Nonlinear Control 18-776

Final Project Report

Siddharth Gangadhar, Yash Jain, & Eric Xu

Electrical & Computer Engineering

Carnegie Mellon University

Pittsburgh PA

December 13, 2021

Contents

List of Figures **iv**

List of Tables **vi**

1 Introduction and Problem Statement **2**

1.1 Introduction 2

1.2 Problem Statement 3

1.3 Related Works 4

2 Drone Mathematical Modeling and Analysis **5**

2.1 Drone Frames of Reference 5

2.1.1 Rotational Matrices 6

2.2 Generalized Velocities 8

2.3 Euler-Lagrange Mechanics 9

2.3.1 The Lagrangian 9

2.3.2 Kinetic and Potential Energies 9

2.3.3 Generalized Exogenous Forces 10

2.3.4 The Euler-Lagrange Equations 10

2.4 State Space Representation 12

3 Assumptions & Proposed Control Strategies **13**

3.1 Assumptions 13

3.2 Proposed Controllers 14
3.3 Drone Parameters 15

4 Model Predictive Control - Control and State Constraints 16

4.1 Introduction 16
4.2 MPC Receding Horizon Optimization Algorithm 17
4.3 Simulation Setup and Profile 20
4.3.1 State Constraints 20
4.3.2 Control Constraints 20
4.4 Simulation Results 21
4.4.1 MPC with Control Constraints Only 21
4.4.2 MPC with Control and State Constraints 24
4.5 Ultimate Bound Analysis 27

5 Nonlinear Dynamic Inversion 29

5.1 Introduction 29
5.2 Controller Formulation 30
5.2.1 Baseline Nonlinear Dynamics Inversion Controller 30
5.2.2 Trajectory tracking with NDI 31
5.3 Simulation Setup 33
5.4 Simulation Results 33

6 Model Adaptive Nonlinear Dynamic Inversion 36

6.1 Introduction 36
6.2 Controller Formulation 37
6.2.1 Unscented Kalman Filter 38
6.3 Indirect adaptive NDI controller formulation 39
6.4 Simulation Setup 40
6.5 Simulation Results 41

6.6 Ultimate Bound Analysis 44

7 PX4 Implementation **45**

7.1 Our Approach 45

7.1.1 Cpp Implementation 45

7.2 Limitations and future work 46

7.2.1 The problems 46

7.2.2 Future work 47

A Video Simulation Links **48**

B C++ code for MPC **49**

Bibliography **66**

List of Figures

2.1 The inertial and body frames of a quadrotor drone 5

4.1 MPC Trajectory Tracking (Positions/Angles) 22
4.2 MPC Trajectory Tracking (Velocities/Angular Rates) 22
4.3 MPC Trajectory Tracking in 3D 23
4.4 MPC Rotor RPM with Control Constraint at ≈ 300 RPM 23
4.5 MPC Trajectory Tracking (Positions/Angles) 24
4.6 MPC Trajectory Tracking (Velocities/Angular Rates) 25
4.7 MPC zoomed in Trajectory Tracking (Velocities/Angular Rates) 25
4.8 MPC Trajectory Tracking in 3D 26
4.9 MPC Rotor RPM with Control and State Constraint at ≈ 300 RPM 26
4.10 Ultimate Bound of MPC with Control and State Constraints 28

5.1 NDI Trajectory Tracking (Positions/Angles) 33
5.2 NDI Trajectory Tracking (Velocities/Angular Rates) 34
5.3 NDI Trajectory Tracking in 3D 34
5.4 NDI Rotor RPM 35

6.1 Block diagram for indirect adaptive control 36
6.2 Adaptive NDI Trajectory Tracking (Positions/Angles) 41
6.3 Adaptive NDI Trajectory Tracking (Velocities/Angular Rates) 41
6.4 Adaptive NDI Trajectory Tracking in 3D 42

6.5 Adaptive NDI Rotor RPM 42
6.6 Adaptive NDI Wind Disturbance Estimation 43
6.7 Ultimate Bound of Adaptive NDI 44

List of Tables

3.1 Drone parameters for an actual drone 15

4.1 Tracking Errors for MPC with Control Constaints only v/s LQR with Saturation 24

4.2 Tracking Errors for MPC with Control and State Constaints v/s LQR with Saturation 27

5.1 Tracking Errors for NDI v/s LQR with Saturation 35

6.1 Tracking Errors for Adaptive NDI v/s LQR with Saturation 43

Acknowledgements

First and foremost, we are deeply grateful to out course professor Dr. Raffaele Romagnoli. His immense knowledge and experience has been a beacon of motivation during our semester. His invaluable advice, continuous support, and patience during our studies have allowed us to explore important areas of control theory and mathematics that have proved to be invaluable for our research work.

