dot{\psi}) := \omega = \frac{1}{u}z \times j + \frac{1}{||y_C \times z||^2}\left(\dot{\psi}z^\top e_3 + \frac{1}{u}(y_C^\top z)y_C^\top(z \times j)\right) z \tag{2.43b}$$

[3] $a^\top(b \times c) = b^\top(c \times a) = c^\top(a \times b)$. $a \times (b \times c) = \left(a^\top c\right) b - \left(a^\top b\right) c$.

2.2.2.3 Computing Angular Acceleration

To compute the angular acceleration α as a function of $\ddot{z}$ and $\ddot{x}$, we similarly apply a cross product with z and x to (2.23) and (2.25).

$$\begin{aligned} z \times \ddot{z} &= \left(z^\top z\right) \alpha - \left(\alpha^\top z\right) z + \left(z^\top \dot{z}\right) \omega - \left(\omega^\top z\right) \dot{z} \\ &= \alpha - (\alpha^\top z) z - (\omega^\top z) \dot{z} \end{aligned} \tag{2.44}$$

And similarly for $\ddot{x}$,

$$x \times \ddot{x} = \alpha - (\alpha^\top x) x - (\omega^\top x) \dot{x} \tag{2.45}$$

Now combine (2.44) and (2.45) to obtain

$$\begin{aligned} \mathcal{T}_\alpha^{-1}(u, a, j, s, \psi, \dot{\psi}, \ddot{\psi}) := \alpha &= z \times \ddot{z} + (\omega^\top z) \dot{z} + \left(\left(x \times \ddot{x} + (\omega^\top x) \dot{x} \right)^\top z \right) z \\ &= z \times \ddot{z} + (\omega^\top z) \dot{z} + \left(y^\top \ddot{x} + (\omega^\top x)(z^\top \dot{x}) \right) z \\ &= z \times \ddot{z} + (\omega^\top z) \dot{z} + \left(y^\top \ddot{x} - (\omega^\top x)(\omega^\top y) \right) z. \end{aligned} \tag{2.46}$$

(2.46) computes the angular acceleration α from the orientation, and the first and second derivatives of the body x and body z axes.

We can compute $\ddot{z}$ from (2.28) and $\ddot{u}$ by projecting (2.28) onto z and making use of $z^\top z = 1 \implies z^\top \dot{z} = 0 \implies z^\top \ddot{z} = -\dot{z}^\top \dot{z}$.

$$\ddot{z} = \frac{1}{u} \left(s - 2\dot{u}\dot{z} - \ddot{u} z \right) \tag{2.47}$$

$$\ddot{u} = s^\top z + u \dot{z}^\top \dot{z} \tag{2.48}$$

To compute $\ddot{x}$, (2.42) can be differentiated. However, in practice, the second gradient of (2.13), along with the unit length constraint and orthogonality constraint between x and z, is used in a linear system to solve for $\ddot{x}$.

2.2.2.4 Linear System Solutions for $\dot{x}$ and $\ddot{x}$

The body axis rates $\dot{x}$ and $\ddot{x}$ are fully constrained by considering the following three constraints on x.

$$x^\top x = 1 \qquad \text{unit length constraint} \tag{2.49}$$

$$x^\top z = 0 \qquad \text{orthonormality constraint} \tag{2.50}$$

$$\tan(\psi) = \frac{x^\top e_2}{x^\top e_1} \qquad \text{Euler ZYX yaw constraint} \tag{2.51}$$

Differentiate (2.49), (2.50), and rearrange (2.30) to obtain

$$x^\top \dot{x} = 0 \tag{2.52}$$

$$z^\top \dot{x} = -x^\top \dot{z} \tag{2.53}$$

$$\begin{bmatrix} -x_2 & x_1 & 0 \end{bmatrix} \dot{x} = \dot{\psi} \left(x_1^2 + x_2^2 \right). \tag{2.54}$$

Now $\dot{x}$ can be found by solving the linear system

$$A\dot{x} = b_1 \tag{2.55}$$

with

$$A = \begin{bmatrix} — & x^\top & — \\ — & z^\top & — \\ -x_2 & x_1 & 0 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ -x^\top \dot{z} \\ \dot{\psi}\left(x_1^2 + x_2^2\right) \end{bmatrix}. \tag{2.56}$$

To find $\ddot{x}$, differentiate (2.52), (2.53), and rearrange (2.31).

$$x^\top \ddot{x} = -\dot{x}^\top \dot{x} \tag{2.57}$$

$$z^\top \ddot{x} = -2\dot{x}^\top \dot{z} - x^\top \ddot{z} \tag{2.58}$$

$$\begin{bmatrix} -x_2 & x_1 & 0 \end{bmatrix} \ddot{x} = \ddot{\psi}\left(x_1^2 + x_2^2\right) + 2\dot{\psi}(x_1\dot{x}_1 + x_2\dot{x}_2) \tag{2.59}$$

Now $\ddot{x}$ can be found by solving the linear system

$$A\ddot{x} = b_2 \tag{2.60}$$

with

$$b_2 = \begin{bmatrix} -\dot{x}^\top \dot{x} \\ -2\dot{x}^\top \dot{z} - x^\top \ddot{z} \\ \ddot{\psi}\left(x_1^2 + x_2^2\right) + 2\dot{\psi}\left(x_1\dot{x}_1 + x_2\dot{x}_2\right) \end{bmatrix}. \tag{2.61}$$

2.3 Feedback Control

In this section, we describe the feedback control architecture and relevant concepts used throughout the remainder of the thesis. Quantities denoted with "ref", as in a_{ref}, denote reference quantities that are assumed given from a higher-level planner. Those denoted with "des", as in a_{des}, denote desired quantities that are derived from reference quantities, vehicle state, or both, and used as inputs to the controller.

2.3.1 Cascaded Control Architecture

Traditional quadrotor controllers use the cascaded controller architecture depicted in Fig. 2.4, where a position controller generates references for an inner attitude controller. This is also known as a *backstepping* controller.

The outer position controller is most commonly a PD controller, the output of which is interpreted as an acceleration vector.

$$a_{\text{des}} = -K_p(p - p_{\text{ref}}) - K_v(v - v_{\text{ref}}) + a_{\text{ref}} \tag{2.62}$$

To compute u_{des}, cascaded controllers typically use one of the following three methods.

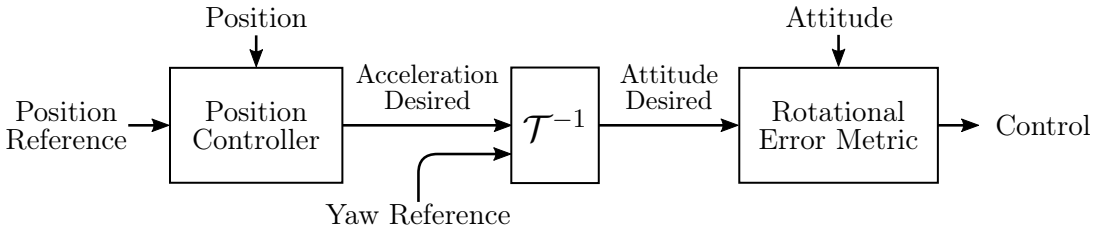


Figure 2.4: The standard cascaded quadrotor controller architecture. The position controller computes a desired acceleration, which is then combined with the yaw reference to generate an attitude reference for the attitude controller. The attitude controller then uses a rotational error metric to compute the controls.

1. Norm of the desired acceleration vector [28]
2. The desired acceleration vector projected onto the actual body z-axis [23, 61]
3. The acceleration necessary to achieve the desired acceleration along the world z-axis [30]

These correspond to the following versions of $\mathcal{T}_u^{-1}$ applied to a_{des} and z.

$$\mathcal{T}_{u1}^{-1}(a) \quad := u = ||a - g|| \tag{2.63a}$$
$$\mathcal{T}_{u2}^{-1}(a, z) := u = (a - g)^\top z \tag{2.63b}$$
$$\mathcal{T}_{u3}^{-1}(a, z) := u = \frac{(a - g)^\top e_3}{z^\top e_3} \tag{2.63c}$$

Note that with $z_{\text{des}} = \mathcal{T}_z^{-1}(a_{\text{des}})$, $\mathcal{T}_{u1}^{-1}(a_{\text{des}}) = \mathcal{T}_{u2}^{-1}(a_{\text{des}}, z_{\text{des}}) = \mathcal{T}_{u3}^{-1}(a_{\text{des}}, z_{\text{des}})$. The most common choice in the literature is the second, and this is what we will use in this thesis.

$$u_{\text{des}} = \mathcal{T}_{u2}^{-1}(a_{\text{des}}, z) = (a_{\text{des}} - g)^\top z \tag{2.64}$$

The desired acceleration vector a_{des} is then combined with the reference yaw ψ_{ref} and higher order references j_{ref}, s_{ref}, $\dot{\psi}_{\text{ref}}$, and $\ddot{\psi}_{\text{ref}}$ to compute the desired orientation R_{des}, desired angular velocity ω_{des}, and desired angular acceleration α_{des} for the attitude controller using the differential flatness transformations $\mathcal{T}_R^{-1}$ from (2.36), $\mathcal{T}_\omega^{-1}$ from (2.43b), and $\mathcal{T}_\alpha^{-1}$ from (2.46).

$$R_{\text{des}} = \mathcal{T}_R^{-1}(a_{\text{des}}, \psi_{\text{ref}}) \tag{2.65}$$
$$\omega_{\text{des}} = \mathcal{T}_\omega^{-1}(u_{\text{des}}, a_{\text{des}}, j_{\text{ref}}, \psi_{\text{ref}}, \dot{\psi}_{\text{ref}}) \tag{2.66}$$
$$\alpha_{\text{des}} = \mathcal{T}_\alpha^{-1}(u_{\text{des}}, a_{\text{des}}, j_{\text{ref}}, s_{\text{ref}}, \psi_{\text{ref}}, \dot{\psi}_{\text{ref}}, \ddot{\psi}_{\text{ref}}) \tag{2.67}$$

Note that the acceleration u_{des} used in the above inverse mappings does not necessarily have to be equal to the acceleration used as the system control input.

The desired orientation, angular velocity, and angular acceleration are then used in a modified-PD attitude controller, which uses a rotational error function e_R: SO(3) $\times$

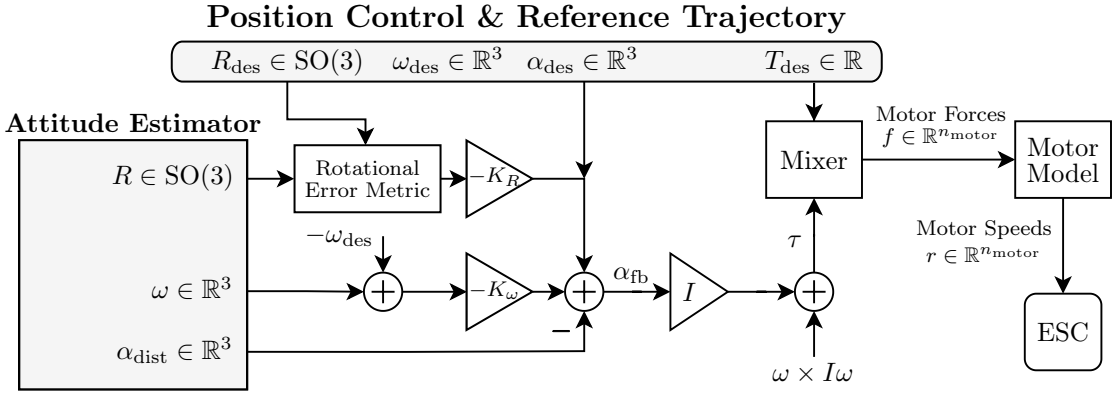


Figure 2.5: Dataflow diagram of the attitude control system. The output of the attitude estimator, combined with the position controller and reference trajectory, is used to compute a desired angular acceleration, which in turn gets converted into desired rotor speeds using the vehicle and motor dynamics models.

$\text{SO}(3) \mapsto \mathbb{R}^3$ to compare the current orientation R to the desired orientation R_{des}.

$$\alpha = -K_R e_R(R, R_{\text{des}}) - K_\omega(\omega - \omega_{\text{des}}) + \alpha_{\text{des}} \tag{2.68}$$

This attitude controller is described in further detail in Section 2.3.2.

2.3.2 Attitude Control

The attitude controller takes in the following as input.

- The attitude state, consisting of the orientation of the vehicle $R \in \text{SO}(3)$, the angular velocity $\omega \in \mathbb{R}^3$, and angular acceleration disturbance estimate $\alpha_{\text{dist}} \in \mathbb{R}^3$.
- The reference trajectory and position controller output, consisting of the desired orientation $R_{\text{des}} \in \text{SO}(3)$, desired angular velocity $\omega_{\text{des}} \in \mathbb{R}^3$, desired angular acceleration $\alpha_{\text{des}} \in \mathbb{R}^3$, and the desired thrust $T_{\text{des}} = mu_{\text{des}} \in \mathbb{R}^+$.

The controller outputs the desired motor speed for each of the n_{motor} motors on the vehicle, $r \in \mathbb{R}^{n_{\text{motor}}}$.

A high level data flow diagram of the controller is shown in Fig. 2.5.

In the first stage, the current attitude state is compared to the desired attitude state to generate an angular acceleration α_{fb} that intends to drive the attitude state to the desired state. A special attitude error function $e_R : \text{SO}(3) \times \text{SO}(3) \mapsto \mathbb{R}^3$ must be used to compare the current attitude to the desired attitude, while the angular velocities are compared using a simple difference.

$$\alpha_{\text{fb}} = -K_R e_R(R, R_{\text{des}}) - K_\omega(\omega - \omega_{\text{des}}) + \alpha_{\text{des}} - \alpha_{\text{dist}} \tag{2.69}$$

Since ω_{des} and α_{des} are typically expressed in either the reference frame defined by $R_{\text{ref}} = \mathcal{T}^{-1}(a_{\text{ref}}, \psi_{\text{ref}})$ or the desired frame defined by $R_{\text{des}} = \mathcal{T}^{-1}(a_{\text{des}}, \psi_{\text{ref}})$, they can be converted into the current body frame by multiplication with $R^\top R_{\text{ref}}$ or $R^\top R_{\text{des}}$. This

choice is relevant when using an imperfect dynamical model and will be explored in Section D.

Converting the control inputs to rotational motor speeds is first achieved by calculating forces for each of the rotors, that when applied, will result in the desired body thrust and angular acceleration. Angular acceleration α is converted into torque τ using Euler's equations of motion for a rigid body.

$$\tau = I\alpha + \omega \times I\omega \tag{2.70}$$

This torque is then combined with the desired thrust along the body z-axis $T_{\text{des}} = mu$ using the vehicle mixer to generate the desired forces at each motor $f \in \mathbb{R}^{n_{\text{motor}}}$. These motor forces are such that, according to the vehicle geometry, applying them on the vehicle will result in the desired torque and thrust. Since the mixer is usually the linear mapping M found in (2.9), computing these forces is a matter of inverting M, though more advanced techniques can be used, for example to prioritize vehicle stability under control input saturation [21].

$$f = M^{\dagger} \begin{bmatrix} T_{\text{des}} \\ \tau \end{bmatrix} \tag{2.71}$$

Finally, each desired rotor force in f is converted into a desired motor speed by inverting the model of the motor's rotor speed to thrust relationship. This is typically the quadratic mapping (2.7), introduced in Section 2.1.3.

$$r_i = f_{\text{motor}}^{-1}(f_i) \tag{2.72}$$

2.3.2.1 Rotational Error Functions

In this section, we give example attitude error functions e_R that can be used with the described attitude control system. Attitude error functions effectively compute a difference between two orientations in SO(3). We provide brief discussions about the advantages and disadvantages of each error function.

Euler Angles The most common choice of error function uses differences of Euler angles. It is simple to implement and think about, but it has many disadvantages, mainly coming from the coupling between angles at large displacements.

Let ϕ, θ, ψ correspond to the Euler angles around the x, y, and z axes (the ZYX Euler angles) for an orientation $R \in \text{SO}(3)$.

$$e_R = \begin{pmatrix} \phi - \phi_{\text{des}} & \theta - \theta_{\text{des}} & \psi - \psi_{\text{des}} \end{pmatrix} \tag{2.73}$$

SO(3) Two orientations can directly be compared in SO(3) using an error function described by [50]. Here, $\vee$ (vee) is an operator that takes a skew symmetric matrix and turns it into a 3 dimensional vector (the inverse of the $\wedge$ (hat) map).

$$e_R = \frac{1}{2\sqrt{1 + \text{tr}[R_{\text{des}}^{\top} R]}} \left(R_{\text{des}}^{\top} R - R^{\top} R_{\text{des}} \right)^{\vee} \tag{2.74}$$

This is a natural error function choice for attitude control since it provides a response in the direction of the shortest path, or Euler axis rotation to the desired orientation. However, it may not be ideal for quadrotors, as it does not decouple yaw from the attitude response.

As written above, the error function has a singularity when the angular error is 180°. This can be avoided by using an error function, shown below, that smoothly approaches zero as the angular difference approaches 180°.

$$e_R = \frac{1}{2}\left(R_{\text{des}}^\top R - R^\top R_{\text{des}}\right)^\vee \tag{2.75}$$

The disadvantage of this is that now the response of the system decreases for angles larger than 90°. However, this error function is still seen in the literature [34]. From a theoretical point of view, they both provide almost global exponential stability for the set of initial attitude errors that excludes errors of 180°.

Quaternion Similar to the SO(3) error function, the computation can be performed using quaternions [13]. Let q be the quaternion representing the rotation represented by R and let q_w and q_v be the scalar and axis part of that quaternion.

$$q_{\text{err}} = q^{-1} \cdot q_{\text{des}} \tag{2.76}$$
$$e_R = -2\,\text{sgn}(q_{\text{err,w}})q_{\text{err,v}} \tag{2.77}$$

This error function turns out to be equivalent to the first SO(3) error function described above, less a factor of 2. It may be more desirable to use than the SO(3) function, since it is simple and more efficient to implement. For example, this is the error function used by the PX4 autopilot[4], where the yaw decoupling discussed in [13] is also implemented.

Thrust Vector The z-axis of a quadrotor is dynamically linked to its position. This is due to the constraint that any linear acceleration imparted by the rotors acts in the direction of the vehicle's z-axis. However, the rotation around the z axis is unconstrained by the position, and thus it makes sense to treat the attitude control problem as two distinct problems: the alignment of the body z-axis to the desired z-axis, and the selection of an angular acceleration around the z-axis. This decomposition is discussed in detail in [48]. We focus only on the z-axis alignment problem here. The acceleration around the z-axis can be chosen to follow a desired yaw trajectory.

The change in rotation required to align two vectors can be computed by taking a cross product. Let z be the current z-axis of the vehicle and z_{des} be the desired z-axis.

$$e_R = z_{\text{des}} \times z \tag{2.78}$$

In practice, this is usually implemented in the body frame where $z = e_3$.

This thrust vector attitude error function has the advantage that it does not couple yaw dynamics with translational vehicle dynamics as the SO(3) and quaternion-based metrics

[4] https://github.com/PX4/Firmware/blob/ab060cdab076fb6934a86a75349cd9330c873432/src/modules/mc_att_control/AttitudeControl/AttitudeControl.cpp#L88

do. One such example is provided in [48]: SO(3)-based metrics result in deviation from a straight line step when there is large initial yaw error. For a second example, consider a diagonal step with a desired yaw of zero. SO(3)-based metrics result in the vehicle following a curved path to the goal while the thrust vector error function follows the straight line path to the goal.

(2.78), similarly to (2.75), has magnitude that decreases as the angle error increases past 90° and reaches zero at 180°. This can be alleviated using a scaling factor similar to the one in (2.74). See Appendix B for the details.

Later, we will see how the attitude controllers discussed here compare to the feedback linearization controller. For a more detailed discussion of various rotational error functions, including simulation case studies, see Appendix B.

2.3.3 Model Predictive Controllers

In addition to the differential flatness and linearization-based controllers discussed in this chapter, there exist model predictive controllers (MPC), which attempt to optimize a trajectory subject to the vehicle dynamics and state and control input constraints. The key idea of MPC is to use a forward model of the system dynamics to predict what will happen as a function of the chosen control input, and select the control input to achieve the desired outcome. In contrast to planners, which typically generate a trajectory and/or control inputs once at the start, an MPC repeats this computation in the control loop, taking into account the current state of the vehicle.

There are a wide variety of such optimizing controllers and they can be roughly categorized by the assumptions they place on the system dynamics, cost function, and constraints. The solution type and complexity will vary according to these assumptions. In general, MPC methods that allow for nonlinear dynamics and nonlinear cost functions and constraints are more computationally expensive than those that make more simplifying assumptions such as linear dynamics, a quadratic cost function, and linear constraints.

MPC methods have been very popular on a wide variety of systems [54, 93]. The use of a forward model in the optimization makes MPC an easy paradigm to combine with (forward) model learning of the system dynamics [5, 19, 52, 86]. MPC has also been applied for active perception on quadrotors [24] and combined with feedforward linearization [35].

While this thesis does not focus on MPC, many of the techniques presented here can be augmented with MPC in a manner similar to [35]. Although differential flatness can adequately account for nonlinear dynamics, state and control input constraints are still a problem, and that is one area where applying an MPC can provide benefit.

2.4 Disturbance Compensation

We use $\mathcal{L}_1$ adaptive control [63] on the position loop and an angular acceleration disturbance observer on the attitude loop as baseline disturbance compensation strategies. $\mathcal{L}_1$ adaptive control is essentially a nonlinear Luenberger state observer with a low pass filter. We briefly outline the implementation used for acceleration disturbance compensation and angular acceleration disturbance compensation.

2.4.1 Acceleration Disturbance Observer

The observer state consists of the estimated velocity $\hat{v} \in \mathbb{R}^3$ and the estimated acceleration disturbance $\hat{a}_d \in \mathbb{R}^3$. The model dynamics are assumed to be

$$\dot{v} = uz + g + a_d. \tag{2.79}$$

The velocity estimate $\hat{v}$ is updated according to the predicted velocity and the velocity estimation error scaled by a gain.

$$\dot{\hat{v}} = uz + g + \hat{a}_d - K_v(\hat{v} - v) \tag{2.80}$$

The acceleration disturbance estimate $\hat{a}_d$ is updated using the velocity estimation error scaled by a gain.

$$\dot{\hat{a}}_d = -K_a(\hat{v} - v) \tag{2.81}$$

Before the acceleration disturbance estimate $\hat{a}_d$ is used in the controller, a low pass filter of bandwidth γ is applied.

$$\dot{\hat{a}}_d^{\text{LPF}} = -\gamma(\hat{a}_d^{\text{LPF}} - \hat{a}_d) \tag{2.82}$$

(2.80), (2.81), (2.82) are implemented at position control rate using Euler integration.

The cascaded controller (2.62) is then augmented to compensate for the estimated disturbance.

$$a_{\text{des}}^{\mathcal{L}_1} = -K_p(p - p_{\text{ref}}) - K_v(v - v_{\text{ref}}) + a_{\text{ref}} - \hat{a}_d^{\text{LPF}} \tag{2.83}$$

2.4.2 Angular Acceleration Disturbance Observer

An estimate of the angular acceleration disturbance is updated by comparing the predicted angular velocity with the true angular velocity. Let $\hat{\alpha}_d \in \mathbb{R}^3$ be the angular acceleration disturbance estimate and $\alpha \in \mathbb{R}^3$ the angular acceleration control input from the attitude controller.

$$\dot{\hat{\omega}} = \alpha + \hat{\alpha}_d - K_\omega(\hat{\omega} - \omega) \tag{2.84}$$

$$\dot{\hat{\alpha}}_d = -K_\alpha(\hat{\omega} - \omega) \tag{2.85}$$

See Section A.2.4 for more details about the attitude disturbance estimation baseline.

2.5 Linear Regression for Model Learning

We use incremental linear regression to enable online learning of system dynamics. We first introduce the linear least squares model learning formulation, then show how the same linear regression can leverage a nonlinear feature space, and finally show how the solution to these regression problems can be computed recursively online.

2.5.1 Linear Regression

Linear regression attempts to find the linear model that best fits a given dataset. The assumed model is of the form

$$y = w^\top x. \tag{2.86}$$

Given a dataset consisting of N pairs of points (x_i, y_i), with $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$, the objective is to find $\hat{w} \in \mathbb{R}^n$, such that the difference between the predicted values $\hat{w}^\top x_i$ and the true values y_i is minimized. This can be encoded using the sum of the squared errors using the following cost function.

$$f(w) = \sum_{i=1}^{N} \frac{1}{2}(w^\top x_i - y_i)^2 \tag{2.87}$$

We can find a minimum of $f(w)$ by differentiating with respect to w. First define $X = \begin{bmatrix} x_1 \cdots x_N \end{bmatrix}^\top \in \mathbb{R}^{N \times n}$ and $Y = \begin{bmatrix} y_1 \cdots y_N \end{bmatrix}^\top \in \mathbb{R}^N$. Here we adopt the notation X_{ij}, X_i, and Y_i to refer to the i'th row j'th column entry of X, i'th row of X, and i'th entry of Y.

$$\begin{aligned}
\frac{\partial f}{\partial w_i} &= \sum_{j=1}^{N} (w^\top X_j - y_j) X_{ji} \\
&= \sum_{j=1}^{N} (w^\top X_j) X_{ji} - \sum_{j=1}^{N} y_j X_{ji} \\
&= \sum_{j=1}^{N} \left(\sum_{k=1}^{n} X_{jk} w_k \right) X_{ji} - \sum_{j=1}^{N} X_{ji} y_j \\
&= \sum_{j=1}^{N} \sum_{k=1}^{n} X_{ji} X_{jk} w_k - \sum_{j=1}^{N} X_{ji} y_j \\
&= \sum_{k=1}^{n} \sum_{j=1}^{N} X_{ji} X_{jk} w_k - \sum_{j=1}^{N} X_{ji} y_j \\
&= \sum_{k=1}^{n} \left(\sum_{j=1}^{N} X_{ji} X_{jk} \right) w_k - \sum_{j=1}^{N} X_{ji} y_j
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{k=1}^{n} \left(\sum_{j=1}^{N} X_{ij}^{\top} X_{jk} \right) w_k - \sum_{j=1}^{N} X_{ij}^{\top} y_j \\
&= \sum_{k=1}^{n} \left(X^{\top} X \right)_{ik} w_k - \sum_{j=1}^{N} X_{ij}^{\top} y_i \\
&= \left(X^{\top} X w \right)_i - \left(X^{\top} Y \right)_i \\
\frac{\partial f}{\partial w_i} &= \left(X^{\top} X w - X^{\top} Y \right)_i
\end{aligned} \tag{2.88}
$$

From (2.88), it follows that

$$
\frac{\partial f}{\partial w} = X^{\top} X w - X^{\top} Y \tag{2.89}
$$

and a local minimum can be found at

$$
\hat{w} = \left(X^{\top} X \right)^{-1} X^{\top} Y. \tag{2.90}
$$

The $n \times n$ matrix $X^{\top} X$ is invertible when there are at least n linearly independent input vectors in the set $\{x_i\}_{i=1}^{N}$. In that case, the pseudoinverse of X is indeed $(X^{\top} X)^{-1} X^{\top}$. In practice, because dynamical models are learned from many data points and $N >> n$, this is almost always the case and $\hat{w}$ represents a least squares solution to the regression problem.

Bayesian Interpretation One can formulate linear regression from a Bayesian point of view using the model form

$$
p(y|x, w) = \mathcal{N}(w^{\top} x, \sigma_n^2) \tag{2.91}
$$

$$
w \sim \mathcal{N}(0, \Sigma_p) \tag{2.92}
$$

where σ_n^2 represents the variance of the output points and Σ_p the uncertainty in a zero prior on w [75]. The *maximum a posteriori* (MAP) estimate is found using

$$
\hat{w} = \left(X^{\top} X + \sigma_n^2 \Sigma_p^{-1} \right)^{-1} X^{\top} Y. \tag{2.93}
$$

The addition of $\sigma_n^2 \Sigma_p^{-1}$ ensures the inverse always exists for any positive σ_n^2 and positive definite Σ_p. When $\Sigma_p = cI_n$, (2.93) is equivalent to *Tikhonov regularization* or *ridge regression*. The uncertainty in the zero prior on w serves as a hyperparameter that controls how strong the model estimates are towards zero. Biasing estimates towards zero is a way to *regularize* the model and is used to avoid overfitting. In the incremental dynamical model learning setting, a zero prior can be useful in the first few learning iterations to avoid spurious and noisy model outputs. When the variance of the zero prior on w is infinite, (2.93) reduces to (2.90).

In addition to an output mean, Bayesian linear regression outputs a full posterior distribution, the variance of which may be useful for downstream applications such as constrained model predictive control [19].

One big drawback with linear regression is that nonlinear functions of the input cannot be accurately captured by a linear model. In the next section, we review how linear regression can be augmented with nonlinear features to alleviate this shortcoming.

2.5.2 Linear Regression with Nonlinear Features

To allow for the model to capture nonlinear relationships between the input x_i and output y_i, we can compute nonlinear features of the input $\phi(x_i)$. The assumed model form is now

$$y = w^\top \phi(x) \tag{2.94}$$

This model is strictly more general than (2.86). Pure linear model learning can be recovered with $\phi(x) = x$. Affine model learning results from $\phi(x) = \begin{bmatrix} x^\top & 1 \end{bmatrix}^\top$ and n'th order univariate polynomial regression can be performed with $\phi(x) = \begin{bmatrix} 1 & x & x^2 & \cdots & x^n \end{bmatrix}$.

The resulting squared error cost function

$$f(w) = \sum_{i=1}^{N} \frac{1}{2}(w^\top \phi(x_i) - y_i)^2 \tag{2.95}$$

can be minimized in much the same way as in standard linear regression. Let $\phi \colon \mathbb{R}^n \mapsto \mathbb{R}^d$ and $\Phi \in \mathbb{R}^{N \times d}$ be the matrix whose i'th row is $\phi(x_i)$. Then, the unregularized least squares solution $\hat{w} \in \mathbb{R}^d$ can be found with

$$\hat{w} = \left(\Phi^\top \Phi\right)^{-1} \Phi^\top Y \tag{2.96}$$

and the regularized solution with

$$\hat{w} = \left(\Phi^\top \Phi + \sigma_n^2 \Sigma_p^{-1}\right)^{-1} \Phi^\top Y. \tag{2.97}$$

Rahimi and Recht [74] showed that sinusoids with randomly chosen frequencies are a particularly effective choice of features and approximate a feature space in which inner products are defined using the squared exponential kernel. The number of random frequencies D can be smoothly varied to trade-off between computation speed and model regressive power. A small number of frequencies is easy to regress with but the model may not be able to accurately capture the nonlinearities of the system. A large number of frequencies increases the model power at the expense of computation time. Random frequencies $\Omega \in \mathbb{R}^{D \times n}$ can be generated by multiplying a D by n matrix of univariate Gaussians by $\text{diag}(M)$, where each entry in $M \in \mathbb{R}^n$ is the inverse of the characteristic length scale of the corresponding input dimension. The length scales allow for the adjustment of the relative importance of each input dimension and are thus hyperparameters that need to be adapted to the application [33]. The nonlinear features as a function of the input data x are

$$\phi(x) = \frac{1}{\sqrt{D}} \begin{bmatrix} \cos(\Omega x) \\ \sin(\Omega x) \end{bmatrix}. \tag{2.98}$$

2.5.3 Incremental Linear Regression

Since the linear regression problem can be solved using a matrix inverse, updating the solution to the linear regression problem given a new data point corresponds to performing a rank-one update on the corresponding matrix inverse. The Sherman-Morrison formula applied to the inverse is one way to do this, while rank-one updates to the Cholesky decomposition is another, typically more numerically stable, approach.

Incremental linear regression using the random features (2.98) is described in [32] and extended to the Bayesian formulation in [33].

A key observation from (2.90), and the analogous solutions (2.93) and (2.97), is that the sizes of $(X^\top X)^{-1} \in \mathbb{R}^{n\times n}$ and $X^\top Y \in \mathbb{R}^n$ are independent of the number of data points in the regression problem N. Thus if $(X^\top X)^{-1}$ and $X^\top Y$ can be computed recursively as additional data points (x_i, y_i) are received, the regression solution $\hat{w}$ can easily be computed in time independent of the number of data points.

$$X^\top Y = \begin{bmatrix} x_1 & \cdots & x_N \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{N} y_i x_i \tag{2.99}$$

(2.99) shows that $X^\top Y$ can be computed using a running sum.

To recursively compute $(X^\top X)^{-1}$, first decompose $X^\top X$ into its Cholesky decomposition $R^\top R$, then perform rank one updates to R using Givens rotations [32]. While the Sherman–Morrison formula can directly be used on $(X^\top X)^{-1}$, the Cholesky decomposition method is more numerically stable. The Cholesky factor R is then used with backsubstitution and $X^\top Y$ to compute $\hat{w}$ in $O(n)$ time.

CHAPTER 3

Inverting Learned Dynamics Models for Quadrotor Trajectory Tracking

We present a control strategy that applies inverse dynamics to a learned acceleration error model for accurate quadrotor control input generation. This allows us to retain accurate trajectory and control input generation despite the presence of exogenous disturbances and modeling errors. Although accurate control input generation is traditionally possible when combined with parameter learning-based techniques, we propose a method that can do so while solving the relatively easier non-parametric model learning problem. We show that our technique is able to compensate for a larger class of model disturbances than traditional techniques can and we show reduced tracking error while following trajectories demanding accelerations of more than $7\,\mathrm{m/s^2}$ in quadrotor simulation and hardware experiments.

3.1 Introduction

3.1.1 Motivation

Computing precise control inputs for a dynamical system often requires accurate knowledge of its dynamics. Chapter 2.2 showed that for a quadrotor, differential flatness can be used to compute the exact inputs required to follow a specified trajectory in x, y, z, and yaw. The computed control inputs are only accurate if the fixed dynamic model and its associated parameters, e.g. mass, inertia, etc., are correct. Often, this fixed dynamic model assumption fails and the estimated parameters are inaccurate. This results in suboptimal trajectory tracking performance.

One possible approach to alleviate this problem is to estimate the model parameters from vehicle trajectory data. This however, can be difficult, and is still suboptimal when the chosen parameterization cannot realize the true vehicle model. On the other hand, non-parametric error models are commonly used and relatively easy to learn but are not readily used in the differential flatness framework. In this chapter, we show how a non-parametric error model can be used to generate control inputs that follow a specified trajectory. We additionally provide an extension to the proposed approach that can deal with *input-dependent* model errors via numerical optimization. We validate the control input generation strategy both in simulation and through experiments on a quadrotor.

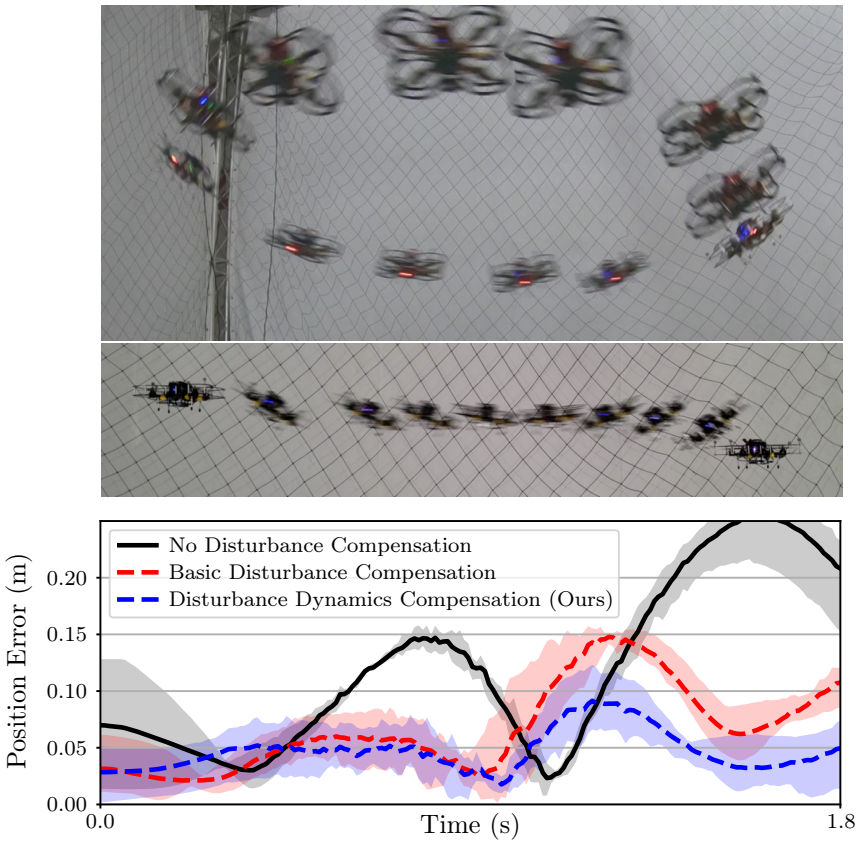


Figure 3.1: Our experimental platform while executing an aggressive circle trajectory (top) and an aggressive line trajectory (middle) using the proposed control input generation strategy that is capable of compensating for dynamic and input-dependent acceleration disturbances (**FF5**). Our method substantially reduces tracking error along the aggressive line trajectory (bottom).

The contents of this chapter first appeared in [80].

3.1.2 Related Works

Accurate and aggressive quadrotor flight has been explored in [21, 61, 65, 85] among others. As for many other robotic platforms, accurate modeling has been shown to improve flight performance [7, 83]. Traditional non-learning based modeling can be achieved via hand crafted experiments, calibration procedures, and computer-aided design [60]. Since this requires significant manual effort and engineering hours, there have been many works exploring automatic parameter estimation methods [15, 16] and non-parametric model learning methods [18, 53] for quadrotor control. In this thesis, we focus on non-parametric model learning methods, since parameter learning methods can be limited in their accuracy by the choice of parameterization [68]. There has also been work on learning control input corrections for aggressive flight without learning a dynamical model [56]. These methods are not a focus of this chapter since they can typically only be applied while executing the trained trajectories or reference quantities. A learned

dynamical model can be applied to any trajectory or reference.

Non-parametric model learning methods for robot control have been employed in [1, 27, 59, 79]. Model learning performed in real-time incrementally has been studied in [6, 32, 37]. Florez, Bellot, and Morel [27] use Locally Weighted Projection Regression (LWPR [90]) while Gijsberts and Metta [32] use Random Fourier Features [74], which was extended to Incremental Sparse Spectrum Gaussian Process Regression (ISSGPR), a Bayesian regression formulation, in Gijsberts and Metta [33]. Droniou, Ivaldi, Padois, and Sigaud [20] evaluated LWPR and ISSGPR for the purposes of robot control and found ISSGPR to perform better. In this chapter, we use both linear regression and ISSGPR.

Once an accurate dynamical system model is known, a Model Predictive Control (MPC) strategy can be used to optimize a desired cost function, subject to the dynamics [5, 18, 54, 57, 86]. These approaches often make approximations to ensure real time feasibility [5, 18] or require expensive numerical optimizations [86]. Furthermore, Desaraju [18] does not perform full inverse dynamics on the disturbance, which can lead to suboptimal performance while tracking aggressive trajectories.

The differential flatness property of quadrotors has been widely exploited for accurate trajectory tracking [23, 25, 42, 61, 65, 78]. Differential flatness of the quadrotor subject to linear drag was shown in Faessler, Franchi, and Scaramuzza [23]. This extends the applicability of the approach to a limited family of disturbances. Faessler, Franchi, and Scaramuzza [23] do not address the issue of nonlinear disturbances as a function of state and/or control input in the flatness computations. Issues arising from singularities, commonly encountered during aggressive flight, were discussed and mitigated in Morrell, Rigter, Merewether, Reid, Thakker et al. [65], increasing the robustness of the differential flatness approach.

Although control inputs computed using the differential flatness framework will automatically take into account dynamical model parameter changes, such as mass, inertia, etc., it is not clear how to incorporate non-parametric model corrections. In this chapter, we build on the differential flatness formulation by extending it to compensate for learned non-parametric dynamic model disturbances. Our approach can compensate for arbitrary disturbances that are a function of vehicle position and velocity, as well as control input dependent disturbances that are a function of vehicle orientation and thrust. This increases the applicability of the approach to a much wider range of realistic flight conditions.

Table 3.1 provides a high level comparison between our method and the methods of Faessler, Franchi, and Scaramuzza [23] and Torrente, Kaufmann, Foehn, and Scaramuzza [86]. [23] can only deal with linear disturbances and thus can neither deal with constraints nor input-dependent disturbances. [86] can deal with nonlinear disturbances, input-dependent disturbances, and constraints, owing to its use of numerical optimization and the MPC paradigm. Our method deals with both nonlinear disturbances and input-dependent disturbances in an efficient matter, but cannot easily handle state or control input constraints.

Table 3.1: A high level comparison between the proposed and two representative methods

Capability	[23]	[86]	**Ours**
Nonlinear	×	✓	✓
Efficient	✓	×	✓
Constraints	×	✓	×
Input-dependent disturbances	×	✓	✓

3.2 Method

We first introduce the problem statement in Section 3.2.1. Section 3.2.2 details our approach for compensating for dynamic disturbances that can be a function of vehicle position, vehicle velocity, or other quantities that are independent of the applied control inputs. Section 3.2.3 extends the approach to compensate for disturbances that are *input-dependent* and can be a function of e.g. the applied vehicle thrust or vehicle orientation. Finally, Section 3.2.4 describes the model learning approach.

3.2.1 Problem Statement

Assume we are given a desired position over time, $p_{\text{ref}}(t) \in \mathbb{R}^3$, along with its first four time derivatives, the velocity, acceleration, jerk, and snap: $v_{\text{ref}}(t)$, $a_{\text{ref}}(t)$, $j_{\text{ref}}(t)$, $s_{\text{ref}}(t)$.

Equation (3.1) shows the standard acceleration model of a quadrotor, as introduced in Section 2.1, augmented with an additive acceleration error term $\boldsymbol{f}_e$.

$$\boldsymbol{a} = u\boldsymbol{z} + \boldsymbol{g} + \boldsymbol{f}_e(\boldsymbol{\eta}, u) \tag{3.1}$$

Here $u \in \mathbb{R}$ is the commanded body acceleration, $\boldsymbol{z} \in \mathbb{R}^3$ is the body z-axis ($||\boldsymbol{z}|| = 1$), $\boldsymbol{g} = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^\top$ is the gravity vector, and $\boldsymbol{f}_e \in \mathbb{R}^3$ is an additive acceleration error model that can, in general, be a function of both vehicle state $\boldsymbol{\eta}$ and control input u.

The objective is to compute the body acceleration u, body z-axis $\boldsymbol{z}$, angular velocity $\boldsymbol{\omega}$, and angular acceleration $\dot{\boldsymbol{\omega}}$ such that integrating $\dot{\boldsymbol{\omega}}$ forwards in time twice results in an orientation with $\boldsymbol{z}$ as the z-axis and that the vehicle acceleration, which is a function of $u\boldsymbol{z}$, equals the desired vehicle acceleration $\boldsymbol{a}_d(t)$. This will ensure that the vehicle follows the specified trajectory $\boldsymbol{x}_d(t)$. Note that while $\boldsymbol{z}$ and $\boldsymbol{\omega}$ are not true control inputs to the system, they are necessary as feedforward references to the attitude feedback controller. Once the body acceleration u and angular acceleration $\dot{\boldsymbol{\omega}}$ are computed, they are multiplied by mass and inertia and used as the feedforward force and torque in the position and attitude feedback controllers respectively.

3.2.2 Input-independent Error Compensation

The simplest version of our control input generation strategy assumes that the disturbance model $\boldsymbol{f}_e$ is a function of the vehicle position and velocity only: $\boldsymbol{f}_e(\boldsymbol{x}, \dot{\boldsymbol{x}})$. In this case, the desired acceleration vector can be computed directly, as shown in (3.2).

$$u\boldsymbol{z} = \boldsymbol{a}_d - \boldsymbol{g} - \boldsymbol{f}_e(\boldsymbol{x}, \dot{\boldsymbol{x}}) \tag{3.2}$$

Since $\boldsymbol{z}$ must be of unit length, both u and $\boldsymbol{z}$ can be computed from $u\boldsymbol{z}$ by computing the magnitude and normalizing.

The angular velocity and angular acceleration are found by first computing the first and second time derivatives of $\boldsymbol{z}$.

Differentiating (3.2) in time results in

$$\dot{u}\boldsymbol{z} + u\dot{\boldsymbol{z}} = \boldsymbol{j}_d - \frac{\partial \boldsymbol{f}_e}{\partial \boldsymbol{x}}\dot{\boldsymbol{x}} - \frac{\partial \boldsymbol{f}_e}{\partial \dot{\boldsymbol{x}}}\ddot{\boldsymbol{x}} = \boldsymbol{j}_d^{\text{eff}} \tag{3.3}$$

Since $\boldsymbol{z}$ is of unit length, and it must remain so, it must be perpendicular to $\dot{\boldsymbol{z}}$. Thus taking a dot product of (3.3) with $\boldsymbol{z}$ allows us to find $\dot{u}$.

$$\dot{u} = \boldsymbol{j}_d^{\text{eff}\top}\boldsymbol{z} \tag{3.4}$$

Inserting $\dot{u}$ into (3.3) gives us $\dot{\boldsymbol{z}}$.

$$\dot{\boldsymbol{z}} = \frac{1}{u}\left(\boldsymbol{j}_d^{\text{eff}} - \left(\boldsymbol{j}_d^{\text{eff}\top}\boldsymbol{z}\right)\boldsymbol{z}\right) \tag{3.5}$$

The body angular velocity can be extracted from $\dot{\boldsymbol{z}}$ by first defining the body x and body y-axes using a desired vehicle yaw, then projecting $\dot{\boldsymbol{z}}$ onto those axes. See Section 2.2.2 for the details.

To find $\ddot{\boldsymbol{z}}$, we differentiate (3.3).

$$\ddot{u}\boldsymbol{z} + 2\dot{u}\dot{\boldsymbol{z}} + u\ddot{\boldsymbol{z}} = \boldsymbol{s}_d - \ddot{\boldsymbol{f}}_e(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{s}_d^{\text{eff}} \tag{3.6}$$

The second time derivative of the learned disturbance $\boldsymbol{f}_e$ is shown in (3.7). Note that the second partial derivative of the error model with respect to its vector inputs is a 3rd order tensor.

$$\begin{aligned}\ddot{\boldsymbol{f}}_e = &\left(\frac{\partial^2 \boldsymbol{f}_e}{\partial \boldsymbol{x}^2}\dot{\boldsymbol{x}} + \frac{\partial^2 \boldsymbol{f}_e}{\partial \boldsymbol{x}\dot{\boldsymbol{x}}}\ddot{\boldsymbol{x}}\right)\dot{\boldsymbol{x}} + \frac{\partial \boldsymbol{f}_e}{\partial \boldsymbol{x}}\ddot{\boldsymbol{x}} + \\ &\left(\frac{\partial^2 \boldsymbol{f}_e}{\partial \dot{\boldsymbol{x}}\boldsymbol{x}}\dot{\boldsymbol{x}} + \frac{\partial^2 \boldsymbol{f}_e}{\partial \dot{\boldsymbol{x}}^2}\ddot{\boldsymbol{x}}\right)\ddot{\boldsymbol{x}} + \frac{\partial \boldsymbol{f}_e}{\partial \dot{\boldsymbol{x}}}\dddot{\boldsymbol{x}}\end{aligned} \tag{3.7}$$

Noting that differentiating $\boldsymbol{z}^\top\dot{\boldsymbol{z}} = 0$ implies $\boldsymbol{z}^\top\ddot{\boldsymbol{z}} = -\dot{\boldsymbol{z}}^\top\dot{\boldsymbol{z}}$ and again taking a dot product with $\boldsymbol{z}$, we can compute $\ddot{u}$.

$$\ddot{u} = \boldsymbol{s}_d^{\text{eff}\top}\boldsymbol{z} + u\dot{\boldsymbol{z}}^\top\dot{\boldsymbol{z}} \tag{3.8}$$

Inserting $\ddot{u}$ into (3.6) gives us $\ddot{\boldsymbol{z}}$.

$$\ddot{\boldsymbol{z}} = \frac{1}{u}\left(\boldsymbol{s}_d^{\text{eff}} - \ddot{u}\boldsymbol{z} - 2\dot{u}\dot{\boldsymbol{z}}\right) \tag{3.9}$$

To compute the body angular acceleration from $\ddot{\boldsymbol{z}}$, we proceed as described in Section 2.2.2.

Note that the above equations for $\dot{\boldsymbol{z}}$ and $\ddot{\boldsymbol{z}}$ are similar to those derived in Section 2.2.2 with the difference that here, the first and second derivatives of the learned dynamics model are incorporated. In this way, the control inputs generated *anticipate* changes in the disturbance.

One practical issue that arises is that the vehicle acceleration, $\ddot{\boldsymbol{x}}$, and jerk, $\dddot{\boldsymbol{x}}$, are not readily available during operation. Computing them from odometry by taking finite-differences will introduce noise. To alleviate this in our experiments, we use the acceleration and jerk demanded by the trajectory, which are good approximations of the true vehicle acceleration and jerk when tracking error is low.

3.2.3 Input-dependent Error Compensation

In many cases, additive dynamics model errors are a function of the applied control input and vehicle orientation, in addition to the vehicle position and velocity. For example, if the mass of the vehicle is not accurately known (or alternatively, the actuators are not properly modeled), the disturbance will be a linear function of the applied acceleration. The input-dependent acceleration model is shown in (3.10).

$$u\boldsymbol{z} = \boldsymbol{a}_d - \boldsymbol{g} - \boldsymbol{f}_e(\boldsymbol{\eta}, \boldsymbol{u}) \tag{3.10}$$

Here, $\boldsymbol{\eta} = \begin{bmatrix}\boldsymbol{x} & \dot{\boldsymbol{x}}\end{bmatrix}^\top$ contains the vehicle position and velocity and $\boldsymbol{u} = u\boldsymbol{z}$.

Without assuming a particular form for the additive error term $\boldsymbol{f}_e$, it is not possible to solve for the required acceleration and orientation analytically. We must resort to solving the problem numerically. Interestingly however, once a solution for the acceleration and orientation is found, the rest of the control inputs can be found analytically in a method similar to the input-independent case described above.

We first rewrite the acceleration model as the functional equation $\boldsymbol{f}(\boldsymbol{u}, t) = 0$ that is only a function of $\boldsymbol{u}$ and time. We compute the time derivative of $\boldsymbol{u}$ by taking a derivative of the above equation and solving the resulting linear system.

$$\dot{\boldsymbol{f}}(\boldsymbol{u}, t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\dot{\boldsymbol{u}} + \frac{\partial \boldsymbol{f}}{\partial t} = 0 \tag{3.11}$$

$$\dot{\boldsymbol{u}} = -\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right)^{-1}\frac{\partial \boldsymbol{f}}{\partial t} \tag{3.12}$$

For our acceleration model, $\boldsymbol{f}(\boldsymbol{u}) = \boldsymbol{u} + \boldsymbol{g} + \boldsymbol{f}_e(\boldsymbol{\eta}, \boldsymbol{u}) - \boldsymbol{a}_d$. The necessary derivatives are shown in (3.13) and (3.14).

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} = \boldsymbol{I}_3 + \frac{\partial \boldsymbol{f}_e}{\partial \boldsymbol{u}} \tag{3.13}$$

$$\frac{\partial \boldsymbol{f}}{\partial t} = \frac{\partial \boldsymbol{f}_e}{\partial \boldsymbol{\eta}} \dot{\boldsymbol{\eta}} - \boldsymbol{j}_d \tag{3.14}$$

To find $\ddot{\boldsymbol{u}}$, we take a derivative of (3.11) and again solve the resulting linear system.

$$\ddot{\boldsymbol{f}}(\boldsymbol{u}, t) = \left(\frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{u}^2} \dot{\boldsymbol{u}} + 2 \frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{u} \partial t} \right) \dot{\boldsymbol{u}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \ddot{\boldsymbol{u}} + \frac{\partial^2 \boldsymbol{f}}{\partial t^2} \tag{3.15}$$

$$\ddot{\boldsymbol{u}} = \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \right)^{-1} \left(- \left(\frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{u}^2} \dot{\boldsymbol{u}} + 2 \frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{u} \partial t} \right) \dot{\boldsymbol{u}} - \frac{\partial^2 \boldsymbol{f}}{\partial t^2} \right) \tag{3.16}$$

The necessary derivatives for our acceleration model are shown in (3.17), (3.18), and (3.19).

$$\frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{u}^2} = \frac{\partial^2 \boldsymbol{f}_e}{\partial \boldsymbol{u}^2} \tag{3.17}$$

$$\frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{u} \partial t} = \frac{\partial^2 \boldsymbol{f}_e}{\partial \boldsymbol{u} \partial \boldsymbol{\eta}} \dot{\boldsymbol{\eta}} \tag{3.18}$$

$$\frac{\partial^2 \boldsymbol{f}}{\partial t^2} = \left(\frac{\partial^2 \boldsymbol{f}_e}{\partial \boldsymbol{\eta}^2} \dot{\boldsymbol{\eta}} \right) \dot{\boldsymbol{\eta}} + \frac{\partial \boldsymbol{f}_e}{\partial \boldsymbol{\eta}} \ddot{\boldsymbol{\eta}} - \boldsymbol{s}_d \tag{3.19}$$

To compute $\dot{\boldsymbol{z}}$ from $\dot{\boldsymbol{u}}$, we take a derivative of $\boldsymbol{u} = u\boldsymbol{z}$ and proceed as before, by projecting onto $\boldsymbol{z}$ and solving first for $\dot{u}$.

$$\dot{\boldsymbol{u}} = \dot{u}\boldsymbol{z} + u\dot{\boldsymbol{z}} \tag{3.20}$$

$$\dot{u} = \dot{\boldsymbol{u}}^\top \boldsymbol{z} \tag{3.21}$$

$$\dot{\boldsymbol{z}} = \frac{1}{u} \left(\dot{\boldsymbol{u}} - \dot{u}\boldsymbol{z} \right) \tag{3.22}$$

To compute $\ddot{\boldsymbol{z}}$ from $\ddot{\boldsymbol{u}}$, and the angular velocity and angular acceleration, we follow the same approach as for the input-independent case.

It should be noted that this approach requires the existence of a solution to $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} \dot{\boldsymbol{u}} = \frac{\partial \boldsymbol{f}}{\partial t}$ and the analogous equation for $\ddot{\boldsymbol{u}}$. Solutions will only fail to exist when the estimated disturbance model is strong enough to completely negate the acceleration imparted by $\boldsymbol{u}$. This may be a concern when learning a model from data, but in practice has not occurred in our experiments.

3.2.4 Model Learning

To estimate $\boldsymbol{f}_e$ from vehicle trajectory data, we fit a model to differences between the observed and the predicted acceleration at every time step. The observed acceleration is computed using finite-differences of the estimated vehicle velocity while the predicted acceleration is $u\boldsymbol{z} + \boldsymbol{g}$.

In principle, any regressive model whose derivatives are available can be used.

3.3 Experiments

We first evaluate the proposed approach on a simulated 2D quadrotor that is subjected to a series of input-independent and input-dependent disturbances. We then evaluate how the approach reduces tracking error on a quadrotor executing aggressive trajectories.

3.3.1 Simulation

The 2D planar quadrotor captures many of the important dynamics present in the 3D quadrotor. Namely, orientation and acceleration are coupled. In fact, the motion of a 3D quadrotor moving in a vertical plane, e.g. in a straight line trajectory, can essentially be described with the 2D quadrotor. As such, we believe a planar simulation is an appropriate testbed for our method.

The 2D quadrotor force model is shown in Fig. 3.2. The dynamics are shown in (3.23) – (3.25), where F is the applied body force and τ is the applied body acceleration. The mass, m, was set to $4.19\,\mathrm{kg}$, gravity g to $10.18\,\mathrm{m/s^2}$, and inertia I to $0.123\,\mathrm{kg\,m^2}$.

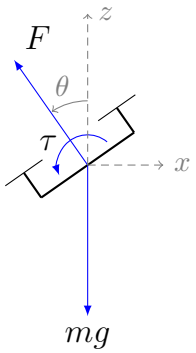


Figure 3.2: Force diagram of the 2D quadrotor used in the simulation experiment. F and τ are the control inputs.

$$m\ddot{x} = -F\sin(\theta) \tag{3.23}$$

$$m\ddot{z} = F\cos(\theta) - mg \tag{3.24}$$

$$I\ddot{\theta} = \tau \tag{3.25}$$

We subject the simulated quadrotor to disturbances selected from Table 3.2. Disturbance 1 is constant and emulates a fixed force field in the x direction, e.g. due to wind. It is not input-dependent and is not dynamic since it does not change along with the vehicle state. Disturbance 2 is velocity dependent and emulates drag in the x direction. Disturbance 3 depends on the vehicle angle and is thus input-dependent. Disturbance 4 is velocity dependent and emulates drag in the z direction. Disturbance 5 is a mass perturbation that adds a disturbance linear in the applied acceleration, which makes it input-dependent.

Table 3.2: Disturbances used in the 2D quadrotor simulation experiment

No.	**Effect**	**Input-dependent?**	**Dynamic?**
1	$\ddot{x}$ -= 4.1	no	no
2	$\ddot{x}$ -= $3.1\dot{x}$	no	yes
3	$\ddot{x}$ += $1.4\sin(\theta)$	yes	yes
4	$\ddot{z}$ -= $3.1\dot{z}$	no	yes
5	m += 2	yes	yes

The vehicle is given a desired trajectory that takes it from $x = 0$, $z = 0$, to $x = 1$, $z = 1$ in one second. The trajectories in x and z are both 7th-order polynomials that have the velocity, acceleration, and jerk equal to zero at each of their endpoints. This ensures that the trajectory starts and ends with the vehicle at rest, at an angle of zero, and with an angular velocity of zero. When Disturbance 1 is in effect, the vehicle's angle is initialized such that maintaining zero acceleration in z also maintains zero acceleration in x. This ensures that the trajectory can be perfectly followed with correct control inputs despite the constant acceleration disturbance in x. In all other cases, the vehicle state starts at 0.

We show x and z tracking error for the following feedforward input generation strategies with and without feedback.

FF1) No disturbance learning

FF2) Basic disturbance compensation (no disturbance dynamics)

FF3) Disturbance compensation w/ numerical optimization

FF4) Dist. comp. w/ disturbance dynamics (ours)

FF5) Dist. comp. w/ num. opt. and disturbance dynamics (ours)

FF1 uses the feedforward generation strategy as presented in Section 2.3.1 and does not do any regression for disturbance learning. **FF2** and **FF3** do not consider the dynamics of the disturbance; they compute the angular velocity and angular acceleration feedforward terms as in Section 2.2.2 while incorporating the learned disturbance in the acceleration model, (3.1). **FF4** is the proposed approach that deals with input-independent disturbances while **FF5** is the proposed approach that deals with input-dependent disturbances.

In this experiment, **FF3** and **FF5** solve (3.10) numerically using the modified Powell method root finder in SciPy [45]. The initial guess for the optimization is the solution from the previous timestep.

Position and angle feedback is provided by PD controllers with gains of 10 on position and velocity errors, 300 on angle errors, and 30 on angular velocity errors. The position PD controller output is added to the desired acceleration and the angle PD controller output is added to the desired angular acceleration.

In all simulation experiments, the feature vector used for linear regression of model errors is shown in (3.26). The features were hand selected to appropriately model the disturbances in Table 3.2.

$$\boldsymbol{\phi}(x, z, \theta, \dot{x}, \dot{z}, F) = \left[x, z, \dot{x}, \dot{z}, \sin(\theta), F\sin(\theta), F\cos(\theta), 1\right]^\top \tag{3.26}$$

The learned model is thus

$$\boldsymbol{f}_e(\boldsymbol{\eta}, \boldsymbol{u}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{\eta}, \boldsymbol{u}) \tag{3.27}$$

$\boldsymbol{w}$ is the result of regressing the projected input data $\boldsymbol{\phi}$ to the observed acceleration errors and minimizing least squared error. In this experiment, $\boldsymbol{w}$ is recomputed after every trajectory execution using data from all past executions. Results reported are on the 3rd run, since we found that only two regression steps were needed to converge to an accurate enough model. This is not surprising, as in this simulation there is no noise and the features used can appropriately reproduce the applied disturbances.

Each control configuration is subjected to the following set of disturbance combinations.

A) Disturbance 1

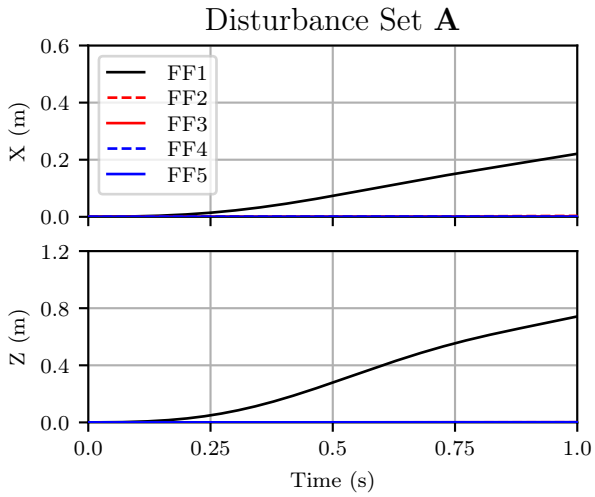
B) Disturbances 1, 2, and 4

C) Disturbances 3 and 5

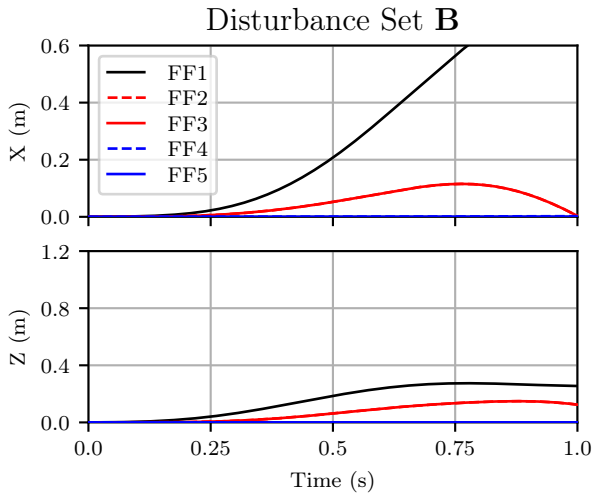
D) Disturbances 1, 2, 3, 4, and 5

Error plots for each of the four disturbance sets without feedback control are shown in Figs. 3.3a – 3.3d. Under only a constant disturbance (Fig. 3.3a), all disturbance compensation strategies work well, since the disturbance is neither input-dependent nor dynamic. When we introduce drag, a dynamic disturbance, in disturbance set **B** (Fig. 3.3b), only the approaches that compensate for disturbance dynamics, **FF4** and **FF5**, achieve low error. Although basic disturbance compensation as in **FF2** helps considerably, accounting for disturbance dynamics improves performance further. Since in disturbance set **B**, the disturbances are still input-independent, the use of numerical optimization to solve the acceleration model (3.1) has no effect.

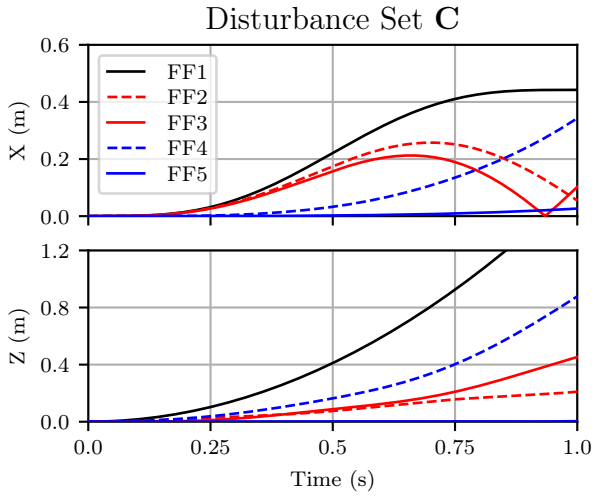
Under input-dependent disturbances, we see that **FF5** is the only approach that achieves low error. This is expected, as for both disturbance sets **C** and **D**, there are dynamic and input-dependent disturbances present.



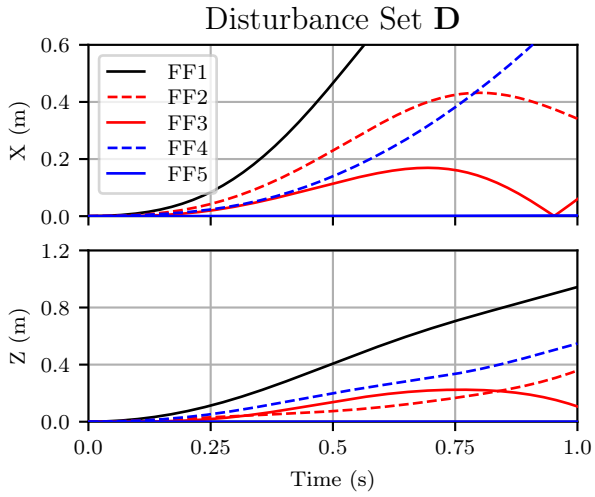
(a) Errors for disturbance set **A**, containing only a constant disturbance. All strategies that compensate for the disturbance perform well.



(b) Errors for disturbance set **B**, containing dynamic, but input-independent disturbances. **FF4** and **FF5**, which compensate for dynamic disturbances, perform the best.



(c) Errors for disturbance set **C**, containing dynamic and input-dependent disturbances. Only **FF5** performs well.



(d) Errors for disturbance set **D**, which contains all considered disturbances. **FF5** performs the best.

Figure 3.3: Absolute x and z position error for all five feedforward strategies without feedback control under each of the four disturbance sets.

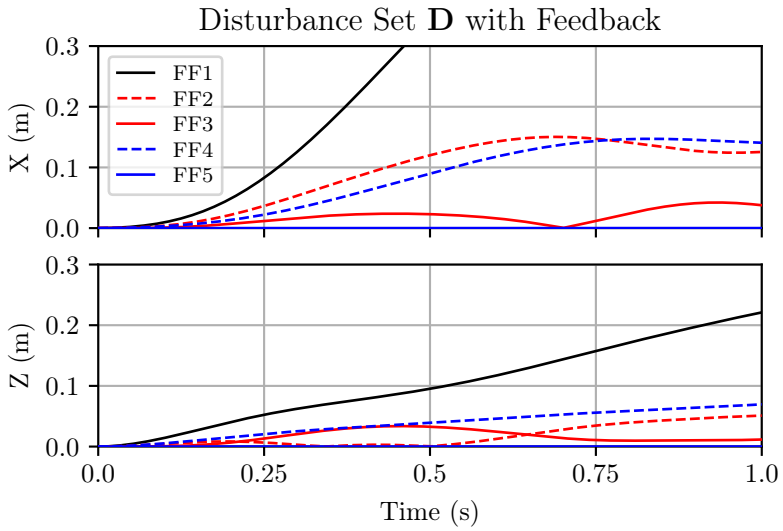


Figure 3.4: Absolute x and z position error for all five feedforward strategies *with* feedback control under disturbance set **D**. **FF5** outperforms all others.

Table 3.3: Maximum absolute position errors, in meters, for each control strategy without feedback in simulation

Dist. Set \ Strategy	**FF1**	**FF2**	**FF3**	**FF4**	**FF5**
A	0.774	**0.003**	**0.002**	**0.001**	**0.001**
B	0.879	0.182	0.182	**0.002**	**0.001**
C	1.703	0.297	0.465	0.943	**0.027**
D	1.580	0.495	0.278	0.933	**0.002**

Error plots for disturbance set **D** *with* feedback control are shown in Fig. 3.4. We see that although feedback can reduce the error, it is not enough to completely eliminate the error. **FF5** still outperforms the other methods, achieving nearly zero error in all trials.

The maximum absolute position errors over the trajectory for all tested configurations are listed in Tables 3.3 and 3.4.

3.3.2 Hardware

3.3.2.1 Platform & Setup

To validate the usefulness of dynamic disturbance compensation and input-dependent disturbance compensation, we compare the five aforementioned feedforward generation strategies, **FF1** through **FF5**, on a 750 g quadrotor while following aggressive trajectories. Figure 3.5 shows the hardware platform and Fig. 3.1 shows the robot while following aggressive circle and line trajectories.

Table 3.4: Maximum absolute position errors, in meters, for each control strategy with feedback in simulation

Dist. Set \ Strategy	**FF1**	**FF2**	**FF3**	**FF4**	**FF5**
A	0.253	**0.000**	**0.000**	**0.000**	**0.000**
B	0.373	0.044	0.044	**0.000**	**0.000**
C	0.274	0.063	0.058	0.070	**0.000**
D	0.589	0.153	0.043	0.159	**0.000**

Figure 3.5: The 750 g quadrotor used for the hardware experiments. Onboard computation is performed by an Odroid XU4 and the Pixhawk 1 Flight Controller.

Position, velocity, and yaw feedback is provided by a motion capture arena at 100 Hz, while pitch, roll, and angular velocity feedback is provided by a Pixhawk PX4 at 250 Hz. Feedback control is performed by a cascaded PD system following Section 2.3.1. The feedforward terms are as computed by **FF1** through **FF5**. **FF3** and **FF5** solve (3.10) numerically using the Newton-Raphson method. All control computation is performed onboard the vehicle's Odroid XU4 computer. The position control loop runs at 100 Hz and the attitude control loop runs at 200 Hz.

For the hardware experiments, we use three test trajectories: a 1.8 s straight line trajectory, a circle trajectory, and a figure 8 trajectory. The trajectories are designed to be near the limit of what the robot can feasibly track. Table 3.5 lists the three trajectories and their maximum absolute derivatives.

Table 3.5: The aggressive trajectories used to evaluate the proposed approach and their maximum derivatives

Traj.	$\boldsymbol{x}$ **(m)**	$\dot{\boldsymbol{x}}$ **(m/s)**	$\ddot{\boldsymbol{x}}$ **(m/s^2)**	$\boldsymbol{x}^{(3)}$ **(m/s^3)**	$\boldsymbol{x}^{(4)}$ **(m/s^4)**
Line	2.7	3.28	6.26	24.31	216
Circle	2.0	2.75	7.56	21.43	64.35
Figure 8	2.0	2.75	7.15	21.43	59.25

3.3.2.2 Model Learning

We use linear regression as the model learning strategy in the hardware experiment. Input data to the regression is a 6 dimensional vector consisting of the vehicle velocity and the commanded acceleration vector $\boldsymbol{u}$. The system starts with an uninitialized model and uses a few test trajectories per trial to regress to the acceleration error. The error model is then held fixed during the remaining trajectories used for error evaluation. Although in principle, the model can be updated incrementally, keeping it fixed allows for a fair comparison between the control strategies.

3.3.2.3 Results

For the line trajectory, each of **FF1**, **FF2**, **FF4**, and **FF5** is evaluated four times. The first four trajectories, run using **FF1**, are used to train the acceleration error model. An overlay of the vehicle executing the 2.7 m line trajectory can be seen in Fig. 3.1. Absolute errors along the trajectory and errors along the vertical axis for the line trajectory are shown in Fig. 3.6. **FF1** performs the worst, especially along the vertical axis, indicating that the robot is underestimating the control input required to maintain hover. **FF2** eliminates much of the error in the vertical axis, but still accumulates significant error along the trajectory, rising above 10 cm consistently. **FF4** and **FF5** provide on average a 30% reduction in the average absolute $x - y$ tracking error along the trajectory when compared to **FF2**. This indicates that taking disturbance dynamics into account can significantly improve tracking performance. This trajectory does not provide sufficient clarity to determine the impact of **FF5**, input-dependent disturbance compensation.

For the circle trajectory, all of the feedforward strategies are evaluated once, with **FF3** and **FF5** receiving two and four more trajectories respectively. An overlay of the vehicle executing the circle trajectory can be seen in Fig. 3.1. Fig. 3.7 shows the resulting error. As expected **FF1**, with no disturbance compensation, performs the worst. **FF2**, **FF4**, and **FF5** all perform similarly well, with **FF2** achieving slightly lower vertical error than the others. **FF3** performs slightly worse here than **FF2**, suggesting that the numerical routine may be failing to converge or that the input's dependence on the acceleration error has not been properly modeled.

Fig. 3.8 shows the error of **FF1**, **FF2**, **FF4**, and **FF5** along the figure 8 trajectory for one trial each. The improvement of **FF4** over **FF2** here is smaller than in the other trajectories, suggesting that dynamic disturbances have relatively less of an impact when following the figure 8, though more experimental trials are warranted to strengthen this claim.

3.4 Conclusion

We have presented a method that allows compensation of dynamic disturbances through evaluation of the derivatives of a learned model. We have shown in both simulation and hardware experiments that our dynamic disturbance compensation method improves

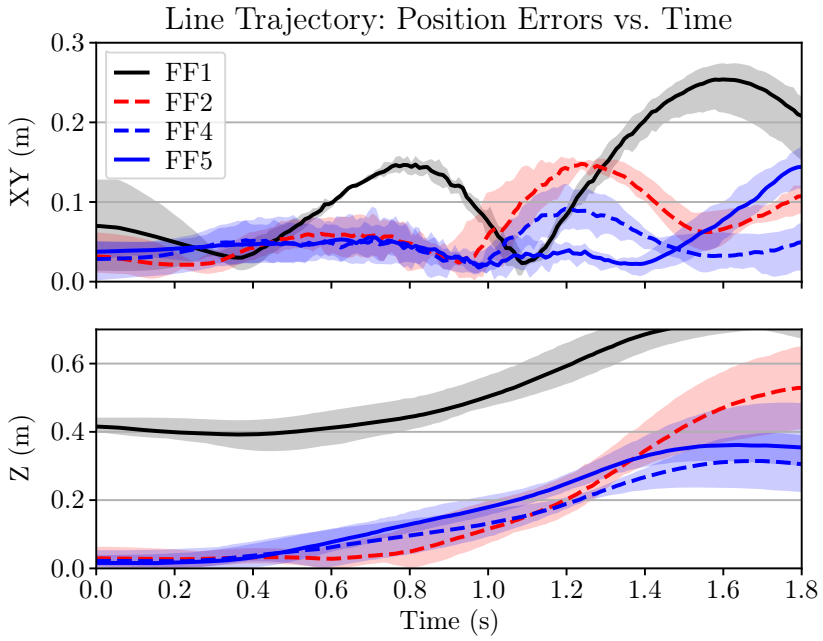


Figure 3.6: Average absolute errors during an aggressive straight line trajectory for four of the five control strategies. Shaded regions denote the minimum and maximum errors per timestep over four trials. Means (m) ($\pm$ std (m)) over the 4 trajectories of the average $|x-y|$ error for **FF1**, **FF2**, **FF4**, and **FF5** respectively are 0.120 $\pm$ 0.003, 0.067 $\pm$ 0.003, **0.047** $\pm$ 0.009, and **0.048** $\pm$ 0.10. Those for the average $|z|$ error are 0.525 $\pm$ 0.023, 0.173 $\pm$ 0.034, **0.142** $\pm$ 0.025, and 0.173 $\pm$ 0.030.

performance over traditional disturbance compensation. We have also shown the usefulness of input-dependent disturbance compensation in simulation and preliminary results on hardware. The versatility of the approach in a realistic robotics application has been verified through evaluation on three distinct test trajectories.

Future work will evaluate nonlinear regression techniques, such as ISSGPR, on hardware platforms, as well as consider regression techniques that explicitly optimize model derivative accuracy. An interesting avenue of future study is to analyze theoretically how the error model accuracy affects the performance of each of the feedforward generation strategies. This is discussed in Appendix D. Lastly, we apply this technique to the attitude dynamics of quadrotors, in order to fully compensate for vehicle disturbances and modeling errors, in Chapter 5.

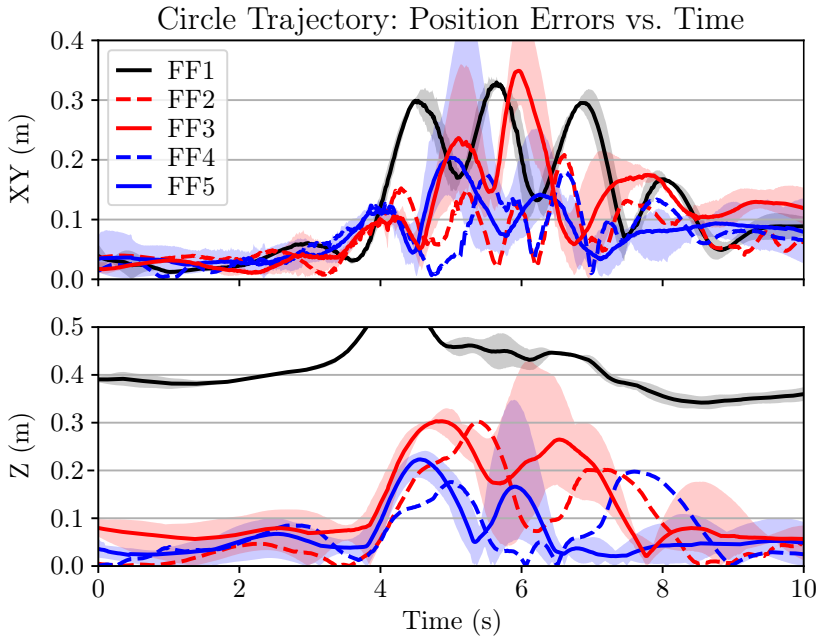


Figure 3.7: Average absolute errors during an aggressive circle trajectory for the five control strategies. Shaded regions denote the minimum and maximum errors per timestep. Avg. $|x - y|$ errors (m) for **FF1**, **FF2**, **FF3**, **FF4**, and **FF5** are 0.118, **0.071**, 0.103, **0.069**, and **0.076**, respectively, while avg. $|z|$ errors (m) are 0.41, 0.084, 0.122, **0.069**, and **0.066**, respectively.

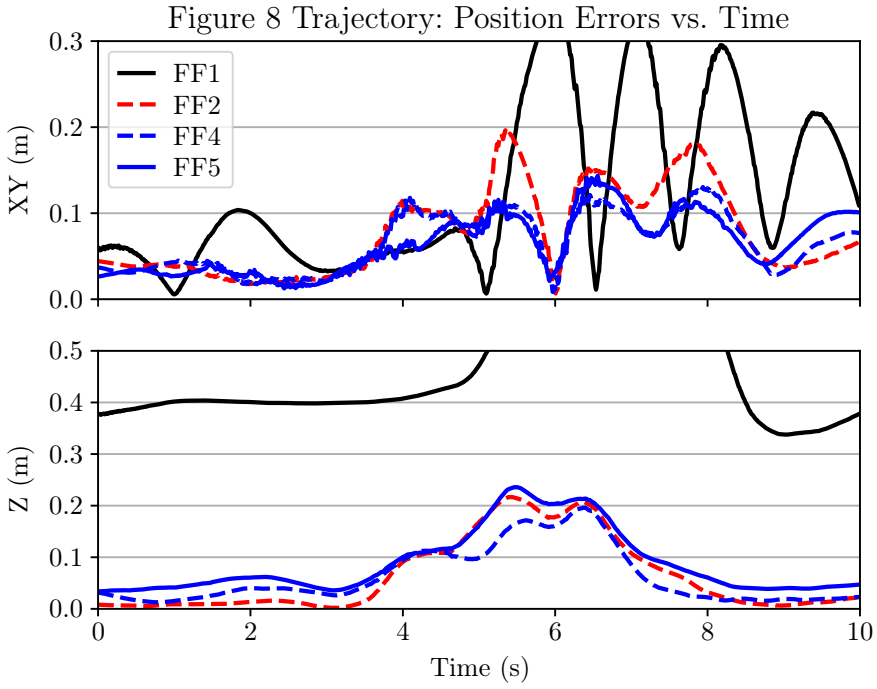


Figure 3.8: Errors during an aggressive figure 8 trajectory for four of the five control strategies. Avg. $|x-y|$ errors (m) for **FF1**, **FF2**, **FF4**, and **FF5** are 0.105, 0.076, **0.063**, and **0.059** respectively, while avg. $|z|$ errors (m) are 0.473, **0.064**, **0.064**, and 0.088, respectively.

CHAPTER 4

Feedback Linearization for Quadrotors with Model Learning and Delay Compensation

This chapter enhances the feedback linearization controller for quadrotors with a learned acceleration error model. Feedback linearization controllers are theoretically appealing but their performance suffers on real systems, where the true system does not match the known system model. We take a step in reducing these robustness issues by learning an acceleration error model, applying this model in the position controller, and further propagating it forward to the attitude controller. We show how this approach improves performance over the standard feedback linearization controller in the presence of unmodeled dynamics and repeatable external disturbances in both simulation and hardware experiments.

4.1 Introduction

In the previous chapter, we showed how model learning can be used to improve the trajectory tracking performance of the traditional cascaded quadrotor controller. In this chapter, we improve the quadrotor feedback linearization controller with a thrust delay model and a learned acceleration error model, analogous to the one for the cascaded controller.

Many practical aerial robotics applications use vision-based algorithms for state estimation [9, 84]. These algorithms can be brittle to environmental conditions and can generate drifting or discontinuous state estimates. Similarly, GPS-based state estimation can exhibit jumps when the signal quality is degraded, leading to jumps in tracking error. Unanticipated tracking error can also arise from gusts of wind and other external disturbances. Thus, the control algorithms used must be able to handle both poorly estimated dynamics and jumps in the state estimate.

By far the most common quadrotor feedback controller discussed in the literature and employed in practice is the cascaded controller. Such cascaded controllers often make the assumption that the inner attitude control loop can track references much faster than the outer position control loop. While approximately correct, this assumption introduces

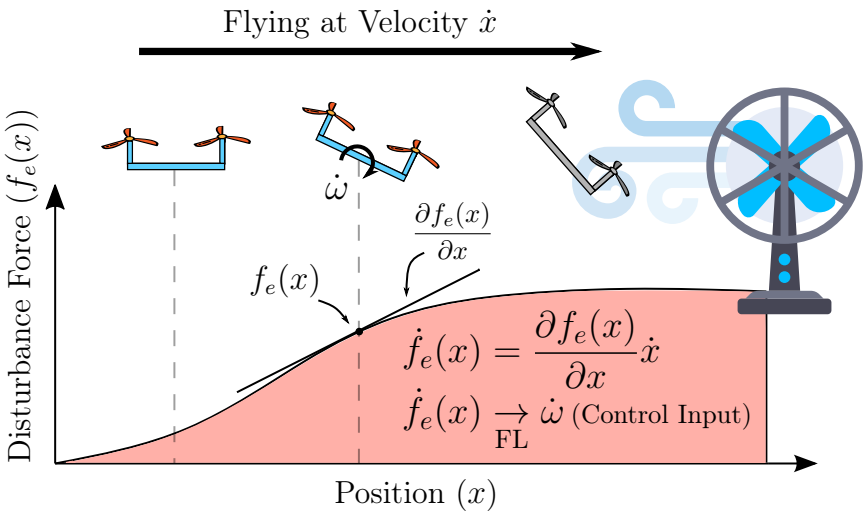


Figure 4.1: An illustration of the proposed acceleration disturbance compensation method. The vehicle is flying with velocity $\dot{x}$ head on into a strong wind field. The acceleration disturbance from the wind field is modeled as a function $f_e(x)$ that depends on the vehicle position among other quantities. Using the derivatives of the learned model, $\frac{\partial f_e(x)}{\partial x}$, and the vehicle velocity $\dot{x}$, the disturbance jerk $\dot{f}_e(x)$ can be computed. The disturbance jerk is then used in a feedback linearizaton (FL) controller that computes the angular acceleration control input $\dot{\omega}$ to mitigate the effects of the wind.

non-ideal characteristics in flight during aggressive or highly dynamic vehicle maneuvers, where the attitude tracking error is likely to be large. In contrast, feedback linearization exactly linearizes the quadrotor system, leading to a linear error response.

A linear error response is advantageous for several reasons. First linear controllers are easy to tune using well known traditional control techniques such as LQR and pole placement. Second, a linear error response ensures exponential convergence to the reference state. For example, for a quadrotor, a step input in position results in the vehicle taking the linear path connecting the start and end positions. Further, any jumps in the state estimate, or disturbances causing large transient error, will be handled linearly. The traditional cascaded quadrotor controllers do not have this property and their use may result in instability when the error is large.

However, feedback linearization is known to be brittle when the dynamical model of the system is not well known. In this chapter, we derive a feedback linearization controller for the quadrotor and show how it can be used with a learned acceleration error model to improve performance in the presence of unmodeled or mismodeled dynamics and external disturbances. We use incremental linear regression with a nonlinear feature space to rapidly learn a disturbance model over the state-space online. This model and its derivatives are then used in the feedback linearization transformations to account for and anticipate changes in the disturbance, as shown in Fig. 4.1.

In addition to correcting for unmodeled acceleration, we also correct for thrust control input delay. Specifically, we show that control input delay, as is always present in real

systems, can significantly impact the performance of the quadrotor feedback linearization controller, and is often not considered in quadrotor control system design. To mitigate this, we augment the quadrotor model with a delay in the thrust control input, and investigate the performance of the resulting feedback linearization controller in both simulation and hardware.

The contributions of this chapter are as follows:

1. A **simple, Euler-angle free, derivation of the feedback linearization controller** for the quadrotor with torque control inputs.
2. A **control input delay mitigation model** for feedback linearization that improves performance in hardware experiments.
3. A **learned acceleration model correction** that is used in the feedback linearization controller to fully compensate for the unmodeled dynamics and external disturbances. This is an extension of the work in [80] for the feedback linearization controller.
4. An **experimental analysis** of the effectiveness of the linear acceleration error compensation and control input delay mitigation model in the feedback linearization controller.

The contents of this chapter first appeared in [82].

4.2 Related Works

Feedback linearization [44] has been applied to quadrotors since at least Mistler, Benallegue, and M'Sirdi [64], although the same technique has been used much earlier in the context of simplified helicopter models [47]. Chang and Eun [17] extends feedback linearization to preserve the positive thrust constraint using a modified dynamic extension and apply feedback linearization chartwise to avoid singularities. Lee, Jin Kim, and Sastry [49] compares feedback linearization to sliding mode control for a quadrotor in the context of robustness, although a small-angle approximation is used.

Some existing works that present feedback linearization for quadrotors do not present hardware results [3, 17], do not compare to cascaded approaches [2, 73, 91], or only use angular velocity, not angular acceleration as control inputs [2, 73]. In this chapter, we provide theoretic, simulation, and hardware comparisons to cascaded approaches, and present the final control law with angular acceleration as a control input. We then extend the feedback linearization approach to incorporate a delay in the thrust control input. Further, our derivation does not use Euler angles or quaternions and instead uses the simpler quantity of the body z-axis to handle the coupling between translational and rotational dynamics.

Although differential flatness and feedback linearization are closely related, differential flatness, as used in the context of feedforward linearization [39], only linearizes the system at the *desired* state and corresponding input, whereas feedback linearization linearizes the

system at the *current* state and input. Thus, differential flatness is unable to properly linearize the system when there is large tracking error and control performance may suffer.

Faessler, Falanga, and Scaramuzza [21] shows that modeling the torque control input delay improves disturbance rejection, but does not consider the delay in the thrust control input and [91] uses Pseudo Control Hedging (PCH) to mitigate delays in the control inputs, among other unmodeled effects.

Model learning for feedback linearizing controllers has been explored for a long time. As for general model learning for controls, the related works in this area can be divided into those that learn forward models and those that learn inverse models. Yeşildirek and Lewis [94] uses neural networks to learn the forward dynamics model, after which it is used in feedback linearization and [87] learns a forward model using a Gaussian Process (GP), whose uncertainty estimates are used to prove a convergence guarantee. Spitzer and Michael [80] learns a forward model and uses its derivatives to compensate for the disturbance dynamics. On the other hand, [92] and [36] learn an inverse model for feedback linearization.

While model learning for quadrotors has been studied extensively, few have studied model learning for the feedback linearizing controller. In this chapter, we learn a forward model for the quadrotor feedback linearization controller and also consider the delay in the thrust control input.

4.3 Method

Section 4.3.1 presents a simple, Euler-angle free derivation of the feedback linearization controller for the standard quadrotor model with torque control inputs. Then, Section 4.3.2 augments the quadrotor model with a delay in the thrust control input and an additive linear acceleration correction, and derives the corresponding feedback linearization controller.

4.3.1 Feedback Linearization with Dynamic Extension

Feedback linearization [44] is a technique to a cancel out the nonlinearities of a system so that it can be controlled using a linear controller. Two classes of feedback linearization are *full state linearization* and *input-output linearization*. Full state linearization seeks to render the entire state controllable via a linear system while input-output linearization seeks to linearize the relationship between the control input and some specified output. For the quadrotor, we will use input-output linearization to linearize the relationship between the control inputs and the vehicle position.

Feedback linearization of multi-input and multi-output systems is contingent on the existence of a well-defined *vector* relative degree. The relative degree is effectively, for each dimension of the output vector, the number of times it should be differentiated such

that all the control inputs that affect it appear[1]. When different control inputs affect different order derivatives of a particular output, the system may fail to be input-output linearizable. This is in fact the case with the quadrotor model (2.6).

To ensure that the system has a well-defined relative degree, we can delay the thrust input u two times, so that it acts on the fourth derivative of position, as does the angular acceleration α. Because delaying the thrust requires integration of a virtual control input $\ddot{u}$ to compute the original system control input u, this converts the feedback linearizing control law from a static feedback into a dynamic feedback. This technique is known as *dynamic extension* of the original nonlinear system.

The augmented dynamic model, with two new virtual states $\xi_1 = u$ and $\xi_2 = \dot{u}$, and virtual control input $\nu = \ddot{u}$, is shown below. We also take as output the vehicle position p and yaw ψ.

$$\boldsymbol{x} = \begin{bmatrix} p \\ v \\ R \\ \omega \\ \xi_1 \\ \xi_2 \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} \nu \\ \alpha \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} p \\ \psi \end{bmatrix} \tag{4.1}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} v \\ \xi_1 R e_3 + g \\ [\omega]_\times R \\ \alpha \\ \xi_2 \\ \nu \end{bmatrix} \tag{4.2}$$

Placing (4.2) into control affine form yields

$$\dot{\boldsymbol{x}} = \begin{bmatrix} v \\ \xi_1 R e_3 + g \\ [\omega]_\times R \\ 0 \\ \xi_2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & I_{3\times 3} \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \boldsymbol{u}. \tag{4.3}$$

Now, we differentiate the position four times.

$$p^{(1)} = v \tag{4.4}$$
$$p^{(2)} = \xi_1 R e_3 + g = \xi_1 z + g \tag{4.5}$$
$$p^{(3)} = \xi_2 z + \xi_1 \dot{z} \tag{4.6}$$
$$= \xi_2 z + \xi_1 (\omega \times z) \tag{4.7}$$
$$p^{(4)} = \nu z + 2\xi_2 \dot{z} + \xi_1 \ddot{z} \tag{4.8}$$
$$= \nu z + 2\xi_2 (\omega \times z) + \xi_1 \left(\alpha \times z + \omega \times (\omega \times z)\right) \tag{4.9}$$

[1]For the precise technical definition, see Section 5.1 of [44].

Note that the control inputs ν and α both appear only in the fourth time derivative and not in any of the lower order derivatives. This is the key property enabled by delaying u.

For the yaw output ψ, we differentiate twice until the control input α, the angular acceleration, appears. To do this, refer back to (2.30), (2.31), (2.24), and (2.25).

$$\dot{\psi} = \frac{-x_2\dot{x}_1 + x_1\dot{x}_2}{x_1^2 + x_2^2} \tag{4.10}$$

$$\ddot{\psi} = \frac{-x_2\ddot{x}_1 + x_1\ddot{x}_2 - 2\dot{\psi}(x_1\dot{x}_1 + x_2\dot{x}_2)}{x_1^2 + x_2^2} \tag{4.11}$$

$$\dot{x} = \omega \times x \tag{4.12}$$

$$\ddot{x} = \alpha \times x + \omega \times \dot{x} \tag{4.13}$$

We can construct the 4×4 "relative degree matrix" $A(\boldsymbol{x})$ and verify it is invertible for a vector relative degree of $\{4, 4, 4, 2\}$.

$$A(\boldsymbol{x}) = \begin{bmatrix} L_{g_\nu} L_f^3 p & L_{g_\alpha} L_f^3 p \\ L_{g_\nu} L_f \psi & L_{g_\alpha} L_f \psi \end{bmatrix} \tag{4.14}$$

$$= \begin{bmatrix} \frac{\partial p^{(3)}}{\partial \xi_2} & \frac{\partial p^{(3)}}{\partial \omega} \\ \frac{\partial \dot{\psi}}{\partial \xi_2} & \frac{\partial \dot{\psi}}{\partial \omega} \end{bmatrix} \tag{4.15}$$

$$= \begin{bmatrix} z & & -\xi_1[z]_\times & \\ 0 & -\frac{x_1 x_3}{x_1^2+x_2^2} & -\frac{x_2 x_3}{x_1^2+x_2^2} & x_1^2 + x_2^2 \end{bmatrix} \tag{4.16}$$

We see that for $\xi_1 \neq 0$ and $x \neq e_3$, $A(\boldsymbol{x})$ is invertible, thus showing that (2.6) has a vector relative degree of $\{4, 4, 4, 2\}$.

4.3.2 Feedback Linearization with Thrust Delay Model and Acceleration Error Correction

In order to account for the vehicle's rotor inertia, we add a first order exponential delay model to the vehicle thrust, u.

$$\dot{u} = -\tau_u(u - u^{\text{des}}) \tag{4.17}$$

Equation (4.17) provides a good approximation to the delay in the thrust produced by the vehicle if the relationship between force and rotor speed is linear, as rotor speeds have been experimentally shown to display first order exponential delay behavior [62]. To keep the resulting feedback linearization controller simple, we do not model a delay in the angular acceleration control input. The augmented system with the thrust control input

delay component, and the thrust u added to the state vector, is shown in (4.18).

$$\boldsymbol{x} = \begin{bmatrix} p \\ v \\ R \\ \omega \\ u \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} u^{\text{des}} \\ \alpha \end{bmatrix}, \quad \dot{\boldsymbol{x}} = \begin{bmatrix} v \\ uz + g + f_e(\xi) \\ [\omega]_\times R \\ \alpha \\ -\tau_u(u - u^{\text{des}}) \end{bmatrix} \tag{4.18}$$

Here, u^{des} is the *desired* linear acceleration along the body z-axis, and is closely tracked by the *estimated* linear acceleration, u, according to a first order exponential delay model with time constant τ_u, given by (4.17).

We apply feedback linearization with dynamic extension as in Section 4.3.1 to the above quadrotor system model (4.18), with the objective of controlling the system position p and yaw ψ. The relative degree of the quadrotor system (4.18) is not defined as written, since u^{des} first appears in the third derivative of the vehicle position, while α first appears in the fourth derivative, through $\ddot{z}$.

As shown above, in the traditional quadrotor case, we need to delay the body thrust u twice, so that it appears in concert with the angular acceleration. For the augmented system model we consider, which models thrust delay, we only need to delay the linear acceleration input once. For that, we move u^{des} into the state and replace it with its derivative, $\dot{u}^{\text{des}}$.

Now to solve for the control inputs $\dot{u}^{\text{des}}$ and α, first differentiate the position output four times.

$$\dot{p} = v \tag{4.19}$$

$$\ddot{p} \equiv a = uz + g + f_e(\xi) \tag{4.20}$$

$$p^{(3)} \equiv j = \dot{u}z + u\dot{z} + \dot{f}_e(\xi) \tag{4.21}$$

$$p^{(4)} \equiv s = \ddot{u}z + 2\dot{u}\dot{z} + u\ddot{z} + \ddot{f}_e(\xi) \tag{4.22}$$

4.3.2.1 Computing Thrust Control Input $\boldsymbol{u}^{\boldsymbol{des}}$

We can project the snap to the body z-axis to solve for $\ddot{u}$, noting that $z^\top \dot{z} = 0$ and $z^\top \ddot{z} = -\dot{z}^\top \dot{z}$.

$$\ddot{u} = s^\top z + u\dot{z}^\top \dot{z} - \ddot{f}_e(\xi)^\top z \tag{4.23}$$

$$= \left(s - \ddot{f}_e(\xi)\right)^\top z + u\dot{z}^\top \dot{z} \tag{4.24}$$

We also note that, according to the thrust delay model (4.17),

$$\ddot{u} = -\tau_u(\dot{u} - \dot{u}^{\text{des}}). \tag{4.25}$$

Combining (4.24), (4.25), and (4.17), we have

$$\dot{u}^{\text{des}} = \frac{(s - \ddot{f}_e(\xi))^\top z + u\dot{z}^\top \dot{z}}{\tau_c} - \tau_c(u - u^{\text{des}}) \tag{4.26}$$

The thrust control input u^{des} and the estimated thrust u are computed from (4.26) and (4.17) respectively, using numerical integration.