We also thank Sherya Ramesh, for all her hard work grading homeworks and projects; as fellow TAs we appreciate all the effort she has put in to make this course edifying and enjoyable.

Finally, we are indebted to our faculty advisor and mentor Dr. Yorie Nakahira. She has been a source of inspiration and support during this difficult semester. We are thankful for that.

Chapter 1

Introduction and Problem Statement

1.1 Introduction

Advances in computer architecture, sensor technology, power electronics, and control & dynamics theory have made autonomous aerial vehicles possible. Over the past few years, they have become central to the functions of various businesses and governmental organizations.

Among the general class of vertical takeoff and landing (VTOL) autonomous aerial vehicles, quadrotor drones are the most popular. Their inertial and aerodynamic simplicity coupled with the need for fewer actuators has made them The particular interest of the research community. The quadrotor design can be linked to two main advantages over comparable vertical take off and landing (VTOL) UAVs, such as helicopters. First, quadrotors do not require complex mechanical control linkages for rotor actuation, relying instead on fixed pitch rotors and using variation in motor speed for vehicle control [1]. Second, the use of four rotors ensures that individual rotors are smaller in diameter than the equivalent main rotor on a helicopter, relative to the airframe size. The individual rotors, therefore, store less kinetic energy during flight, mitigating the risk posed by the rotors should they entrain any objects [2].

However, the small size of these vehicles poses significant challenges. The small sensors used on these systems are much noisier than their larger counterparts. The compact structure of these vehicles also makes them more vulnerable to environmental effects [3].

Historically, quadcopters have been approximated as linear and solved with fast, effective approaches such as Linear Quadratic Regulator(LQR) and PID. However, linear approximations only hold true close to their set point, and grossly miss the true values outside this neighborhood. The small angle approximation assumes the quadcopter will remain near the neutral hovering state. While this assumption works well for applications in cinematography and surveillance, the model fails for use cases that require high-speed, dynamic maneuvering such as drone-racing and target tracking.

In this work, we analyze the nonlinear dynamics of the quadcopter and use it study and develop nonlinear control algorithms that can help achieve robust performance while maintaining reasonable control efforts and push the limits of what linear control can do.

1.2 Problem Statement

This work is composed of three main parts. First, we model the dynamics of the quadcopter. Second, we analyze and develop nonlinear controllers to track a figure-8 reference trajectory parameterized by time. Third, we study their characteristics and juxtapose their performance with the previously developed linear control algorithms. More specifically, we analyze the mean square tracking (and estimation error where necessary) and examine the ultimate bound of each controller for a given exogenous disturbance such as wind. Further, we also deploy one of the nonlinear control algorithms proposed on PX4 and simulate its performance in a more realistic setting.

1.3 Related Works

The Quadrotor has been in use for decades now has has become a prevalent vehicle in our modern society. Dating back from 1920, Etienne Oehmichen was the first (published and recorded) person to have experimented with rotor-based vehicle designs and among those designs, most prevalent was the multicoptor which has four rotors, hence the inception of the quadrotor. A couple years later in 1922, Dr. George de Bothezat and Ivan Jermore would develop a six-bladed rotor coptor; this coptor however had underwhelming performance in that it was highly unreliable and would often have power issues. Then in 1956 came the Convertawings Model A Quadrotor which was essentially the prototype to military quadrotor helicopters. This model would start the craze about quadrotors despite its initial lack of demand for its creation.

Moving onto the control of quadrotors; there are several papers which go into detail about control techniques for these vehicles. Many works have already shown that linearization of the dynamics and applying the usual kinds of linear control to this linearized model works quite well [1] [2] [4]. However, we can certainly improve performance of our controller via methods using more modern nonlinear techniques that have been developed. There are a wide array of papers that have already demonstrated the incredible power of nonlinear control such as those who have used techniques of feedback linearization/dynamic inversion [5] [6] [7] [3].

In this paper, we will discuss several nonlinear control techniques that are comparable and some, even surpassing regular linearized control techniques (i.e. the major example being LQR).