4.3.2.2 Computing Angular Acceleration Control Input α

From (4.22), we can solve for $\ddot{z}$.

$$\ddot{z} = \frac{1}{u}\left(s - \ddot{f}_e(\xi) - \ddot{u}z - 2\dot{u}\dot{z}\right) \tag{4.27}$$

Noting that $\dot{z} = \omega \times z$ and $\ddot{z} = \alpha \times z + \omega \times \dot{z}$, the vector triple product gives

$$z \times \ddot{z} = z \times (\alpha \times z) + z \times (\omega \times (\omega \times z)) \tag{4.28}$$

$$= \alpha - (\alpha^\top z)z + (\omega^\top z)z \times \omega \tag{4.29}$$

Thus the angular acceleration less that along the z axis, or the angular acceleration along the body x and body y axes, denoted α^{xy}, can be found using

$$\alpha^{xy} = z \times \ddot{z} - (\omega^\top z)z \times \omega \tag{4.30}$$

$$= \frac{1}{u}\left(z \times (s - \ddot{f}_e(\xi)) - 2\dot{u}(z \times \dot{z})\right) + (\omega^\top z)\omega \times z \tag{4.31}$$

$$= \frac{1}{u}\left(z \times (s - \ddot{f}_e(\xi)) - 2\dot{u}\omega^{xy}\right) + (\omega^\top z)\omega \times z \tag{4.32}$$

The component of the angular acceleration along the body z-axis is undetermined by the position and its derivatives and thus can be chosen to follow a desired yaw trajectory. This is computed as in (2.46) with $\ddot{x}$ computed using (2.60).

$$\alpha^z = \mathcal{T}_\alpha^{-1}(u, \mathcal{T}_a(z,u), \mathcal{T}_j(z,u,\dot{u},\omega), s_{\text{des}}, \mathcal{T}_\psi(x), \mathcal{T}_{\dot{\psi}}(x,\omega), \ddot{\psi}_{\text{des}}) \tag{4.33}$$

Eqs. (4.26) and (4.32) together provide the control inputs that are needed to achieve snap s. Snap is typically the result of a linear feedback controller that tracks a desired position, velocity, acceleration, and jerk, as shown in (4.34), where $x_{\text{err}} = x - x_{\text{ref}}$.

$$s_{\text{des}} = -K_1 p_{\text{err}} - K_2 v_{\text{err}} - K_3 a_{\text{err}} - K_4 j_{\text{err}} + s_{\text{ref}} \tag{4.34}$$

Gains for this controller, $K_i \in \mathbb{R}^{3\times3}$, can be chosen by any linear control technique, such as LQR and pole placement. When the acceleration and jerk are not readily available, as is the case with most state estimators, Eqs. (4.20) and (4.21) can be used for feedback. Further, to support implementation on an embedded platform, the disturbance model f_e and its derivatives can be evaluated on a ground control station. In that case, let $s^{\text{ff}} = -K_1 p_{\text{err}} - K_2 v_{\text{err}} + K_3(a_{\text{ref}} - f_e(\xi)) + K_4(j_{\text{ref}} - \dot{f}_e(\xi)) + s_{\text{ref}} - \ddot{f}_e(\xi)$ and the desired snap becomes

$$s_{\text{des}} = s^{\text{ff}} - K_3(uz + g) - K_4(\dot{u}z + u\dot{z}) + \ddot{f}_e(\xi) \tag{4.35}$$

We can combine (4.35) with (4.32) to further simplify the expression for angular acceleration along the body x and y axes.

$$\alpha^{xy} = \frac{1}{u}\left(z \times s^{\text{ff}} - K_3 z \times g - 2\dot{u}\omega^{xy}\right) - K_4\omega^{xy} - (\omega^\top z)z \times \omega \tag{4.36}$$

To obtain the control input α^{xy} in the body frame, we left-multiply (4.36) by $R_{\mathcal{W}}^{\top}$ to obtain.

$$\alpha_{\mathcal{B}}^{xy} = \frac{1}{u}\left(e_3 \times s_{\mathcal{B}}^{\text{ff}} - K_3 e_3 \times g_{\mathcal{B}} - 2\dot{u}\omega_{\mathcal{B}}^{xy}\right) - K_4\omega_{\mathcal{B}}^{xy} - \omega_{\mathcal{B}}^{z} e_3 \times \omega_{\mathcal{B}} \tag{4.37}$$

We similarly transform the computation of $\dot{u}_{\text{des}}$ into the body frame, after combining (4.35) with (4.26), for a simpler implementation on an embedded platform.

$$\dot{u}^{\text{des}} = \frac{s_{\mathcal{B}}^{\text{ff}\top} e_3 - K_3(u + g_{\mathcal{B}}^{\top} e_3) - K_4\dot{u} + u||\omega_{\mathcal{B}}^{xy}||^2}{\tau_c} - \tau_c(u - u^{\text{des}}) \tag{4.38}$$

Note that implementing (4.37) and (4.38) only requires knowing the gravity vector, the angular velocity, and s^{ff} in the body frame. The gravity vector in the body frame is directly measured by an IMU that is at the center of the body frame and the angular velocity in the body frame is directly measured by a gyroscope that is anywhere on the rigid body. Further, this controller requires no trigonometric operations, such as sin, cos, tan, or their inverses, and can run very quickly on an embedded system. Notably, the control law does not depend on the yaw trajectory of the vehicle.[2]

To compute the desired yaw acceleration $\ddot{\psi}_{\text{des}}$ needed to implement (4.33), a linear PD controller in the flat yaw space is used.

$$\ddot{\psi}_{\text{des}} = -k_{p,\psi}(\mathcal{T}_{\psi}(x) \ominus \psi_{\text{ref}}) - k_{d,\psi}(\mathcal{T}_{\dot{\psi}}(x,\omega) - \dot{\psi}_{\text{ref}}) + \ddot{\psi}_{\text{ref}} \tag{4.39}$$

(4.39) makes use of $\mathcal{T}_{\psi}$ defined in (2.29) and $\mathcal{T}_{\dot{\psi}}$ defined in (2.30). Here, $\ominus$ is a difference function defined on $\mathcal{S}^1$ and ensures the shortest path around the circle is taken.

More advanced difference functions on $\mathcal{S}^1$ that take into account the angular velocity can also be used. This can be useful if large angular velocities around z are to be encountered by enabling the planning of a smooth "unwinding" of the state space without an oscillating angle feedback term. The angle feedback component can "plan" to bring the yaw to an angle several revolutions away from the current angle, in order to accommodate the gradual reduction in angular velocity of a rapidly spinning robot. See [8] for more details about the challenges that spaces such as $\mathcal{S}^1$ pose for continuous and stable control.

4.3.3 Gain Matching

While the cascaded control architecture uses gains on position, velocity, orientation, and angular velocity, the feedback linearization controller uses gains on position, velocity, acceleration, and jerk. When comparing the control methods experimentally, it is desirable that the first order response is the same between both. Here we linearize both control methods at hover and provide relationships between cascaded control gains and feedback linearization gains.

[2] Although the full orientation is required to compute s^{ff} in the body frame if the position feedback and trajectory planning are with respect to the fixed frame.

Cascaded Controller The cascaded feedback controller from (2.68) is shown below for a zero trajectory reference at hover ($\omega_{\text{des}} = \alpha_{\text{des}} = 0$).

$$\alpha_{\text{casc}} = -K_R^{\text{casc}} e_R(R, R_{\text{des}}) - K_\omega^{\text{casc}} \omega \tag{4.40}$$

We choose the thrust vector error function (2.78) from Section 2.3.2.1 and recall a_{des} from (2.62), R_{des} from (2.65), and $z_{\text{des}} = \mathcal{T}_z^{-1}(a_{\text{des}})$ from (2.33).

$$e_R(R, R_{\text{des}}) = z_{\text{des}} \times z \tag{4.41}$$

$$a_{\text{des}} = -K_p^{\text{casc}} p - K_v^{\text{casc}} v \tag{4.42}$$

$$z_{\text{des}} = \frac{a_{\text{des}} - g}{||a_{\text{des}} - g||} \tag{4.43}$$

To compute the gain of the angular acceleration control input with respect to the position, we take the partial derivative of (4.40).

$$\frac{\partial \alpha_{\text{casc}}}{\partial p} = -K_R^{\text{casc}} \frac{\partial e_R}{\partial z_{\text{des}}} \frac{\partial z_{\text{des}}}{\partial a_{\text{des}}} \frac{\partial a_{\text{des}}}{\partial p} \tag{4.44}$$

The derivative of a normalized vector $\hat{n} = \frac{n}{\sqrt{n^\top n}} \in \mathbb{R}^3$, such as z_{des}, can be derived using the inverse vector triple product.

$$\begin{aligned}
D(\hat{n})x &= \frac{||n|| I_3 - n \left(\frac{1}{2}\right) \frac{1}{||n||}(2) n^\top}{||n||^2} x \\
D(\hat{n})x &= \frac{I_3 - \hat{n}\hat{n}^\top}{||n||} x \\
D(\hat{n})x &= \frac{x - \hat{n}\hat{n}^\top x}{||n||} \\
D(\hat{n})x &= \frac{(\hat{n}^\top \hat{n})x - (\hat{n}^\top x)\hat{n}}{||n||} \\
D(\hat{n})x &= \frac{\hat{n} \times (x \times \hat{n})}{||n||} \\
D(\hat{n})x &= -\frac{\hat{n} \times (\hat{n} \times x)}{||n||} \\
D(\hat{n})x &= -\frac{[\hat{n}]_\times^2}{||n||} x \\
\implies D(\hat{n}) &= -\frac{[\hat{n}]_\times^2}{||n||}
\end{aligned} \tag{4.45}$$

Now expanding (4.44),

$$\frac{\partial \alpha_{\text{casc}}}{\partial p} = -K_R^{\text{casc}} (-[z]_\times) \left(-\frac{[z_{\text{des}}]_\times^2}{||a_{\text{des}} - g||} \right) (-K_p^{\text{casc}}). \tag{4.46}$$

Evaluating the above at hover with $p = 0$, $v = 0$, and thus $z_{\text{des}} = e_3$, and $z = e_3$ results in

$$\left.\frac{\partial \alpha_{\text{casc}}}{\partial p}\right|_{\text{hover}} = \frac{1}{||g||} K_R^{\text{casc}} [e_3]_\times^3 K_p^{\text{casc}} \tag{4.47}$$

$$= -\frac{1}{||g||} K_R^{\text{casc}} [e_3]_\times K_p^{\text{casc}}. \tag{4.48}$$

Analogously for velocity, we have

$$\left.\frac{\partial \alpha_{\text{casc}}}{\partial v}\right|_{\text{hover}} = -\frac{1}{||g||} K_R^{\text{casc}} [e_3]_\times K_v^{\text{casc}}. \tag{4.49}$$

For tilt z, we have

$$\frac{\partial \alpha_{\text{casc}}}{\partial z} = -K_R^{\text{casc}} \frac{\partial e_R}{\partial z} \tag{4.50}$$

$$= -K_R^{\text{casc}} [z_{\text{des}}]_\times \tag{4.51}$$

$$\left.\frac{\partial \alpha_{\text{casc}}}{\partial z}\right|_{\text{hover}} = -K_R^{\text{casc}} [e_3]_\times \tag{4.52}$$

and for angular velocity ω,

$$\frac{\partial \alpha_{\text{casc}}}{\partial \omega} = -K_\omega^{\text{casc}}. \tag{4.53}$$

Feedback Linearization To compute the linearization of the feedback linearization controller, differentiate (4.36).

$$\frac{\partial \alpha_{\text{fblin}}}{\partial p} = \frac{1}{u} [z]_\times \frac{\partial s^{\text{ff}}}{\partial p} \tag{4.54}$$

$$= \frac{1}{u} [z]_\times (-K_1^{\text{fbl}}) \tag{4.55}$$

At hover, $u = ||g||$, and $z = e_3$.

$$\left.\frac{\partial \alpha_{\text{fblin}}}{\partial p}\right|_{\text{hover}} = -\frac{1}{||g||} [e_3]_\times K_1^{\text{fbl}} \tag{4.56}$$

Analogously, the gain for velocity is

$$\left.\frac{\partial \alpha_{\text{fblin}}}{\partial v}\right|_{\text{hover}} = -\frac{1}{||g||} [e_3]_\times K_2^{\text{fbl}}. \tag{4.57}$$

Differentiating with respect to z, we have

$$\frac{\partial \alpha_{\text{fblin}}}{\partial z} = \frac{1}{u} \left(-[s^{\text{ff}}]_\times + K_3^{\text{fbl}} [g]_\times\right) + \omega^\top z [\omega]_\times - \omega \left(z \times \omega\right)^\top \tag{4.58}$$

At hover, $s^{\text{ff}} = 0$, $u = ||g||$, and $\omega = 0$, and

$$\left.\frac{\partial \alpha_{\text{fblin}}}{\partial z}\right|_{\text{hover}} = -K_3^{\text{fbl}}[e_3]_\times. \tag{4.59}$$

For angular velocity, we have

$$\frac{\partial \alpha_{\text{fblin}}}{\partial \omega} = -\frac{2\dot{u}}{u} I_3 - K_4^{\text{fbl}} - (\omega^\top z)[z]_\times - z(z \times \omega)^\top, \tag{4.60}$$

which at hover is

$$\left.\frac{\partial \alpha_{\text{fblin}}}{\partial \omega}\right|_{\text{hover}} = -K_4^{\text{fbl}}. \tag{4.61}$$

Gain Relationships Now equating (4.48) with (4.56), (4.49) with (4.57), (4.52) with (4.59), and (4.53) with (4.61), we get the following gain relationships.

$$K_1^{\text{fbl}} = K_R^{\text{casc}} K_p^{\text{casc}} \tag{4.62a}$$
$$K_2^{\text{fbl}} = K_R^{\text{casc}} K_v^{\text{casc}} \tag{4.62b}$$
$$K_3^{\text{fbl}} = K_R^{\text{casc}} \tag{4.62c}$$
$$K_4^{\text{fbl}} = K_\omega^{\text{casc}} \tag{4.62d}$$

Since the feedback linearization controller controls the first derivative of the thrust, the first order response cannot be matched to that of the cascaded controller, which controls the thrust directly.

4.3.4 Acceleration Model Learning

To estimate $f_e(\xi)$ from vehicle trajectory data, we fit a model to differences between the observed and the predicted acceleration at every time step. We use a subset of the state as input to the model: the position p and velocity v. Although this limits the types of disturbances that can be accounted to those that are functions of vehicle position and velocity, and notably not disturbances that are functions of the control input, this is a suitable choice to highlight the proposed control algorithm, given the experiments performed in Section 4.4. The observed acceleration is computed using finite-differences of the estimated vehicle velocity sampled in intervals of ΔT, while the predicted acceleration is $uz + g$. Thus, the training examples used to fit the model are pairs of data points (ξ_t, y_t), defined below.

$$\xi_t = \begin{pmatrix} p_t & v_t \end{pmatrix}^\top \tag{4.63}$$
$$y_t = \frac{1}{\Delta T}(v_t - v_{t-1}) - (u_{t-1} z_{t-1} + g) \tag{4.64}$$

For the experiments in this chapter, we use Incremental Sparse Spectrum Gaussian Process Regression (ISSGPR) [33] as the regression strategy. ISSGPR projects the input

data into a nonlinear feature space defined by sinusoids with random frequencies [74] and then applies regularized linear regression incrementally using rank-one matrix updates. Let d_x be the dimension of the input vector ξ and let N be the number of random frequencies chosen. The random frequencies $\Omega \in \mathbb{R}^{N \times d_x}$ can be generated by multiplying an N by d_x matrix of univariate Gaussians by $\text{diag}(M)$, where each entry in $M \in \mathbb{R}^{d_x}$ is the inverse of the characteristic length scale of the corresponding input dimension. The length scales allow for the adjustment of the relative importance of each input dimension and are thus hyperparameters that need to be adapted to the application.

The output of the model is shown below. Here, $W \in R^{2N \times 3}$ is the matrix of weights learned by linear regression, where each column corresponds to a dimension in the output.

$$f_e(\xi) = \frac{1}{\sqrt{N}} W^\top \begin{pmatrix} \cos(\Omega\xi) \\ \sin(\Omega\xi) \end{pmatrix} \tag{4.65}$$

The derivatives of f_e with respect to the input ξ, which are required to implement the proposed control strategy, are shown below. Here, ξ_k is the k'th entry of ξ, Ω_k is the k'th column of Ω, and $\odot$ is the Hadamard, or component-wise, product.

$$\frac{\partial f_e(\xi)}{\partial \xi_i} = \frac{1}{\sqrt{N}} W^\top \begin{pmatrix} -\sin(\Omega\xi) \odot \Omega_i \\ \cos(\Omega\xi) \odot \Omega_i \end{pmatrix} \tag{4.66}$$

$$\frac{\partial^2 f_e(\xi)}{\partial \xi_i \xi_j} = \frac{1}{\sqrt{N}} W^\top \begin{pmatrix} -\cos(\Omega\xi) \odot \Omega_i \odot \Omega_j \\ -\sin(\Omega\xi) \odot \Omega_i \odot \Omega_j \end{pmatrix} \tag{4.67}$$

Using (4.66) and (4.67), $\dot{f}_e$ and $\ddot{f}_e$ can be computed using the chain rule and knowledge of $\dot{\xi}$ and $\ddot{\xi}$, as depicted in Fig. 4.1.

4.4 Experiments

We aim to show the following three results:

R1 The feedback linearization controller presented handles large step responses with less deviations along unexcited axes than traditional cascaded approaches.

R2 Accounting for rotor delay using a delayed thrust model increases performance of the feedback linearization controller during step inputs in the presence of control input delays.

R3 The learned acceleration error model improves control performance of the feedback linearization controller in the presence of unmodeled dynamics and external disturbances.

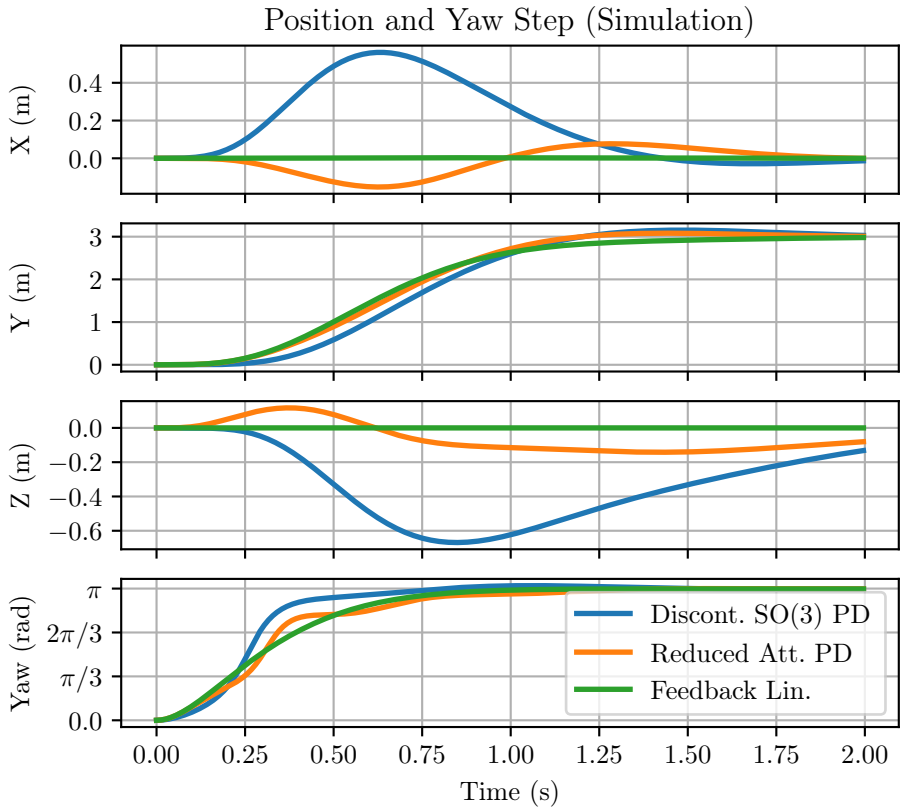


Figure 4.2: Position and yaw during a step from $(x, y, z, \psi) = (0, 0, 0, 0)$ to $(0, 3, 0, \pi - 0.01)$ for a traditional cascaded controller (blue), reduced attitude control (orange), and feedback linearization (green). Only feedback linearization follows the straight path from $p = (0, 0, 0)$ to $(0, 3, 0)$.

4.4.1 Position and Yaw Step Response

To highlight the advantages of feedback linearization over traditional controllers during large tracking error, we simulate the vehicle executing a simultaneous step along position and yaw. We compare against traditional discontinuous SO(3) control [29], as well as reduced attitude control [12]. The simulation performance of feedback linearization and the baseline controllers for a step response from $(x, y, z, \psi) = (0, 0, 0, 0)$ to $(0, 3, 0, \pi - 0.01)$ is shown in Fig. 4.2. The traditional attitude controller, which uses a metric on SO(3), suffers deviations along the x and z-axes, despite the position step lying along the y-axes. Although reduced attitude control is able to reduce these deviations, feedback linearization eliminates them entirely, thus showing **R1**.

4.4.2 Control Input Delay

To test the performance of the control input delay mitigation strategy on the hardware vehicle, we execute ten 3 m steps in position with the feedback linearization controller and various values of τ_u. Figure 4.3 shows the trajectory side view for the 3 m steps executed on the hardware platform. Ten trials are executed for the following: feedback linearization

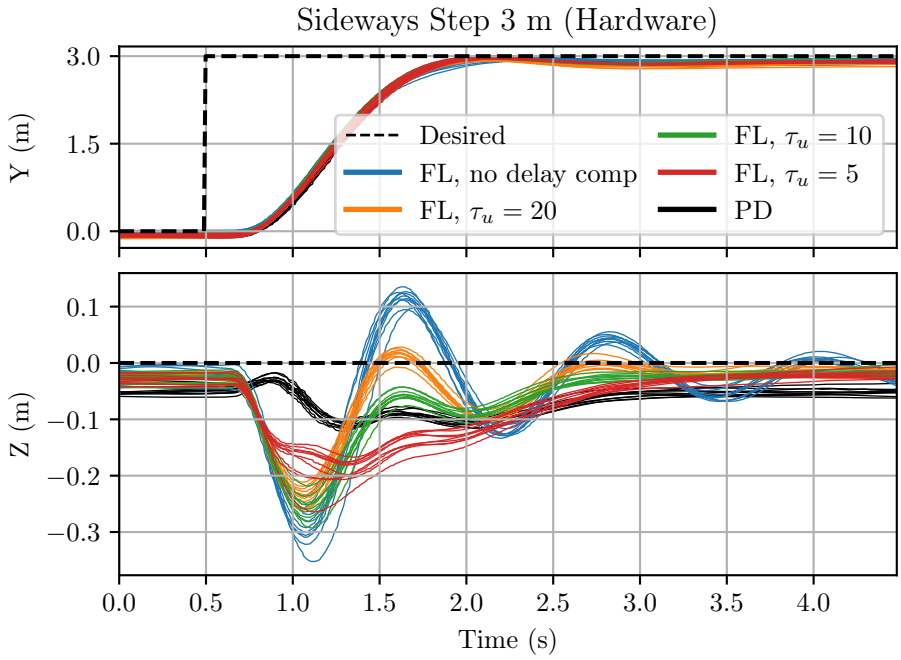


Figure 4.3: Position during ten 3 m sideways steps on the hardware platform for various feedback linearizaton configurations. Our proposed approach, shown here assuming a thrust delay with time constants of 5, 10, and 20, maintains height better, and reduces oscillations compared to standard feedback linearization. The traditional cascaded PD controller maintains height better than all tested feedback linearization methods.

without delay compensation, feedback linearization assuming thrust control input delays of $\tau = 5$, 10, and 20, and traditional cascaded PD control. We see that considering the thrust input delay significantly reduces the altitude oscillations and altitude error when compared to standard feedback linearization, supporting **R2** on the hardware platform. However, the traditional PD controller still maintains height better than all tested feedback linearization methods. For the remaining hardware experiments, we use $\tau_u = 10$, as it provides good reduction in oscillations, without greatly overestimating the delay.

4.4.3 Learned Acceleration Error Model

We test the effectiveness of the learned acceleration error model in the presence of unmodeled dynamics and external disturbances by attaching a cardboard plate to a quadrotor weighing 650 g and flying in a turbulent wind field, as shown in Fig. 4.4. The cardboard plate serves to accentuate disturbances due to the wind field. Position feedback is provided at 100 Hz and sent to the vehicle via a 2.4 GHz radio, while attitude control is run onboard the vehicle at 500 Hz.

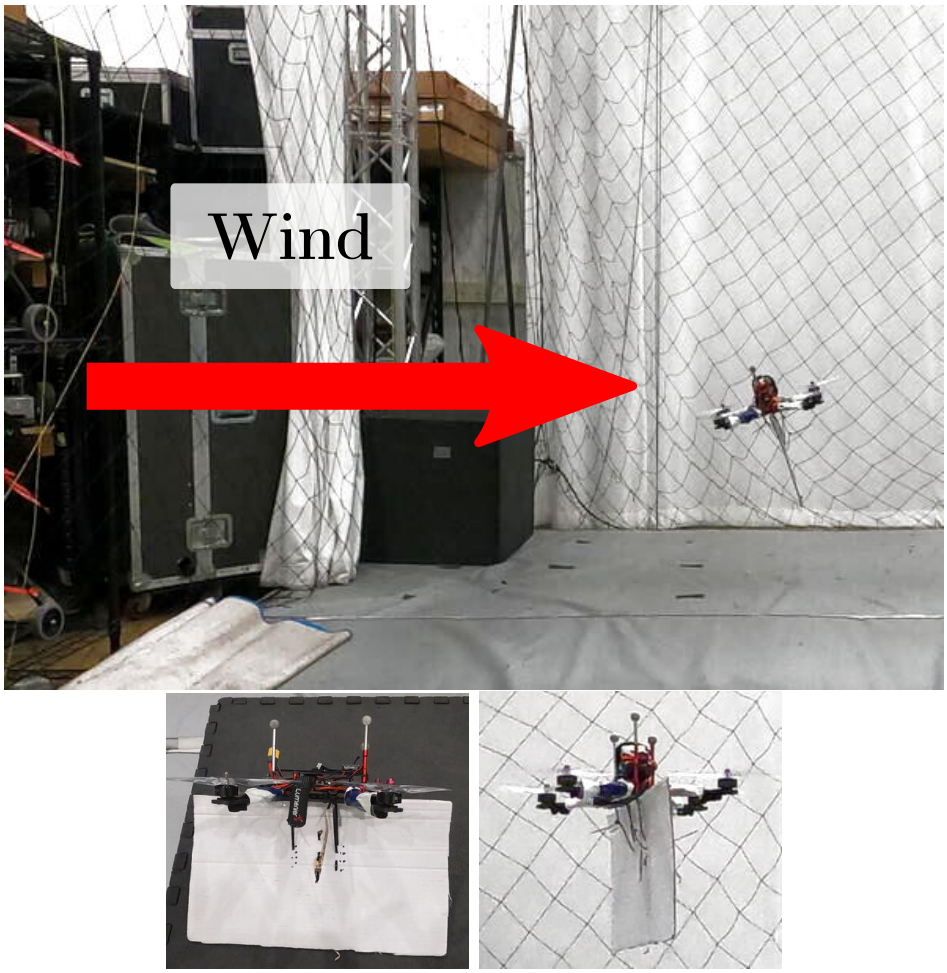


Figure 4.4: The quadrotor vehicle during the yaw in place experiment, with wind from two industrial fans coming from the left (top), the cardboard plate attached to amplify wind effect (bottom left), and the vehicle hovering with the cardboard plate without wind (bottom right).

Feedback linearization gains for the vehicle are

$$K_1 = \text{diag}(1040, 1040, 1900) \tag{4.68}$$
$$K_2 = \text{diag}(600, 600, 1140) \tag{4.69}$$
$$K_3 = 190 I_3 \tag{4.70}$$
$$K_4 = 25 I_3 \tag{4.71}$$
$$k_{p,\psi} = 30 \tag{4.72}$$
$$k_{d,\psi} = 10. \tag{4.73}$$

These were chosen to provide the same first order response in angular acceleration as

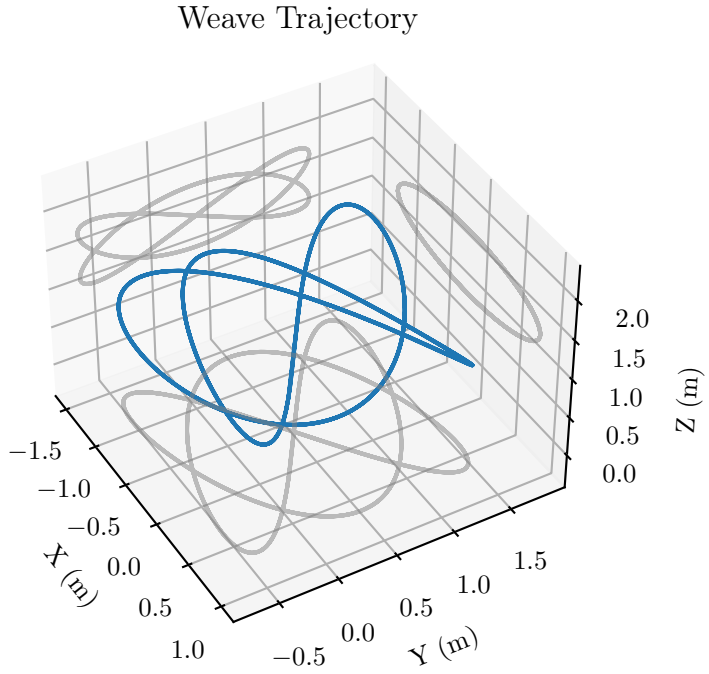


Figure 4.5: The 3D weave trajectory used in the experimental evaluation of the control strategy with per-axis shadows shown in gray.

the cascaded gains

$$K_p = \text{diag}(5.47, 5.47, 10.0) \tag{4.74}$$
$$K_v = \text{diag}(3.16, 3.16, 6.0) \tag{4.75}$$
$$K_R = \text{diag}(190, 190, 30) \tag{4.76}$$
$$K_\omega = \text{diag}(25, 25, 10) \tag{4.77}$$

using (4.62a) - (4.62d). Note that the first order response in thrust cannot be matched as it is a dynamic first order process for the feedback linearization controller.

We test the approach in two scenarios:

1. 3D weave pattern through the wind field
2. Fast yaw in place in the wind field

The 25 s weave trajectory, shown in Fig. 4.5, has a maximum velocity of 2.7 m/s, a maximum acceleration of 5.5 m/s^2, and moves in a sinusoidal pattern along x, y, and z. The equations for the x, y, and z positions are shown below, for $t \in [0, 25]$.

$$x(t) = 1.0\cos(0.72\pi t) \tag{4.78}$$
$$y(t) = 1.0\sin(0.48\pi t) \tag{4.79}$$
$$z(t) = 0.4\sin(0.72\pi t) \tag{4.80}$$

For this experiment, the performance of the proposed model learning strategy is compared to a controller using *L1 Adaptive Control* (L1AC) [63] for acceleration disturbance

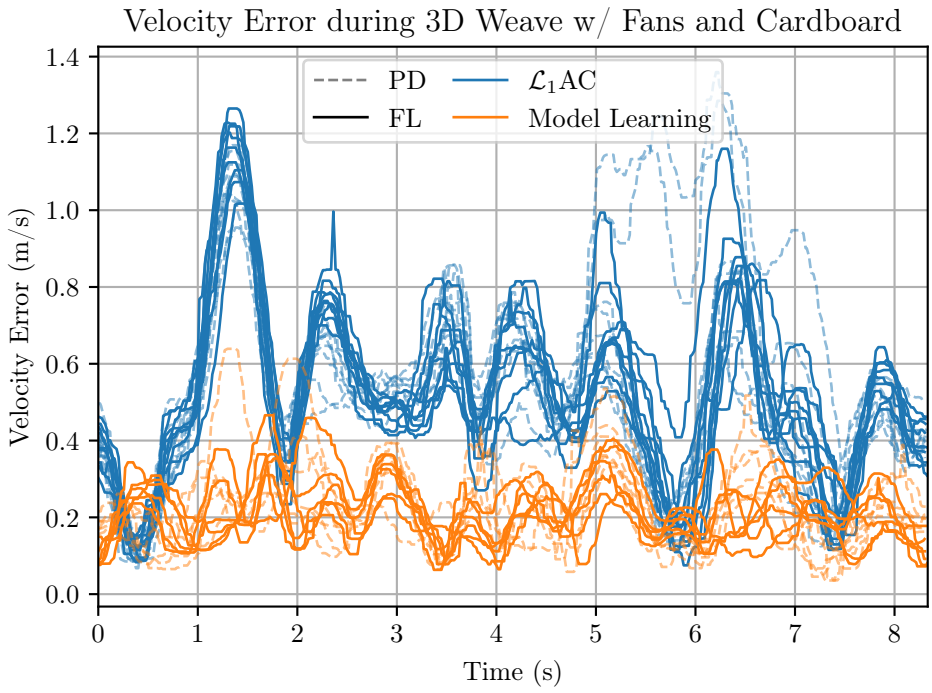


Figure 4.6: Smoothed velocity error for repeated trials of the 3D weave for the baseline adaptive controller (blue, 9 trials) and the proposed model learning strategy (orange, 6 trials shown). Baseline PD methods are shown in lighter dashed lines, while the proposed feedback linearization methods are solid. The model learning strategy reduces the mean velocity tracking error by roughly 60% compared to the adaptive controller. As expected, feedback linearization performs similar to cascaded control during trajectory tracking.

compensation and a cascaded PD controller running the model learning strategy from [80]. Since we are correcting for model errors during trajectory following, a low tracking error will mean that differences in feedback strategy are negligible, and we expect the cascaded PD approach to perform similarly to the feedback linearization approach. For this experiment, we use $N = 50$ random frequencies for ISSGPR and length scales of 1.0 for each dimension of the position and velocity. The velocity error during the weave trajectories for each strategy is shown in Fig. 4.6. Only the second and third trials (each trial consists of three circuits of the weave) for the two strategies using model learning are included to highlight the performance after the acceleration error model has converged. The model learning strategy is able to reduce the mean velocity error from 54 cm/s to 22 cm/s, a 60% improvement relative to the adaptive controller. As expected, feedback linearization performs similarly to the cascaded approach, for both with and without model learning. Fig. 4.7 shows the error of each of the four strategies per revolution. While the adaptive methods' errors remain high for each of the nine revolutions, the model learning methods converge to their lowest error after only three revolutions.

For the second test, seen in Fig. 4.4, the vehicle yaws in place in the wind field at a rate of 120 °/s for 4 revolutions. To enable the vehicle to model the changing acceleration disturbance as the vehicle yaws, we include two extra features in ξ in addition to the

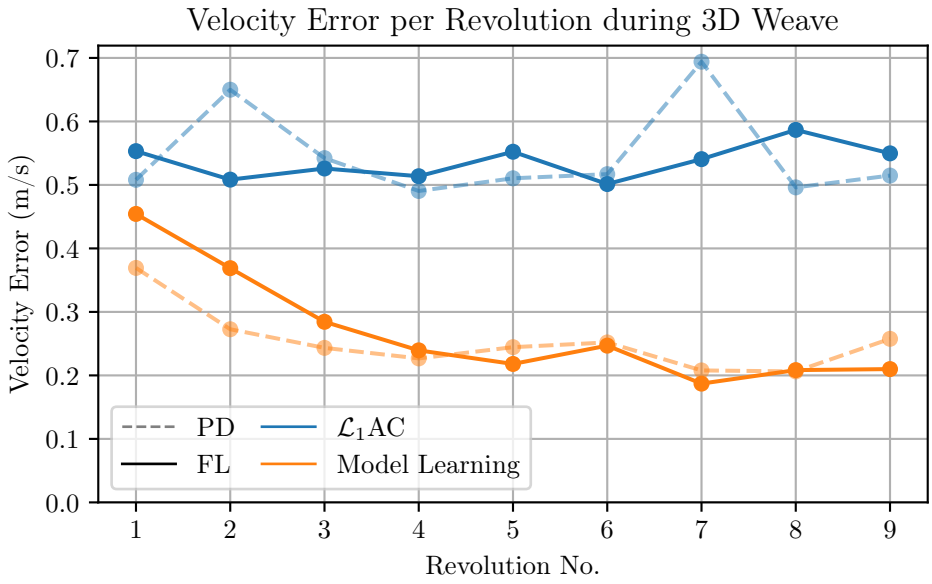


Figure 4.7: Average velocity error per revolution of the 3D weave for the baseline adaptive controller (blue) and the proposed model learning strategy (orange). The model learning strategy converges to its lowest error after three revolutions.

position and velocity: $\sin(\psi)$ and $\cos(\psi)$. The length scales used for this experiment are 5.0 for position and velocity and 0.5 for the yaw terms.

In this test, we compare the performance of the proposed model learning strategy with *L1 Adaptive Control* as before, and as an ablation test, with the proposed strategy but with disturbance dynamics neglected, i.e. $\dot{f}_e = \ddot{f}_e = 0$.

The average speed during the experiment for all three strategies is shown in Fig. 4.8. As before, the first trajectory for the model learning strategies is omitted, to remove transients from the model learning process. The adaptive controller struggles to stay still since it cannot react to the disturbance fast enough. The model learning strategies can predict the disturbance ahead of time, and reduce the speed by roughly 50%, showing **R3**. Compensating for disturbance dynamics helps to further reduce the speed error.

To gain insight into why model learning improves performance over the adaptive controller, we can look at how well the learned models match the observed data. Figure 4.9 shows the raw acceleration disturbance measurements, the disturbance estimated by the adaptive controller, and the disturbance predicted by our learned model for the two hardware experiments. The adaptive controller's disturbance estimate lags the true disturbance, while the learned model's prediction accurately captures the mean of the disturbance, leading to superior control performance. Increasing the adaptation rate (bandwidth) of the baseline disturbance estimator to attempt to reduce the lag would introduce noise and instability into the system. The learning algorithm appropriately averages sensor data to learn a smooth model from noisy input points.

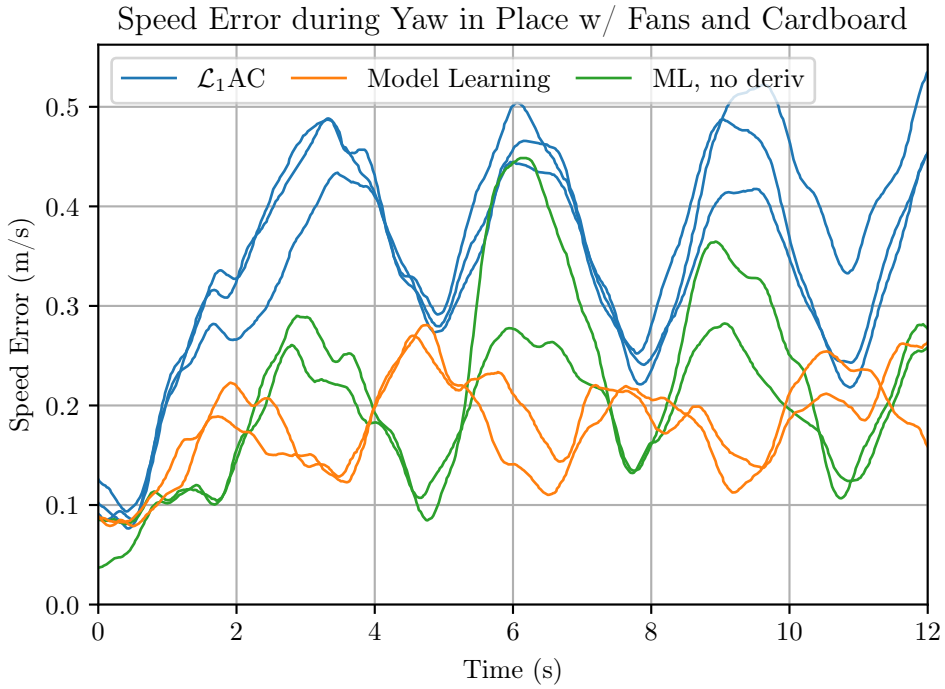


Figure 4.8: Smoothed speed error for multiple trials of the yaw in place test for the baseline adaptive controller (blue), the proposed model learning strategy without disturbance dynamics (green) and the proposed model learning strategy with full disturbance dynamics compensation (orange). The model learning strategy is able to stay still with roughly 50% lower speed compared to the adaptive controller. Including disturbance dynamics further helps the vehicle maintain position.

4.4.4 Iterative Learning Control Application

We further test the feedback linearization controller combined with Iterative Learning Control (ILC) [4]. The virtual linear system provided by feedback linearization is a natural space to apply ILC corrections, which are computed under the assumption of a linear system arising from the linearization of the full system along a trajectory. We task the vehicle with carrying a heavy 300 g PVC pipe along an aggressive linear trajectory, and compare the results against a cascaded controller running ILC. The ILC for the cascaded controller corrects the desired body z-axis acceleration u and desired angular acceleration α, while the ILC for feedback linearization corrects the desired snap s_{des}.

Fig. 4.10 shows the position errors for both the cascaded controller and feedback linearization and Fig. 4.11 shows the trajectory side view for both methods. Only feedback linearization converges to the desired trajectory. The cascaded control method fails to converge, likely because of the nonlinear relationship between the ILC corrections, u and α, and the desired output, position p. This highlights one of the benefits that the linearized system from feedback linearization can provide.

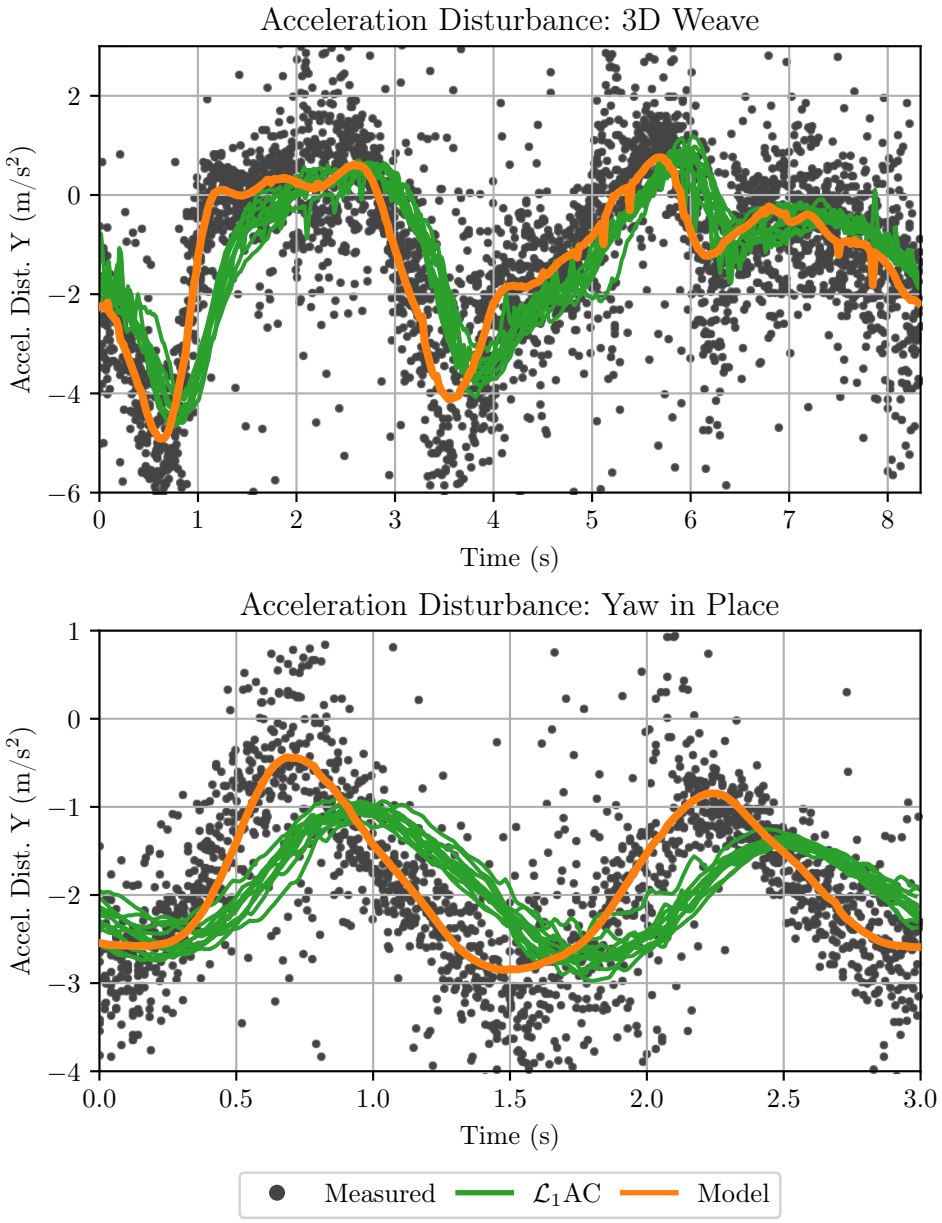


Figure 4.9: The raw acceleration disturbance measurements (black), the disturbance estimated by the adaptive controller (green), and the disturbance predicted by our learned model (orange) for the weave (top) and yaw in place (bottom) experiments. The disturbance predicted by the learned model captures the mean of the disturbance well, while the disturbance estimated by the adaptive controller lags.

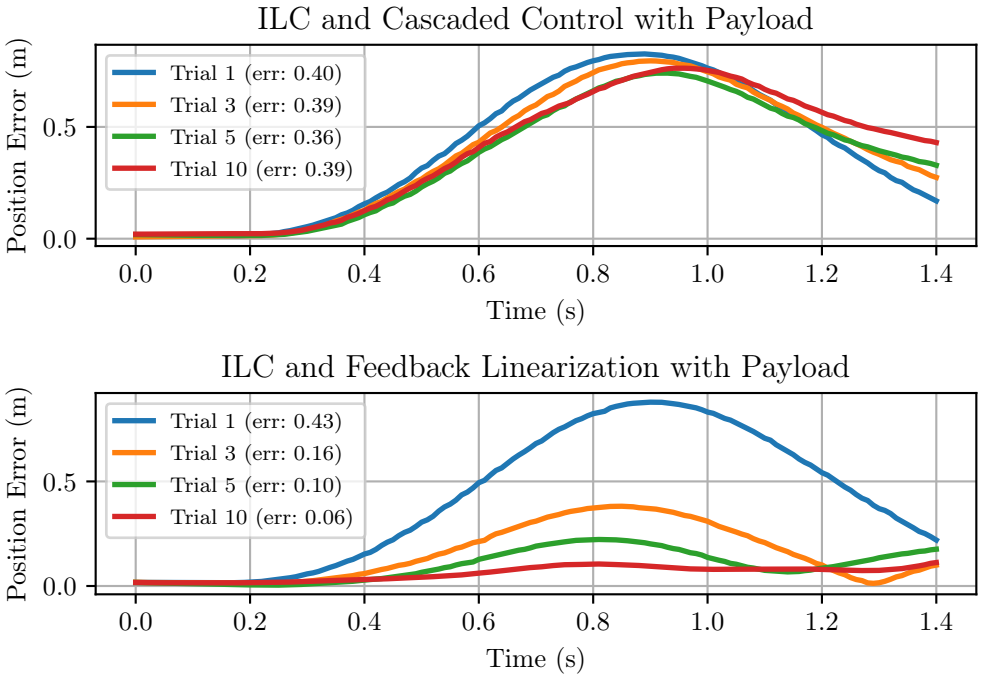


Figure 4.10: Position errors over time for trials 1, 3, 5, and 10 for both the cascaded controller (top) and feedback linearization (bottom). Only feedback linearization achieves low error and converges to the desired trajectory.