Chapter 2

Drone Mathematical Modeling and Analysis

2.1 Drone Frames of Reference

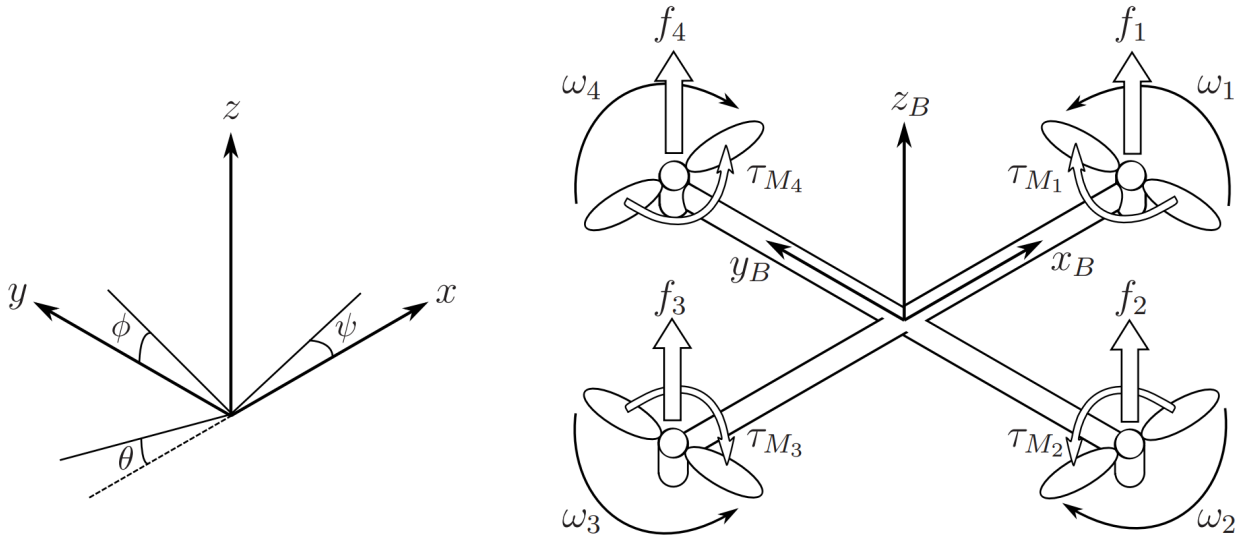

Figure 2.1: The inertial and body frames of a quadrotor drone

Figure 2.1 shows a quadrotor drone along with the reference inertial and body frames. We define the translational coordinates of the center of mass (C.M.) of the drone in the inertial

frame as:

$$\xi = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} \tag{2.1}$$

The drone attitude, i.e principle Euler angles, can be seen in Figure 2.1 and are defined in the inertial frame as:

$$\eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{2.2}$$

With this, we can define the generalized coordinate system of the drone as:

$$\mathbf{q} = \begin{bmatrix} \xi \\ \eta \end{bmatrix} \in \mathbb{R}^6 \tag{2.3}$$

Here, we see that $\dim(q) = 6$, which implies that the drone has six degrees of freedom. [8]

2.1.1 Rotational Matrices

As a result of the Euler angles being defined in the body frame of reference, all the exogenous rotor forces and moments will exist in the body frame and therefore would have to be transformed to the inertial frame. To do this, we make use of the ZYX Tait-Bryan rotation

matrices (typically used in aerospace applications) [8]

$$\mathcal{R}_\phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \tag{2.4}$$

$$\mathcal{R}_\theta = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \tag{2.5}$$

$$\mathcal{R}_\psi = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

The overall rotational matrix that transforms the inertial frame to the body frame of reference will be the multiplication of the individual rotation matrices $\mathcal{R}_\phi, \mathcal{R}_\theta,$ and $\mathcal{R}_\psi$ shown below:

$$\mathcal{R}_i^b = \mathcal{R}_\phi \mathcal{R}_\theta \mathcal{R}_\psi \tag{2.7}$$

$$\mathcal{R}_i^b = \begin{pmatrix} \cos\theta\cos\psi & \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix} \tag{2.8}$$

Further, 3D rotation groups form an isometry (preserves Euclidean distances) and therefore are unitary in nature. Consequently, their matrix inverse is the same as their transpose. This gives is the the transform from the body frame to the inertial [8]:

$$\mathcal{R}_b^i = \mathcal{R}_i^{b^{-1}} = \mathcal{R}_i^{b^T} \tag{2.9}$$

With this, we can transform forces and moments in the body frame to the inertial frame.

2.2 Generalized Velocities

Next, we obtain the generalized velocities to compute the canonical momenta for the drone. The transnational velocities are defined as:

$$\mathbf{v} = \dot{\xi} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{2.10}$$

The generalized rotational velocities, i.e the Euler rates, are defined as follows:

$$\mathbf{r} = \dot{\eta} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.11}$$

It is important to note that the generalized rotational velocities are **not the same** as the angular velocities, since they need to undergo a rotational transformation [8] as shown below:

$$\Omega = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \mathcal{R}_{\phi} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathcal{R}_{\theta}\mathcal{R}_{\phi} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{2.12}$$

$$\Omega = \underbrace{\begin{bmatrix} 1 & 0 & -sin(\theta) \\ 0 & cos(\phi) & cos(\theta)sin(\phi) \\ 0 & -sin(\phi) & cos(\phi)cos(\theta) \end{bmatrix}}_{W_{\eta}(\eta)} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.13}$$

$$\text{Therefore, we get: } \Omega = W_{\eta}(\eta)\dot{\eta} \tag{2.14}$$

2.3 Euler-Lagrange Mechanics

2.3.1 The Lagrangian

Lagrangian mechanics is a formulation of classical mechanics and is founded on the stationary action principle [8]. We define the Lagrangian as the difference between the total kinetic energy and total potential energy of the system to minimize a quantity called **Action** as follows:

$$A = \int_{t_i}^{t_f} \underbrace{\mathcal{T}(\mathbf{q}, \dot{\mathbf{q}}) - \mathcal{V}(\mathbf{q})}_{\mathcal{L}(\mathbf{q},\dot{\mathbf{q}})} dt \tag{2.15}$$