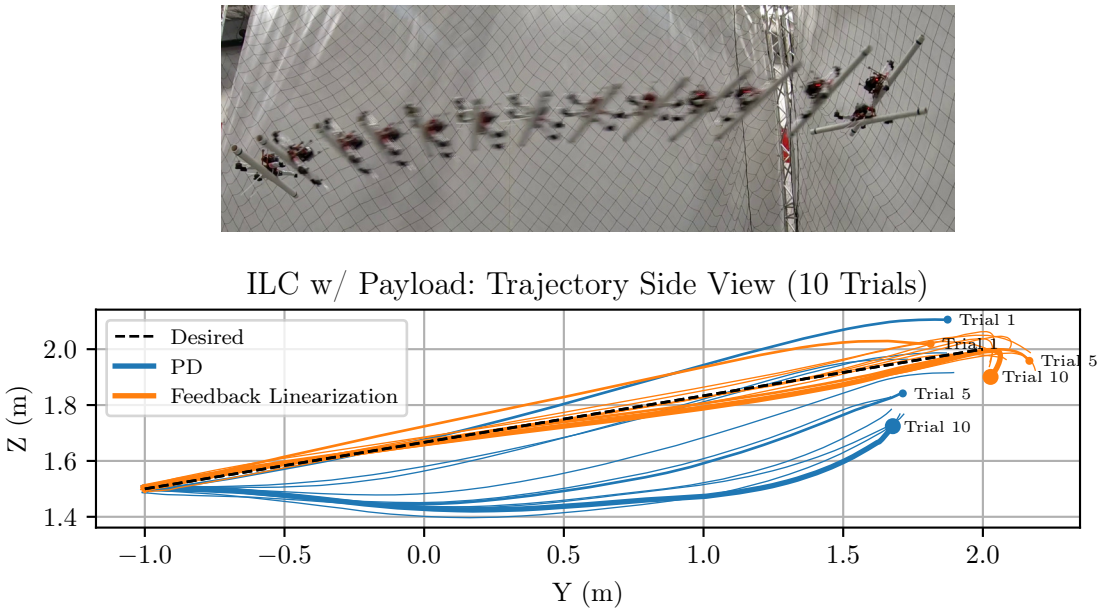


Figure 4.11: Trajectory side view during the ILC experiment. The traditional cascaded controller (blue) does not converge to the desired trajectory, while feedback linearization (orange) does.

4.5 Conclusion

We have presented a feedback linearization controller for the quadrotor that outperforms existing cascaded approaches during aggressive step responses in simulation (**R1**), mitigates the negative impact of control input delays (**R2**), and reduces errors arising from external disturbances through modeling and inversion of the learned disturbance (**R3**). We have further validated the feedback linearization controller using a challenging Iterative Learning Control experiment with a heavy payload.

Limitations arise from the need to model the disturbance as a function of the state space. The current incremental strategy cannot properly handle time-varying disturbances that often arise in practical conditions. Recent advances in deep learning can enable more capable model learning methods that require less hand-engineered features and parameters.

Future work should consider the delay in the angular acceleration control input, as well as the delay in the thrust control input, which is considered here. Further, a theoretical analysis of the control performance differences between feedforward linearization and feedback linearization in the presence of model error will enable intelligent selection of controllers and possible incorporation of uncertainty bounds into the control strategy. This is considered in Appendix D.

CHAPTER 5

Model Learning for Quadrotor Attitude Control

5.1 Introduction

The prior two chapters used model learning to learn a more accurate acceleration model of the quadrotor. This chapter explores dynamical model learning for the rotational dynamics of the quadrotor. Since the accuracy of the inner attitude controller directly affects the accuracy of the outer position controller, model errors in the inner loop can result in poor position tracking performance. Thus, improving the attitude dynamical model by learning a more accurate angular acceleration model should improve the overall control performance of the vehicle.

5.1.1 Problem

Quadrotor aerial vehicles use an IMU tightly coupled with a high rate control loop to allow precise and stable flight. Since attitude dynamics are faster than the position dynamics, we seek to learn a dynamics model using the data available onboard the vehicle firmware.

5.1.2 Challenges

5.1.2.1 Data

Data available on board the vehicle comes from an IMU. The IMU provides data at a high rate, but also comes with several aspects that make it difficult to directly learn models with:

1. Inherent sensor noise
2. Data biases, e.g. sensor scale errors, transform offsets
3. Time-dependence, e.g. temperature, biases exhibit random walks
4. Sources we are not interested in: vibration motion

Some of these challenges can be mitigated using calibration. For example, accelerometer scale and gyroscope bias calibrations are an established step of preparing a vehicle for

flight. However, the time-varying nature of these biases, for example due to temperature, or noises due to the vibration, render calibration less effective than we would like.

5.1.2.2 Regression

Once a sanitized data stream is available, model learning approaches typically optimize a model that predicts a desired quantity from a given "feature" quantity. Choosing the features to regress against is a difficult problem broadly known as "feature selection" and can have huge consequences on the accuracy of the model. An inherent difficulty is the balance between model expressiveness and generalization. Too little features and the model will not be able to explain any of the data. Too many features and the model will "memorize" the data and fail to generalize (overfitting).

After feature selection, most regression algorithms have hyperparameters that greatly impact performance. This necessitates hyperparameters optimization, which can be expensive or difficult to run online.

5.1.3 State of the Art

Established approaches, such as the Luenberger observer described in Section 2.4.2, estimate point disturbances that can be used to mitigate errors that are constant over time. However, these approaches fail when the model or disturbance rapidly changes. Further, their performance does not improve over time as more data is collected.

5.1.4 Requirements

1. A model learning strategy that can accurately predict observed IMU data.
2. Model must incorporate high rate IMU data in real-time.
3. A controller that can use a learned model to improve control performance.
4. Model and controller pair must outperform reactive approaches (e.g. Luenberger observer / L1AC).

5.2 Modeling

Following Section 2.1, we model the rotational dynamics as a rigid body. The Euler equation (2.70) is used to compute the angular acceleration.

$$\alpha = I^{-1}\left(\tau - \omega \times I\omega\right) = f_{\text{nom}}(\omega, \tau) \tag{5.1}$$

Here, we denote this *nominal* dynamics model by f_{nom}, which is a function of the vehicle angular velocity and the commanded torque.

We model the true angular acceleration of the robot as the sum of the nominal model f_{nom} and an additional error model f_{err}.

$$\alpha = f_{\text{nom}}(\omega, \tau) + f_{\text{err}} \tag{5.2}$$

5.2.1 Rotor Angular Momentum

In this section, we show that neglecting the angular momentum of the vehicle's spinning rotors can lead to large model prediction errors during aggressive yaw maneuvers.

To begin, we compute the vehicle's total angular momentum L, considering the four spinning rotors. Let ω_i be the speed of rotor i and $\omega_r = \omega_1 + \omega_2 - \omega_3 - \omega_4$. We assume that motors 1 and 2 spin counter-clockwise and motors 3 and 4 spin clockwise. Thus, ω_r is the sum of the angular velocities of all four rotors.

$$L = I\omega + I_r\omega_r e_3 \tag{5.3}$$

Here, I_r is the moment of inertia of the motor and propeller combination around its spinning axis, which is aligned with the body z axis of the vehicle. ω is the angular velocity of the body expressed in the body frame.

We proceed by equating the time derivative of the angular momentum with the sum of all the external torques acting on the vehicle, τ_{ext}.

$$\frac{dL}{dt} = I\dot{\omega} + I_r\dot{\omega}_r e_3 + \omega \times (I\omega + I_r\omega_r e_3) = \tau_{\text{ext}} \tag{5.4}$$

Solving for the angular acceleration of the vehicle $\alpha = \dot{\omega}$, we get

$$\alpha = I^{-1}\left(\tau_{\text{ext}} - I_r\dot{\omega}_r e_3 + \omega \times (I\omega + I_r\omega_r e_3)\right) \tag{5.5}$$

If we neglect the rotor inertia I_r, we see that (5.5) becomes the standard Euler equation for a rotating rigid body.

$$\alpha = I^{-1}\left(\tau_{\text{ext}} - \omega \times I\omega\right) \tag{5.6}$$

To see the effect of considering the rotor angular momentum, we compare predictions of angular acceleration from (5.6) with those from (5.5). To compute $\dot{\omega}_r$, the acceleration of each rotor $\dot{\omega}_i$ is computed from the commanded rotor speed ω_i^{des} using a first order delay model with $\dot{\omega}_i = -\tau_{\text{motor}}(\omega_i - \omega_i^{\text{des}})$.

Figure 5.1 shows the resulting predictions along the z-axis for a step yaw change of 45°. We see that the true angular acceleration more closely matches the prediction that considers rotor angular momentum using (5.5). The rotor inertia I_r was set to $5 \times 10^{-5}\,\text{kg}\,\text{m}^2$ by hand to closely match the true data.

Although we are able to identify a physics-based explanation for model error in the case of fast yaw changes, we still would like to be able to identify and correct for model errors without manually tweaked models.

5.2.2 Control with Disturbance Model

The feedback law shown in (2.69) assumes a disturbance estimate α_{dist} that is constant. This is appropriate for disturbance estimates that are the output of a filter, but the use of a disturbance model requires a choice of a query point.

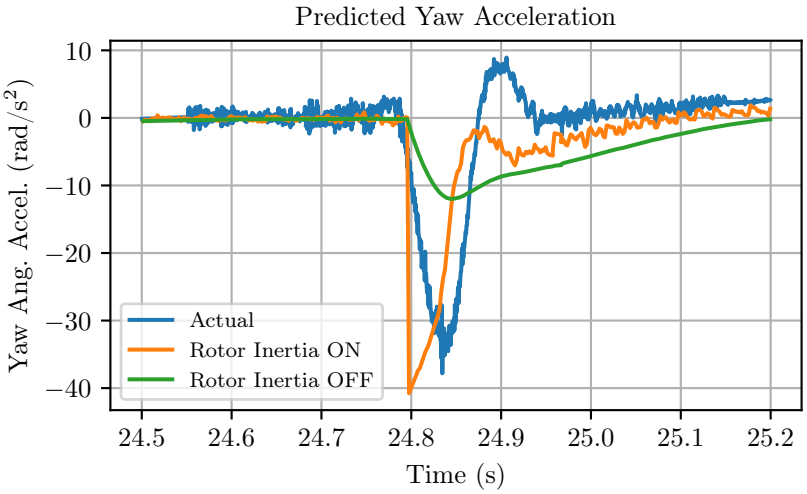


Figure 5.1: Angular acceleration predictions with and without considering rotor angular momentum during a step yaw change of 45°. The prediction considering rotor inertia matches the true angular acceleration more closely than the prediction without rotor inertia.

There are two natural choices for the query point: the current state and the desired state. Using the current state corresponds to *feedback linearization*, while using the desired state corresponds to *feedforward linearization* [39].

For example, if the disturbance model is a function of vehicle angular velocity, the feedback linearization law would be

$$\alpha_{\text{fb}} = -K_R e_R(R, R_{\text{des}}) - K_\omega(\omega - \omega_{\text{des}}) + \alpha_{\text{des}} - \alpha_{\text{dist}}(\omega) \tag{5.7}$$

while the feedforward linearization law would be

$$\alpha_{\text{fb}} = -K_R e_R(R, R_{\text{des}}) - K_\omega(\omega - \omega_{\text{des}}) + \alpha_{\text{des}} - \alpha_{\text{dist}}(\omega_{\text{des}}). \tag{5.8}$$

Although the feedback linearization law may feel more natural, feedforward linearization does not suffer from state estimation or sensor noise, nor does it suffer from the robustness issues of feedback linearization [38].

The disturbance model may further be a function of the control input α. With feedback linearization, a numerical optimization routine must be used to solve the below equation.

$$\alpha_{\text{fb}} = -K_R e_R(R, R_{\text{des}}) - K_\omega(\omega - \omega_{\text{des}}) + \alpha_{\text{des}} - \alpha_{\text{dist}}(\omega, \alpha_{\text{fb}}). \tag{5.9}$$

With feedforward linearization however, no such optimization is required as the desired angular acceleration is independent of the applied angular acceleration.

$$\alpha_{\text{fb}} = -K_R e_R(R, R_{\text{des}}) - K_\omega(\omega - \omega_{\text{des}}) + \alpha_{\text{des}} - \alpha_{\text{dist}}(\omega_{\text{des}}, \alpha_{\text{des}}). \tag{5.10}$$

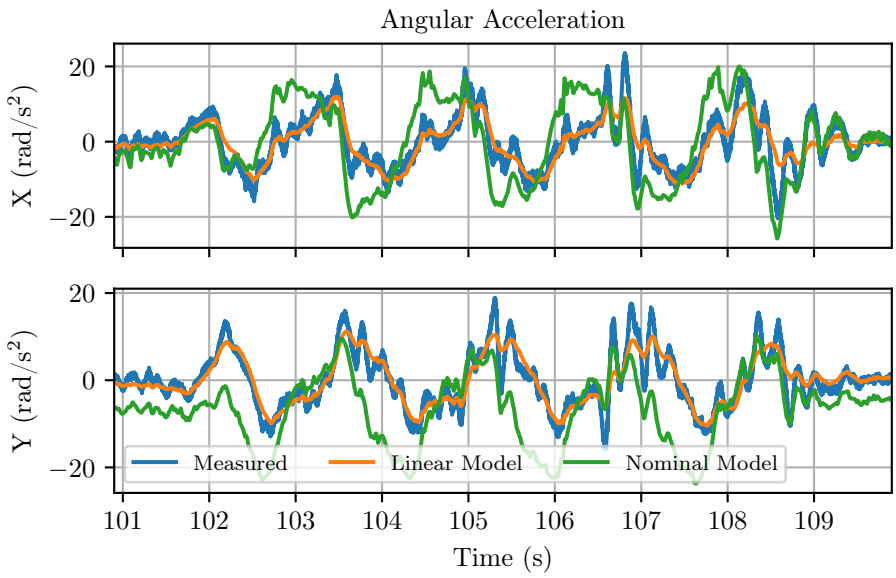


Figure 5.2: Angular acceleration while following aggressive circles. The nominal model is inaccurate and the learned linear model corrects the error well.

5.2.3 Model Learning

To set up a regression problem, we solve for f_{err}.

$$f_{\text{err}} = \alpha - f_{\text{nom}}(\omega, \tau) \tag{5.11}$$

Natural choices for the inputs to the regression are the angular velocity of the vehicle, the orientation of the vehicle, and the RPMs of the motors. Feature selection is a hard problem however, and we leave this as an open problem for now. Model performance on datasets can be helpful in evaluating what features are best to use.

Results from an initial trial on flight data are shown in Figs. 5.2 and 5.3. Average absolute angular acceleration error along X and Y is reduced by around 60% and 80% percent on the training set. Features used in the linear regression are the angular velocity, the motor RPMs, and a constant term.

5.3 Experiments

5.3.1 Baseline System Evaluation

We first establish performance characteristics of the baseline system through a series of trajectory following experiments.

5.3.1.1 Disturbance Filter with Acceleration Disturbance Compensation

We compare the tracking performance of the angular acceleration disturbance (AAD) estimation filter described in Section A.2.4 to the tracking performance of the system

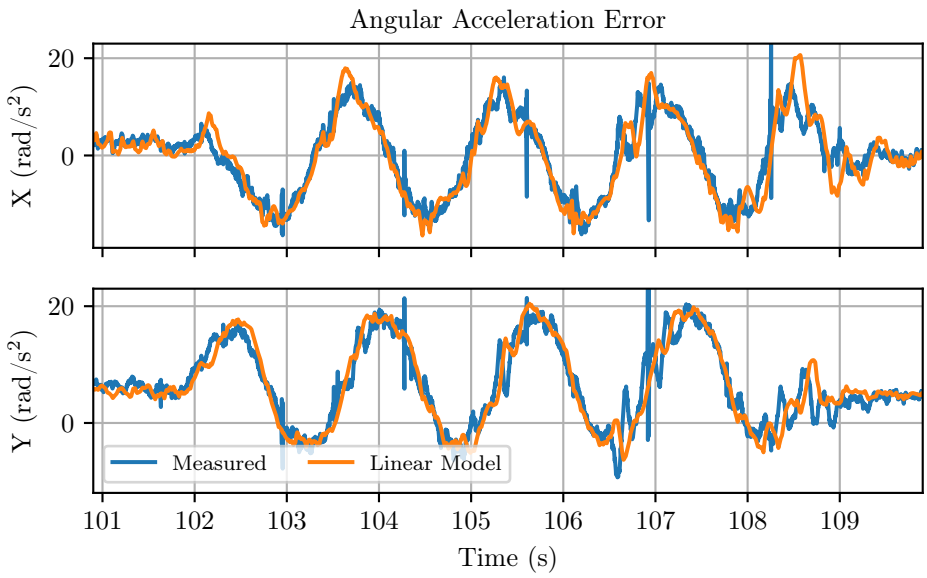


Figure 5.3: Angular acceleration error during aggressive circles after fitting a linear model. The model matches the measured angular acceleration disturbance well.

with only gyro bias estimation as described in Section A.2.3 (no angular acceleration disturbance estimation). Figs. 5.4 and 5.5 show the control performance of a 660 g quadrotor executing circular trajectories at two different speeds while carrying a 160 g metal weight (simulating a payload).

During the circular trajectories, AAD compensation improves angular tracking performance for the slower circles (Fig. 5.4), but has the opposite effect for the fast circles (Fig. 5.5). This suggests that the disturbance estimate has difficulty in improving performance when the disturbance is changing quickly. For both trials however, adding AAD compensation increases the position error. This suggests there is modeling error in the outer loop that cannot be accounted for by the AAD observer.

5.3.2 Flying in a Wind Field with Cardboard Plate

To test the proposed algorithms in a challenging environment, we task the vehicle to fly in a turbulent wind field generated by 6 high-velocity fans while carrying a rigidly attached cardboard plate for added drag disturbances. Figure 4.4 shows the robot hovering with the cardboard plate in the wind field. In order to remain still, the vehicle must hover at a non-trivial angle to fight the linear force imparted by the wind. The wind further induces a large roll torque as the entire cardboard plate is below the center of mass of the vehicle.

The forces and torques generated by the oncoming wind are heavily dependent on the angle of the cardboard plate with respect to the velocity of the wind. Thus, as a measure of how well our disturbance compensation algorithms are performing, we task the vehicle to yaw in place with various speeds as it hovers in the wind field.

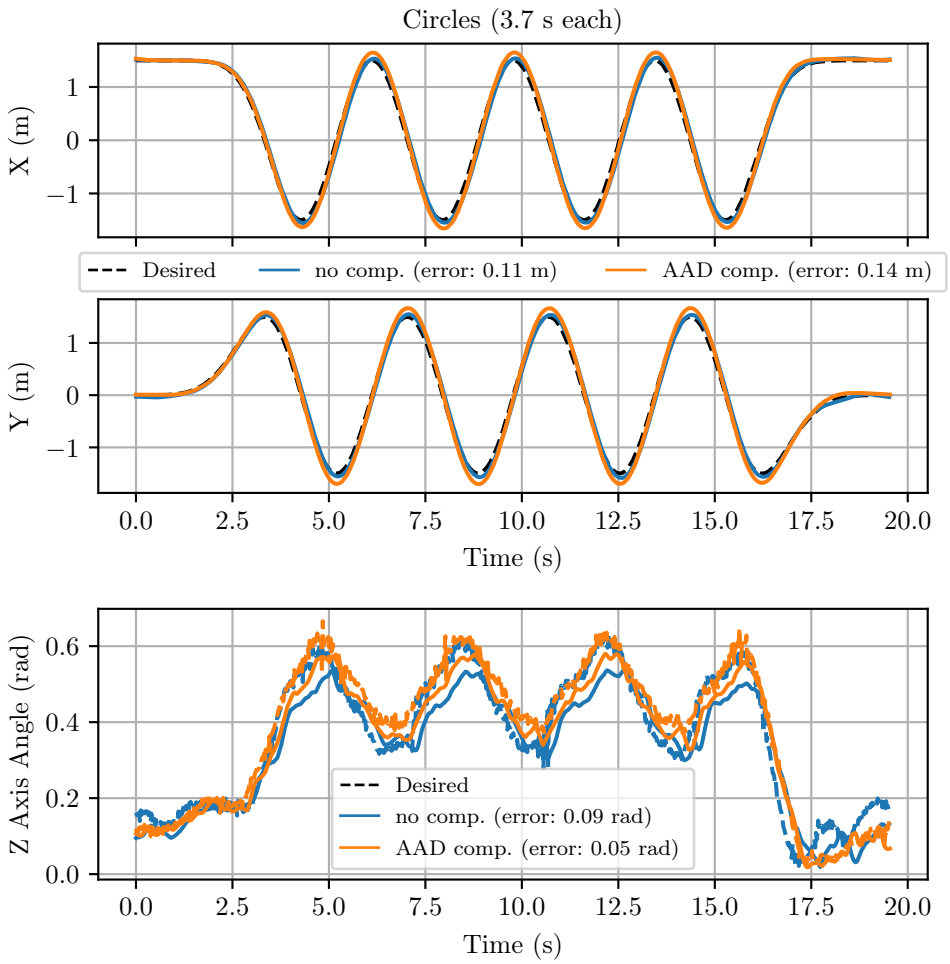


Figure 5.4: Position (top) and angle of the body z-axis (bottom) during 3 1.5 m radius circles of 3.7 s each at a speed of 2.57 m/s, while carrying a 160 g metal weight.

First, we compare the performance of the angular acceleration disturbance (AAD) estimation filter described in Section A.2.4 with the baseline system without angular acceleration disturbance compensation. Both systems use $\mathcal{L}_1$ adaptive control with a bandwidth of 1.0 along the x and y axes for linear acceleration disturbance compensation. Video of the experiment for the baseline system can be seen at https://youtu.be/T9GPuptJnlQ while that for the system with AAD compensation can be seen at https://youtu.be/xdR7y_xiW8I.

Figure 5.6 shows attitude control performance as the vehicle executes a 360° rotation in yaw in increments of 45°. Angular acceleration disturbance compensation increases both yaw tracking and body z-axis tracking performance, as expected. The large transient yaw error just before $t = 20$ s for AAD comp. is likely due to motor saturation.

Figures 5.7 and 5.8 show attitude and position control performance as the vehicle executes 3 revolutions in yaw at a constant yaw rate of 120 °/s. Angular acceleration disturbance compensation again increases both yaw tracking and body z-axis tracking performance. However, it has no noticeable effect on position tracking. This is likely due

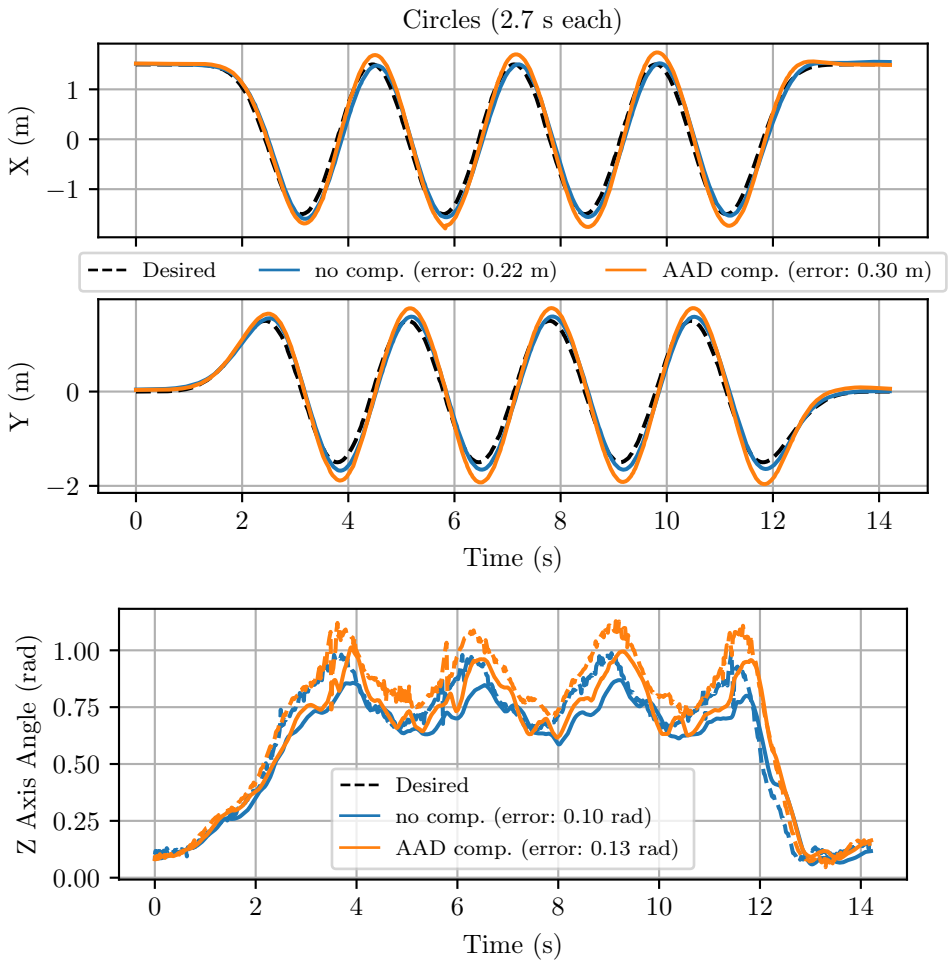


Figure 5.5: Position (top) and angle of the body z-axis (bottom) during 3 1.5 m radius circles of 2.7 s each at a speed of 3.53 m/s, while carrying a 160 g metal weight weighing.

to the rapidly changing linear acceleration that is not accounted for quickly enough by the position loop disturbance observer.

5.3.2.1 Angular Acceleration Disturbance (AAD) Compensation with Acceleration Model Learning

To see if AAD compensation will have more of a positive impact on position control performance if the accelerations on the robot are more accurately modeled, we apply the linear acceleration model learning approach from Chapter 3.

Yaw-in-place in Front of Fans The robot is asked to execute 4 revolutions in yaw at a rate of 120 °/s while maintaining its position in front of the fans. We test the following four control system configurations:

1. L1AC in the position loop with a bandwidth of 1.0 in the x and y axis and no

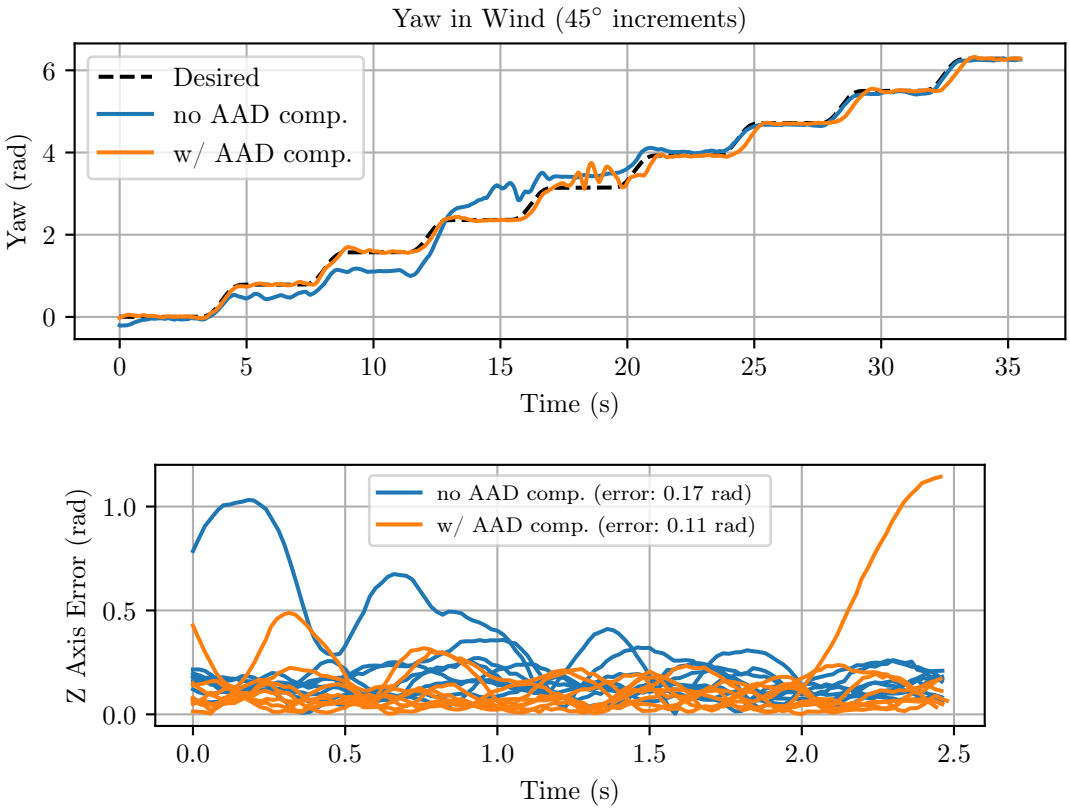


Figure 5.6: Yaw tracking performance (top) and body z-axis error (bottom) during 8 trials each of a trajectory spanning 45° in yaw in the wind field. Angular acceleration disturbance compensation greatly improves attitude tracking performance.

AAD compensation (**Filter** / **None**)[1]

2. Online model learning in the position loop from [80], with full disturbance dynamics inversion, and no AAD compensation (**ML** / **None**). The position loop model learning uses position and the vehicle body y-axis as the input feature vector for ISSGPR.
3. L1AC as above with AAD compensation (**Filter** / **Filter**)
4. Model learning as above with AAD compensation (**ML** / **Filter**)[2]

Each configuration executes the test trajectory at least five times.

Figure 5.9 shows the position control performance for all four configurations during the experiment, broken down into position error and speed. Although differences in position error are quite small, we see that model learning in the outer loop, combined with AAD compensation in the inner loop, outperforms all others approaches in terms of speed error, suggesting that the vehicle is able to stay still much better as it rotates in the turbulent wind field. The fact that this gain in speed error does not translate into a gain

[1]Video of **Filter** / **None** for 120 deg/s is at https://www.youtube.com/watch?v=5e1HlEimJhI.
[2]Video of **ML** / **Filter** for 120 deg/s is at https://www.youtube.com/watch?v=hhJhfkse0w4.

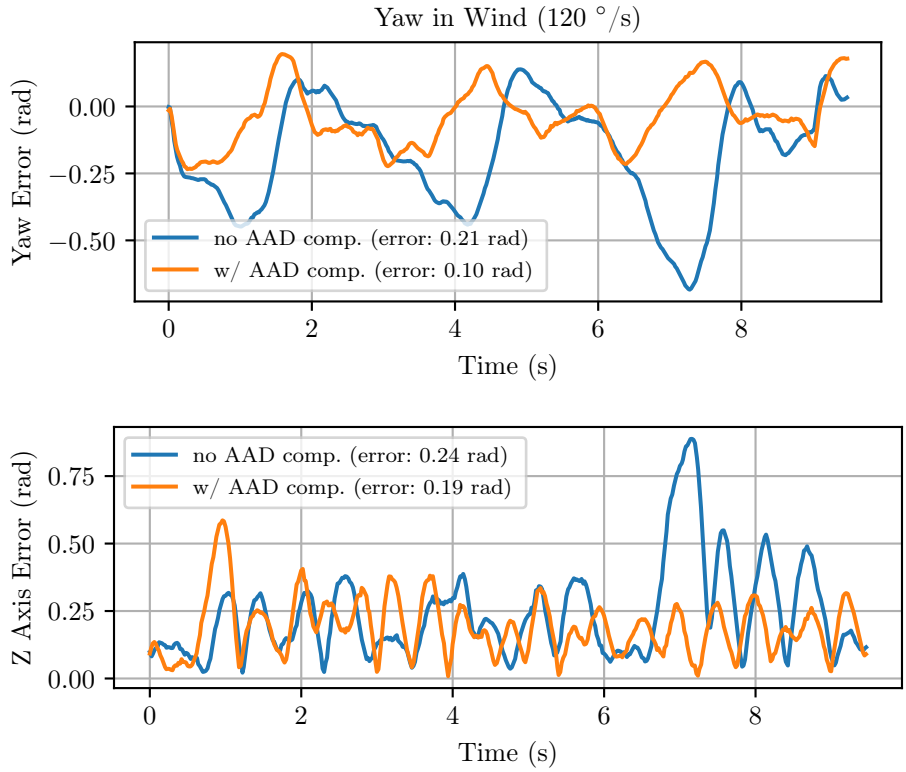


Figure 5.7: Yaw tracking performance (top) and body z-axis error (bottom) during 3 revolutions at 120 °/s in the wind field. Angular acceleration disturbance compensation greatly reduces the yaw deviations and improves body z-axis tracking performance.

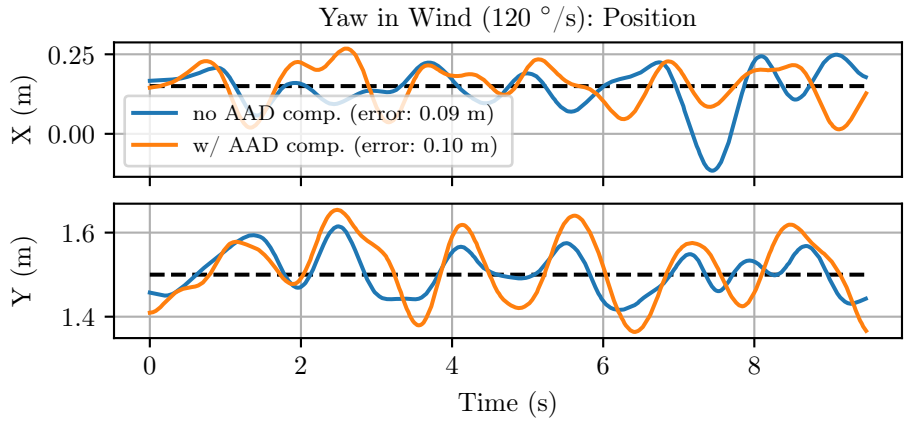


Figure 5.8: Position tracking performance during 3 revolutions at 120 °/s in the wind field. Angular acceleration disturbance compensation has no noticeable effect on position tracking.

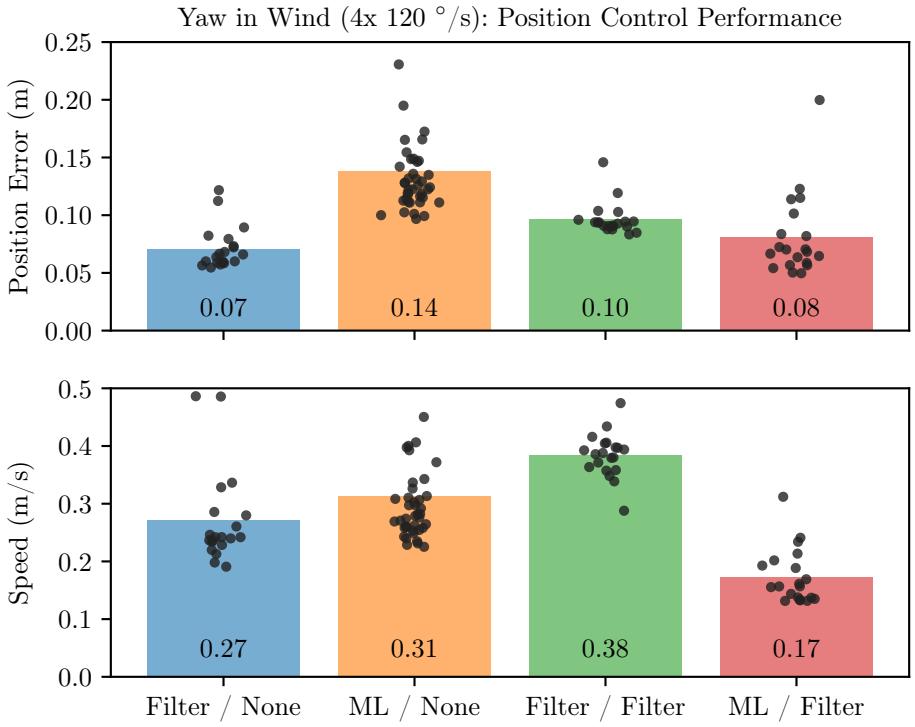


Figure 5.9: Position error (top) and speed (bottom) for the four tested combinations while rotating 4 times around yaw in the wind field at 120 °/s. Points shown represent one revolution with mean errors displayed in the bar (lower is better). Each configuration is tested at least five times. Online model learning in the position loop, combined with AAD compensation using the onboard filter, (**ML** / **Filter**, red), achieves lower speed error and similar position error when compared to the **Filter** / **None** baseline.

in absolute position error suggests that there is a small, but persistent position error offset during the duration of the trial. :his will be investigated in future work.

Figure 5.10 shows the yaw error and z-axis angle error for the four configurations, broken down into yaw error and z-axis angle error. Both approaches using AAD compensation achieve substantially lower yaw error than the approached without AAD compensation. However, only **ML** / **Filter**, the configuration combining acceleration model learning and AAD compensation, achieves the lowest z-axis angle error. **Filter** / **Filter** struggles to improve the z-axis angle tracking error over **Filter** / **None**, likely because the desired z-axis coming from the position control loop is changing rapidly due to the imprecise position feedback. This can be seen in the speed graph, where **Filter** / **Filter** sustains the highest speed error over all of the configurations.

The results here suggest that for highly dynamic trajectories, there is a synergistic component to disturbance compensation between the outer loop, which estimates acceleration disturbances, and the inner loop, which estimates angular acceleration disturbances. Using either one alone fails to improve performance in a significant way. Improved control performance under dynamic disturbances requires both adequate acceleration compensation and adequate angular acceleration compensation running together.

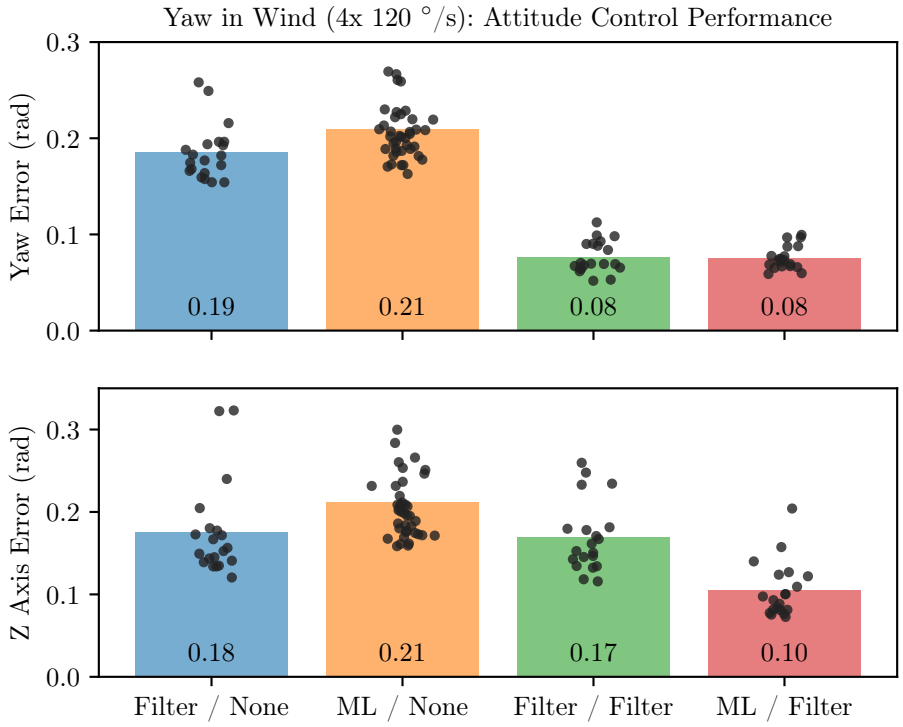


Figure 5.10: Yaw error (top) and z-axis error (bottom) for the four tested combinations while rotating 4 times around yaw in the wind field at 120 °/s. Points shown represent one revolution with mean errors displayed in the bar (lower is better). Online model learning in the position loop, combined with AAD compensation using the onboard filter, (**ML / Filter**, red) performs best in terms of z-axis angle error, while both approaches that use AAD compensation perform best in terms of yaw error.

AAD Model at 180 °/s We further test the impact of position control loop model learning when AAD compensation is enabled using a faster yaw trajectory of 180 °/s. We also test a configuration that uses a learned AAD model that is fixed at runtime.

Below are the configurations tested.

1. L1AC in the position loop with a bandwidth of 5.0 in the x and y axis and AAD compensation (**Filter / Filter**)[3]. The position loop acceleration disturbance filter used for these trials differs from the one used in the trials above with a yaw rate of 120 °/s, in the source used for the predicted acceleration. For these trials, the predicted acceleration is computed as the commanded acceleration along the body z-axis, multiplied by the actual vehicle z-axis. In the trials at 120 °/s, the predicted acceleration is the desired acceleration vector that is output by the position control loop. This change allows for a much higher bandwidth without encountering instability, since the predicted acceleration, and thus estimated disturbance, cannot change as rapidly as position error changes. Consequently, the bandwidth is increased from 1.0 to 5.0 for this trial.

[3]Video of **Filter / Filter** for 180 °/s is at https://www.youtube.com/watch?v=FJkUeA0nxGs.

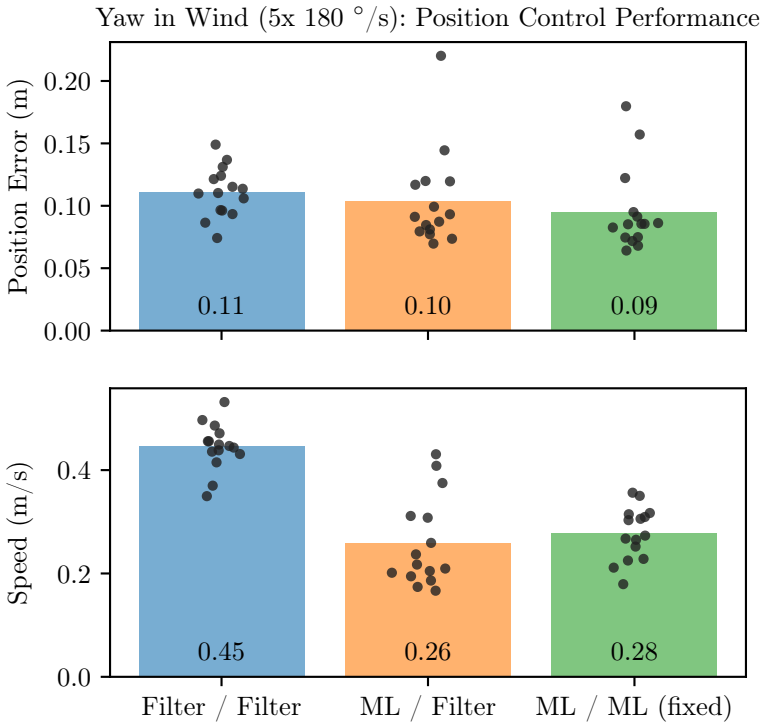


Figure 5.11: Position error (top) and speed (bottom) for the three tested configurations while rotating 5 times around yaw in the wind field at 180 °/s. Points shown represent one revolution with mean errors displayed in the bar (lower is better). Each configuration is tested three times. All tested configurations perform comparably with respect to position error, while both configurations that use model learning outperform the filtering configuration with respect to speed error.

2. Online model learning in the position loop from [80], as before, and AAD compensation (**ML / Filter**).[4]
3. Position loop model learning as above, with a fixed model for the AAD, with position and the vehicle y-axis as the input feature vector to ISSGPR. The model is trained on separate flight data and is not updated at test time. (**ML / ML**)

Figure 5.11 shows the position control performance for the three configurations, again broken down into position error and speed. All configurations perform comparably in terms of position error, while the configurations using model learning outperform the filtering configuration in terms of speed error. Results here are consistent with the results for the trials at 120 °/s, with no noticeable improvement by using the AAD model.

Figure 5.12 shows the attitude control performance, broken down into yaw error and z-axis angle error. In terms of attitude, the configuration using the AAD model outperforms all by a slim margin. More experimental tests are needed to determine if this is a significant difference.

[4]Video of **ML / Filter** for 180 °/s is at https://www.youtube.com/watch?v=rUE6sdRQJ-w.

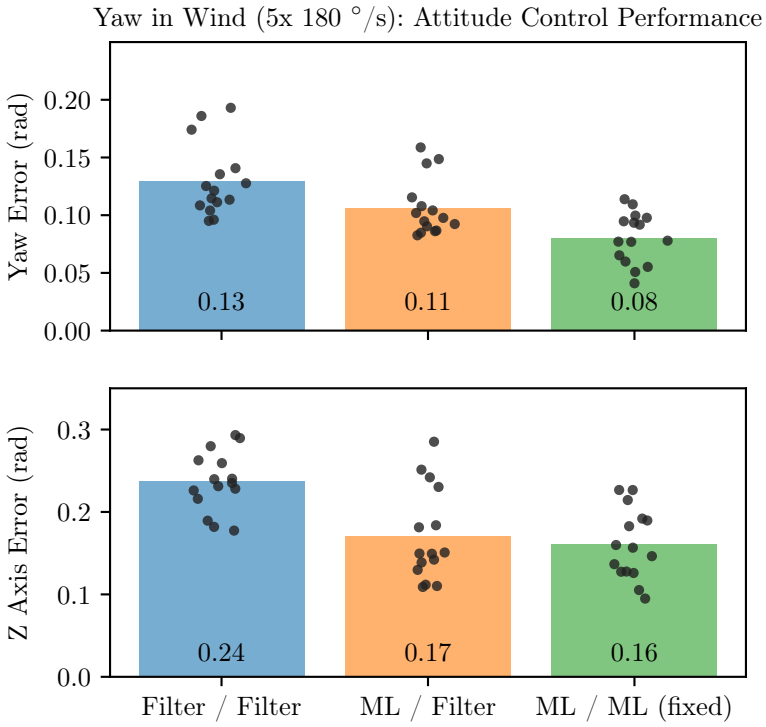


Figure 5.12: Yaw error (top) and z-axis error (bottom) for the three tested combinations while rotating 5 times around yaw in the wind field at 180 °/s. Points shown represent one revolution with mean errors displayed in the bar (lower is better). The configuration which uses a learned AAD model (**ML / ML**) outperforms with respect to yaw error, while both configurations that use model learning outperform the filtering configuration with respect to z-axis angle error.

The experiment using a fixed, learned AAD model is preliminary and future work can expand on it, by

- Learning the AAD model online.
- Using angular velocity and commanded RPMs in the feature vector.

5.3.3 Flying with a Cardboard Plate

We test the ability of the vehicle to track a trajectory while carrying the cardboard plate and yawing at a rate of 180 °/s. For this experiment, the AAD model is a function of velocity, thrust, and yaw. The attitude gains were lowered when running with AAD compensation to avoid instability. Figure 5.13 shows the speed and z-axis error for various strategies during a 3D Weave trajectory. Although the approach that uses the learned AAD model combined with the learned acceleration error model performs the best out of all the approaches with the same gains, the filtering-based AAD compensation achieves lower error with the original, higher attitude gains (HG).

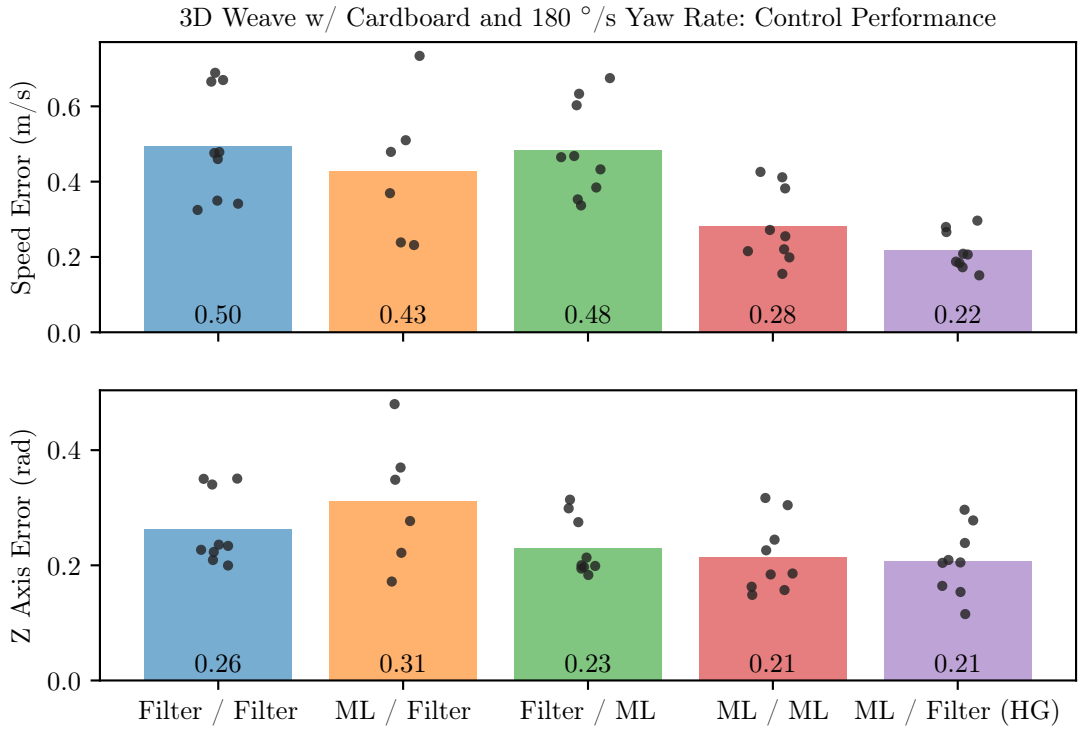


Figure 5.13: Speed (top) and z-axis error (bottom) for various strategies while executing the 3D Weave trajectory, as in https://www.youtube.com/watch?v=E1-LK__Ssmo.

5.3.3.1 Feedforward Linearization for AAD Model

We compare using a disturbance model that is a function of vehicle angular velocity and motor RPMs with both feedback linearization and feedforward linearization. As in the previous experiment, attitude gains had to be lowered (LG) when using this model with actual sensor data due to instability. When using feedforward linearization however, the original attitude gains (HG) could be used. This highlights the robustness advantage of feedforward linearization over feedback linearization.

Position and attitude performance during the 3D Weave while spinning around yaw are shown in Figs. 5.14 and 5.15. The learning-based AAD compensation approach with feedforward linearization performs similarly to the filtering approach in position and speed error and achieves slightly lower z-axis error. This is a positive result for model learning as it shows the model learning method is competitive despite not using data from the current vehicle operation. The filtering method has the benefit of fresh data. Model learning can outperform, despite using "stale data", when there is a repeatable and thus predictable angular acceleration disturbance.

5.4 Conclusion

The experiments presented highlight the benefits of modeling disturbances on the attitude dynamics of the quadrotor. We have shown that while model learning approaches

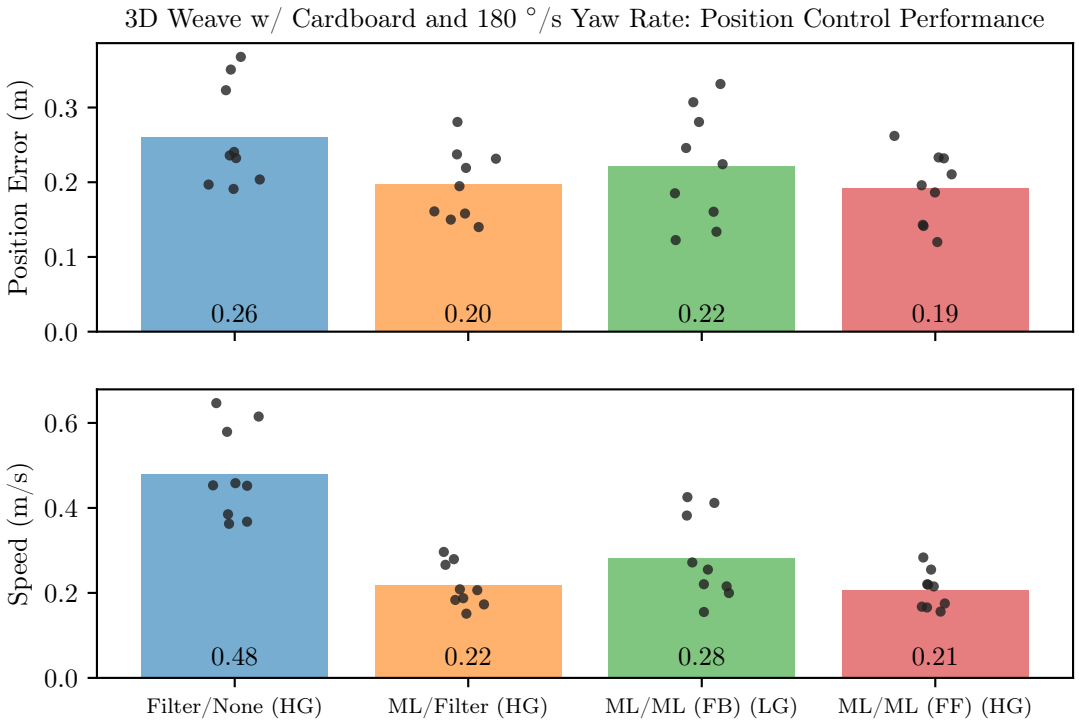


Figure 5.14: Position error (top) and speed error (bottom) while executing the 3D Weave trajectory for feedback linearization and feedforward linearization on the inner loop.

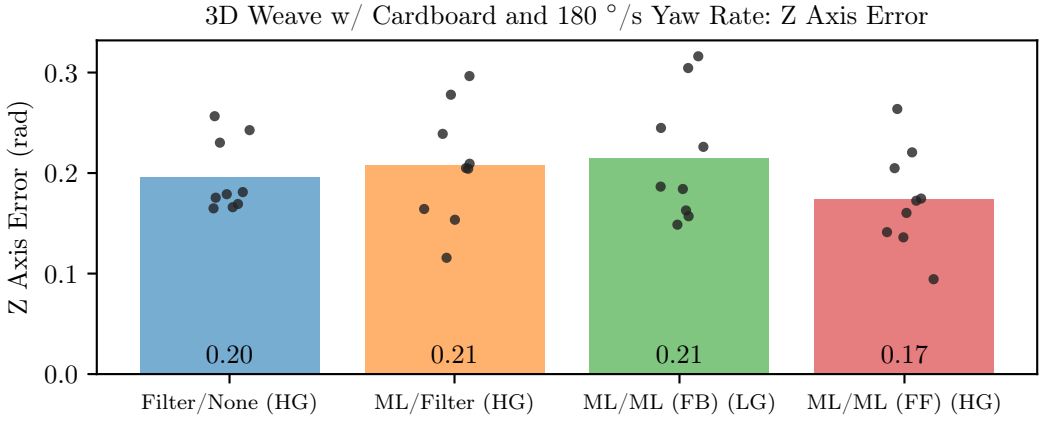


Figure 5.15: z-axis error while executing the 3D Weave trajectory for feedback linearization and feedforward linearization on the inner loop.

on the attitude dynamics can improve performance, the margin of improvement over filtering-based approaches is smaller than when learning position dynamics as in the previous two chapters. The much faster attitude dynamics are likely a large reason for this. We have also shown that feedforward linearization provides a way to use learned angular acceleration disturbance models without sacrificing robustness and is a promising direction for future study.

CHAPTER 6

Model Learning for High Speed Outdoor Flight

In the prior chapters, we used dynamical model learning to improve control performance of a quadrotor flying with reliable state estimation. In practice, quadrotors flying in realistic scenarios rely mostly on GPS and vision-based state estimation. GPS data can be unreliable and is often unavailable indoors or in obstructed environments. Vision-based data is more readily available but can suffer from robustness issues and can fail due to motion blur, a lack of visual richness in the scene, or extreme lighting conditions. As a result, a state estimate derived from vision data is often noisier and less accurate than that from a motion capture arena.

In this chapter, we show how to apply the model learning strategies presented earlier to scenarios with less than ideal state estimation and validate them by flying a quadrotor outdoors at high speed.

This work advances the thesis statement in two ways:

1. shows significance of model learning's impact on control performance in a challenging and realistic outdoor scenario, and
2. enhances and validates the applicability of the methods presented under a degraded state estimate.

6.1 Quadrotor

The vehicle used in the outdoor experiments, shown in Fig. 6.1 is a quadrotor weighing 1.1 kg and a thrust to weight ratio of over 4. The parts used on the vehicle are provided in Table 6.1.

The firmware running on the Pixracer is as described in Appendix A, with the attitude feedback controller running at 1000 Hz and commands sent to the ESC using the DShot protocol. The onboard NVIDIA Tegra TX2 computer, on which all control and model learning algorithms are executed, runs ROS Melodic on a minimized Ubuntu 18.04.

Table 6.1: Outdoor quadrotor parts list

Part	Model
Frame	Armattan Chameleon Ti LR 7 inch
Motor	4x T-Motor F60 Pro II 1750kv
ESC	Lumenier BLHeli32 50 A 4-in-1
Propeller	4x DALProp Cyclone T7056C
Battery	Lumenier 4s N2O 1500 mAh 120c
Flight Controller	Pixracer
GPS	ublox MAX-8
Vision-based S.E.	Intel RealSense T265
Computer	NVIDIA Tegra TX2
Carrier Board	Auvidea J121

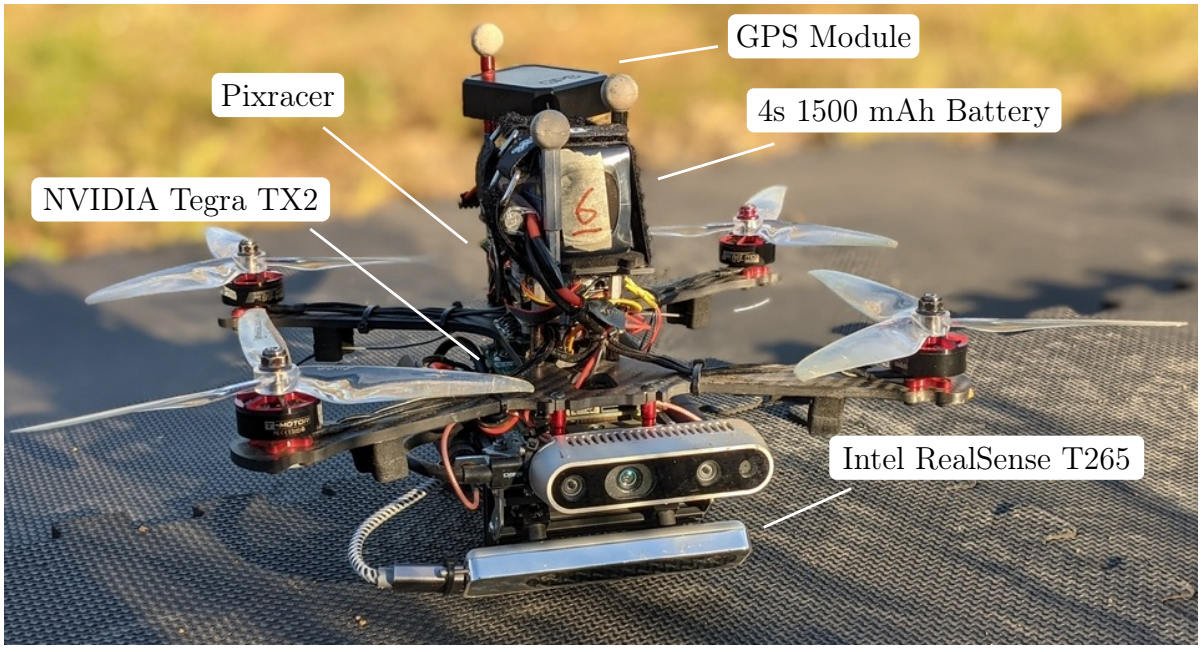


Figure 6.1: The 1.1 kg quadrotor used in the outdoor field experiments has a thrust to weight ratio of over 4 and a maximum flight time of near 6 minutes. Onboard are the Tegra TX2 computer for position control and model learning, the Intel RealSense T265 for vision-based state estimation, and a GPS unit.

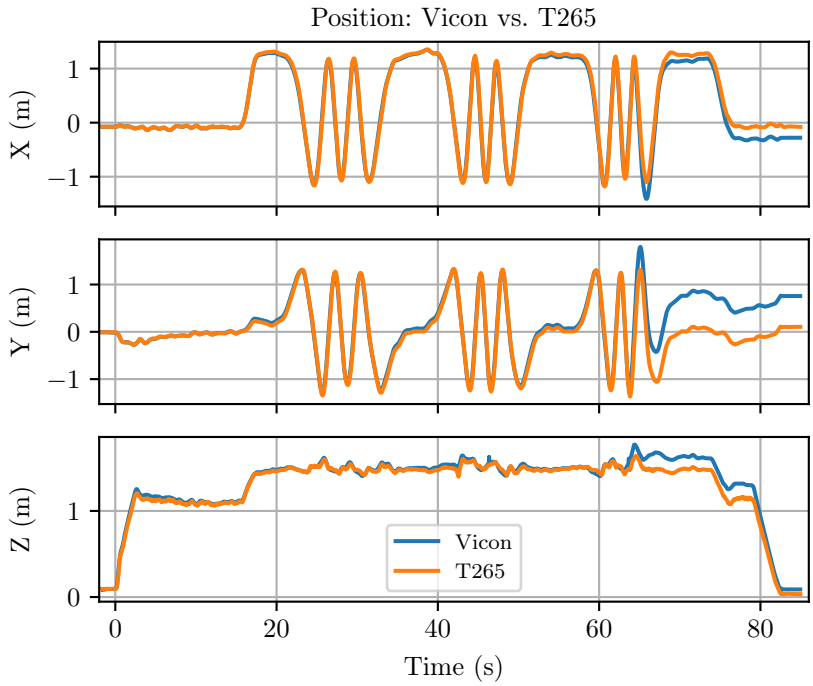


Figure 6.2: T265 vision-based position estimates (orange) compared to Vicon position estimates (blue) while following three aggressive circle trajectories indoors with an added wind disturbance. The trajectories demand speeds of up to 3.5 m/s and accelerations of 9.8 m/s^2. Although experiencing drift, the T265 estimates the position well enough for control purposes.

6.2 State Estimation

To provide vision-based state estimates, an Intel RealSense T265 tracking camera is installed on the vehicle, along with a GPS unit. While the T265 provides high rate state feedback, its performance suffers during high speed and high acceleration trajectories.

Fig. 6.2 shows the estimated vehicle positions of both the T265 and Vicon as the vehicle executes an aggressive circle trajectory three times with an added wind disturbance in the arena. The trajectories reach speeds of up to 3.5 m/s and accelerations of 9.8 m/s^2. Despite the large accelerations demanded, the T265 performs well enough for its use in the control loop.

Fig. 6.3 shows the T265 performance vs. GPS during a high speed outdoor trajectory that reaches speeds of 12 m/s and accelerations of 8 m/s^2. In this flight, the T265 estimate drifts to a degree that renders it infeasible for control. Fig. 6.4 shows the resulting flight path of the vehicle. The poor performance of the T265 is likely a result of both motion blur in the camera leading to a lack of features, as well as the larger vibrations from the higher RPMs during fast outdoor flight. A big drawback to black-box modules such as the T265 is that diagnosing failure is very difficult. A certain way to improve these vision-based estimates would be to use a notch filtered IMU, as in Appendix A.2.2, in

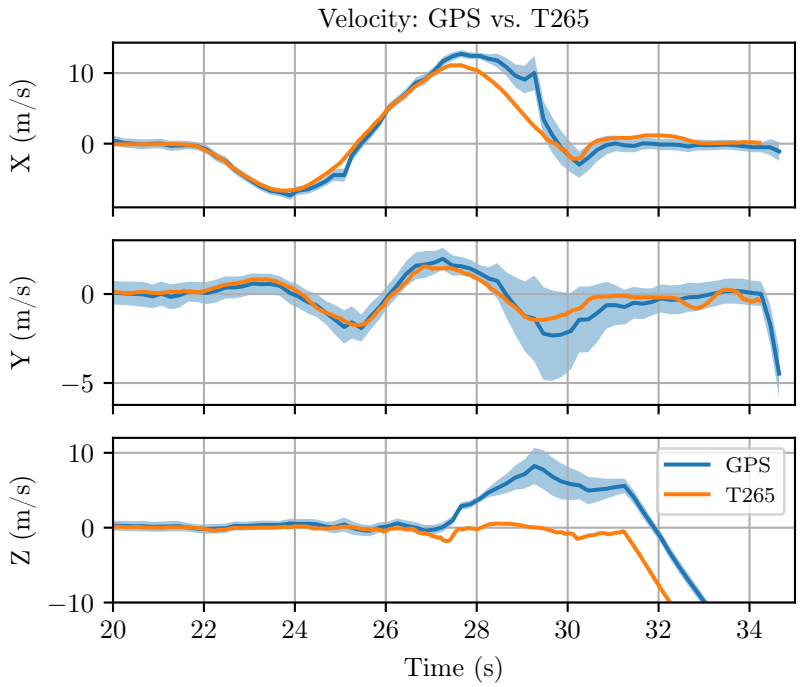


Figure 6.3: T265 vision-based velocity estimates (orange) compared to GPS velocity estimates (blue) during a fast outdoor flight, reaching a speed of 12 m/s and an acceleration of 8 m/s^2. The T265 experiences catastrophic drift starting at around the 27 s mark with the z-velocity estimate, leading to an eventual crash. The GPS accuracy bounds are shown in shaded blue.

Figure 6.4: The vision-based estimates from the T265 experience extreme drift in the z-axis, which leads to a deviation from the desired trajectory and eventual crash.

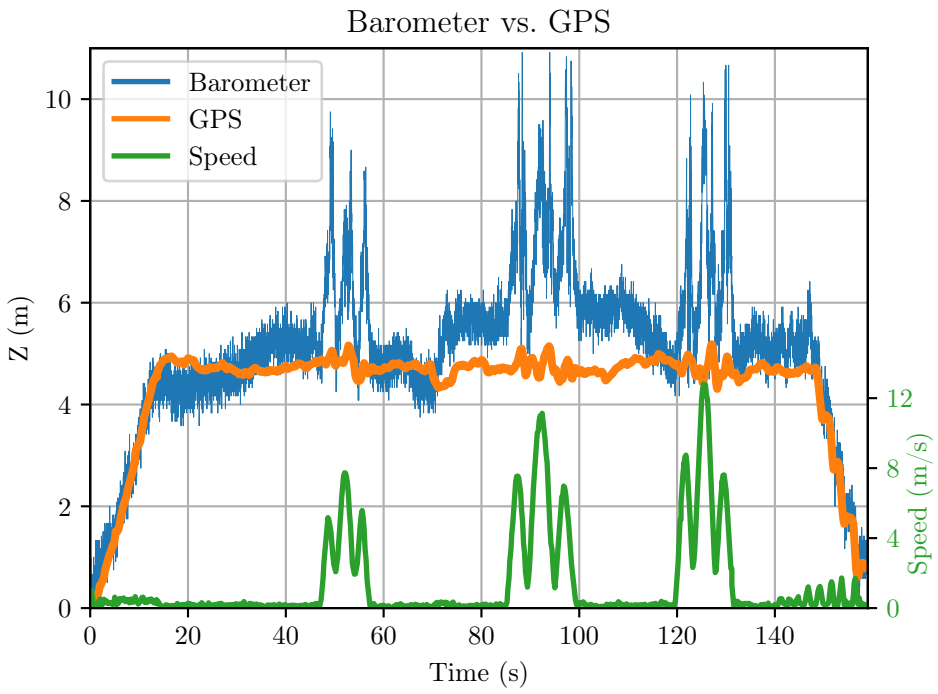


Figure 6.5: Barometer estimate of z position (blue) vs that of GPS (orange) during three fast outdoor figure 8s. The barometer estimate is coupled with the speed of the vehicle (green) due to an increase in air pressure.

the optimization. This would alleviate many of the issues arising from vibrations and subsequent noise in the IMU readings, which leak into the state estimate and potentially cause optimization failure. Unfortunately, the T265 uses an internal IMU that cannot be replaced.

These results show that while the vision-based estimates from the T265 may work during aggressive trajectories in some situations, such as indoors, the T265's use as the primary state estimation source for control during fast trajectories outdoors is too risky and leads to crashes.

GPS too has its shortcomings. Most GPS solutions are more precise in the horizontal component than in the vertical. This can lead to poor altitude control, causing crashes into the ground or an escape from communication range. Air pressure can be used as a secondary estimate of altitude, but not without introducing additional complexity. Figure 6.5 shows an estimate of z position derived from the vehicle's onboard barometer, compared to that of GPS. In addition to the poor precision of the barometer, its estimate of z position is corrupted during periods of high vehicle speed. It may be possible to learn a sensor model of the barometer to remove the bias introduced as a function of the speed. While this is an interesting direction for future work, we deemed it too complex for the purposes of our experiments.

Ultimately however, the trajectories executed during these experiments have a desired height well above the typical error bound of the GPS altitude, and thus crashing into

Table 6.2: Control gains used in the outdoor experiments

Gain	**T265 (vision)**	**GPS**
Position X, Y	1.47	0.4
Velocity X, Y	2.16	1.8
Position Z	3.0	2.2
Velocity Z	3.0	2.5
Attitude X, Y	160	160
Angular Velocity X, Y	14	14
Yaw	20	20
Angular Velocity Z	5	5

the ground is not likely. Issues due to communication range were only seen when the vehicle is out of line of sight and not when the vehicle flies a few meters above its desired altitude.

One trick to improve the vehicle's vertical control performance is to use integrated GPS velocity as the z-axis position state feedback instead of the GPS altitude reported by the receiver. The altitude reported from the GPS receiver attempts to be absolutely correct, while the velocity reported is an additional measurement unaffected by jumps in the altitude estimate. In this way, absolute accuracy can be sacrificed for more precision. Although this can be done for all three axis, we choose to use integrated GPS velocity only for the z position in these experiments, as position estimate jumps were most severe in the vertical direction.

A second drawback of using GPS is the low data rate. The GPS receiver onboard the vehicle returns a GPS solution at 5 Hz. This poses a number of challenges for both control and model learning.

1. Control gains must be lowered. Flying the vehicle with the control gains used for T265 flight is not possible due to instability. Table 6.2 shows the control gains used for both state estimation configurations.

2. The position controller must combine high rate trajectory information with low rate state feedback. It is important to compute error feedback with respect to the trajectory only at times at which state information is available. Computing the error at times that lie in between state updates requires state prediction due to the changing desired state. For the experiments in this chapter, we only compute error at the times the state is available and keep it constant while the remaining higher order trajectory desires are sampled at the attitude control command rate (100 Hz).

3. Model learning must learn accurate models with 1/20'th the available data. Model learning's strength lies in averaging out the noise after assigning noisy points to the state space. With less data, the learned model variance will be higher and thus potentially inaccurate.

Figure 6.6: The outdoor field testing location used to validate model learning at high speed.

When GPS is used for position and velocity feedback, yaw feedback is still provided by the T265.

6.3 Safety

In order to ensure safety while flying at high speeds, a means of commanding the vehicle via radio control (RC) is developed. First, the vehicle's motors can immediately killed via the flip of a switch on the RC transmitter. Second, the RC allows an operator to switch control of the vehicle from the onboard computer, which is following trajectories autonomously, to the RC using the two control sticks. The RC control mode uses the same onboard attitude controller, but the attitude commands are computed from the incoming RC data. Since a yaw estimate is not available on the firmware, absolute yaw control is turned off in RC mode and the only the angular velocity around the body z-axis is controlled.

6.4 Experiments

Prior flights for validating model learning strategies have taken place in simulation and inside a 5 m by 4 m by 3 m caged Vicon arena. While simulation is useful for validating theory and large scale data analysis, drawing conclusions about the practical usefulness of an approach from simulation is difficult. Due to the limited size of the caged arena, the speed and acceleration attainable by the vehicle inside is limited.

Table 6.3: ISSGPR parameters used for the four outdoor trajectories

Traj.	**Vel. (m/s)**	N	$\boldsymbol{\sigma_n}$	$\boldsymbol{\sigma_f}$	**Length Scales (m/s)**
1	8	50	4.1	1.1	1.0
2	12	50	0.5	1.0	2.5
3	14	50	0.5	1.0	2.5
4	17	50	0.5	1.0	2.5

To validate the proposed approaches in more realistic scenarios, flights are conducted outdoors in a large open field, shown in Fig. 6.6[1]. While the environment is free of obstacles, frequent wind gusts and changing lighting conditions pose additional challenges for control and vision-based state estimation.

The regression technique used for model learning in these experiments is, unless otherwise indicated, ISSGPR with velocity as input and 50 random features. The regression is performed online incrementally, using updates to the Cholesky decomposition, and an initial Tikhonov regularization factor of σ_n^2. When linear regression is used, it is performed incrementally in the same manner with the same initial regularization factor. Table 6.3 shows the hyperparameters used by ISSGPR for the four trajectories tested:

- N, the number of random features
- σ_n, the square root of the Tikhonov regularization factor, used to initialize the Cholesky decomposition of the data matrix inverse
- σ_f, the signal standard deviation of the Gaussian process kernel function, scales the output of the random Fourier features
- Length scales, bandwidth of the input features, inversely scales input (velocity in our experiments)

In Section 6.7, an offline model, denoted SSGPR, is learned using the same parameters as that used by ISSGPR on the vehicle. All experiments shown in this chapter consider first and second-order disturbance dynamics to correct the desired angular velocity and desired angular acceleration, as described in Chapter 3.

6.5 Trajectories

While previous flights in the Vicon arena have used aggressive trajectories derived using simple polynomials of short duration, outdoor flights require longer trajectories with higher velocities to take full advantage of the available space and to experience larger disturbance forces due to drag. To generate these trajectories, we use the method from [77] as implemented by the open source package at https://github.com/ethz-asl/mav_trajectory_generation.

We generate elongated figure 8 trajectories reaching speeds of 8 m/s, 12 m/s, 14 m/s,

[1]Special thanks to Bob Bittner and Jake Lammott from NREC for the Gascola field testing site.

Table 6.4: Parameters used to generate the outdoor trajectories

Traj.	**Time Penalty k_T**	**Vel. Constr. (m/s)**	**Acc. Constr. (m/s^2)**
1	1200	10	6
2	200	15	6
3	600	16	9
4	2200	20	12

Table 6.5: Outdoor trajectories, their higher order derivatives, and the relative speed error improvement of our model learning method when compared to the adaptive baseline controller.

Traj.	**Span (m)**	**Vel. (m/s)**	**Acc. (m/s^2)**	**State Est.**	**% Err ↓**
1	20	8.4	6.1	Vision, GPS	60.7
2	40	11.8	6.1	GPS Only	41.0
3	40	14.3	9.1	GPS Only	34.3*
4	40	17.0	12.4	GPS Only	4.1

*Excluding transient first trial.

and 17 m/s. A visualization of the 20 m 8 m/s figure 8 is shown in Fig. 6.7, while the three 40 m figure 8 trajectories are shown in Fig. 6.8.

While the fast trajectories were attempted using both T265 and GPS control, only GPS provided reliable enough state estimates to perform the experiments. As a result, for all but the slowest 8 m/s outdoor trajectory, we report results using GPS control only.

The varied parameters used to generate these trajectories are provided in Table 6.4 while those common to all, as well as more details about the method used, are given in Appendix C.

6.6 Results

A high speed outdoor figure 8 trajectory reaching speeds of up to 8 m/s is executed first with a baseline adaptive control method and then using the proposed acceleration model learning strategy. A visual overlay of this trajectory is shown in Fig. 6.7. The speed, position error and speed error for this experiment are shown in Figs. 6.9, 6.10, and 6.11. While the baseline method roughly attains the desired speed, with a mean absolute error of 0.33 m/s, adding model learning improves the error by 61%, bringing it to 0.13 m/s.

Figs. 6.12, 6.13, and 6.14 show the speed for both the baseline method and model learning for the three faster figure 8s. In all cases, model learning improves speed tracking performance. However, the performance improvement is minimal for the 17 m/s trajectory, likely due to poor model fit.

A summary of the percent speed error improvement for all four trajectories is provided in Table 6.5 and an error comparison to the baseline is shown in Fig. 6.15. Two ob-

Figure 6.7: Overlay of the 8 m/s 20 meter figure 8 trajectory at Gascola, executed using GPS control.

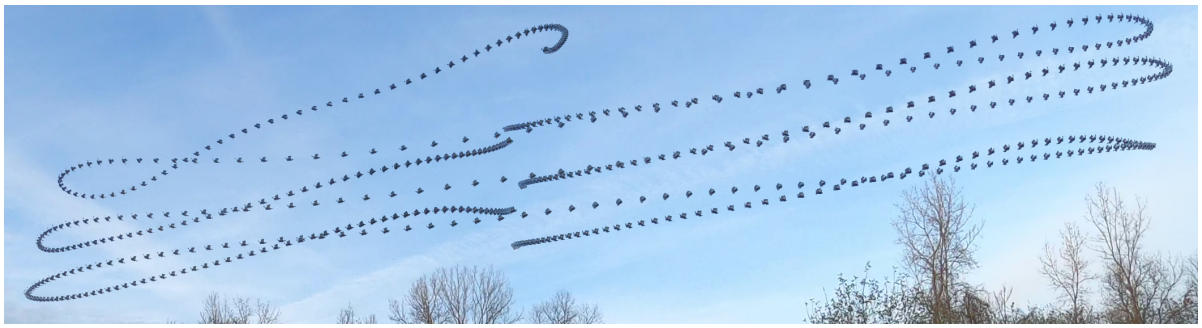


Figure 6.8: Overlay of three high-speed 40 m figure 8 trajectories at Gascola, executed using GPS control. The bottom-most trajectory was executed first, with the middle second, and the topmost third. Note the large vertical drift and comparatively lower lateral drift.

servations are worth highlighting. First, the error incurred by both methods increases with increasing trajectory speed. This is unsurprising, as higher speeds come with higher drag disturbances on the vehicle. Second, the variance in the error of the model learning method is higher than that of the baseline adaptive method. Model learning reduced the bias in the error, but increased the variance. This is likely a result of the incremental model learning performed and can be eliminated, at least to some degree, by holding the model fixed during execution of the trajectory.

6.7 Learned Model Analysis

Figure 6.16 shows the learned model output for each of the six iterations of the 20 m figure 8 overlaid onto the time axis. Since the model is learned online, the first iteration (red) shows some oscillations. The remaining five however, track the mean of the raw disturbance data (blue) well. This improved prediction error corresponds well to the improved position and speed tracking performance shown in Figs. 6.10 and 6.11.

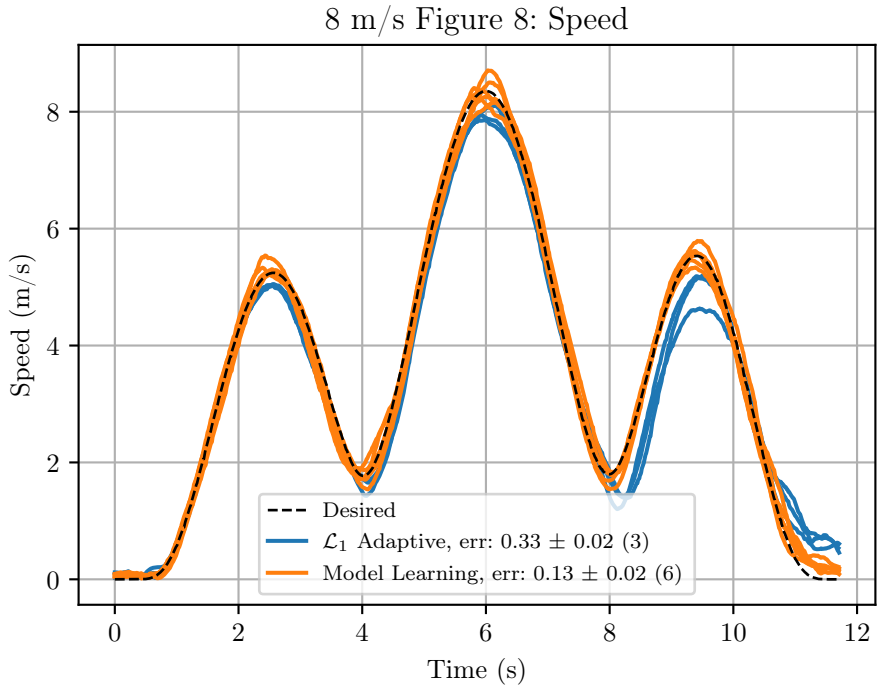


Figure 6.9: Speed for both the baseline and the proposed model learning method during the 8 m/s outdoor figure 8 trajectory.

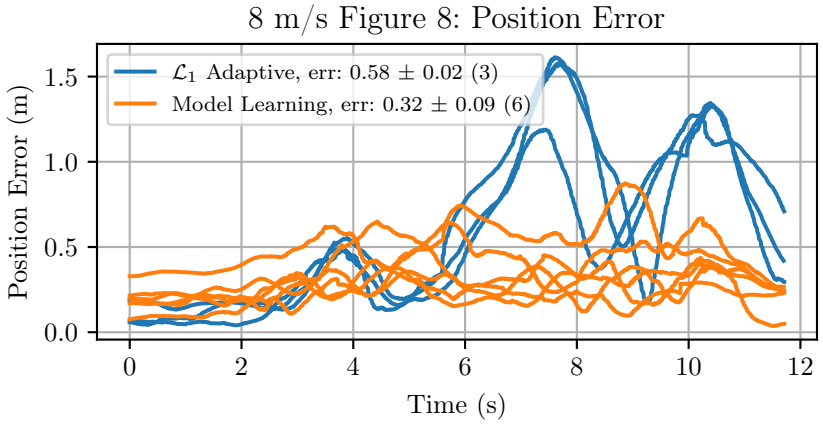


Figure 6.10: Position error for both the baseline and the proposed model learning method during the 8 m/s outdoor figure 8 trajectory.

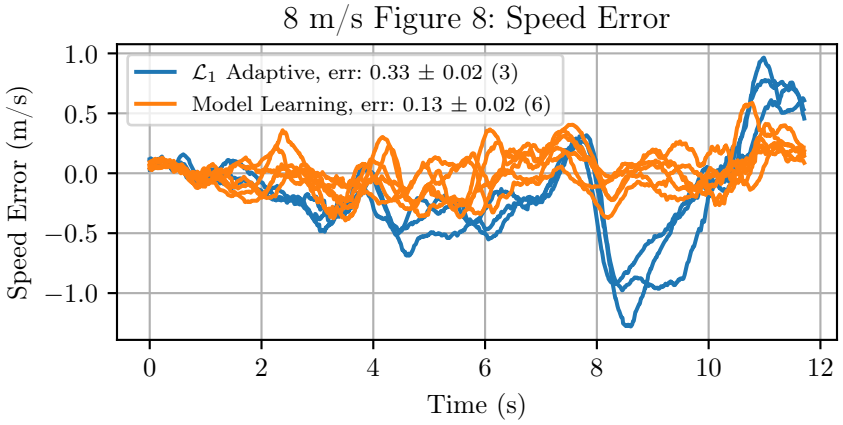


Figure 6.11: Speed error for both the baseline and the proposed model learning method during the 8 m/s figure 8 trajectory.

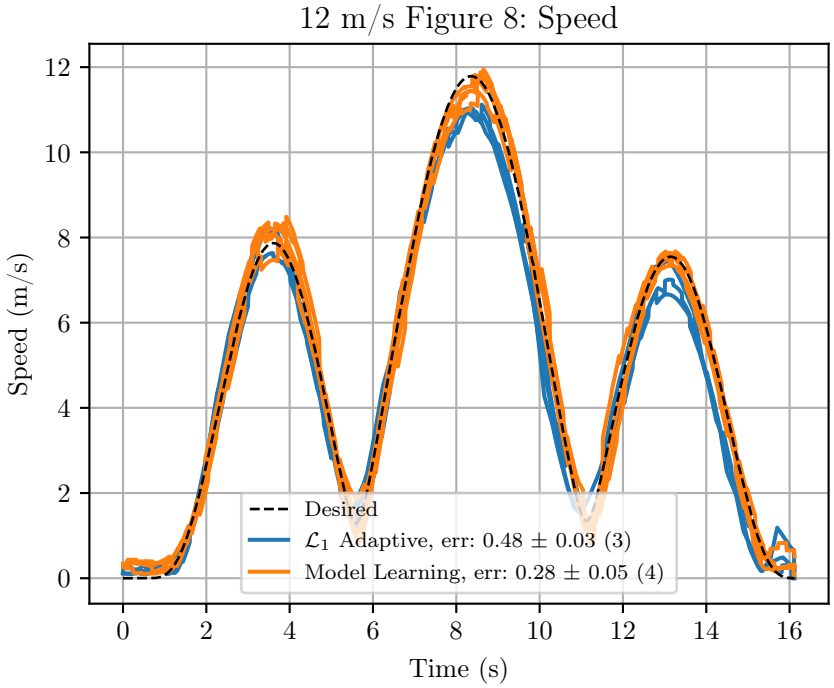


Figure 6.12: Speed for both the baseline and the proposed model learning method during the 12 m/s figure 8 trajectory.

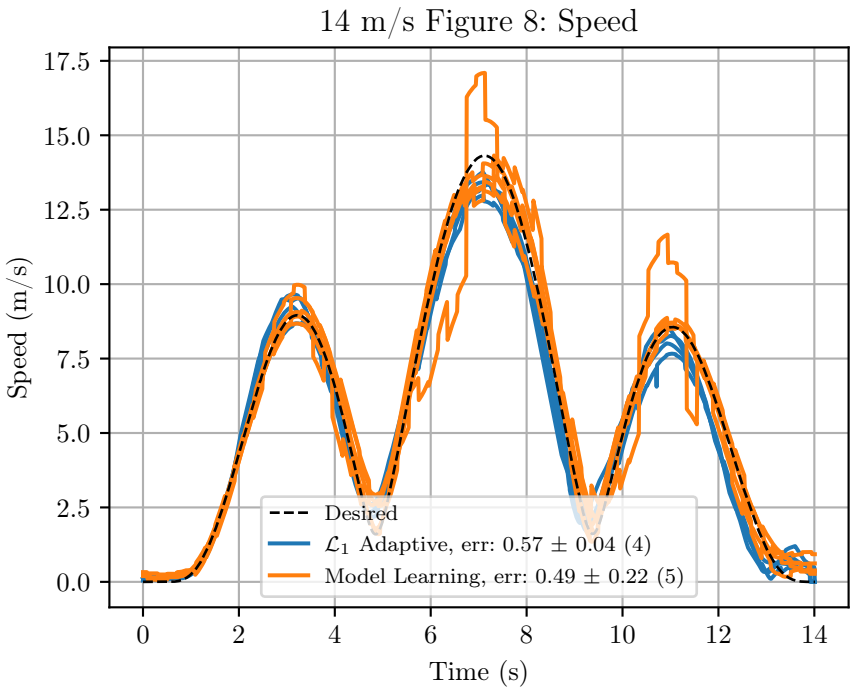


Figure 6.13: Speed for both the baseline and the proposed model learning method during the 14 m/s figure 8 trajectory. The first iteration has poor performance since the model has not seen this trajectory before. The average speed error for the remaining four trajectories is 0.38 m/s ± 0.05 (4).

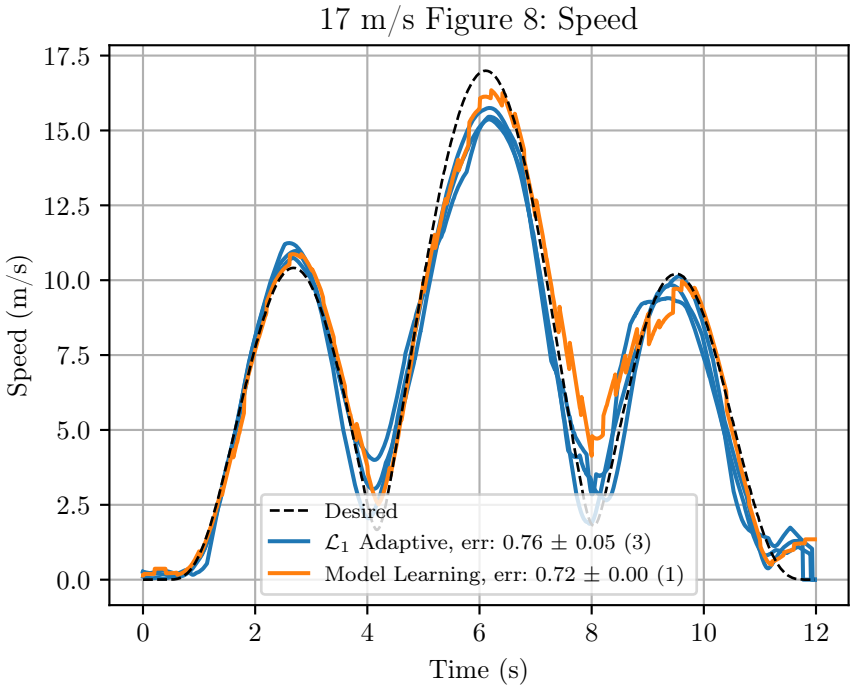


Figure 6.14: Speed for both the baseline and the proposed model learning method during the 17 m/s figure 8 trajectory.

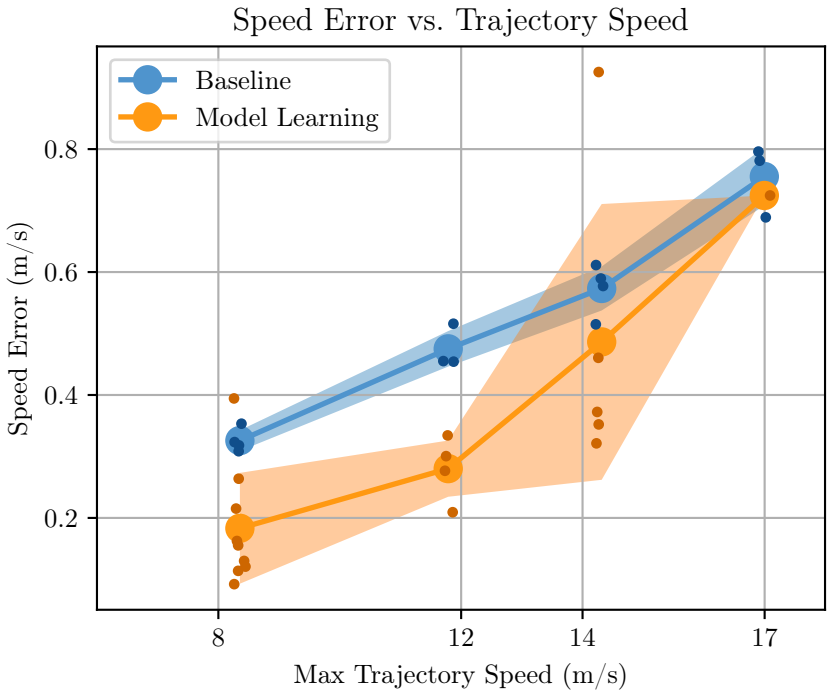


Figure 6.15: Speed error summary for the baseline (blue) and model learning (orange) on all four high-speed figure 8 trajectories. Individual trials are small dots, while the mean and standard deviation over all trials are the large dots and shaded regions. Model learning reduces the error for the 8 m/s, 12 m/s, and 14 m/s trajectories.

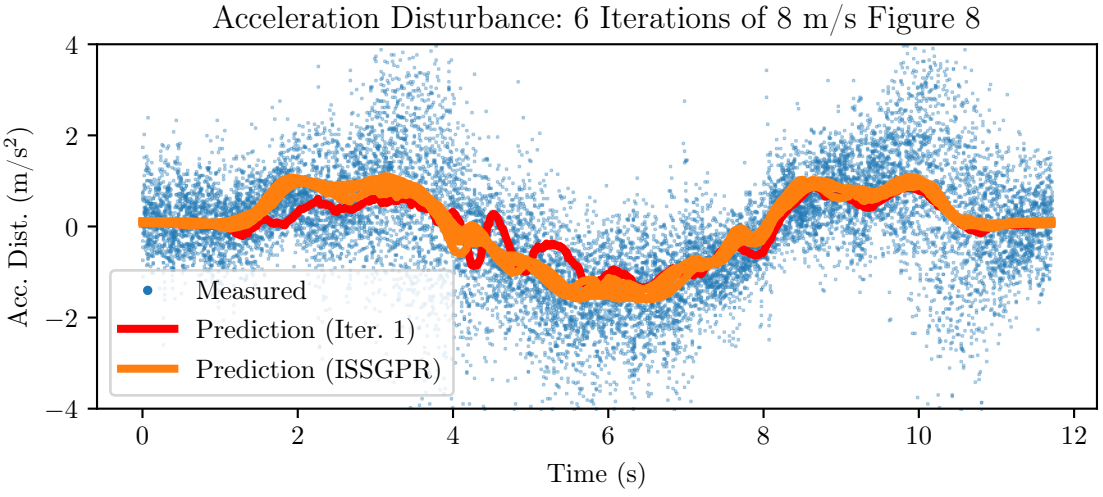


Figure 6.16: The learned model prediction (orange) and the raw disturbance data (blue) during the six executions of the 20 m figure 8.

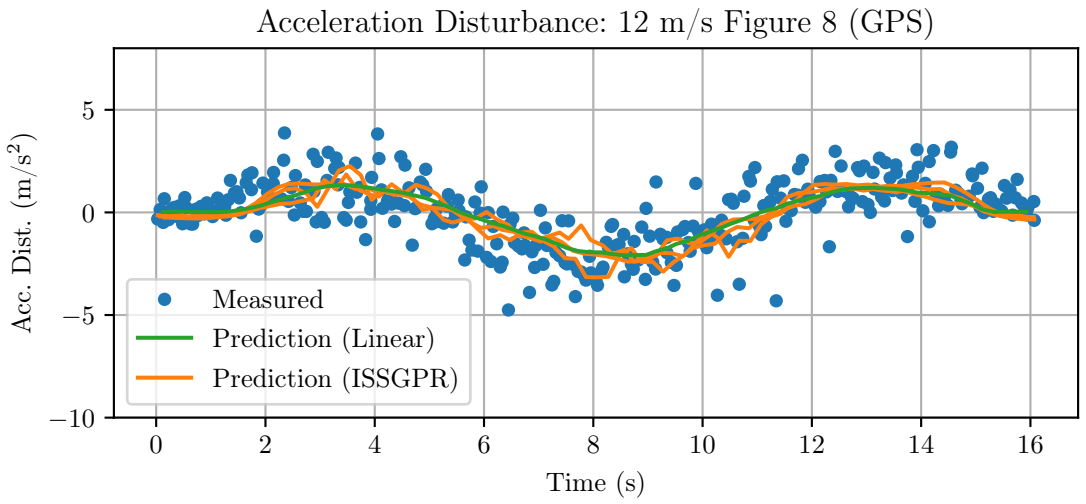


Figure 6.17: The learned model prediction from ISSGPR (orange), a linear model (green), and the raw disturbance data (blue) during the 12 m/s 40 m figure 8 trials.

Figures 6.17, 6.18, and 6.19 show the learned models for the three 40 m figure 8s that use GPS control. While the two slowest figure 8s show good model agreement with the raw data, the sparsity of the data leads to a poor model fit for the fastest trajectory. This is likely the reason for the poor control performance improvement from model learning seen in Fig. 6.14. Also note the relative sparsity of the data relative to Fig. 6.16. While the T265 provides data at 200 Hz, the GPS module only provides data at 5 Hz, making model learning more difficult. Despite this, the incremental regression is able to capture the mean of the disturbance well, without destabilizing oscillations, for the 12 m/s and 14 m/s trajectories. The results at 17 m/s highlight the difficulty in learning dynamical models with data sparsity and with only a single trajectory execution.

Figure 6.20 shows the acceleration disturbances felt by the vehicle as a function of maximum trajectory speed, along with the residual disturbance after learning offline linear and SSGPR models. First, we notice that the acceleration disturbance without a learned model increases with increasing trajectory speed. As mentioned above, this is due to the stronger air resistance acting on the vehicle. Second, we see that the proposed SSGPR model learning technique is able to significantly reduce the disturbance on the vehicle at all four tested speeds. Finally, we note that the linear model is competitive for the three slower speeds, but does not perform as well for the fastest speed, 17 m/s. This indicates that the nonlinearity of the disturbance due to the drag force increases with trajectory speed, and is significant at speeds at or greater than 17 m/s, which is consistent with the conventional model of drag $\sim$ the square of the velocity.