To minimize this integral, we require that the Lagrangian $\mathcal{L}(q, \dot{q})$ to satisfy the following PDE [8]:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}} - \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} = -\mathcal{F}_{\text{ext}} \tag{2.16}$$

This is called the Euler-Lagrange Equation. Here, $\mathcal{F}_{\text{ext}}$ is the sum of all external generalized forces acting on the system.

2.3.2 Kinetic and Potential Energies

Here, we define the linear and rotational kinetic energies of the drone, under the **assumption** that it is a rigid body along with the gravitational potential energy [8].

$$\mathcal{T}_\xi = \frac{1}{2} m \dot{\xi}^T \dot{\xi} \quad \text{Translational Kinetic Energy} \tag{2.17}$$

$$\mathcal{T}_\eta = \frac{1}{2}\Omega^T \mathcal{I} \Omega = \frac{1}{2}\dot{\eta}^T \underbrace{W_\eta^T \mathcal{I} W_\eta}_{\mathcal{J}(\eta)} \dot{\eta} \quad \text{Rotational Kinetic Energy} \tag{2.18}$$

$$\mathcal{V} = mgz \quad \text{Total Potential Energy} \tag{2.19}$$

Where, m is the mass of the drone and $\mathcal{I} \in \mathbb{R}^{3\times3}$ is the inertia tensor.

2.3.3 Generalized Exogenous Forces

Next, we derive the generalized exogenous forces acting on the drone:

$$\mathcal{F}_{\text{ext}} = \begin{bmatrix} \mathcal{F}_\xi \\ \mathcal{F}_\eta \end{bmatrix} \tag{2.20}$$

$$\mathcal{F}_\xi = \mathcal{R}_b^i \mathbf{T}_B \quad \text{Total Exogenous Thrust in the inertial frame} \tag{2.21}$$

$$\mathcal{F}_\eta = \mathcal{R}_b^i \mathbf{M}_B \quad \text{Total Exogenous Moment in the inertial frame} \tag{2.22}$$

From [4], we see that $f_i = K_F \omega_i^2$ and $\tau_i = K_M \omega_i^2$, and therefore the exogenous forces and moments acting in the body-frame of reference are as follows:

$$\mathbf{T}_B = \begin{bmatrix} 0 \\ 0 \\ f_1 + f_2 + f_3 + f_4 \end{bmatrix} \quad \text{Total Thrust Body Frame} \tag{2.23}$$

$$\mathbf{M}_B = \begin{bmatrix} l(-f_2 + f_4) \\ l(-f_1 + f_3) \\ \frac{K_M}{K_F}(-f_1 + f_2 - f_3 + f_4) \end{bmatrix} \quad \text{Total Moment in Body Frame} \tag{2.24}$$

Where, l is the distance between opposite rotors, as see in Figure 2.1.

2.3.4 The Euler-Lagrange Equations

Next, we compute individual partial derivatives shown in Eqn. 1.16:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \xi} \\ \frac{\partial \mathcal{L}}{\partial \eta} \end{bmatrix} \quad \frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \dot{\xi}} \\ \frac{\partial \mathcal{L}}{\partial \dot{\eta}} \end{bmatrix} \tag{2.25}$$

We can now compute the individual partial derivatives for the translational and rotational coordinates separately:

$$\frac{\partial \mathcal{L}}{\partial \xi} = \mathcal{Q}_\xi = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \qquad \frac{\partial \mathcal{L}}{\partial \dot{\xi}} = m\dot{\xi} \qquad \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\xi}} = m\ddot{\xi} \tag{2.26}$$

$$\frac{\partial \mathcal{L}}{\partial \eta} = \mathcal{Q}_\eta \qquad \frac{\partial \mathcal{L}}{\partial \dot{\eta}} = J(\eta)\dot{\eta} \qquad \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\eta}} = J(\eta)\ddot{\eta} + \dot{J}(\eta)\dot{\eta} \tag{2.27}$$