6.8 Conclusion

The presented outdoor flight experiments show how model learning can improve control performance for aggressive trajectories outdoors with imperfect state estimation. A number of issues and questions remain however:

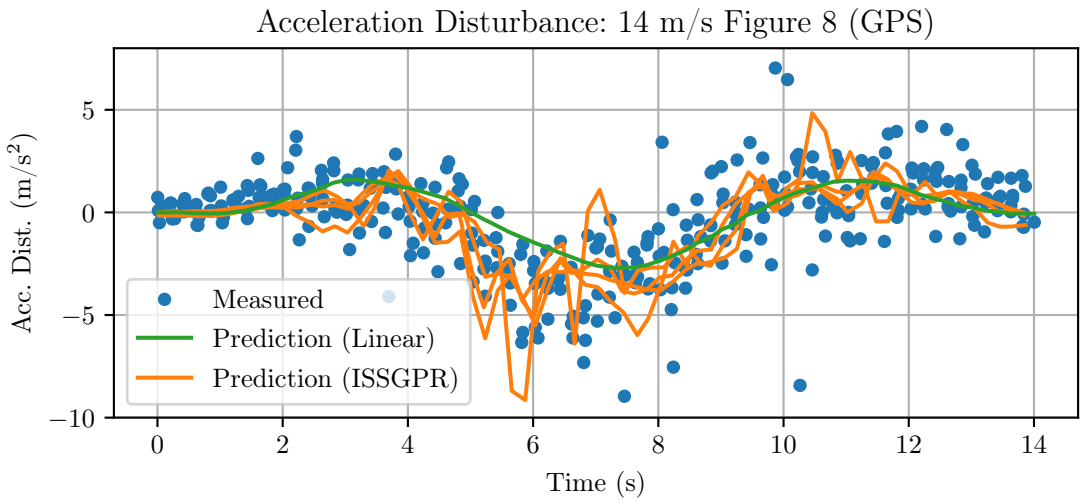


Figure 6.18: The learned model prediction from ISSGPR (orange), a linear model (green), and the raw disturbance data (blue) during the 14 m/s 40 m figure 8 trials. The oscillatory prediction corresponds to the large error trial from Fig. 6.13.

- The learned model output can be sensitive to the regression hyperparameters. Figure 6.18 shows that the linear model output is smoother than that of ISSGPR. Hyperparameter optimization of the ISSGPR length scales can perhaps alleviate this issue somewhat.
- The model learning performed for these experiments was done incrementally with each incoming data point and single-step differentiation of the vehicle state to obtain the measured acceleration. There are intermediate possibilities between this single-step model learning and offline model learning that may provide a better bias/variance trade-off, particularly when the state estimate is noisy or low rate, as is the case when using GPS data. For example, vehicle states can be non-causally smoothed using advanced filtering algorithms to reduce the noise of the regression algorithm input. This cannot be done for adaptive methods or feedback control, where there is no "memory" or model used.
- *How important are the disturbance dynamics?* Chapters 3 and 4 showed that considering disturbance dynamics improves control performance when using a learned disturbance model. While this also likely holds outdoors, it has not been verified experimentally using an ablation study for the configurations tested in this chapter. A reasonable hypothesis is that disturbance dynamics are most relevant when rapidly accelerating or decelerating, so that the drag force disturbance acting on the vehicle is rapidly changing. Further, their relevance may be determined through inspection of the magnitude of the learned model's first and second derivatives, and how that propagates to changes in the vehicle's attitude references and control inputs.
- *How can an accurate dynamics model alleviate the burden of state estimation?* A primary reason to learn a more accurate model is to follow trajectories with

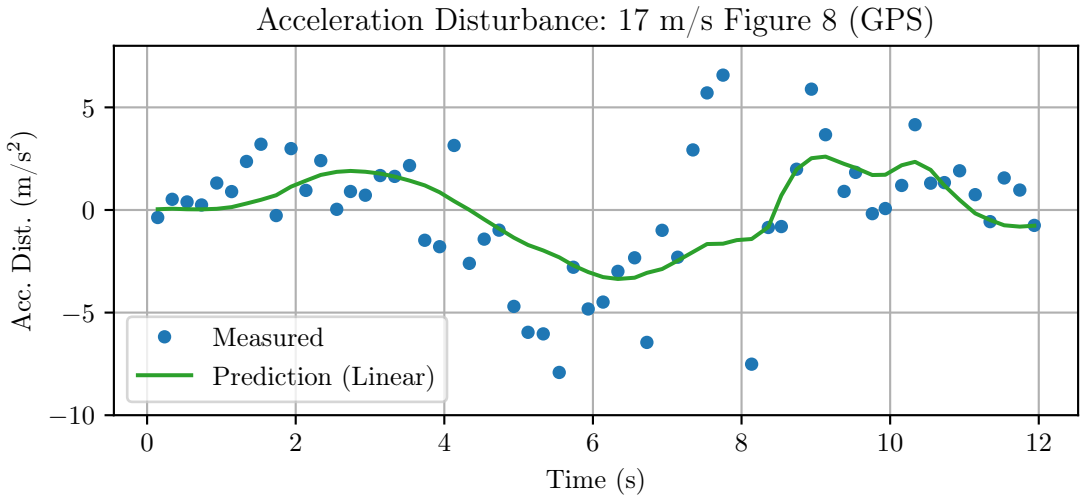


Figure 6.19: The learned model prediction from a linear model (green) and the raw disturbance data (blue) during the 17 m/s 40 m figure 8.

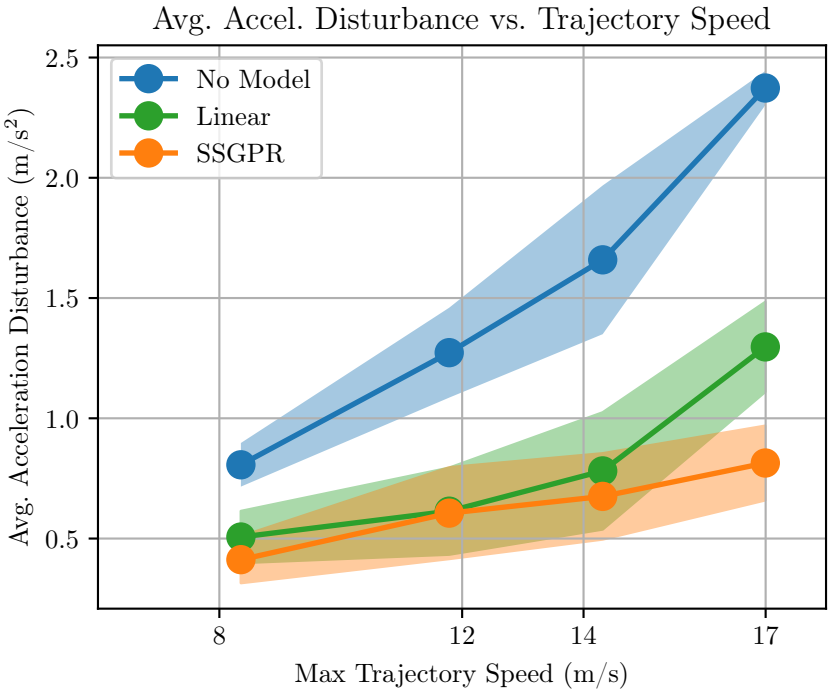


Figure 6.20: The average acceleration disturbances experienced along the outdoor trajectories (blue), the remaining disturbance after applying a linear model (green), and the remaining disturbance after applying Sparse Spectrum Gaussian Process Regression (orange). Standard deviation over trajectory executions is denoted by the shaded regions. The importance of nonlinear model learning increases as trajectory speed increases.

less error. A secondary reason may be to reduce the reliance on precise state estimation by the lowering of control gains. As the presented experiments show, both accurate and precise state estimation outdoors at high speed remains difficult. Model learning allows a technique to aggregate large amounts of data to learn accurate models, and subsequently lower control gains to achieve similar trajectory tracking performance but with less susceptibility to state estimate noise and jumps. Through careful theoretical analysis, quantitative relationships should be derived between dynamical model accuracy, trajectory tracking error, state estimate noise, and control and filter gains. We believe this "model learning-as-feedback" strategy to be a promising area for future study.

CHAPTER 7
Conclusion

Quadrotors are agile vehicles whose potential as of yet remains somewhat untapped. In this thesis, we have strived to enable more accurate aggressive flight in nonideal conditions arising from external disturbances, poorly modeled dynamics, and state estimation difficulties. While our model learning strategies have improved control performance in most of the scenarios tested, the problem of flying a vehicle with unknown dynamics in unpredictable conditions remains largely unsolved. Here, we outline a summary of the contributions made in this thesis and conclude with promising directions for future work.

7.1 Summary of Contributions

In Chapter 3, we highlight the importance of disturbance dynamics and introduce a method capable of inverting learned models that capture disturbance dynamics of large magnitude. We also augment the method with the ability to invert models that depend on the vehicle control inputs using efficient online nonlinear root finding. We show the performance improvements arising from using the method in both simulation and hardware experiments.

In Chapter 4, we take steps to render the feedback linearization controller for quadrotors feasible for aggressive flight in uncertain conditions. First, we augment the feedback linearization controller with an analogous acceleration disturbance model learning strategy to that from Chapter 3 and show how to propagate the disturbance dynamics to the attitude controller. We show that this controller is efficient, provides a better error response than traditional quadrotor attitude controllers, and validate it on a hardware quadrotor in a motion capture arena. We then provide a way to reduce the detrimental impact of motor delay while also reducing the order of the dynamic feedback from two to one. We show that this delay model improves step responses for the feedback linearization controller on the hardware platform. While we have not yet demonstrated improved trajectory tracking performance relative to a traditional cascaded approach on a hardware platform, we believe the feedback linearization approach is viable and should be explored further.

Chapter 5 details the development of model learning for quadrotor attitude controllers. We show the synergistic relationship between linear acceleration disturbance compensation and angular acceleration disturbance compensation through challenging experiments

on a hardware platform.

Finally, in Chapter 6, we validate our approach in challenging outdoor conditions. We show that our model learning method works with a degraded and low rate state estimate and in an environment with unpredictable disturbances coming from wind gusts.

7.2 Limitations and Future Work

Many areas neighboring to this topic are left unexplored by this thesis and the limitations that result can transition nicely into future work.

7.2.1 Learning Models

The incremental model learning presented does not consider data forgetting. Every additional data point affects the learned model's parameters less than the previous, eventually leading to a model that is sluggish to adapt. While this is adequate for a few short flights, longer autonomous operation requires a strategy for handling this. A simple sliding window approach, combined with rank-one *downdates* of the inverse data matrix, can be used with incremental linear regression to forget old data. More complicated data management strategies can be used to ensure that the model remains well-defined and non-singular in all relevant directions, not unlike the strategies used for keyframe retention and node marginalization in visual-inertial state estimators.

The regression strategy used is designed to deal with noise in the output points, e.g. the measured acceleration. However, since the model input is the vehicle state, the regression strategy should also consider probabilistic input points by design [58]. In linear acceleration learning for quadrotors, the model output is acceleration, which is computed by differentiating the vehicle velocity. This presents a further difficulty, as the noise in the output (acceleration) is dependent on the noise in the input (velocity). Extending the probabilistic regression framework to handle this noise coupling between the data input and the data output may improve the model accuracy.

While this thesis explores learning corrections to both linear and angular dynamics, there are a number of components where structured calibration remains essential for quadrotors and robotics systems in general. For example, the quadratic relationship between motor speed and force produced is characterized using a specialized rig offline. In theory, this relationship can be subsumed by both the linear and angular non-parametric components, but it may be easier from a learning perspective to inject some domain knowledge or structure into the regression. Gradient descent can be used online during vehicle operation to update the quadratic coefficients.

Minimizing trajectory prediction error is a natural and simple to implement cost function for learning models. However, altering or changing the cost function may result in more accurate models, models that are better for control performance, or both. First, optimizing for model multi-step prediction accuracy has been shown to improve both multi-step predictions and control performance, and can be done efficiently [89]. Second,

optimizing tracking error directly with respect to the model parameters can provide better convergence than optimizing prediction error [76]. We believe that exploring the relative strengths and weaknesses of prediction error and tracking error as a signal for model optimization can provide insights. For example, an optimization that minimizes a weighted sum of both may be a more flexible cost function.

7.2.2 Using Learned Models

First, the theoretical analysis can be greatly expanded. The connection between model accuracy (or uncertainty) and resulting trajectory tracking performance has not been made fully explicit. Such a result would allow an autonomous vehicle to make much more informed decisions about both trajectory planning and model learning. Active learning [95] could then be used to reach the level of model accuracy required to complete a particular objective. Dynamical models that are aware of their uncertainties can inform planners that may want to prioritize safety, and thus avoid trajectories that take the vehicle into a region of the state space where the dynamics are not well known.

The results in this thesis have compared adaptive control and disturbance estimation methods with model learning methods, but there is no real obstacle to combining the two. In such a system, the adaptive component should handle the unpredictable and time-varying disturbances, such as wind gusts, while the modeling component can handle the predictable disturbances arising from modeling error and other regularities in the environment [41]. In that case, it is important that the dynamical model used in the adaptive disturbance estimator is the same learned model used in the controller. The interplay between a model that is learned incrementally and an adaptive component should be theoretically analyzed for stability and performance. Both incremental model learning and adaptive disturbance estimation methods have parameters that control the speed of their adaptation, and these should be set from a principled probabilistic framework.

Improved dynamical models are useful for more than just control performance. The state estimator process model can be augmented using the learned acceleration corrections to improve the state estimate [69], particularly in situations where state measurements are degraded or unavailable.

7.2.3 Future

In general, the goal of a robotic system that can handle arbitrary degradation gracefully online remains elusive. As a guiding question, one may ask: *what is the best way for a mobile robot to learn how to optimally control itself in an uncertain and dynamic environment?* The role of large scale simulation can be used to meta-learn an agent that can learn in a variety of environments. Such an agent can self-supervise through the goal of minimizing prediction error of its sensor observations, as well as minimizing tracking error for various trajectories. This simulation technique has the advantage of not requiring time-consuming and potentially dangerous hardware experiments and similar techniques have recently been employed with success [46]. With enough variety

in the simulated training data, such learned agents can potentially handle degradation where current more structured approaches cannot. Furthermore, such techniques have the ability to circumvent the need for an accurate state estimator, as the agent can learn to act directly on sensor observations.

Minimizing prediction error as self-supervision with large-scale learning has recently seen tremendous success in the field of natural language processing [14], but has seen slower adoption in robotics. The adaptive robotic systems of the future will leverage the ever increasing power of offline computation to perform at the limits of their capabilities in a larger variety of environments.

APPENDIX

A Firmware and Attitude Estimation

A.1 System

Figure A.1 shows the architecture of the embedded firmware implementation. The system is organized into three threads: (1) the main control thread that samples the IMU, reads from serial, runs the attitude estimator, and runs the controller, (2) the CAN thread that sends commands to the ESCs, and (3) the logger thread that writes diagnostics at a high-rate to the onboard SD card.

A.2 Sensing

The primary sensor available on the firmware is the IMU, which provides accelerometer and gyroscope data along three axes.

Figure A.2 shows an example timeseries of unfiltered accelerometer data from a vehicle at hover. It is clear that the noise dominates over the small changes in the gravity vector during hover, as the peak to peak amplitude is over 2 Gs of acceleration. The same type of noise can be seen in sample gyroscope data, shown in Fig A.3.

As shown in Section 2.1, estimating the angular acceleration of the vehicle from IMU data requires differentiating gyroscope data. Thus, we strive to use filtering methods to extract the smoothest possible signal in real-time. This will be useful for control, in addition to model learning.

A.2.1 IMU Logging

The communication system used to send commands to the robot in flight is a 921600 baud serial link. Recording the available IMU data at 2 kHz only allows streaming 57 bytes per sampling time. Further, any additional burden on this link will impact the latency and bandwidth available for the control commands, which will negatively impact control performance. Since we wish to log IMU data without impacting control performance, we need an alternative system to collect the data.

The vehicle has onboard a high capacity SD card that can be used to store data that is then transferred manually after a flight has completed. However, the standard PX4 logger is not readily configured to log at 2 kHz. Further, since we do not use any of

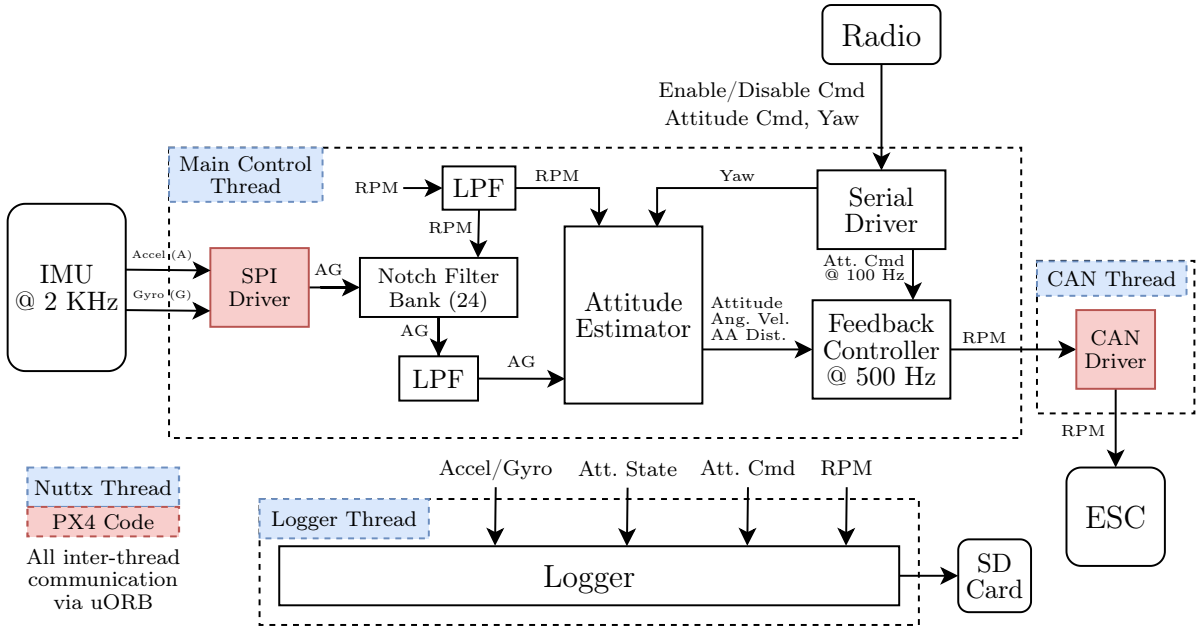


Figure A.1: Pixracer firmware implementation system diagram. Threads are delineated with dashed boxes. Inter-thread communication is done using the uORB PX4 module. All code is in-house with the exception of the SPI IMU driver and the CAN driver.

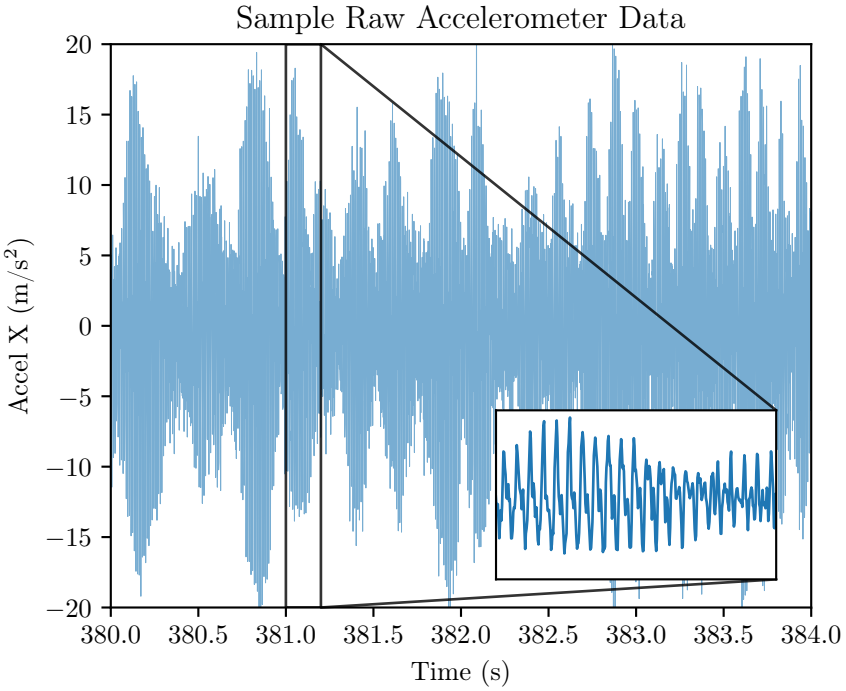


Figure A.2: Raw accelerometer data from a quadrotor at hover.

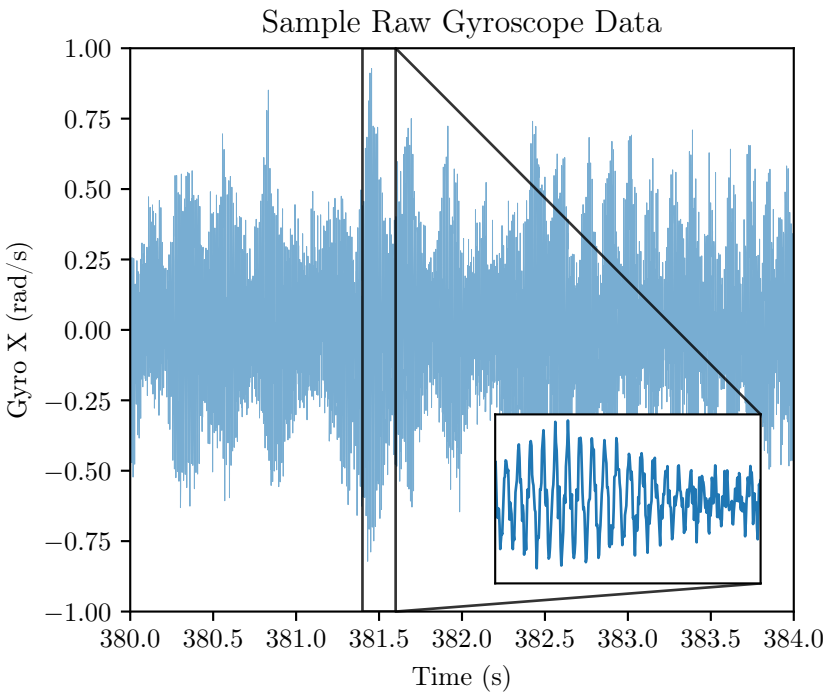


Figure A.3: Raw gyroscope data from a quadrotor at hover.

the standard PX4 modules with the exception of uORB, we do not want to bring in the overburdened PX4 logger module. We implemented a simple logging system that directly interfaces with the `monolith` control module and can record data at 2 kHz.

In order to minimize the writing bandwidth required, IMU data is logged in its original 16-bit representation. In addition, RPMs sent to the ESCs are also logged as 16-bit integers.

The logging binary format is roughly described using the below C structs for IMU and RPM logging. Endianness has not been sensitively treated, although it will be clear if incorrect on the log reading side.

```
struct imu_msg {
  uint8_t msg_type = 1;
  uint64_t timestamp; // Time since firmware boot in microseconds.
  int16_t gyro[3]; // X, Y, and Z gyroscope data.
  int16_t accel[3]; // X, Y, and Z accelerometer data.
};
struct imu_msg {
  uint8_t msg_type = 2;
  uint16_t rpms[4]; // Commanded RPMs of all four motors.
};
```

To avoid filesystem and hardware delays, the Nuttx `write` method is only called after an internal buffer of configurable size has been filled. It is important not to call the `write` method in the control thread, since it may block while I/O operations are completing.

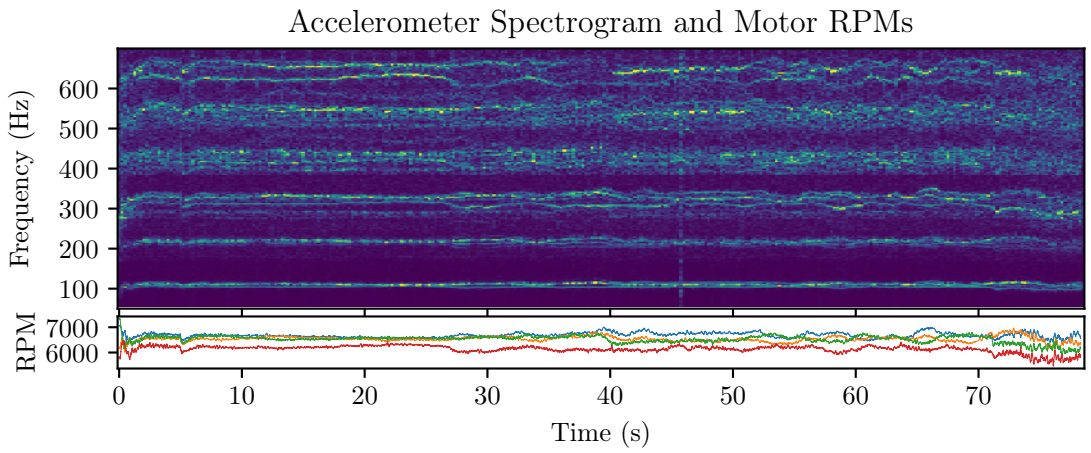


Figure A.4: Spectrogram of x-axis accelerometer data aligned with smoothed desired RPMs from a vehicle at hover. Blue corresponds to low amplitude and yellow corresponds to high amplitude. Note that coloring has been normalized for each of the 6 bands of noise present in the spectrogram.

The large latencies will add too much jitter to the timing of the control commands.

The solution we employ is to call `write` in a separate Nuttx thread, so that the control loop can continue to run while the I/O operation is processed. IMU and RPM values are passed from the main control thread to the logging thread using uORB messages. Despite the dedicated logging thread, the internal buffer size used to control how often `write` is called is still important to avoid timing deadline overruns. A buffer size of 256 bytes was eventually settled on. Both larger and smaller buffers increased the frequency and magnitude of control timing deadline misses. A more rigorous analysis of the logging subsystem may be performed in future work.

A.2.2 Notch Filtering

The periodic nature of the noise present in the IMU data suggests that the noise may be restricted to a narrow frequency band. To visualize this, we compute a spectrogram of the IMU data.

Figure A.4 shows the spectrogram of the x-axis accelerometer aligned with motor RPMs during hover, and displays 6 bands of noise that track very closely with the RPM values. The 6 bands of noise correspond to 6 harmonics of the frequency equal to the RPM of the motors. For example, 6500 RPM is $\sim$108 Hz, and that is approximately where the first band is seen in the spectrogram.

To filter out this noise, we use digital filters that are tuned to narrowly remove specific frequencies from the signal. We can compute the frequencies we want to remove if we know the RPMs of all four motors. Since at this time we do not have high-rate RPM feedback from the ESCs, we resort to using the desired RPM values sent from the Flight Controller. Since the true RPMs cannot instantaneously achieve the commanded RPMs due to motor inertia, we smooth the commanded RPMs before using them to compute

the notch filter frequencies.

Since there are four motors on a typical quadrotor and we see six main harmonics of noise in the spectrogram, we need to filter out 24 different frequencies. We rely on digital biquad filters for high-rate operation on an embedded system. Biquad filters are 2nd order notch filters that filter out one frequency and have five free parameters. In order to filter out each of the 24 frequencies of interest, we need to serially cascade 24 notch filters. Implementing this cascaded filter digitally can be done either by convolving each of the filter coefficients to obtain a single filter of a much higher order, or by applying each of the filters sequentially. Although the former method may be computationally faster, quantization of the coefficients of a very high-order filter can lead to massive instabilities. Thus, the latter is preferred and used here.

Designing the notch filter is out of scope for this thesis. Here we discuss the computation required to compute filter coefficients and implement it on the platform. We also compare notch filter formulations found on standard open-source drone firmware.

Second order biquad filters have the following computational form.

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} - a_1 y_{n-1} - a_2 y_{n-2} \tag{A.1}$$

Here, the coefficients b_0, b_1, and b_2 correspond to the numerator of the filter transfer function and encode its zeros. The coefficients a_1 and a_2 correspond to the denominator of the transfer function and encode the poles. a_0 is implicitly assumed to be one here. Equation (A.1) implements Direct Form I. Although Direct Form II is more efficient and only requires two delay elements[1], it does not allow changing coefficients during operation, as its delay elements require multiplication with the coefficients.

Second order notch filters require specification of two parameters, the desired frequency to filter out, and the desired width of the notch. We know that we need to set the desired frequency to that corresponding to the particular harmonic of a particular motor. The notch width should be wide enough to maximally filter out frequency from that motor at the harmonic, and it should be narrow enough to minimize latency from high frequency changes in the signal not arising from vibrations.

A.2.2.1 Implementation

Let f_s be the sampling frequency of the system, f_0 the desired frequency to filter, and Q the quality factor, which specifies the width of the notch (a higher Q corresponds to a more narrow notch).

The PX4 codebase implements the notch filter as follows.[2]. This matches the implemen-

[1] https://ccrma.stanford.edu/~jos/filters/Direct_Form_II.html

[2] https://github.com/PX4/Firmware/blob/b12a655c5bd39ecf65b8001b6f9256d2a5dcce70/src/lib/mathlib/math/filter/NotchFilter.hpp#L171

Table A.1: Average time to apply 24 notch filters to accelerometer and gyroscope data

Method	**Time (stddev) in µs**
Full trig	130
Poly. approx. trig	82.51 (1.45)

tation of SciPy's `iirnotch`[3], which is derived from [71], pg. 575.

$$\omega = \frac{2\pi f_0}{f_s} \quad \beta = \tan\left(\frac{\omega}{2Q}\right) \tag{A.2}$$

$$b_0 = b_2 = \frac{1}{1+\beta} \quad b_1 = a_1 = -\frac{2\cos(\omega)}{1+\beta} \quad a_2 = \frac{1-\beta}{1+\beta} \tag{A.3}$$

The Betaflight codebase implements the notch filter slightly differently.[4]

$$\omega = \frac{2\pi f_0}{f_s} \quad \beta = \frac{\text{sin\_approx}(\omega)}{2Q} \tag{A.4}$$

$$b_0 = b_2 = \frac{1}{1+\beta} \quad b_1 = a_1 = -\frac{2\cos(\omega)}{1+\beta} \quad a_2 = \frac{1-\beta}{1+\beta} \tag{A.5}$$

Betaflight's implementation does two things differently: (1) β is computed slightly differently and (2) polynomial approximations are used in place of the true trigonometric functions. This will be important to ensuring that filter parameters can be updated at the rate at which RPMs are received. It is important to note that both implementations have identical zero locations and filter out the same frequency. It is the gain, or the width of the notch as a function of the quality factor Q, that differs.

We implement the "more widely used" method from PX4 and [71] and inspect the filtering timing using PX4's `perf_counters` to see the impact of trigonometric approximations. Table A.1 shows the time it takes to apply 24 filters for the full trigonometric functions as compared to the polynomial approximations. Since applying the full trigonometric functions adds a non negligible amount of time (50 µs) to the control loop relative to the total sample time of 500 µs, we use approximate trigonometric tan and cos.

A.2.2.2 Results

Offline We apply notch filtering to a dataset from a quadrotor at hover. We use a quality factor of $Q = 5.0$ and an exponential smoothing time constant of 20.0 for the RPMs.

Figures A.5 and A.6 show the results for the x-axis accelerometer and x-axis gyroscope data. There is a clear reduction in the noise after filtering. The standard deviation from the mean before and after filtering is $6.065\,\text{m/s}^2$ and $0.595\,\text{m/s}^2$ for the accelerometer data and $0.229\,\text{rad/s}$ and $0.067\,\text{rad/s}$ for the gyroscope data.

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.iirnotch.html

[4] https://github.com/betaflight/betaflight/blob/b2e99c66510e364903b02db1624d1ec78c9ca527/src/main/common/filter.c#L107

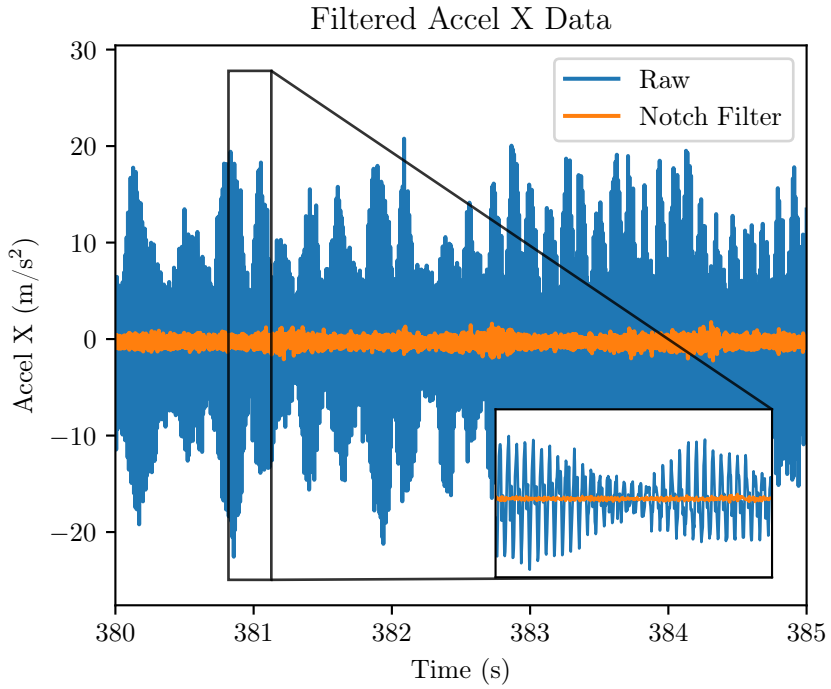


Figure A.5: Filtered accelerometer data from a quadrotor at hover.

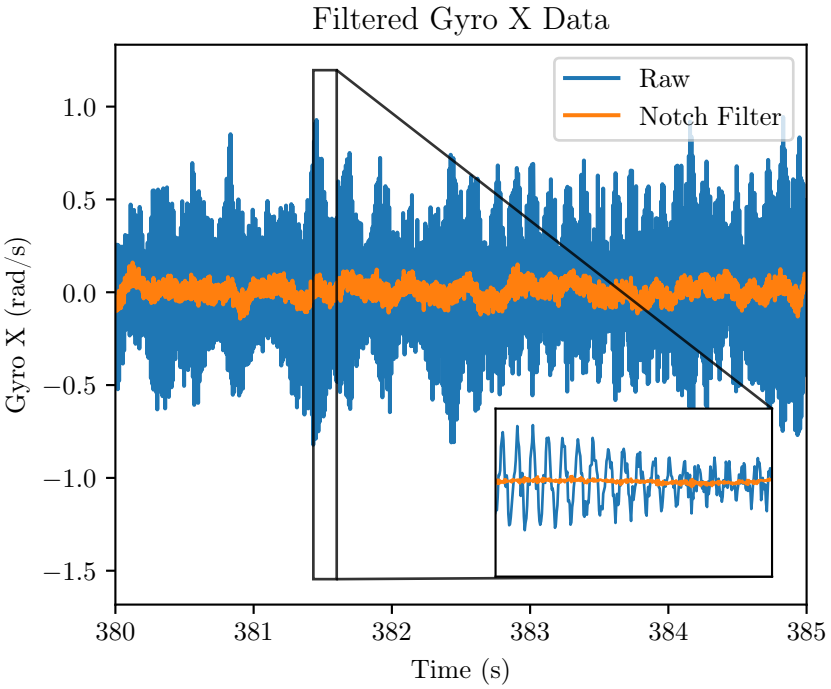


Figure A.6: Filtered gyroscope data from a quadrotor at hover.

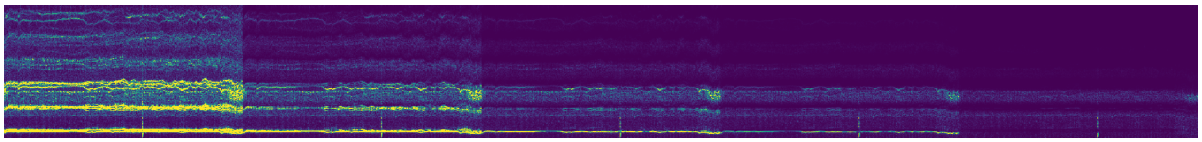


Figure A.7: The spectrogram of the raw accelerometer data is on the left and each successive spectrogram is after notch filtering all six harmonics of an additional motor.

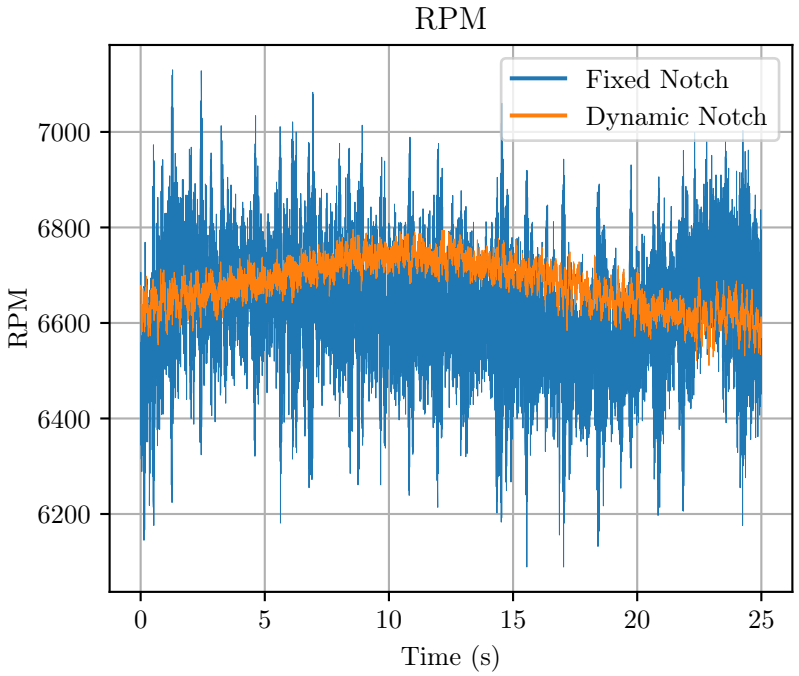


Figure A.8: Comparison between motor speeds at hover with and without the dynamic notch filter.

To visualize the effects of notch filtering at frequencies from *all* four motors and to visualize the precision of the individual notch filtering, we compute spectrograms after filtering of all harmonics for each motors is completed. The results are shown in Fig. A.7. We see that filtering from all four motors is required to completely eliminate the visible noise bands in the spectrogram for the chosen quality factor $Q = 5$.

Online Figure A.8 compares the RPM values for 25 seconds during hover with the old fixed frequency notch and the dynamic notch. Figure A.9 shows the standard deviations. The dynamic filter has drastically reduced the noise in the outputted RPM commands. Figures A.10 and A.11 compare the standard deviations in unfiltered IMU data. Although the standard deviation has decreased along some axes, there does not appear to be a drastic reduction in noise in the unfiltered IMU data. This suggests that the large reduction in RPM output noise has not resulted in a large enough change in vehicle motion to be detected by the IMU.

To show the reduction in noise by applying the dynamic notch filter, we compare the

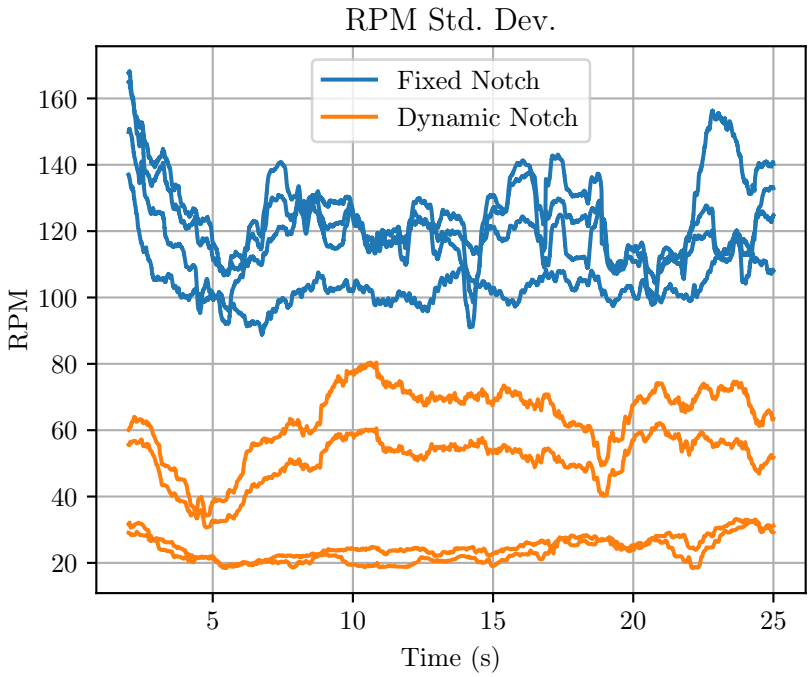


Figure A.9: Comparison between RPM standard deviations over a 2 second window during hover with and without the dynamic notch filter.

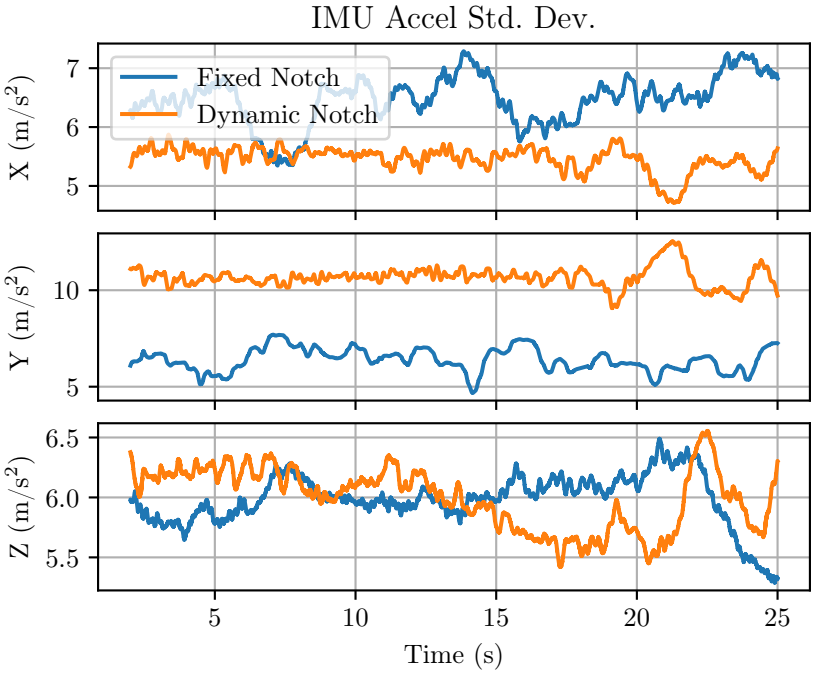


Figure A.10: Comparison between unfiltered accelerometer data standard deviations over a 2 second window during hover with and without the dynamic notch filter.

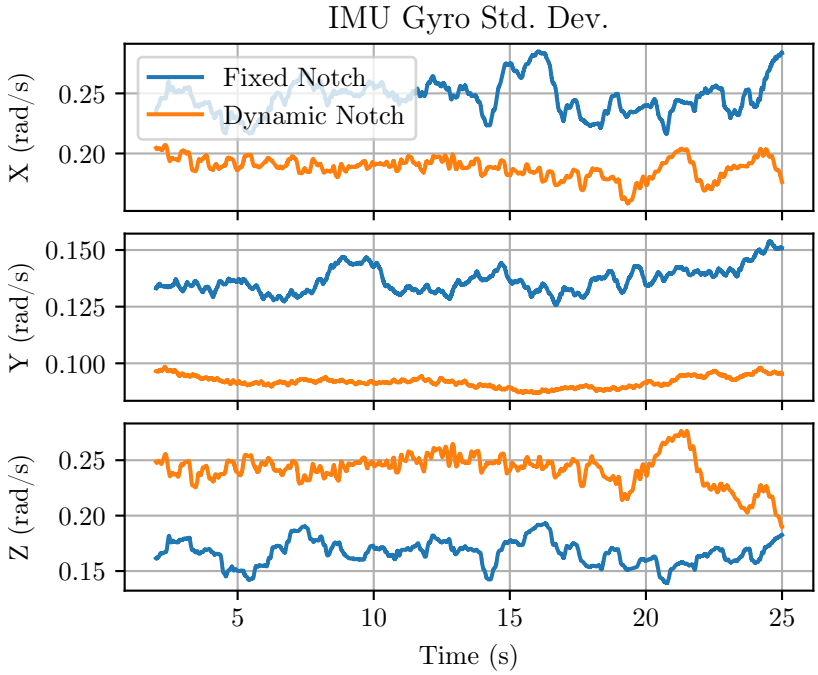


Figure A.11: Comparison between unfiltered IMU gyroscope data standard deviations over a 2 second window during hover with and without the dynamic notch filter.

variance of filtered IMU data with the raw IMU data and the IMU data filtered using 6 harmonics of a static notch. The static notch filter frequency is chosen to match the average RPM for the duration of the vehicle's hover and its Q value is the same as that used in the dynamic notch filter, $Q = 5$. Table A.2 shows the variance in all six IMU axes for all three approaches. Figures A.12 and A.13 show the variance over time.

We see that while static notch filtering substantially reduces the noise of both accelerometer and gyroscope data, dynamic notch filtering further reduces the noise by at least an order of magnitude for the accelerometer and a factor of around 3 for the gyroscope.

Table A.2: Accelerometer (m/s^2) and gyroscope (rad/s) average data variance during a 25 second hover for the raw IMU data, IMU data filtered using a static notch filter, and IMU data filtered using a dynamic RPM-based notch filter

	Raw IMU Data	**Static Notch**	**Dynamic Notch**
Accel X	29.86	1.50	0.09
Accel Y	114.04	5.70	0.09
Accel Z	35.99	2.50	0.68
Gyro X	3.55e-2	0.31e-2	0.11e-2
Gyro Y	0.84e-2	0.12e-2	0.06e-2
Gyro Z	5.87e-2	0.40e-2	0.05e-2

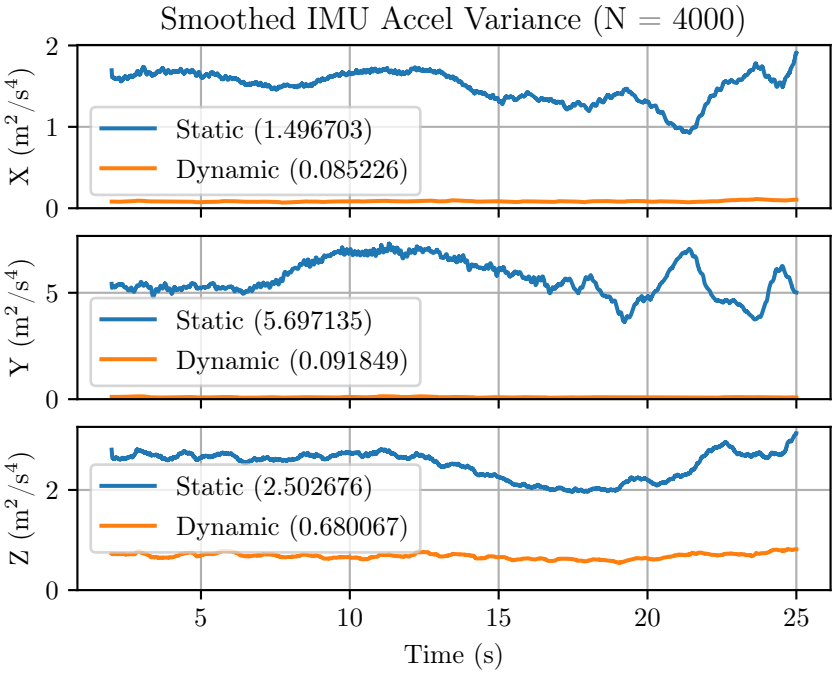


Figure A.12: Comparison between IMU accelerometer data variances when filtered with a static and dynamic notch.

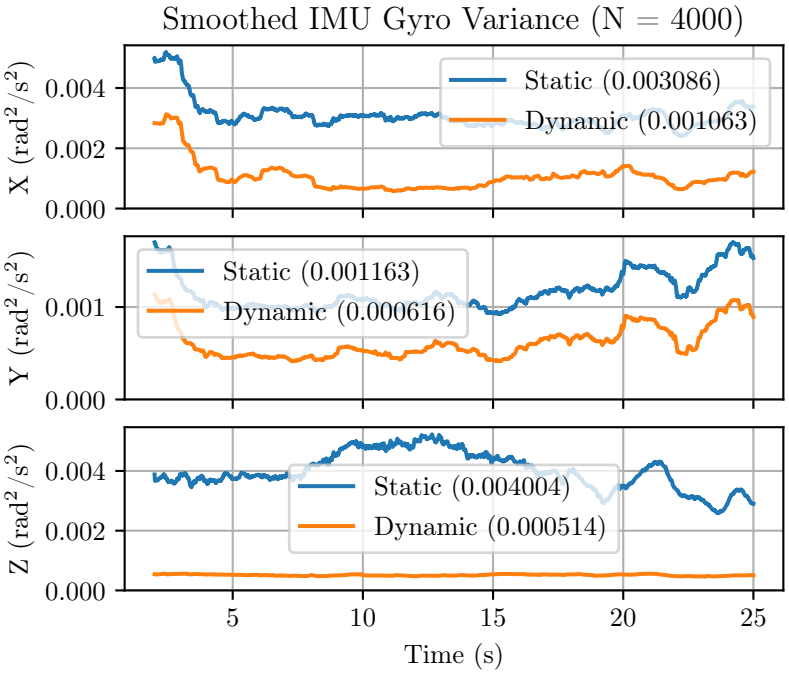


Figure A.13: Comparison between IMU gyroscope data variances when filtered with a static and dynamic notch.