Comparing Eqn. 1.26 and 1.27 to Eqn. 1.16, we can obtain the second order dynamics of the system in the translational and rotational coordinates as follows:

$$m\ddot{\xi} = -\overbrace{A\dot{\xi}}^{\text{Drag}} + \mathcal{Q}_\xi + \mathcal{R}_b^i G_\xi u \tag{2.28}$$

$$J(\eta)\ddot{\eta} = -\underbrace{\dot{J}(\eta)\dot{\eta}}_{\text{Coriolis}} + \mathcal{Q}_\eta + \mathcal{R}_b^i G_\eta u \tag{2.29}$$

Assuming that the control input $u = \begin{bmatrix} f_1 & f_2 & f_3 & f_4 \end{bmatrix}^T$, we get the following

$$\mathbf{T}_B = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{G_\xi} u \quad \mathbf{M}_B = \underbrace{\begin{bmatrix} 0 & -l & 0 & l \\ -l & 0 & l & 0 \\ -\frac{K_M}{K_F} & \frac{K_M}{K_F} & -\frac{K_M}{K_F} & \frac{K_M}{K_F} \end{bmatrix}}_{G_\eta} u \tag{2.30}$$

2.4 State Space Representation

With our second order dynamics established, we can obtain the state-space representation in control-affine form as follows:

$$\dot{x} = \underbrace{\begin{bmatrix} \dot{\xi} \\ \frac{1}{m}(-A\dot{\xi} + \mathcal{Q}_\xi) \\ \dot{\eta} \\ J(\eta)^{-1}(-\dot{J}(\eta)\dot{\eta} + \mathcal{Q}_\eta) \end{bmatrix}}_{f(x)} + \underbrace{\begin{bmatrix} \mathbf{0} \\ \frac{1}{m}\mathcal{R}_b^i G_\xi \\ \mathbf{0} \\ J(\eta)^{-1}\mathcal{R}_b^i G_\eta \end{bmatrix}}_{g(x)} u \tag{2.31}$$

Where, the state-vector $x = \begin{bmatrix} \xi^T & \eta^T & \dot{\xi}^T & \dot{\eta}^T \end{bmatrix}^T \in \mathbb{R}^{12}$

Chapter 3

Assumptions & Proposed Control Strategies

3.1 Assumptions

In this chapter, we explore three nonlinear control strategies and discuss their advantages over traditional linear controllers, as well as their potential limitations. We have chosen these specific controllers due to their wide applicability in the industry, practicality, and, specifically for the quadcopter, their ability to track high-speed reference signals with low tracking error and high level of robustness. We must mention a few basic assumptions we have made while choosing these control policies:

- As discussed in [9], typical drone brushless motors come with a Kv (velocity constant) rating ranging from 600 rpm/V to 2500 rpm/V. This is rpm the motor would have to rotate at to generate 1V of back-emf. Conversely, when converted to rads/Vs, this would also be equal to the current needed to generate 1 Nm of torque. For our drone, we have assumed a Kv rating of 120 rpm/V and a battery voltage of 3.3V, which means that at steady state our motors would at most spin at 800×3.3 rpm or 2,640 rpm at no load.

- With the propeller acting as a damped inertial load, with force constant of $K_f = 2.980 \times 10^{-3}$ and moment constant of $K_m = 1.49 \times 10^{-3}$, we can assume that the max motor rpm will drop to 1,320 rpm. Therefore, this would be our saturation rpm [4].

- For exogenous disturbances, we assume wind in the form of a linear force perturbing the linear velocity derivatives in each direction. To make the simulation tractable, we assume a smooth bandlimited disturbance, as opposed to infinite bandwidth white Gaussian noise.

- The reference trajectory used in our simulations is a Lemniscate of Bernoulli. The mathematical expression is as follows:

$$x = \frac{Acos(wt - \frac{\pi}{2})}{(1 + sin^2(wt - \frac{\pi}{2}))(1 - e^{\frac{-t}{\tau}})} \tag{3.1}$$

$$y = \frac{Asin(wt - \frac{\pi}{2})cos(wt - \frac{\pi}{2})}{(1 + sin^2(wt - \frac{\pi}{2}))(1 - e^{\frac{-t}{\tau}})} \tag{3.2}$$

$$z = A(1 - e^{\frac{-t}{\tau}}) \tag{3.3}$$

Where, A = 20m, $w = \frac{2\pi}{20}$, and $\tau = 10$

3.2 Proposed Controllers

With these assumptions clearly stated. We list our proposed control policies.

- Model Predictive Control - With State and Control Constraints

- Nonlinear Dynamic Inversion

- Model Adaptive Nonlinear Dynamic Inversion - With State Filtering

We analyze, simulate, and discuss these controllers at length in subsequent chapters.

3.3 Drone Parameters

Drone Parameter	Value	Units
m	0.468	kg
l	0.225	m
I_x	4.856	gm^2
I_y	4.865	gm^2
I_z	8.801	gm^2
K_f	2.980×10^{-3}	kgm
K_m	1.490×10^{-3}	kgm^2
A	0.01	Ns/m

Table 3.1: Drone parameters for an actual drone

Table 3.1, shows our choice of parameters for the drone we will be simulating with. These values were chosen from an actual drone that I worked on previously.

Chapter 4

Model Predictive Control - Control and State Constraints

4.1 Introduction

As previously mentioned, drone rotors have limited actuation capability and are likely to saturate while performing aggressive maneuvers. Further, we would also want to limit the linear velocities of the drone to ensure safety in a working environment. These are important consideration to take into account when designing control algorithms for drones that must meet strict safety requirements and operate within actuator constraint, be it for safety or improving battery life. We can realize these actuator and speed constraints by employing techniques in Constrained Optimal Control, which is formulated as follows:

$$u_k = \arg\min_{u \in \mathbb{R}^m} \sum_{i=k}^{\infty} l_i(x_i, u_i) \tag{4.1}$$

$$\textbf{s.t } x_{i+1} = f(x_i, u) \tag{4.2}$$

$$x_{i+1} \in \mathcal{X} \subset \mathbb{R}^n \tag{4.3}$$

$$u \in \mathcal{U} \subset \mathbb{R}^m \tag{4.4}$$

Where, l_k is the run-time cost, $\mathcal{X}$ and $\mathcal{U}$ are state and control constraint sets respectively. However, such an infinite horizon optimization problem with constraints are computationally intractable. Therefore we consider a finite horizon optimization problem. The key idea behind MPC is that we are designing an infinite horizon sub-optimal controller by repeatedly solving a series of finite time optimal control problems in a receding horizon fashion. We only consider a portion of the original cost in the finite horizon problem that we solve at every time step; and we use the terminal cost to approximate the value function of the remainder of the infinite horizon optimal control problem. While this is one way to solve the dynamic programming problem, it does come with challenges, notably those associated with computational complexity, run time requirements, feasibility, and stability.

4.2 MPC Receding Horizon Optimization Algorithm

The general form of the finite receding horizon constrained optimal control problem of horizon size N is as follows:

$$\min_{u_k, u_{k+1}, \ldots, u_{k+N-1}} \sum_{i=k}^{k+N-1} l_i(x_i, u_i) \tag{4.5}$$

$$\textbf{s.t } x_{i+1} = f(x_i, u) \ \forall i \in \{k, k+1, k+2, \ldots, k+N-1\} \tag{4.6}$$

$$x_{i+1} \in \mathcal{X} \subset \mathbb{R}^n \ \forall i \in \{k, k+1, k+2, \ldots, k+N-1\} \tag{4.7}$$

$$u \in \mathcal{U} \subset \mathbb{R}^m \ \forall i \in \{k, k+1, k+2, \ldots, k+N-1\} \tag{4.8}$$

For our situation, however, we consider a quadratic runtime cost and locally linear discrete-time dynamics of the drone, such that:

$$l(x_k, u_k) = \frac{1}{2} x_k^T Q x_k + \frac{1}{2} u_k^T R u_k \tag{4.9}$$

$$x_{k+1} = A_d x_k + B_d u_k \tag{4.10}$$

To make the optimization problem simple and tractable, we examine the standard form of the quadratic program:

$$x^* = \arg\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T P x + q^T x \tag{4.11}$$

$$\textbf{s.t } Gx \leq h \tag{4.12}$$

Next, we want to rearrange the MPC optimization problem to this standard canonical form. To do this we compute the lifted dynamics of the system as follows:

$$\tilde{X}_k = \tilde{A}_d x_k + \tilde{B}_d \tilde{U}_k \tag{4.13}$$

$$\text{Where, } \tilde{X}_k = \begin{bmatrix} x_k^T & x_{k+1}^T & x_{k+2}^T & \dots & x_{k+N}^T \end{bmatrix} \tag{4.14}$$

$$\text{And, } \tilde{U}_k = \begin{bmatrix} u_k^T & u_{k+1}^T & u_{k+2}^T & \dots & u_{k+N-1}^T \end{bmatrix} \tag{4.15}$$

Where, the matrices $\tilde{A}$ and $\tilde{B}$ are defined as follows:

$$\tilde{A}_d = \begin{bmatrix} I \\ A_d \\ A_d^2 \\ \vdots \\ A_d^N \\ A_d^{N+1} \end{bmatrix} \tag{4.16}$$

$$\tilde{B}_d = \begin{bmatrix} B_d & 0 & 0 & \dots & 0 \\ A_d B_d & B_d & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_d^{N-1} B_d & A_d^{N-2} B_d & \dots & A_d B_d & B_d \end{bmatrix} \tag{4.17}$$