A.2.2.3 Implications

Smoother IMU data enables improvements in two areas, with appropriate hyperparameter optimization:

1. Better attitude estimation performance by updated complementary filter gains. This is a tradeoff between reducing bias and noise.
2. Better attitude control loop performance by updated control gains. Lower noise in the gyroscope and attitude estimates should allow for stronger gains.

Complementary Filter Gains The attitude estimator used onboard the vehicle is a complementary filter that combines incoming accelerometer measurements $a \in \mathbb{R}^3$ and gyroscope measurements $\omega \in \mathbb{R}^3$. The filter state is the gravity direction in the body frame, or the third row of the rotation matrix representing the orientation of the vehicle with respect to the world frame.

$$R^\top g^{\mathcal{W}} = g^{\mathcal{B}} \tag{A.6}$$

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}^\top \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = r_3 \tag{A.7}$$

We assume the 3-axis accelerometer is a direct measurement of this vector, $g^{\mathcal{B}}$. We will drop the superscript in this section and refer to the quantity estimated by the filter as $\hat{g}$. This assumption notably fails when the vehicle is following trajectories with non-negligible acceleration. Linear acceleration compensation is addressed in Section A.2.5.

The complementary filter is a weighted average of two estimates of g: the estimate from the accelerometer a, and the estimate from the gyroscope. The estimate from the gyroscope is computed by rotating the current estimate $\hat{g}$ by the rotation indicated by the gyroscope measurement ω. To compute this, we differentiate (A.6).

$$\dot{g} = \dot{R}^\top e_3 \tag{A.8}$$

$$= \left(R[\omega]_\times\right)^\top e_3 \tag{A.9}$$

$$= [\omega]_\times^\top R^\top e_3 \tag{A.10}$$

$$= -[\omega]_\times g \tag{A.11}$$

$$\dot{g} = g \times \omega \tag{A.12}$$

The complementary filter with weight w is shown below.

$$\hat{g}_{t+1} = w\frac{a_t}{||a_t||} + (1-w)\left(\hat{g}_t + \hat{g}_t \times \omega_t dt\right) \tag{A.13}$$

To maintain the unit norm constraint on g, we normalize $\hat{g}$ after every filter step. Although not a filter "on the manifold", this filter is simple and avoids expensive computations that "manifold-based" filters may require such as matrix exponentials.

Pitch, θ, and roll, ϕ, (as defined by ZYX Euler angles) are computed from $\hat{g}$ using trigonometric operations, shown below.

$$\theta = \arctan2\left(\hat{g}_2, \hat{g}_3\right) \tag{A.14}$$

$$\phi = \arcsin\left(-\hat{g}_1\right) \tag{A.15}$$

To find the optimal filter weight, we perform an optimization with several simplifying assumptions:

1. The axes are decoupled; we analyze along one dimension.
2. The norm of the accelerometer reading is constant.
3. We assume there is no gyroscope bias.

With these assumptions, we take the variance of (A.13) and optimize with respect to w by differentiating. We further assume that the variance is at steady state, i.e. $\text{Var}[\hat{g}_{t+1}] = \text{Var}[\hat{g}_t]$.

$$\text{Var}[\hat{g}] = \frac{w^2}{||a||^2}\text{Var}[a] + (1-w)^2\left(\text{Var}[\hat{g}] + dt^2\text{Var}[\omega]\right) \tag{A.16}$$

Let $\sigma_1^2 = \frac{\text{Var}[a]}{||a||^2}$ and $\sigma_2^2 = dt^2\text{Var}[\omega]$. Solving for the variance of the estimate $\text{Var}[\hat{g}]$, we get

$$\text{Var}[\hat{g}] = \frac{w^2\sigma_1^2 + (1-w)^2\sigma_2^2}{w(2-w)} \tag{A.17}$$

Symbolically optimizing this for w results in

$$w = \frac{\sqrt{\sigma_2^2(4\sigma_1^2 + \sigma_2^2)} - \sigma_2^2}{2\sigma_1^2} \tag{A.18}$$

With (A.18) we can compute the theoretically optimal complementary filter weight given variances of accelerometer and gyroscope readings. Table A.3 shows the optimal weights and predicted variance for the filter options. Although we see a dramatic reduction in predicted variance between the static notch filter and the raw data, the optimal filter weight has not changed much. The optimal filter weight increases by around a factor of 3 when using the dynamic notch filter, suggesting that the reduced noise in the accelerometer reading makes it slightly more useful. The expected variance for the dynamic rpm notch filter using the optimal weight from the static notch filter is 6.02e-7, around double of what it would be if we used the optimal weight. Although a factor of 2 increase in variance may be significant, it's not clear how well the predicted variances match the true output variances, especially given the simplifying assumptions made. Further, the attitude estimates are used in concert with gyroscope estimates, which have higher variances than the outputted pitch and roll estimates. It may be the case that variance / noise in the attitude estimate is dwarfed by the variance in the filtered gyroscope data when concerned with overall control loop performance.

Table A.3: Optimal complementary filter weights based on (A.18) using statistics from the raw IMU data, IMU data filtered using a static notch filter, and IMU data filtered using a dynamic RPM-based notch filter

	Raw IMU Data	**Static Notch**	**Dynamic Notch**
Weight w	1.31e-4	1.43e-4	4.20e-4
Predicted Variance	1.62e-5	2.04e-6	3.66e-7

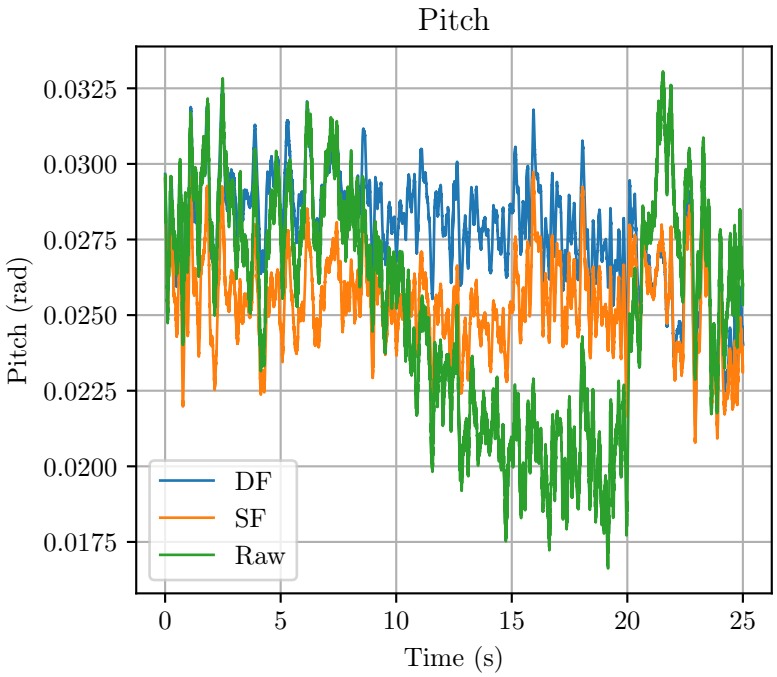


Figure A.14: Pitch computed using raw IMU data (Raw), IMU data filtered using a static notch (SF), and IMU data filtered using a dynamic notch (DF).

To get a sense for how well the variance of the complementary filter's output matches the predicted variance, we run the complementary filter using the three filtering options and their respective optimal weights and an additional baseline weight of w = 1e-4. Unfortunately, the gyroscope readings contain non negligible gyroscope bias. To mitigate this for this analysis, we take average gyroscope readings during the 25 second hover and subtract it out to avoid drifting pitch and roll estimates. Gyroscope bias estimates are added to the filter state in Section A.2.3.

Figures A.14 and A.15 show the resulting pitch and roll estimates, while Figs. A.16 and A.17 show smoothed variances of the pitch and roll estimates. The variances are computed with respect to a mean computed by filtering the pitch and roll estimates using a Savitzky–Golay filter[5] with a window size of 501 (0.5 seconds) and 3rd order polynomials.

The variance data shows three things. First, that there is an ordering in variance between

[5] https://en.wikipedia.org/wiki/Savitzky-Golay_filter

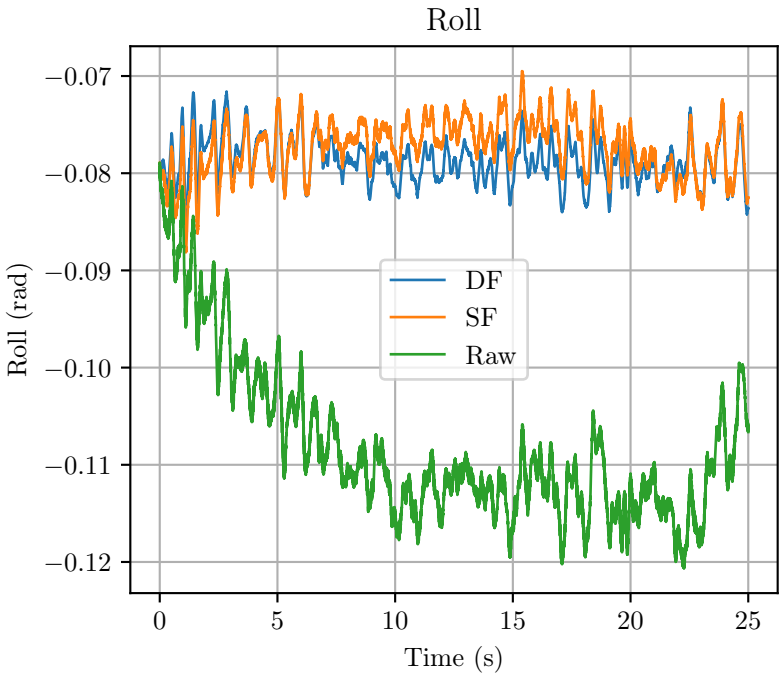


Figure A.15: Roll computed using raw IMU data (Raw), IMU data filtered using a static notch (SF), and IMU data filtered using a dynamic notch (DF).

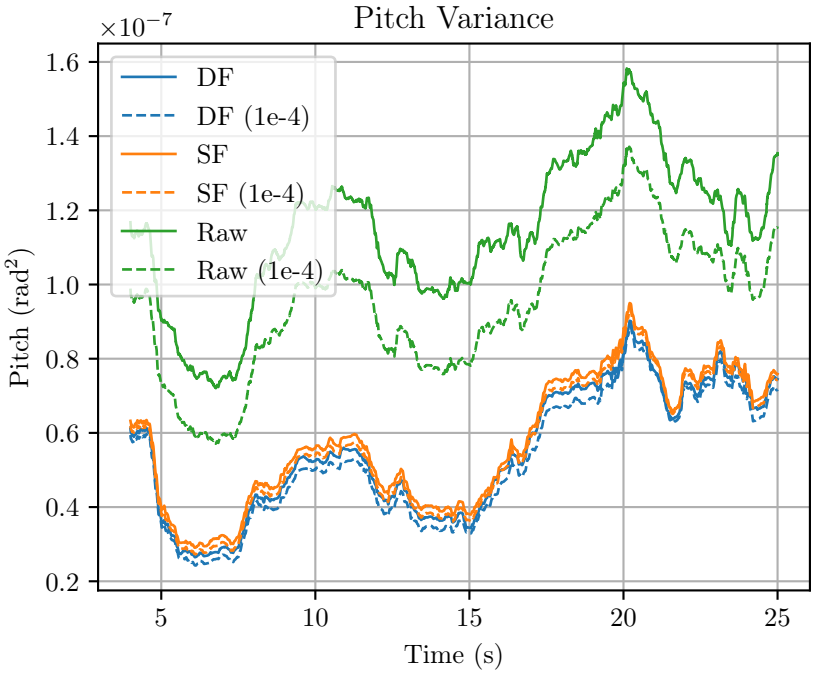


Figure A.16: Variances of pitch computed using raw IMU data (Raw), IMU data filtered using a static notch (SF), and IMU data filtered using a dynamic notch (DF).

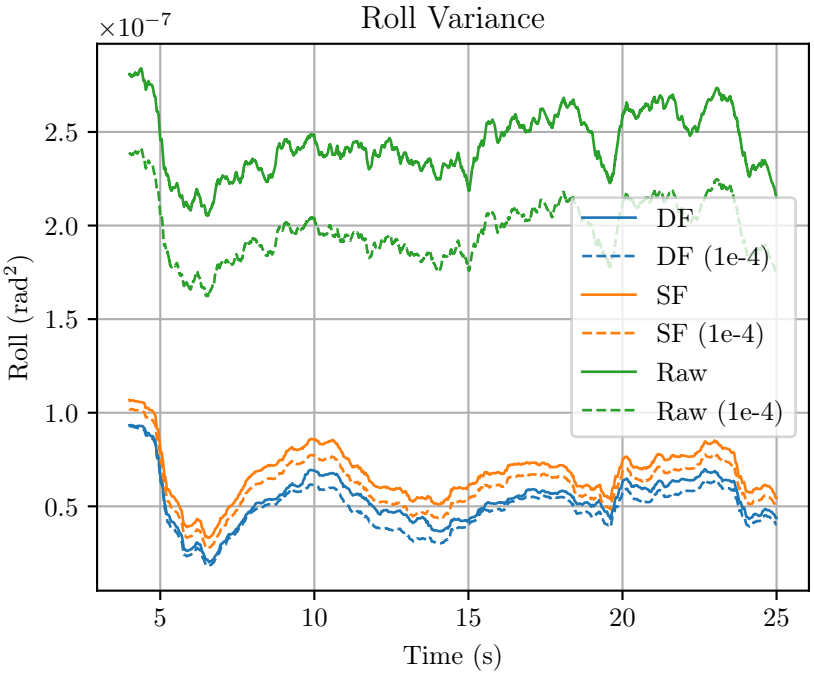


Figure A.17: Variances of roll computed using raw IMU data (Raw), IMU data filtered using a static notch (SF), and IMU data filtered using a dynamic notch (DF).

using raw data, static filtering, and dynamic filtering. This makes sense as smoother data in results in smoother data out. Second, it shows that the outputted variances do not match the predicted variances from Table A.3. This is a result of simplifying assumptions as well as a lack of ground truth. It's not clear what the true mean of the pitch and roll estimates are. The Savitzky–Golay filter is a reasonable choice to showcase the noise reduction more clearly.

Third, we see that the optimal weight may not result in the lowest variance. For example, Fig A.17 shows that the dynamic filter with a weight of 1e-4 achieves slightly lower variance than with its predicted optimal weight of 4.2e-4. Again, this is likely a result of simplifying assumptions and tells us that there is not much to be gained by optimizing the filter weight according to the observed *input* data variance. Practically, the filter weight should look at the *output* data variance and finding the optimal value requires some trial and error. However, this analysis shows that the benefits of finding the optimal filter weight are quite small anyway.

Control Gains The filtered gyroscope data is directly used as the angular velocity signal in the attitude control loop. Since the noise present in the attitude estimate is lower than that present in the gyroscope data, reducing gyroscope data noise should allow for stronger gains, improving control performance without destabilizing the vehicle.

A simple linearized analysis can be used to predict what increase in gains are feasible given the reduction in input data variance.

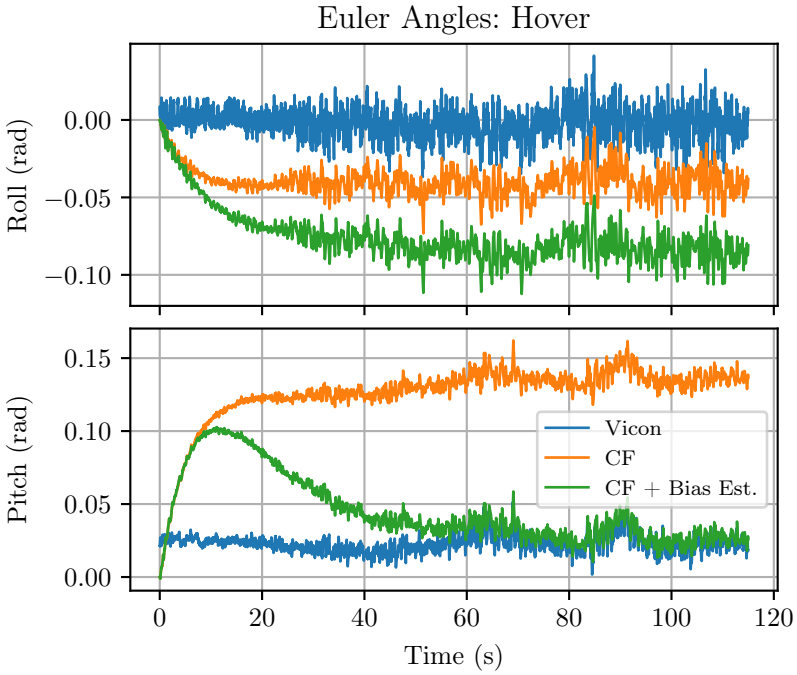


Figure A.18: Comparison of estimated pitch and roll during hover between a complementary filter with bias estimation and one without.

A.2.3 Gyro Bias Estimation

We extend the complementary filter described in Section A.2.2.3 to estimate gyroscope biases, $b \in \mathbb{R}^3$. We rewrite (A.13) using the quantities $\hat{g}_t^{\text{acc}} = \frac{a_t}{||a_t||}$ and $\hat{g}_t^{\text{gyro}} = \hat{g}_t + \hat{g}_t \times (\omega_t - b_t)\, dt$ representing the gravity vector estimate from the accelerometer and the gyroscope respectively. Note the gravity vector estimate from the gyroscope includes compensation for the estimated biases b_t.

$$\hat{g}_{t+1} = w\hat{g}_t^{\text{acc}} + (1-w)\hat{g}_t^{\text{gyro}} \tag{A.19}$$

To update our estimate of the bias b_t we use $\hat{g}^{\text{acc}} \times \hat{g}^{\text{gyro}}$, which represents the delta rotation between the gravity vector estimate from the accelerometer and that from the gyroscope, multiplied by a gain w_b. Essentially, we assume that any difference, on average, between the gravity vector estimated by the accelerometer and that estimated by the gyroscope is due to gyro bias.

$$b_{t+1} = b_t - w_b \left(\hat{g}^{\text{acc}} \times \hat{g}^{\text{gyro}}\right) dt \tag{A.20}$$

Figure A.18 shows the pitch and roll estimates of a complementary filter with the proposed bias estimation strategy and those from one without on data from a vehicle at hover. Figure A.19 shows the resulting biases estimated by the filter. For this experiment, $w = 0.0001$ and $w_b = 0.01$. We see that the biases converge slowly and result in the estimated pitch more closely matching the pitch measured by motion capture, while the

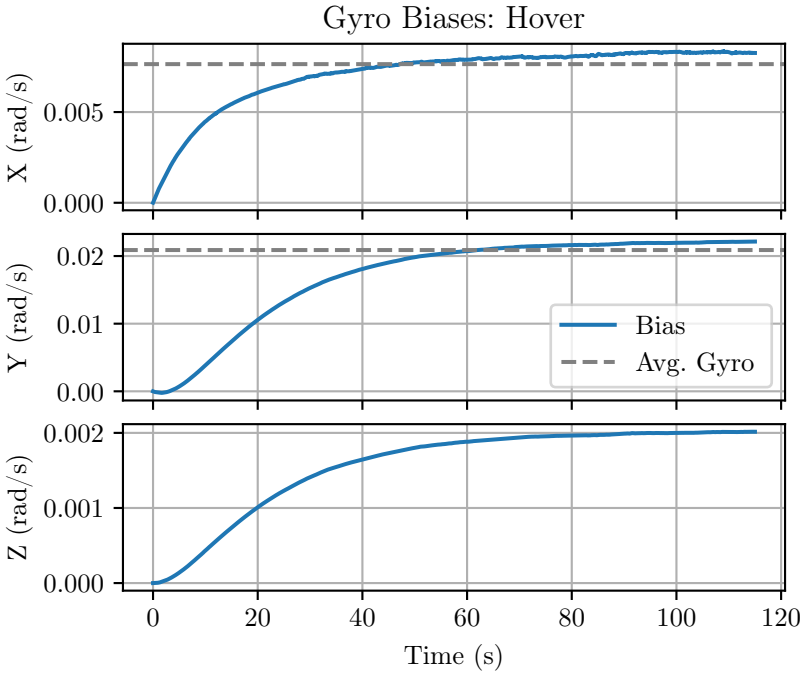


Figure A.19: Estimated gyro biases during the hover from Fig. A.18.

estimated roll matches motion capture less closely. This suggests that there is likely a transform offset between the center of the IMU and the center of the vehicle's motion capture model.

To test the stability and robustness of the complementary filter with gyro bias estimation, we test it on a trajectory where the vehicle is executing aggressive circles.

Figures A.20 and A.21 show the resulting pitch and roll and bias estimates respectively. We see that although the gyro biases do deviate slightly while the vehicle is following the circular trajectories, they remain relatively stable. There is some pitch and roll drift present at around 90 seconds that is present in both the filter with and without gyro bias estimation. This suggests that the likely culprit is the violation of the assumption that there is no linear acceleration acting on the vehicle, which would affect both filters. A strong acceleration is required to keep the vehicle moving in a circular motion. Compensation of vehicle linear acceleration is discussed in Section A.2.5.

A.2.4 Kalman Filter for Disturbance Estimation

We formulate a Kalman filter to estimate the orientation state, gyroscope biases, and angular acceleration disturbance acting on the vehicle. This filter serves as a baseline for the model learning work done later.

In the filter state, we include

- $z \in \mathcal{S}^2$, the z-world axis expressed in the body frame

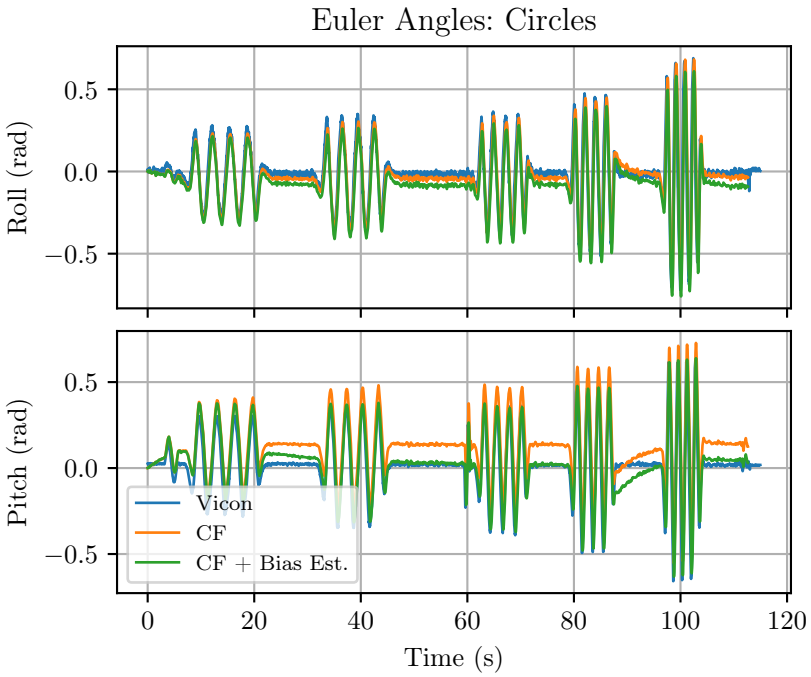


Figure A.20: Comparison of estimated pitch and roll between a complementary filter with bias estimation and one without for a vehicle following several aggressive circular trajectories.

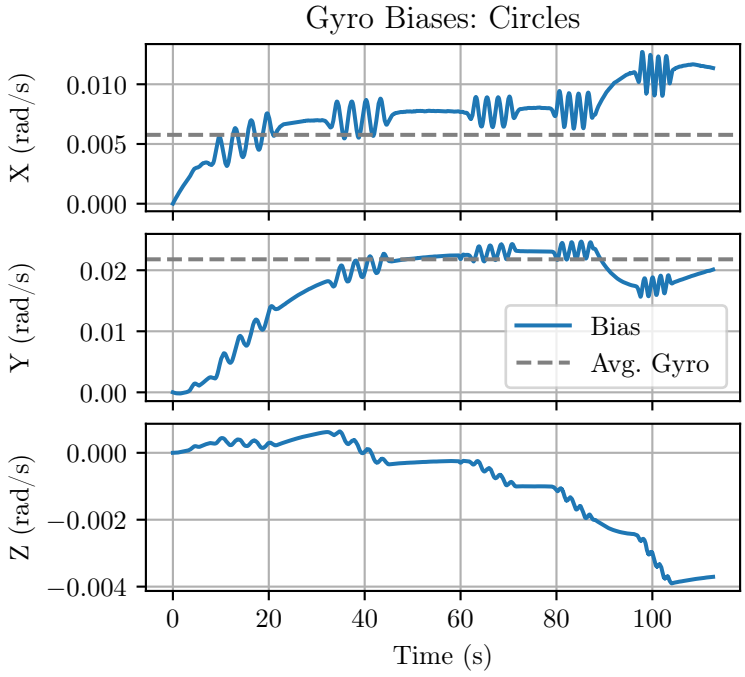


Figure A.21: Estimated gyro biases during the flight from Fig. A.20.

- $\omega \in \mathbb{R}^3$, the angular velocity in the body frame
- $b \in \mathbb{R}^2$, the x and y gyroscope biases
- $d \in \mathbb{R}^3$, the angular acceleration disturbance

Although it's possible to include the position state of the vehicle and leverage accelerometer data for higher rate and higher accuracy position feedback, we do not include it in this baseline for the following reasons.

1. Adding position state would include a lot of complexity and as the filter needs to run on the vehicle firmware, we strive to keep computational costs low.
2. Accelerometer data is quite noisy despite the filtering done and the gains from including it to estimate position may be low.

To reduce complexity further, we also eliminate yaw information. This allows the filter to be decoupled from any motion capture or vision-based state estimation source, while still keeping all state entries observable. This precludes estimation of the z gyroscope bias while the vehicle is level.

For the system control input, we use

- $\alpha \in \mathbb{R}^3$, the body frame angular acceleration

For the system observations, we use

- $a \in \mathbb{R}^3$, the 3-axis accelerometer measurement
- $g \in \mathbb{R}^3$, the 3-axis gyroscope measurement

A.2.4.1 Process Model

We use a discrete process model with a step size of T, where z and ω are corrupted with Gaussian noise at every step, and b and d follow a random walk. The discrete model is derived using single step integration of the continuous dynamics.

The z axis estimate is driven by the angular velocity estimate.

$$z_{i+1} = z_i + (z_i \times \omega)T + \epsilon_z \tag{A.21}$$

The angular velocity is updated by the control, α, and the disturbance estimate, d. Since d is in $\mathbb{R}^2$, we abuse notation here and let d refer to a vector in $\mathbb{R}^3$ with the third component equal to zero.

$$\omega_{i+1} = \omega_i + (\alpha_i + d_i)T + \epsilon_\omega \tag{A.22}$$

The bias and disturbance follow random walks, which is equivalent to added noise in the discrete setting.

$$b_{i+1} = b_i + \epsilon_b \tag{A.23}$$
$$d_{i+1} = d_i + \epsilon_d \tag{A.24}$$

A.2.4.2 Observation Model

The accelerometer measurement includes gravity, which points in the direction of the world z-axis, and a linear acceleration component. Here, a_g is the gravitational acceleration constant approximately equal to $9.81\,\mathrm{m/s^2}$.

$$a_i = a_g z_i + a_{\text{lin}} + \epsilon_a \tag{A.25}$$

The gyroscope measurement is a sum of the angular velocity and the biases.

$$g_i = \omega_i + b_i + \epsilon_g \tag{A.26}$$

A.2.4.3 Algorithm

Algorithm choices include

- EKF
- UKF
- Asymptotic Kalman filter / LTI filter

For simplicity, we choose an asymptotic Kalman filter. This entails solving the discrete Ricatti equation to compute the asymptotic Kalman gain. The dynamics model used to propagate the covariance in the pre-computation is the linearization of the model at hover.

Although this precludes estimating covariances, we feel those are of marginal benefit for the following reasons.

1. Sensor noise during flight is very consistent, barring any failures in notch filtering.
2. The vehicle spends most of its time near-hover. State estimate covariances changes due to changing flight conditions are transient and should not affect the optimal state estimator much.

A large benefit of the asymptotic Kalman filter is its computational efficiency over an EKF or UKF, making an embedded implementation relatively easier.

A.2.4.4 Parameters

Implementing the asymptotic Kalman filter requires the selection of variances for the following random variables.

1. Process noise of world z-axis, ϵ_z from (A.21)
2. Process noise of angular velocity, ϵ_ω from (A.22)
3. Process noise of gyro bias, ϵ_b from (A.23)
4. Process noise of angular acceleration disturbance, ϵ_d from (A.24)
5. Measurement noise of accelerometer, ϵ_a from (A.25)

Table A.4: Attitude Kalman filter noise parameters

Parameter	Value
ϵ_z	10^{-12}
ϵ_ω	10^{-4}
ϵ_b	10^{-15}
ϵ_d	10^{-2}
ϵ_a	5×10^{-1}
ϵ_g	2×10^{-4}

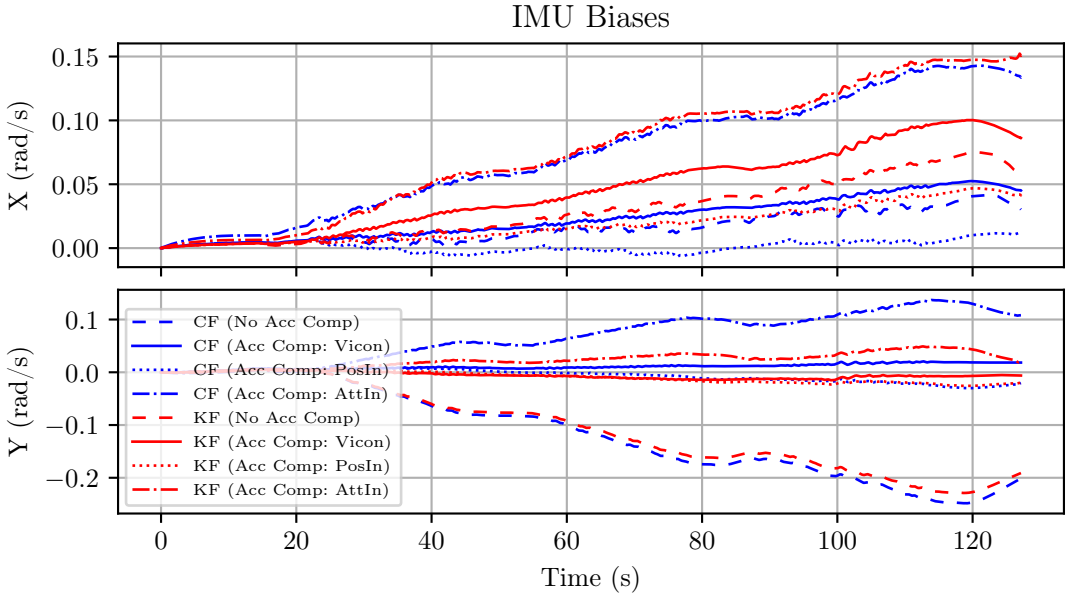


Figure A.22: Gyro biases estimated during flight while following an aggressive 3D trajectory for various linear acceleration compensation strategies for both the complementary filter (CF) and the Kalman filter (KF). Strategies that do not compensate for linear acceleration experience severe gyro bias drift.

6. Measurement noise of gyroscope, ϵ_g from (A.26)

Table A.4 shows the parameters used in this thesis. The Kalman filter update weights are computed by solving the discrete algebraic Ricatti equation offline.

A.2.5 Linear Acceleration Compensation

We test linear acceleration compensation in the both the complementary and Kalman filter attitude estimators using the following linear acceleration sources.

1. Acceleration measured by Vicon
2. Desired acceleration from the reference trajectory (input to the position controller, PosIn)

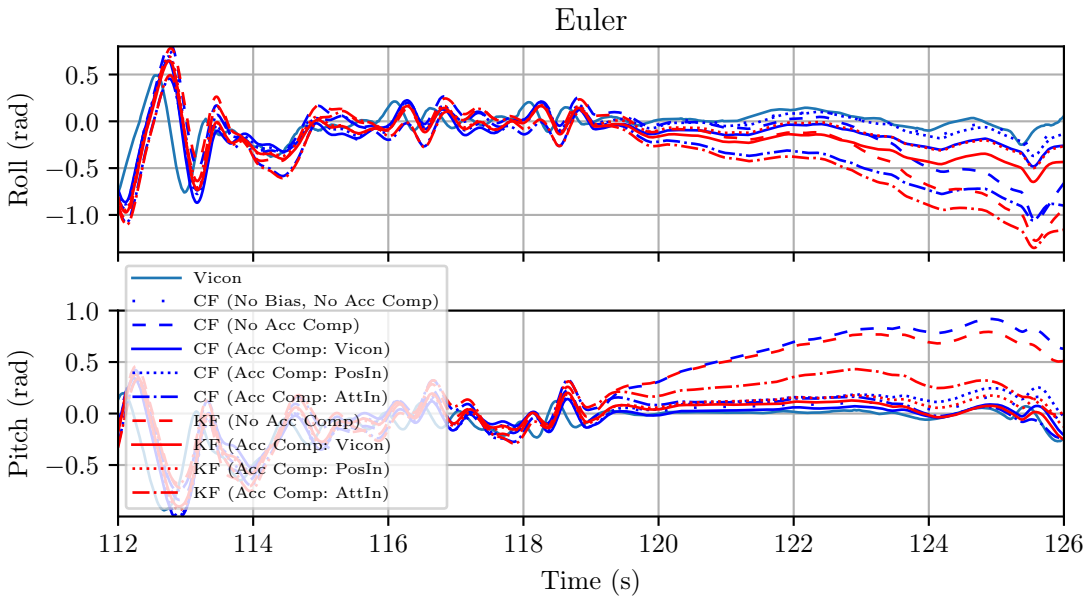


Figure A.23: Roll and pitch angle estimated after following an aggressive 3D trajectory for various linear acceleration compensation strategies for both the complementary filter (CF) and the Kalman filter (KF). Strategies that do not compensate for linear acceleration experience several pitch and roll drift, as a result of bias estimation drift. Note that the strategy that does not estimate biases (sparse blue dots) does not suffer from pitch and roll drift, despite not compensating for linear acceleration.

3. Desired acceleration input to the attitude controller (after position and velocity feedback, AttIn)

Estimated gyro biases and tilt angles after post-processing an aggressive flight dataset are shown in Figs. A.22 and A.23. Strategies that compensate for linear acceleration experience much less gyro bias drift, and thus pitch and roll drift, than those that do not. Interestingly, the complementary filter without bias estimation does not suffer without linear acceleration compensation, indicating that the linear acceleration's effect on attitude estimate accuracy is not direct, but rather indirect through the estimated gyro biases. This behavior is likely heavily dependent on the particular filter weights chosen ($w = 0.0001$ and $w_b = 0.01$ for the complementary filter.). Lower gyroscope bias weights, corresponding to slower gyro bias estimation, may reduce the impact of uncompensated linear acceleration.

We choose to use the acceleration measured by Vicon, as it performs well during post-processing and unlike the desired acceleration, provides the true acceleration of the vehicle.

APPENDIX B

Rotational Error Metric Case Studies

The contents of this section first appeared in [81] and were presented at the IROS 2020 workshop: *Perception Learning and Control for Autonomous Agile Vehicles*.

B.1 Introduction

Here we introduce the concept of rotational error functions to analyze the performance and stability of various attitude controllers in the literature.

We analyze and compare various attitude error metrics that have been proposed for use in quadrotor attitude controllers.

For brevity, we consider regulation, not trajectory tracking, but all of the results here can be extended to work for trajectory tracking, where the desired angular velocity ω_{des} and the desired angular acceleration α_{des} are nonzero.

The structure of quadrotor attitude controllers that we consider is a PD controller on rotational error, as defined by a rotational error metric e_R, scaled by a diagonal attitude gain matrix K_R, and angular velocity ω, scaled by a diagonal angular velocity gain matrix K_ω.

$$\alpha = -K_R e_R(R, R_{\text{des}}) - K_\omega \omega \tag{B.1}$$

Several works in the literature have noted that, since the quadrotor's position dynamics depend solely on its tilt, or body z-axis, it makes sense to prioritize vehicle tilt over vehicle yaw [13, 22, 31, 48, 66]. We confirm this, and show that such metrics amount to inducing a response in the direction of the cross product between the current and desired body z-axes, along with a response around the body z-axis. We show that this decomposition linearizes the vehicle error response better than the traditional attitude controller, which uses the full rotation error. Following straight paths in the presence of large tracking error can be beneficial for overall control performance.

In this section, we place various controllers used in the literature in the framework described above. First, as in Mueller [66], we define rotational error R_e as

$$R_e = R_{\text{des}}^\top R \tag{B.2}$$

and its corresponding angle and axis as ρ_e and n_e.

The vehicle attitude R transforms vectors from the body frame $\mathcal{B}$ into the fixed frame, and is represented using a matrix whose columns x, y, and z are the coordinate frames axes of the body.

$$R = \begin{pmatrix} x & y & z \end{pmatrix} \tag{B.3}$$

We define the *thrust vector* error R_{tv} as the rotation between the z-axes of the current and desired attitude, along with its associated angle ρ_{tv} and axis n_{tv}. This rotation can be defined via its angle and axis in terms of the full rotation error R_e as shown below [66].

$$\rho_{tv} = \cos^{-1}(e_3^\top R_e e_3) \tag{B.4}$$

$$n_{tv} = \frac{(R_e^\top e_3) \times e_3}{||(R_e^\top e_3) \times e_3||} \tag{B.5}$$

To help visualize n_{tv}, note that it is just the normalized cross product of the desired z-axis $z_{\text{des}}^{\mathcal{B}} = R^\top R_{\text{des}} e_3$ with the current z-axis of the body $z^{\mathcal{B}} = e_3$, expressed in the body frame. As a result, $n_{tv}^\top e_3 = 0$.

$$n_{tv} = \frac{z_{\text{des}}^{\mathcal{B}} \times z^{\mathcal{B}}}{||z_{\text{des}}^{\mathcal{B}} \times z^{\mathcal{B}}||} \tag{B.6}$$

We define the yaw error R_{yaw} as the remaining error around the body z-axis, i.e. the rotation needed to align the current rotation to the desired rotation after aligning the z axes.

$$R_{\text{yaw}} = R_e R_{tv}^{-1} \tag{B.7}$$

ρ_{yaw} represents the angle associated with R_{yaw} and n_{yaw} represents the axis, which is either $e_3 = (0, 0, 1)^\top$ or $-e_3$, and thus perpendicular to n_{tv}.

B.2 Full Rotation Metrics

A rotational error metric directly in SO(3) is used in [34, 51, 61]. As shown in [66], this metric is proportional to the sine of the angle error ρ_e and in the direction of the shortest path n_e.

$$e_R^{\mathbf{A}}(R, R_{\text{des}}) = \frac{1}{2}\left(R_e - R_e^\top\right)^\vee = \sin\rho_e n_e \tag{B.8}$$

Lee [50] notes the issues with using a rotational error metric that is proportional to the sine of the error, namely that the response is strongest at an error of 90° and weakens as the error increases, reaching zero at an error of 180°. While this ensures smoothness of the response, it results in slow convergence when the error is high. Lee [50] proposes rescaling the rotational error. [1]

$$e_R^{\mathbf{B}}(R, R_{\text{des}}) = \frac{2}{\sqrt{1 + \text{tr}\,(R_e)}} \sin\rho_e n_e \tag{B.9}$$

[1] We add a factor of 2 to match the linearization around $\rho_e = 0$ of (B.8).

Using the fact that $\operatorname{tr}(R_e) = 2\cos\rho_e + 1$ [66] and the sine half angle identity $\sin\left(\frac{\theta}{2}\right) = \sqrt{\frac{1-\cos\theta}{2}}$, (B.9) can be rewritten as

$$e_R^{\mathbf{B}}(R, R_{\text{des}}) = 2\sin\left(\frac{\rho_e}{2}\right) n_e. \tag{B.10}$$

(B.10) is mathematically equivalent to the rotational error used in Fresk and Nikolakopoulos [29], which is the axis component of the error quaternion. The axis component of a quaternion is the sine of the half angle multiplied by the axis.

Similarly, one can use an error response that is directly proportional to the angle.

$$e_R^{\mathbf{C}}(R, R_{\text{des}}) = \rho_e n_e. \tag{B.11}$$

This is likely not as widely used, since unlike (B.8) and (B.9), evaluating (B.11) from a rotation matrix or quaternion attitude representation requires inverse trigonometric function evaluations, which may be expensive on an embedded platform.

B.3 Thrust Vector – Yaw Decomposition Metrics

In this section we will characterize the attitude controls used by works that decompose attitude into a thrust vector and yaw, also known as "reduced attitude control", with an emphasis on the control around the body x and y axes. Define, $\omega_{XY} = z \times \dot{z} = \omega - (\omega^\top z)\, z$.

Kooijman, Schoellig, and Antunes [48] decomposes the attitude representation into $\mathcal{S}^2 \times \mathcal{S}^1$, which amounts to controlling the thrust vector direction, an element of $\mathcal{S}^2$, independently from the rotation around the thrust vector, an element of $\mathcal{S}^1$. The control response around the x and y axes[2] in Kooijman, Schoellig, and Antunes [48] is given as

$$\begin{aligned} \omega_{XY}^{\mathcal{B}} &= R^\top (z \times u_v) && \text{(B.12)} \\ &= e_3 \times (R^\top u_v) && \text{(B.13)} \\ &= \begin{pmatrix} -y^\top u_v & x^\top u_v & 0 \end{pmatrix}^\top, && \text{(B.14)} \end{aligned}$$

with u_v defined as

$$u_v = \begin{cases} k_1 z_{\text{des}} & \rho_{tv} \leq \frac{\pi}{2} \\ \frac{k_1}{\sqrt{1-(z^\top z_{\text{des}})^2}} z_{\text{des}} = \frac{k_1}{\sin\rho_{tv}} z_{\text{des}} & \frac{\pi}{2} < \rho_{tv} < \pi. \end{cases} \tag{B.15}$$

Substituting (B.15) into (B.12), we get

$$\omega_{XY}^{\mathcal{B}} = \begin{cases} -k_1 z_{\text{des}}^{\mathcal{B}} \times z^{\mathcal{B}} = -k_1 \sin\rho_{tv} n_{tv} & \rho_{tv} \leq \frac{\pi}{2} \\ -k_1 \frac{z_{\text{des}}^{\mathcal{B}} \times z^{\mathcal{B}}}{\sin\rho_{tv}} = -k_1 n_{tv} & \frac{\pi}{2} < \rho_{tv} < \pi. \end{cases} \tag{B.16}$$

[2]For simplicity, we omit the yaw controller from Kooijman, Schoellig, and Antunes [48].

Assuming a P controller for angular velocity, from (B.16) we can extract the rotational error metric e_R as

$$e_R^{\mathbf{D}}(R, R_{\text{des}}) = \begin{cases} \sin \rho_{tv} n_{tv} & \rho_{tv} \leq \frac{\pi}{2} \\ n_{tv} & \frac{\pi}{2} < \rho_{tv} < \pi. \end{cases} \tag{B.17}$$

(B.17) has the interesting property that the magnitude of the response is maximum at an angle of $\rho_{tv} = \frac{\pi}{2}$ and remains constant as the angle increases from $\frac{\pi}{2}$ to π. For $\rho_{tv} \leq \frac{\pi}{2}$, (B.17) is equivalent to (B.8) for axis-aligned errors.

Brescianini and D'Andrea [12] also decomposes the thrust vector and the yaw, but does so using two quaternions. The final control law is a sum of a quaternion error representing the error in the thrust direction and a quaternion error representing the error in the angle around the body z-axis. The use of quaternions implies that the rotational error metric is proportional to the sine of the half angle for both the thrust vector error and the yaw error. The resulting rotational error metric is shown below, with again, a factor of 2 added to match linearizations with the other metrics.

$$e_R^{\mathbf{E}}(R, R_{\text{des}}) = 2 \sin\left(\frac{\rho_{tv}}{2}\right) n_{tv} + 2 \sin\left(\frac{\rho_{\text{yaw}}}{2}\right) n_{\text{yaw}} \tag{B.18}$$

(B.18) is equivalent to (B.10) for axis-aligned errors.

Mueller [66] proposes a new rotational error metric that is effectively a linear interpolation between (B.11) and only controlling the thrust vector. The rotational error metric is shown below, with $\alpha_{\text{yaw}} \in [0, 1]$ used to weight the control of the yaw angle.

$$e_R^{\mathbf{F}}(R, R_{\text{des}}) = \alpha_{\text{yaw}} \rho_e n_e + (1 - \alpha_{\text{yaw}}) \rho_{tv} n_{tv} \tag{B.19}$$

Unlike, (B.17) and (B.18), (B.19) doesn't decouple the yaw control from the thrust vector control for $\alpha_{\text{yaw}} > 0$. As we will show below, this leads to suboptimal performance in certain situations with large angle errors. As an alternative, we propose to use the decoupling metric from (B.18), but proportional to the angle, as shown below.

$$e_R^{\mathbf{G}}(R, R_{\text{des}}) = \rho_{tv} n_{tv} + \rho_{\text{yaw}} n_{\text{yaw}} \tag{B.20}$$

Table B.1 provides a tabulated summary of rotational error metric configurations discussed.

B.4 Simulation Case Studies

To evaluate the rotational error metrics for the purposes of quadrotor control, we use the following three scenarios.

1. Direction change.
2. Axis-aligned step with large initial yaw error.

Table B.1: Comparison matrix of various rotational error metrics

Scaling / Direction	**Full**: n_e	**Decomposed**: $n_{tv} + n_{\text{yaw}}$
$\sin\rho$	$e_R^{\mathbf{A}}$: [34, 51, 61]	$e_R^{\mathbf{D}^*}$: [31, 48]
$2\sin\frac{\rho}{2}$	$e_R^{\mathbf{B}}$: [29, 50]	$e_R^{\mathbf{E}}$: [12, 13, 22, 66]
ρ	$e_R^{\mathbf{C}}$, $e_R^{\mathbf{F}^\dagger}$: [66]	$e_R^{\mathbf{G}}$: Proposed

*For $\rho < \frac{\pi}{2}$. †For $\alpha_{\text{yaw}} > 0$.

3. Diagonal step.

In order to isolate the effects of the rotational error function, we add an additional linearizing term to all controllers inspired by the feedback linearization controller: the final term of (4.37). In effect, this should make the results equivalent to the results using a quadrotor with angular velocity as a control input instead of angular acceleration. The largest impact of this is seen during the second scenario, a simultaneous position and yaw step, where the term allows the decomposing error metrics to follow the straight line path to the goal (compare to Fig. 4.2). To ensure fairness, the term is applied to all error metrics.

For the last two experiments, we additionally compare against an Euler angle-based rotational error metric, defined using the Z-Y-X convention. The yaw ψ, pitch θ, and roll ϕ, define a rotation matrix using the following map $F(\psi, \theta, \phi)$, with $c = \cos$ and $s = \sin$.

$$F = \begin{pmatrix} c\psi c\theta & c\psi s\theta s\phi - c\phi s\psi & s\psi s\phi + c\psi c\phi s\theta \\ c\theta s\psi & c\psi c\phi + s\psi s\theta s\phi & c\phi s\psi s\theta - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{pmatrix} \tag{B.21}$$

The rotational error metric is then defined using the inverse of F, making sure to compute the shortest angular distance for each angle.

$$e_R^{\mathbf{ZYX}}(R, R_{\text{des}}) = F^{-1}(R) \ominus F^{-1}(R_{\text{des}}) \tag{B.22}$$

Videos of the three scenarios discussed here, along with an interactive simulation, can be found at https://alspitz.github.io/roterrormetrics.html.