Further, for the state and control constraints, we consider linear matrix inequalities, i.e,

polyhedra sets for $\mathcal{X}$ and $\mathcal{U}$ as follows:

$$\mathcal{X} = \{x \in \mathbb{R}^n : L_x x \leq b_x\} \tag{4.18}$$

$$\mathcal{U} = \{u \in \mathbb{R}^m : L_u u \leq b_u\} \tag{4.19}$$

We also lift these matrices for all states in the prediction horizon, as follows:

$$\tilde{L}_x = \begin{bmatrix} L_x & 0 & 0 & \dots & 0 \\ 0 & L_x & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & L_x \end{bmatrix} \quad \tilde{b}_x = \begin{bmatrix} b_x \\ b_x \\ \vdots \\ b_x \end{bmatrix} \tag{4.20}$$

$$\tilde{L}_u = \begin{bmatrix} L_u & 0 & 0 & \dots & 0 \\ 0 & L_u & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & L_u \end{bmatrix} \quad \tilde{b}_u = \begin{bmatrix} b_u \\ b_u \\ \vdots \\ b_u \end{bmatrix} \tag{4.21}$$

$$\tag{4.22}$$

Further, the state and control penalty matrices are also lifted as follows:

$$\tilde{Q} = \begin{bmatrix} Q & 0 & 0 & \dots & 0 \\ 0 & Q & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & Q \end{bmatrix} \tag{4.23}$$

$$\tilde{R} = \begin{bmatrix} R & 0 & 0 & \dots & 0 \\ 0 & R & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & R \end{bmatrix} \tag{4.24}$$

With this, we can formulate the QP as follows to compute the MPC control trajectories:

$$\tilde{U}_k^*(x_k) = \arg\min_{\tilde{U}_k} \frac{1}{2}\tilde{U}_k^T[\tilde{B}^T\tilde{Q}\tilde{B} + \tilde{R}]\tilde{U}_k + (\tilde{B}^T\tilde{Q}\tilde{A}x_k)^T\tilde{U}_k \tag{4.25}$$

$$\textbf{s.t} \quad \begin{bmatrix} \tilde{L}_u \\ \tilde{L}_x\tilde{B} \end{bmatrix} \tilde{U}_k \leq \begin{bmatrix} \tilde{b}_u \\ \tilde{b}_x - \tilde{L}_x\tilde{A}_d x_k \end{bmatrix} \tag{4.26}$$

With this MPC algorithm setup, we are ready to formulate MPC for our quadrotor drone.

4.3 Simulation Setup and Profile

4.3.1 State Constraints

We have chosen to constrain the linear velocity of the drone in the z-direction as follows:

$$-1.5\text{m/s} \leq v_z \leq 1.5\text{m/s} \tag{4.27}$$

This gives us the following linear matrix inequality:

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{L_x} x_k \leq \underbrace{\begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}}_{b_x} \tag{4.28}$$

4.3.2 Control Constraints

For the control constraints, we limit the rotor speeds to 1.75 times the hover speed as the upper bound and 0.25 as the lower bound, which gives us the following linear matrix

inequality:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}}_{L_u} u_k \leq \underbrace{\begin{bmatrix} 1.75 \\ 1.75 \\ 1.75 \\ 1.75 \\ 1.75 \\ 1.75 \\ 1.75 \\ 1.75 \end{bmatrix}}_{b_u} \tag{4.29}$$

Further, the Q and R matrices are chosen as follows:

$$Q_{\text{mpc}} = \text{diag}(100, 100, 50, 5, 5, 5, 15, 15, 15, 1, 1, 1) \tag{4.30}$$

$$R_{\text{mpc}} = \text{diag}(0.001, 0.001, 0.001, 0.001) \tag{4.31}$$

We choose these rather drastic values to demonstrate the effectiveness of the control and state constraints.

4.4 Simulation Results

4.4.1 MPC with Control Constraints Only

To juxtapose the performance of the MPC with and without State Constraints, we first explore the performance of the controller with control constraints only.