B.4.1 Direction Change

The quadrotor is tasked to reach a goal position at $(0, 3, 0)$ after starting at the origin with a velocity of $(0, -5, 0)$ and an initial roll angle of 80°. This initial condition is designed to induce an initial roll error greater than 90°, so that differences in the rotational error metric responses at large angle errors are highlighted. Figure B.1 shows the resulting side view and roll trajectory. The important thing to note here is that the

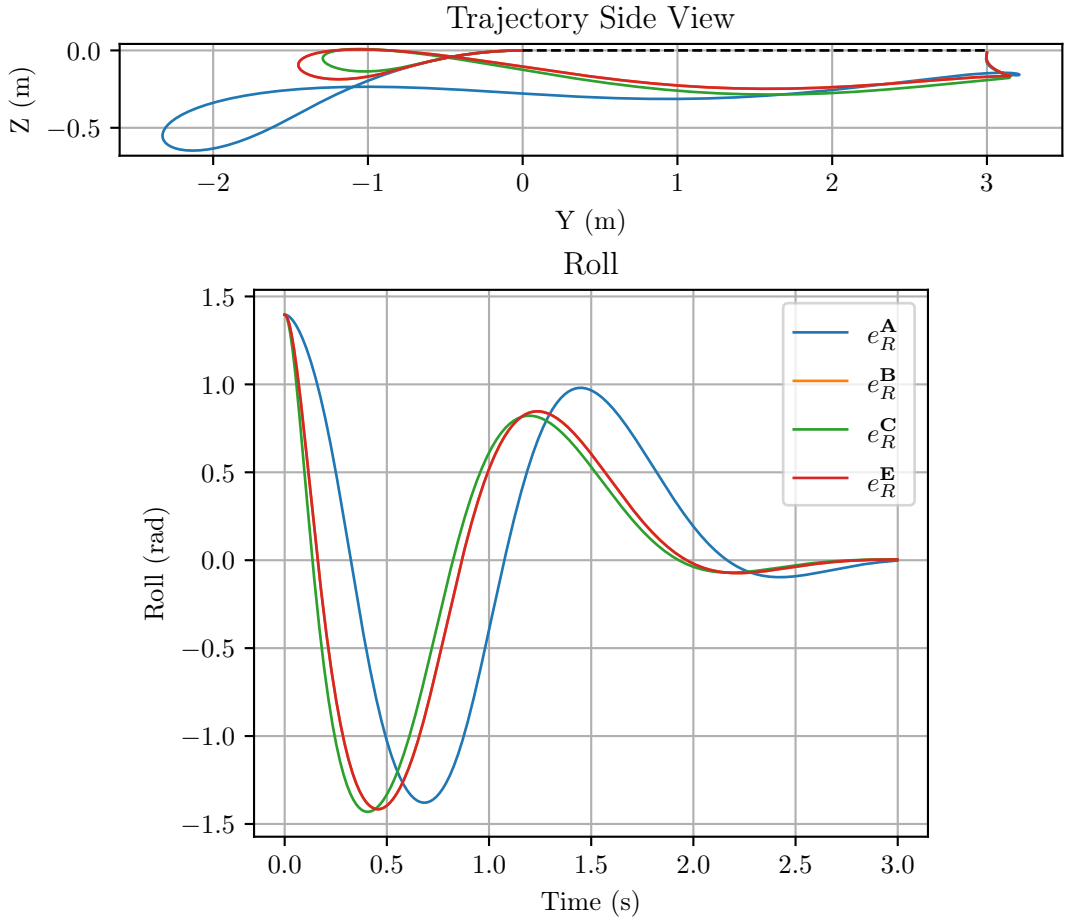


Figure B.1: Side view of the trajectory followed (top) and roll (bottom) during the quick direction change test for various rotational error metrics. Metrics with a response proportional to the sine of the angle, **A**, converge much slower than those with a response proportional to sine of the half-angle, **B** and **E**, or the angle, **C**. Since the trajectory lies in the YZ plane, **B** and **E** perform identically.

rotational error metric $e_R^{\mathbf{A}}$, which has a response proportional to the sine of the angle error, converges slower than the other metrics. This example shows that metrics that induce a response proportional to the sine of the angle can suffer from slow convergence when there is large attitude error.

B.4.2 Step with Yaw Error

In this test, the quadrotor executes a position step and yaw step simultaneously. The vehicle starts at the origin at a yaw of 90° and moves to $(0, 3, 0)$ at a yaw of 0°. Figure B.2 shows the side view and x position during the experiment. We see that metrics that use the full rotation error, result in trajectories that do not maintain the desired x position. Metrics that decompose the response, **E** and **G**, maintain the desired x position.

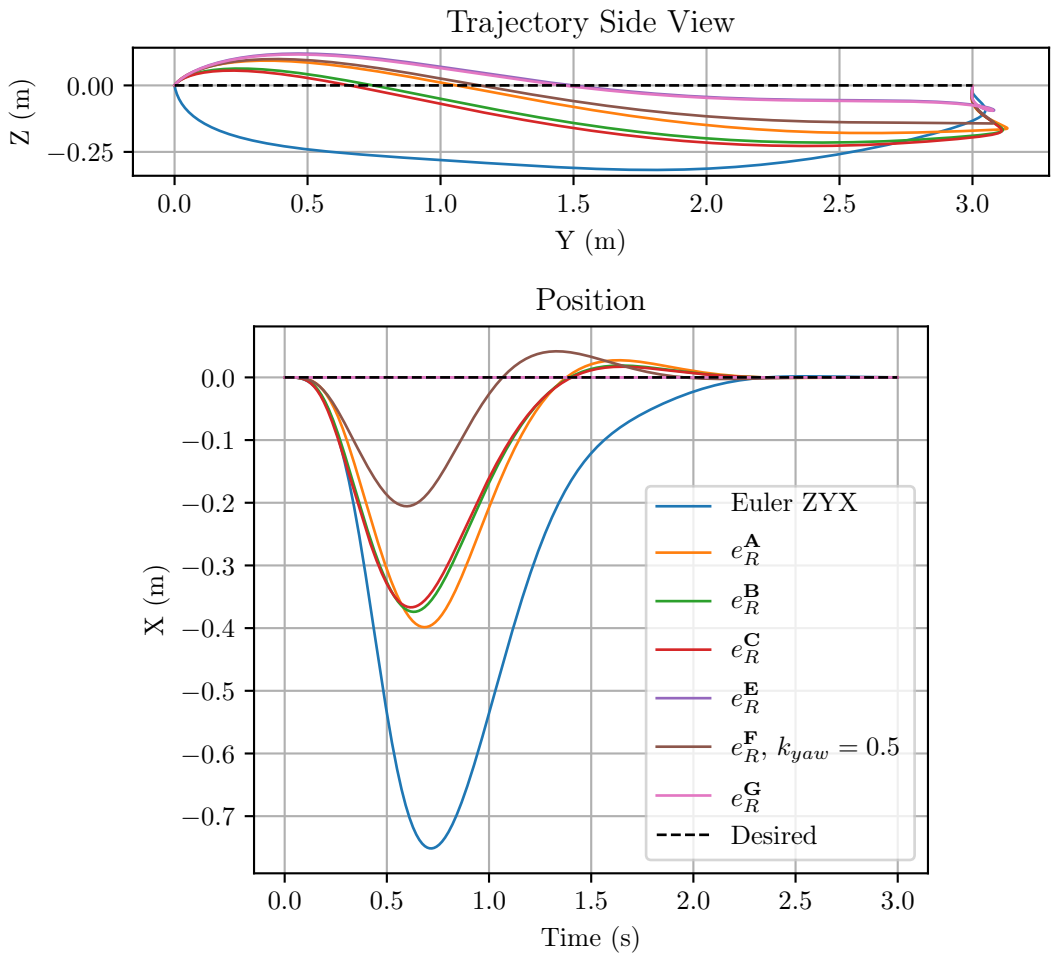


Figure B.2: Side view of the trajectory followed (top) and x position (bottom) during a simultaneous 3 m position step and 90° yaw step for various rotational error metrics. Only metrics that decompose the thrust vector from the yaw, **E** and **G**, maintain the desired x position.

B.4.3 Diagonal Step

In this test, we show that yaw error is not necessary for the rotational error metrics that do not decouple the thrust vector from yaw to exhibit suboptimal performance. The quadrotor is given a diagonal step from the origin to $(3, 3, 0)$, with a desired yaw of 0° for the duration of the test. Figure B.3 shows the top view of the trajectories followed during the experiment for the various error metrics. Metrics that use the full rotation error, **A**, **B**, **C**, and **F**, deviate from the diagonal path. This is because the shortest path in SO(3) from the identity to an orientation that results in the vehicle accelerating diagonally at the same yaw angle, takes the vehicle through orientations that accelerate in directions other than the diagonal direction. In other words, the thrust vector projected onto the horizontal plane does not point in the direction of the desired position.

Metrics that decompose the error into a thrust vector component and a yaw component,

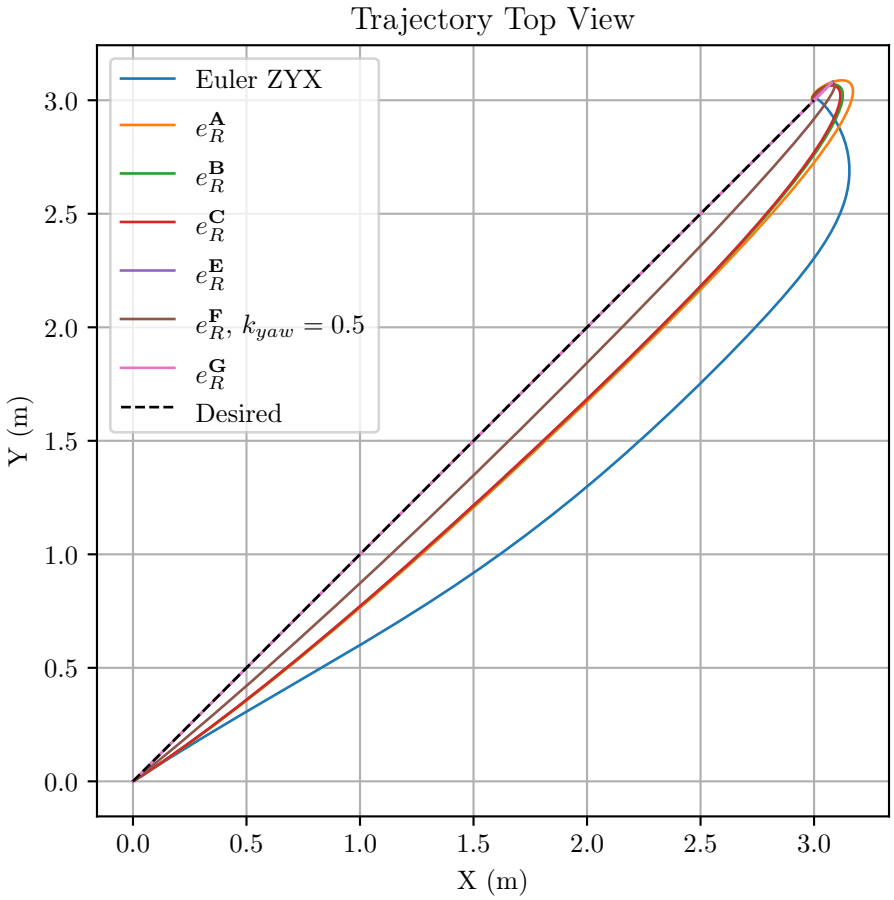


Figure B.3: Top view of the trajectory followed during a diagonal step from the origin to $(3, 3, 0)$ for various rotational error metrics. Only metrics that decompose the thrust vector from the yaw, **E** and **G**, stay on the straight-line diagonal path.

E and **G**, follow the straight-line diagonal path to the desired position.

B.5 Discussion

We have shown that many quadrotor attitude controllers in the literature can be neatly characterized using their rotational error metrics' scaling and direction.

We have shown that rotational error metrics that decompose the attitude error in a thrust vector component and a yaw component provide superior control performance than those that use the full attitude error for (1) simultaneous large errors in position and yaw and (2) diagonal steps.

APPENDIX C

Outdoor Trajectory Generation Details

Richter, Bry, and Roy [77] optimizes the integral of a higher order derivative together with the total duration of a polynomial trajectory satisfying equality constraints at provided positions. For the experiments in Chapter 6, we generate trajectories that attain desired velocity and acceleration limits while passing through user-specified waypoints.

The (x, y) position waypoints used for the 20 m and 40 m figure 8 trajectories are shown in Table C.1. The desired z-position and yaw are held constant for the experiments.

The parameters shared among all trajectories flown during the outdoor experiments are shown in Table C.2.

The optimization method from [77] can be sensitive to initial segment times. The initial trajectory segment time for segment i constrained between points $p_i, p_{i+1} \in \mathbb{R}^3$ is computed according to

$$\hat{t}_i = \frac{||p_{i+1} - p_i||}{v_{\max}/2} \tag{C.1}$$

$$t_i = \hat{t}_i \left(1 + \frac{v_{\max}}{a_{\max}} \exp\left(-\hat{t}_i\right)\right) \tag{C.2}$$

where $v_{\max}$ and $a_{\max}$ are the velocity and acceleration constraints[1]. The final segment times are optimized using the nonlinear optimization described in [77].

The resulting trajectories and their higher order derivatives are shown in Figs. C.1 and C.2.

[1] We use ETH Zurich Autonomous Systems Lab's open-source implementation.

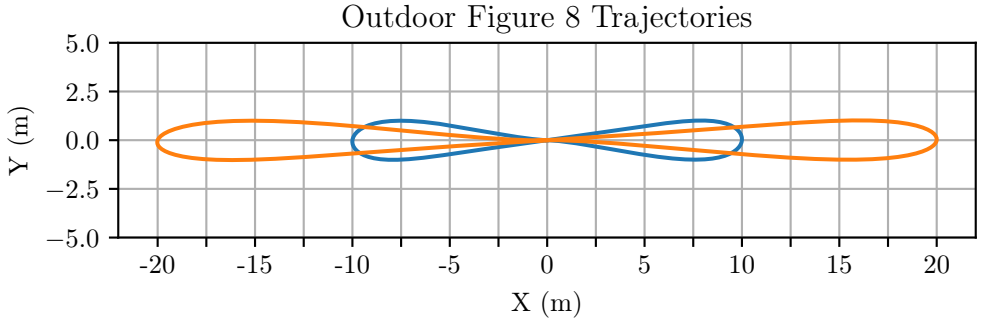


Figure C.1: The 20 m and 40 m figure 8s used in the outdoor experiments.

Table C.1: Position waypoints of outdoor figure 8 trajectories

$20\,\mathbf{m}$	$40\,\mathbf{m}$
$(0, 0)$	$(0, 0)$
$(-3.5, 0.5)$	$(-7.5, 0.5)$
$(-7.5, 1)$	$(-15, 1)$
$(-10, 0)$	$(-20, 0)$
$(-7.5, -1)$	$(-15, -1)$
$(-3.5, -0.5)$	$(-7.5, -0.5)$
$(0, 0)$	$(0, 0)$
$(3.5, 0.5)$	$(-7.5, 0.5)$
$(7.5, 1)$	$(15, 1)$
$(10, 0)$	$(20, 0)$
$(7.5, -1)$	$(15, -1)$
$(3.5, -0.5)$	$(7.5, -0.5)$
$(0, 0)$	$(0, 0)$

Table C.2: Common parameters for outdoor trajectories

Parameter	**Value**
Polynomial Order	9 ($N = 10$)
Derivative Minimized	4 (snap)
Soft Constraint Weight	10^2
Initial Relative Stepsize	10^{-5}
Inequality Constraint Tolerance	10^{-1}
Random Seed	0

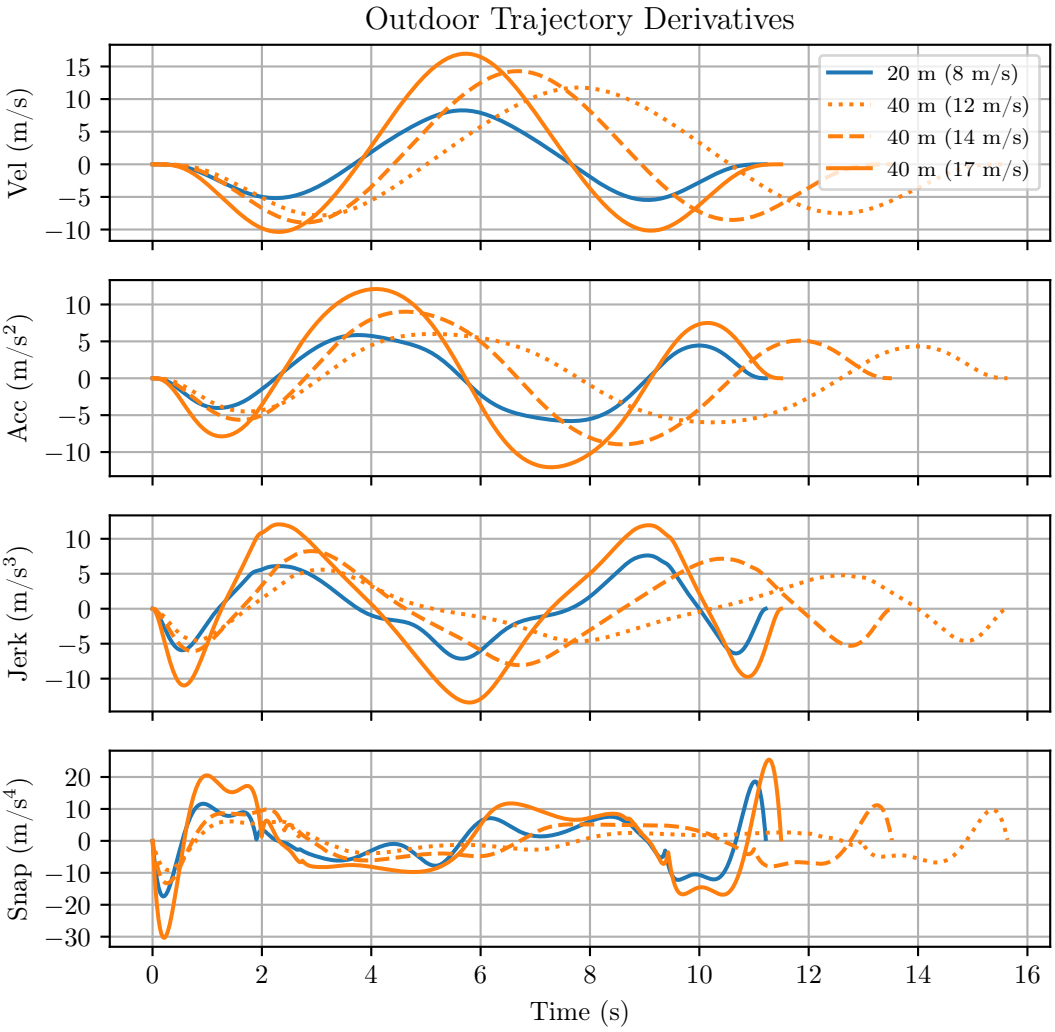


Figure C.2: The higher order derivatives of the four figure 8 trajectories used in the outdoor experiments.

APPENDIX

D

Theoretical Comparison between Feedback Linearization and Feedforward Linearization

While we have examined both feedforward linearization and feedback linearization for quadrotor controls, the choice of one or the other is unclear and relies on trial and error.

We propose a theoretical error analysis that will provide conditions and bounds on the relative performance improvement of feedforward linearization over feedback linearization. The goal is to answer one or all of the following questions.

1. Under what conditions does feedforward linearization achieve lower tracking error than feedback linearization?
2. What is the most appropriate representation of model error for carrying out this analysis?
3. How can feedback linearization and feedforward linearization be combined to achieve lower tracking error than either one?

This work seeks to supplement the thesis statement by providing a principled theoretical basis and framework for control strategy selection as a function of operating parameters and known vehicle and environment uncertainties.

D.1 Related Works

Prior theoretical work on analyzing feedforward linearization and feedback linearization has largely focused on stability and robustness [38]. Experimental work from the same authors has compared the two approaches using a ball and plate experiment [40] and found that feedforward linearization achieves slightly lower tracking error and better robustness over feedback linearization. There are few works to be found that provide theoretical results on the relative tracking error performance difference between feedback and feedforward linearization in the presence of model uncertainty.

In Chapter 2, we showed the differential flatness of the quadrotor and derived the mapping $\mathcal{T}$ from the flat state to the state and control.

Feedback linearization computes the control α using

$$\alpha = \mathcal{T}(a, j, s_{\text{des}}, \psi, \dot{\psi}, \ddot{\psi}), \tag{D.1}$$

while feedforward linearization computes the control α using

$$\alpha = \mathcal{T}(a_{\text{ref}}, j_{\text{ref}}, s_{\text{des}}, \psi_{\text{ref}}, \dot{\psi_{\text{ref}}}, \ddot{\psi_{\text{des}}}), \tag{D.2}$$

where s_{des} is as in (4.34).

D.2 Outcomes

This work seeks to achieve the following outcomes.

1. Derive conditions on model error, trajectory statistics, and system dynamics under which feedforward linearization achieves lower tracking error than feedback linearization in a meaningful and quantitative way.
2. Propose a control strategy that can leverage the above conditions to outperform both feedback linearization and feedforward linearization in a quantitatively guaranteed manner.

D.3 Theory

Consider a single-input single-output system of the form

$$\dot{x} = f^*(x) + u \tag{D.3}$$

with an associated reference x_{ref} and $\dot{x}_{\text{ref}} = v_{\text{ref}}$.

Define state error as

$$x_e = x - x_{\text{ref}} \tag{D.4}$$

and state cost as

$$x_c = \frac{1}{2} x_e^2. \tag{D.5}$$

Evaluating the time derivative of the cost:

$$\begin{aligned} \dot{x}_c &= x_e \dot{x}_e && \text{(D.6)} \\ &= x_e(\dot{x} - v_{\text{ref}}) && \text{(D.7)} \\ &= x_e(f^*(x) + u - v_{\text{ref}}) && \text{(D.8)} \end{aligned}$$

Now let's consider controllers of the form

$$u = -k x_e + v_{\text{ref}} - f(z) \tag{D.9}$$

where z is a stand in for either x or x_{ref}. Here f represents the best available guess of the true system dynamics f^*.

$$\dot{x}_c = x_e(f^*(x) - kx_e - f(z)) \tag{D.10}$$
$$= -kx_e^2 + x_e(f^*(x) - f(z)) \tag{D.11}$$
$$= -2kx_c + x_e(f^*(x) - f(z)) \tag{D.12}$$

We consider two different controllers: *feedback linearization* (FB) and *feedforward linearization* (FF). FB uses the true state to negate the dynamics, while FF uses the reference state.

$$z_{\text{FB}} = x \tag{D.13}$$
$$z_{\text{FF}} = x_{\text{ref}} \tag{D.14}$$

D.3.1 Cost Analysis

Plugging each control law into the time-derivative of the cost (D.12) we get

$$\dot{x}_c^{\text{FB}} = -2kx_c + x_e(f^*(x) - f(x)) \tag{D.15}$$
$$\dot{x}_c^{\text{FF}} = -2kx_c + x_e(f^*(x) - f(x_{\text{ref}})) \tag{D.16}$$

To analyze the control performance of feedback linearization (FB) relative to feedforward linearization (FF), we compute the difference in the time-derivative of the cost, at a fixed x_e and thus a fixed x.

$$\dot{x}_c^{\text{FB}} - \dot{x}_c^{\text{FF}} = x_e\left(f(x_{\text{ref}}) - f(x)\right) \tag{D.17}$$

Interestingly, the difference in performance only depends on the estimated system model f and not on the true system model f^*. We can see that FB performs worse (has a greater time derivative of the cost than FF) when x_e and $f(x_{\text{ref}}) - f(x)$ have the same sign. For example, if f is a decreasing function, then FF performs better.

Another thing to note, is that one can make FF arbitrarily better than FB (and vice-versa) by choosing f. However, such a choice is likely to make the control performance arbitrary low.

In some sense, the condition that f is a decreasing function is enforcing that the system under the estimated model is stable in the no-control case, suggesting that FF should be preferred in those situations.

D.3.2 Local Analysis

We wish to analyze the behavior of the system under model mismatch, when starting with zero error. That is, the initial conditions are such that $x = x_{\text{ref}}$ and thus $x_e = x_c = 0$.

We first introduce a variable representing the "dynamics mismatch" e_d.

$$e_d = f^*(x) - f(z) \tag{D.18}$$

Here, z is stand-in for either x, corresponding to the feedback linearization controller, or x_{ref}, corresponding to the feedforward linearization controller.

Applying (D.18) to (D.12),

$$\dot{x}_c = -2kx_c + x_e e_d \tag{D.19}$$

We see that at $x_c = 0$, the first derivative of the cost is also zero. We differentiate again.

$$\ddot{x}_c = -2k\dot{x}_c + \dot{x}_e e_d + x_e \dot{e}_d \tag{D.20}$$

Computing $\dot{x}_e$ in terms of e_d, we get

$$\dot{x}_e = e_d - kx_e \tag{D.21}$$

Thus the second derivative of the cost at $x_e = 0$ is

$$\ddot{x}_c(x = x_{\text{ref}}) = e_d^2 \tag{D.22}$$

We see that as long as there is model mismatch, i.e. $e_d \neq 0$, the error will begin to grow. However, since $x = x_{\text{ref}}$, $e_d^{\text{FB}} = e_d^{\text{FF}}$, implying that both FB and FF have the same second derivative of the cost at the start. We cannot determine which control strategy accumulates more error using (D.22).

We differentiate (D.20) to get

$$\begin{aligned} x_c^{(3)} &= -2k\ddot{x}_c + \ddot{x}_e e_d + \dot{x}_e \dot{e}_d + \dot{x}_e \dot{e}_d + x_e \ddot{e}_d && \text{(D.23)} \\ &= -2k\ddot{x}_c + \ddot{x}_e e_d + 2\dot{x}_e \dot{e}_d + x_e \ddot{e}_d && \text{(D.24)} \end{aligned}$$

Evaluating (D.24) at $x = x_{\text{ref}}$, we get

$$\begin{aligned} x_c^{(3)}(x = x_{\text{ref}}) &= -2ke_d^2 + (\dot{e}_d - k\dot{x}_e)e_d + 2e_d\dot{e}_d && \text{(D.25)} \\ &= -2ke_d^2 + 3e_d\dot{e}_d - ke_d^2 && \text{(D.26)} \\ &= 3(e_d\dot{e}_d - ke_d^2) && \text{(D.27)} \end{aligned}$$

Now, we compute $\dot{e}_d$ for both FB and FF, at $x = x_{\text{ref}}$.

$$\begin{aligned} \dot{e}_d &= f_x^*\dot{x} - f_x\dot{z} && \text{(D.28)} \\ \dot{e}_d^{\text{FB}} &= f_x^*(f^* - kx_e + v_{\text{ref}} - f) - f_x(f^* - kx_e + v_{\text{ref}} - f) && \text{(D.29)} \\ &= (f_x^* - f_x)(e_d + v_{\text{ref}}) && \text{(D.30)} \\ \dot{e}_d^{\text{FF}} &= f_x^*(f^* - kx_e + v_{\text{ref}} - f) - f_x v_{\text{ref}} && \text{(D.31)} \\ &= f_x^* e_d + (f_x^* - f_x)v_{\text{ref}} && \text{(D.32)} \end{aligned}$$

We compute the difference in the third derivative of the cost between FB and FF.

$$x_c^{(3),\mathrm{FB}} - x_c^{(3),\mathrm{FF}} = 3e_d(\dot{e}_d^{\mathrm{FB}} - \dot{e}_d^{\mathrm{FF}}) \tag{D.33}$$
$$= 3e_d(-f_x e_d) \tag{D.34}$$
$$= -3f_x e_d^2 \tag{D.35}$$

We note that when FB has a smaller third derivative of the cost, FB's error accumulates slower than FF. This occurs when $f_x > 0$, i.e. when the gradient of the estimated model is positive. On the other hand, when $f_x < 0$, FF will accumulate error slower than FB.

Bibliography

[1] Pieter Abbeel, Varun Ganapathi, and Andrew Y. Ng. Learning vehicular dynamics, with application to modeling helicopters. *Proceedings of the 18th International Conference on Neural Information Processing Systems*, NIPS'05, pp. 1–8. 2005. (see page: 42)

[2] Markus W. Achtelik, Simon Lynen, Margarita Chli, and Roland Siegwart. Inversion based direct position control and trajectory following for micro aerial vehicles. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2933–2939. 2013. (see page: 59)

[3] Adeel Akhtar, Steven L. Waslander, and Christopher Nielsen. Path following for a quadrotor using dynamic extension and transverse feedback linearization. *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 3551–3556. 2012. (see page: 59)

[4] Suguru Arimoto, Sadao Kawamura, and Fumio Miyazaki. Bettering operation of Robots by learning. In *Journal of Robotic Systems*, vol. 1(2):123–140, 1984. (see page: 76)

[5] Anil Aswani, Humberto Gonzalez, S. Shankar Sastry, and Claire Tomlin. Provably safe and robust learning-based model predictive control. In *Automatica*, vol. 49(5):1216–1226, 2013. (see pages: 34, 42)

[6] Sreeram V. Balakrishnan. Fast incremental adaptation using maximum likelihood regression and stochastic gradient descent. *INTERSPEECH*. 2003. (see page: 42)

[7] Moses Bangura and Robert Mahony. Nonlinear dynamic modeling for high performance control of a quadrotor. *Australasian conference on robotics and automation*, pp. 1–10. 2012. (see page: 41)

[8] Sanjay P. Bhat and Dennis S. Bernstein. A topological obstruction to continuous global stabilization of rotational motion and the unwinding phenomenon. In *Systems & Control Letters*, vol. 39(1):63–70, 2000. (see page: 65)

[9] Karen Bodie, Maximilian Brunner, Michael Pantic, Stefan Walser, Patrick Pfändler, et al. An Omnidirectional Aerial Manipulation Platform for Contact-Based Inspection. *Robotics: Science and Systems XV*. 2019. (see pages: 12, 57)

[10] Karen Bodie, Zachary Taylor, Mina Kamel, and Roland Siegwart. Towards Efficient Full Pose Omnidirectionality with Overactuated MAVs. *Proceedings of the 2018 International Symposium on Experimental Robotics (ISER)*, vol. 11, pp. 85–95. 2020. (see page: 21)

[11] Rogerio Bonatti, Yanfu Zhang, Sanjiban Choudhury, Wenshan Wang, and Sebastian Scherer. Autonomous Drone Cinematographer: Using Artistic Principles to Create Smooth, Safe, Occlusion-Free Trajectories for Aerial Filming. *Proceedings of the 2018 International Symposium on Experimental Robotics (ISER)*, vol. 11, pp. 119–129. 2020. (see page: 12)

[12] Dario Brescianini and Raffaello D'Andrea. Tilt-Prioritized Quadrocopter Attitude Control. In *IEEE Transactions on Control Systems Technology*, vol. 28(2):376–387, 2020. (see pages: 70, 145, and 146)

[13] Dario Brescianini, Markus Hehn, and Raffaello D'Andrea. Nonlinear Quadrocopter Attitude Control: Technical Report. Technical report, ETH Zurich, 2013. (see pages: 33, 142, and 146)

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901. 2020. (see page: 118)

[15] Michael Burri, Michael Bloesch, Dominik Schindler, Igor Gilitschenski, Zachary Taylor, et al. Generalized Information Filtering for MAV Parameter Estimation. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3124–3130. 2016. (see page: 41)

[16] Michael Burri, Janosch Nikolic, Helen Oleynikova, Markus W. Achtelik, and Roland Siegwart. Maximum Likelihood Parameter Identification for MAVs. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4297–4303. 2016. (see page: 41)

[17] Dong Eui Chang and Yongsoon Eun. Global Chartwise Feedback Linearization of the Quadcopter With a Thrust Positivity Preserving Dynamic Extension. In *IEEE Transactions on Automatic Control*, vol. 62(9):4747–4752, 2017. (see page: 59)

[18] Vishnu R. Desaraju. *Safe, Efficient, and Robust Predictive Control of Constrained Nonlinear Systems*. Doctoral, Carnegie Mellon University, 2017. (see pages: 41, 42)

[19] Vishnu R Desaraju, Alexander E Spitzer, Cormac O'Meadhra, Lauren Lieu, and Nathan Michael. Leveraging experience for robust, adaptive nonlinear MPC on computationally constrained systems with time-varying state uncertainty. In *The International Journal of Robotics Research*, vol. 37(13-14):1690–1712, 2018. (see pages: 34, 37)

[20] Alain Droniou, Serena Ivaldi, Vincent Padois, and Olivier Sigaud. Autonomous online learning of velocity kinematics on the iCub: a comparative study. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3577–3582. 2012. (see page: 42)

[21] Matthias Faessler, Davide Falanga, and Davide Scaramuzza. Thrust Mixing, Saturation, and Body-Rate Control for Accurate Aggressive Quadrotor Flight. In *IEEE Robotics and Automation Letters*, vol. 2(2):476–482, 2017. (see pages: 32, 41, and 60)

[22] Matthias Faessler, Flavio Fontana, Christian Forster, and Davide Scaramuzza. Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1722–1729. 2015. (see pages: 142, 146)

[23] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories. In *IEEE Robotics and Automation Letters*, vol. 3(2):620–626, 2018. (see pages: 22, 30, 42, and 43)

[24] Davide Falanga, Philipp Foehn, Peng Lu, and Davide Scaramuzza. PAMPC: Perception-Aware Model Predictive Control for Quadrotors. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8. 2018. (see page: 34)

[25] Jeff Ferrin, Robert Leishman, Randy Beard, and Tim McLain. Differential Flatness Based Control of a Rotorcraft For Aggressive Maneuvers. *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2688–2693. 2011. (see page: 42)

[26] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. On Differentially Flat Nonlinear Systems. In *IFAC Proceedings Volumes*, vol. 25(13):159–163, 1992. (see page: 21)

[27] Juan Manuel Florez, Delphine Bellot, and Guillaume Morel. LWPR-Model Based Predictive Force Control for Serial Comanipulation in Beating Heart Surgery. *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 320–326. 2011. (see page: 42)

[28] Simone Formentin and Marco Lovera. Flatness-based control of a quadrotor helicopter via feedforward linearization. *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pp. 6171–6176. 2011. (see page: 30)

[29] Emil Fresk and George Nikolakopoulos. Full Quaternion Based Attitude Control for a Quadrotor. *2013 European Control Conference (ECC)*, pp. 3864–3869. 2013. (see pages: 70, 144, and 146)

[30] Oliver Fritsch, Paul De Monte, Michael Buhl, and Boris Lohmann. Quasi-static Feedback Linearization for the Translational Dynamics of a Quadrotor Helicopter. *2012 American Control Conference (ACC)*, pp. 125–130. 2012. (see page: 30)

[31] Kanishke Gamagedara, Mahdis Bisheban, Evan Kaufman, and Taeyoung Lee. Geometric Controls of a Quadrotor UAV with Decoupled Yaw Control. *2019 American Control Conference (ACC)*, pp. 3285–3290. 2019. (see pages: 142, 146)

[32] Arjan Gijsberts and Giorgio Metta. Incremental learning of robot dynamics using random features. *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 951–956. 2011. (see pages: 39, 42)

[33] Arjan Gijsberts and Giorgio Metta. Real-time model learning using Incremental

Sparse Spectrum Gaussian Process Regression. In *Neural Networks*, vol. 41:59–69, 2013. (see pages: 38, 39, 42, and 68)

[34] Farhad Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric Nonlinear PID Control of a Quadrotor UAV on SE(3). *2013 European Control Conference (ECC)*, pp. 3845–3850. 2013. (see pages: 33, 143, and 146)

[35] Melissa Greeff and Angela P. Schoellig. Flatness-Based Model Predictive Control for Quadrotor Trajectory Tracking. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6740–6745. 2018. (see page: 34)

[36] Melissa Greeff and Angela P. Schoellig. Exploiting Differential Flatness for Robust Learning-Based Tracking Control Using Gaussian Processes. In *IEEE Control Systems Letters*, vol. 5(4):1121–1126, 2021. (see page: 60)

[37] Daniel H Grollman and Odest Chadwicke Jenkins. Sparse Incremental Learning for Interactive Robot Control Policy Estimation. *2008 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3315–3320. 2008. (see page: 42)

[38] Veit Hagenmeyer. *Robust nonlinear tracking control based on differential flatness*. Doctoral, University of Paris, France, 2003. (see pages: 83, 153)

[39] Veit Hagenmeyer and Emmanuel Delaleau. Exact feedforward linearization based on differential flatness. In *International Journal of Control*, vol. 76(6):537–556, 2003. (see pages: 59, 83)

[40] Veit Hagenmeyer, Stefan Streif, and Michael Zeitz. Flatness-Based Feedforward and Feedback Linearisation of the Ball & Plate Lab Experiment. In *IFAC Proceedings Volumes*, vol. 37(13):219–224, 2004. (see page: 153)

[41] Drew Hanover, Philipp Foehn, Sihao Sun, Elia Kaufmann, and Davide Scaramuzza. Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors. In *IEEE Robotics and Automation Letters*, vol. 7(2):690–697, 2022. (see page: 117)

[42] Markus Hehn and Raffaello D'Andrea. Real-Time Trajectory Generation for Quadrocopters. In *IEEE Transactions on Robotics*, vol. 31(4):877–892, 2015. (see page: 42)

[43] J. D. Hunter. Matplotlib: A 2D graphics environment. In *Computing in Science & Engineering*, vol. 9(3):90–95, 2007.

[44] Alberto Isidori. *Nonlinear control systems*. 1995. (see pages: 59, 60, and 61)

[45] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. 2001–. (see page: 49)

[46] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Matthias Müller, Vladlen Koltun, et al. Deep Drone Acrobatics. *Robotics: Science and Systems XVI*. 2020. (see page: 117)

[47] T. John Koo and Shankar Sastry. Output tracking control design of a helicopter

model based on approximate linearization. *Proceedings of the 37th IEEE Conference on Decision and Control (CDC)*, vol. 4, pp. 3635–3640. 1998. (see page: 59)

[48] Dave Kooijman, Angela P. Schoellig, and Duarte J. Antunes. Trajectory Tracking for Quadrotors with Attitude Control on $\mathcal{S}^2 \times \mathcal{S}^1$. *2019 18th European Control Conference (ECC)*, pp. 4002–4009. 2019. (see pages: 33, 34, 142, 144, and 146)

[49] Daewon Lee, H. Jin Kim, and Shankar Sastry. Feedback Linearization vs. Adaptive Sliding Mode Control for a Quadrotor Helicopter. In *International Journal of Control, Automation and Systems*, vol. 7(3):419–428, 2009. (see pages: 12, 59)

[50] Taeyoung Lee. Exponential stability of an attitude tracking control system on SO(3) for large-angle rotational maneuvers. In *Systems & Control Letters*, vol. 61(1):231–237, 2012. (see pages: 32, 143, and 146)

[51] Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). *49th IEEE Conference on Decision and Control (CDC)*, pp. 5420–5425. 2010. (see pages: 12, 143, and 146)

[52] Ian Lenz, Ross Knepper, and Ashutosh Saxena. DeepMPC: Learning Deep Latent Features for Model Predictive Control. *Robotics: Science and Systems XI*. 2015. (see page: 34)

[53] Qiyang Li, Jingxing Qian, Zining Zhu, Xuchan Bao, Mohamed K. Helwa, et al. Deep Neural Networks for Improved, Impromptu Trajectory Tracking of Quadrotors. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5183–5189. 2017. (see page: 41)

[54] Weiwei Li and Emanuel Todorov. Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems. *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics*, pp. 222–229. 2004. (see pages: 34, 42)

[55] Sergei Lupashin, Markus Hehn, Mark W. Mueller, Angela P. Schoellig, Michael Sherback, et al. A platform for aerial robotics research and demonstration: The Flying Machine Arena. In *Mechatronics*, vol. 24(1):41–54, 2014. (see page: 22)

[56] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1642–1648. 2010. (see page: 41)

[57] Zachary Manchester and Scott Kuindersma. DIRTREL: Robust trajectory optimization with ellipsoidal disturbances and LQR feedback. *Robotics: Science and Systems (RSS)*. 2017. (see pages: 12, 42)

[58] Andrew McHutchon and Carl Edward Rasmussen. Gaussian Process Training with Input Noise. *Advances in Neural Information Processing Systems*, vol. 24. 2011. (see page: 116)

[59] Christopher D. McKinnon and Angela P. Schoellig. Learning multimodal models for robot dynamics online with a mixture of gaussian process experts. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 322–328, 2017. (see page: 42)

[60] Daniel Mellinger. *Trajectory Generation and Control for Quadrotors*. Doctoral, University of Pennsylvania, 2012. (see page: 41)

[61] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2520–2525. 2011. (see pages: 22, 30, 41, 42, 143, and 146)

[62] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The GRASP Multiple Micro-UAV Testbed. In *IEEE Robotics & Automation Magazine*, vol. 17(3):56–65, 2010. (see page: 62)

[63] Buddy Michini and Jonathan How. L1 Adaptive Control for Indoor Autonomous Vehicles: Design Process and Flight Testing. *Proceeding of AIAA Guidance, Navigation, and Control Conference*, pp. 5754–5768. 2009. (see pages: 13, 35, and 73)

[64] V. Mistler, A. Benallegue, and N.K. M'Sirdi. Exact linearization and noninteracting control of a 4 rotors helicopter via dynamic feedback. *Proceedings of the 10th IEEE International Workshop on Robot and Human Interactive Communication. ROMAN 2001*, pp. 586–593. 2001. (see page: 59)

[65] Benjamin Morrell, Marc Rigter, Gene Merewether, Robert Reid, Rohan Thakker, et al. Differential Flatness Transformations for Aggressive Quadrotor Flight. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5204–5210. 2018. (see pages: 41, 42)

[66] Mark Wilfried Mueller. Multicopter attitude control for recovery from large disturbances. In *arXiv [cs]*, 2018. (see pages: 142, 143, 144, 145, and 146)

[67] Richard M. Murray, Muruhan Rathinam, and Willem Sluis. Differential Flatness of Mechanical Control Systems: A Catalog of Prototype Systems. *Proceedings of the 1995 ASME International Congress and Exposition*, p. 9. 1995. (see page: 22)

[68] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. In *Cognitive Processing*, vol. 12(4):319–340, 2011. (see page: 41)

[69] Barza Nisar, Philipp Foehn, Davide Falanga, and Davide Scaramuzza. VIMO: Simultaneous Visual Inertial Model-Based Odometry and Force Estimation. In *IEEE Robotics and Automation Letters*, vol. 4(3):2785–2792, 2019. (see page: 117)

[70] Sammy Omari, Pascal Gohl, Michael Burri, Markus Achtelik, and Roland Siegwart. Visual Industrial Inspection Using Aerial Robots. *Proceedings of the 2014 3rd International Conference on Applied Robotics for the Power Industry*, pp. 1–5. 2014. (see page: 12)

[71] Sophocles J. Orfanidis. *Introduction to signal processing*. Prentice Hall signal processing series. 1996. (see page: 124)

[72] Alberto Pretto, Stephanie Aravecchia, Wolfram Burgard, Nived Chebrolu, Christian Dornhege, et al. Building an Aerial-Ground Robotics System for Precision Farming: An Adaptable Solution. In *IEEE Robotics & Automation Magazine*, 2020. (see page: 12)

[73] Thomas Raffler, Jian Wang, and Florian Holzapfel. Path Generation and Control for Unmanned Multirotor Vehicles Using Nonlinear Dynamic Inversion and Pseudo Control Hedging. In *IFAC Proceedings Volumes*, vol. 46(19):194–199, 2013. (see page: 59)

[74] Ali Rahimi and Benjamin Recht. Random Features for Large-Scale Kernel Machines. *Advances in Neural Information Processing Systems*, pp. 1177–1184. 2007. (see pages: 38, 42, and 69)

[75] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. 2006. (see page: 37)

[76] Nathan Ratliff, Franziska Meier, Daniel Kappler, and Stefan Schaal. DOOMED: Direct Online Optimization of Modeling Errors in Dynamics. In *Big Data*, vol. 4(4):253–268, 2016. (see page: 117)

[77] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments. *Robotics Research*, vol. 114, pp. 649–666. 2016. (see pages: 12, 104, and 150)

[78] G. Rivera and O. Sawodny. Flatness-Based Tracking Control and Nonlinear Observer for a Micro Aerial Quadcopter. *AIP Conference Proceedings*, vol. 1281, pp. 386–389. 2010. (see page: 42)

[79] Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar. Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning. In *Applied Intelligence*, vol. 17(1):49–60, 2002. (see page: 42)

[80] Alexander Spitzer and Nathan Michael. Inverting Learned Dynamics Models for Aggressive Multirotor Control. *Robotics: Science and Systems XV*. 2019. (see pages: 41, 59, 60, 74, 88, and 92)

[81] Alexander Spitzer and Nathan Michael. Rotational Error Metrics for Quadrotor Control. In *arXiv [cs]*, 2020. (see page: 142)

[82] Alexander Spitzer and Nathan Michael. Feedback Linearization for Quadrotors with a Learned Acceleration Error Model. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6042–6048. 2021. (see page: 59)

[83] James Svacha, Kartik Mohta, and Vijay Kumar. Improving Quadrotor Trajectory Tracking by Compensating for Aerodynamic Effects. *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 860–866. 2017. (see page: 41)

[84] Wennie Tabib, Kshitij Goel, John Yao, Curtis Boirum, and Nathan Michael. Autonomous Cave Surveying With an Aerial Robot. In *IEEE Transactions on Robotics*, pp. 1–17, 2021. (see pages: 12, 57)

[85] Ezra Tal and Sertac Karaman. Accurate Tracking of Aggressive Quadrotor Trajectories using Incremental Nonlinear Dynamic Inversion and Differential Flatness. *2018 IEEE Conference on Decision and Control (CDC)*, pp. 4282–4288. 2018. (see page: 41)

[86] Guillem Torrente, Elia Kaufmann, Philipp Foehn, and Davide Scaramuzza. Data-Driven MPC for Quadrotors. In *IEEE Robotics and Automation Letters*, 2021. (see pages: 34, 42, and 43)

[87] Jonas Umlauft, Thomas Beckers, Melanie Kimmel, and Sandra Hirche. Feedback linearization using Gaussian processes. *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 5249–5255. 2017. (see page: 60)

[88] Michiel J. Van Nieuwstadt and Richard M. Murray. Real-time trajectory generation for differentially flat systems. In *Int. J. Robust Nonlinear Control*, vol. 8(11):995–1020, 1998. (see page: 22)

[89] Arun Venkatraman, Roberto Capobianco, Lerrel Pinto, Martial Hebert, Daniele Nardi, et al. Improved Learning of Dynamics Models for Control. *2016 International Symposium on Experimental Robotics (ISER)*, Springer Proceedings in Advanced Robotics, pp. 703–713. 2017. (see page: 116)

[90] Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Incremental Online Learning in High Dimensions. In *Neural Computation*, vol. 17(12):2602–2634, 2005. (see page: 42)

[91] Jian Wang, Thomas Bierling, Leonhard Höcht, Florian Holzapfel, Sebastian Klose, et al. Novel Dynamic Inversion Architecture Design for Quadrocopter Control. *Advances in Aerospace Guidance, Navigation and Control*, pp. 261–272. 2011. (see pages: 59, 60)

[92] Tyler Westenbroek, David Fridovich-Keil, Eric Mazumdar, Shreyas Arora, Valmik Prabhu, et al. Feedback Linearization for Uncertain Systems via Reinforcement Learning. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1364–1371. 2020. (see page: 60)

[93] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James Rehg, et al. Information Theoretic MPC for Model-Based Reinforcement Learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721. 2017. (see page: 34)

[94] A. Yeşildirek and F.L. Lewis. Feedback Linearization Using Neural Networks. In *Automatica*, vol. 31(11):1659–1664, 1995. (see page: 60)

[95] Weixuan Zhang, Marco Tognon, Lionel Ott, Roland Siegwart, and Juan Nieto. Active Model Learning using Informative Trajectories for Improved Closed-Loop Control on Real Robots. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4467–4473. 2021. (see page: 117)