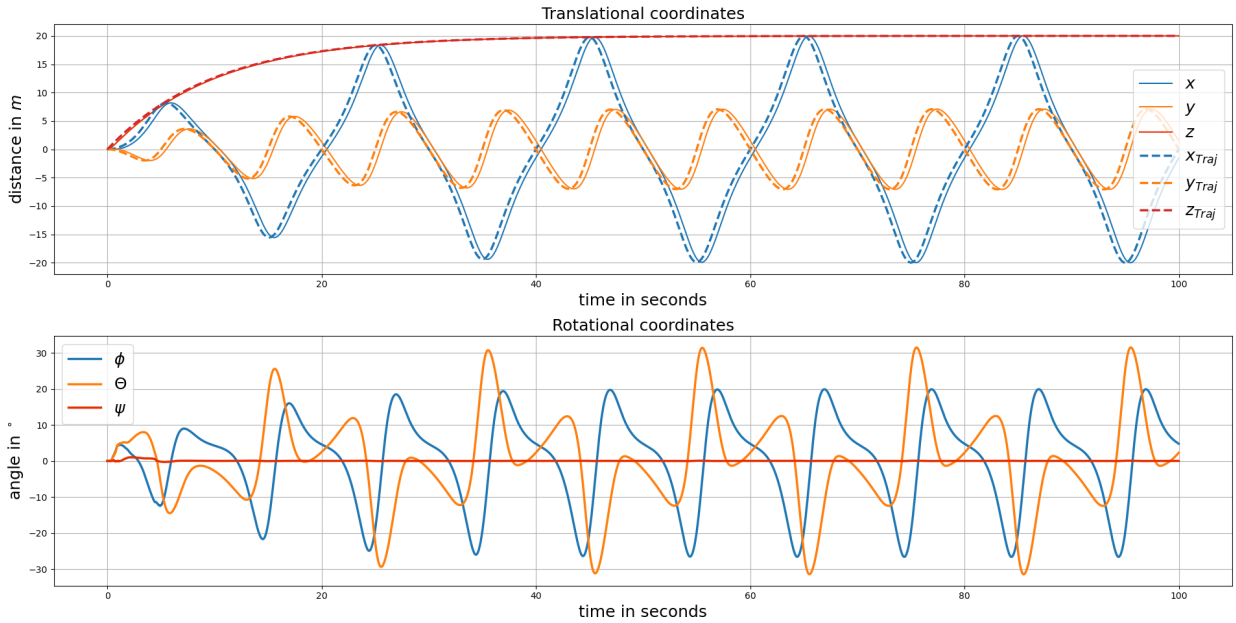

Figure 4.1: MPC Trajectory Tracking (Positions/Angles)

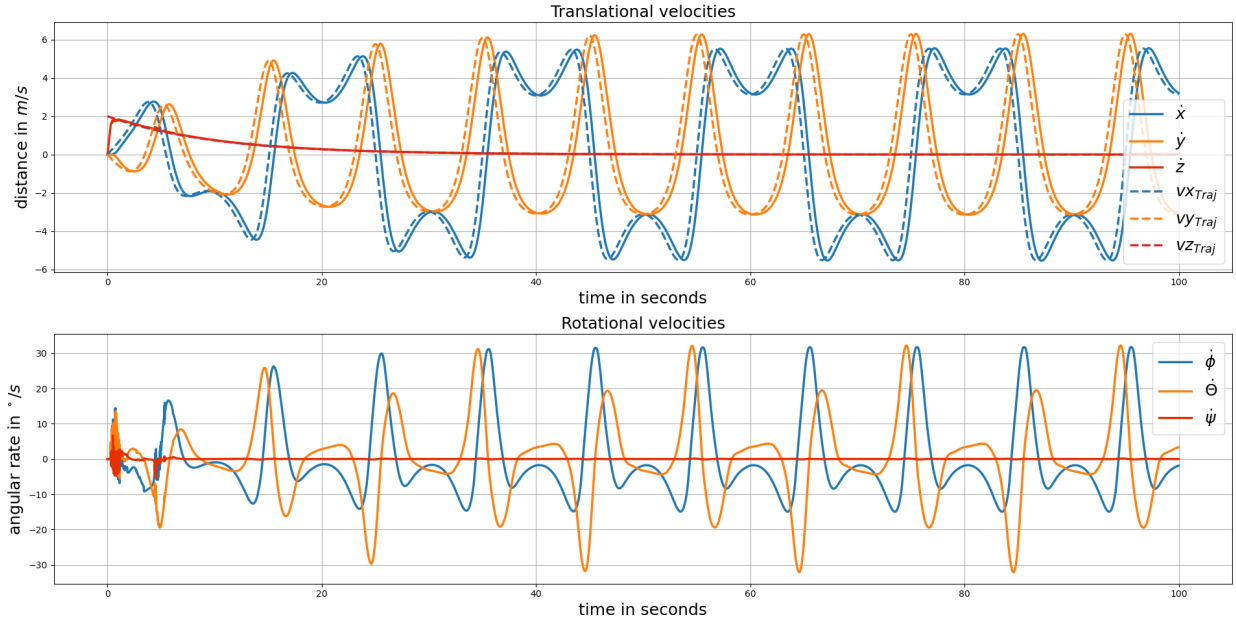

Figure 4.2: MPC Trajectory Tracking (Velocities/Angular Rates)

From Figures 4.1 and 4.2, we can see that the MPC tracks the given reference signal better than the LQR controller from Project Part - A. And while MPC and LQR are similar, the main advantage of MPC is that the state penalty matrix can be made large to aggressively

track the reference signal but at the same time, we can guarantee that the control efforts will stay within the chosen bounds, unlike with LQR where we are likely to run into saturation issues.

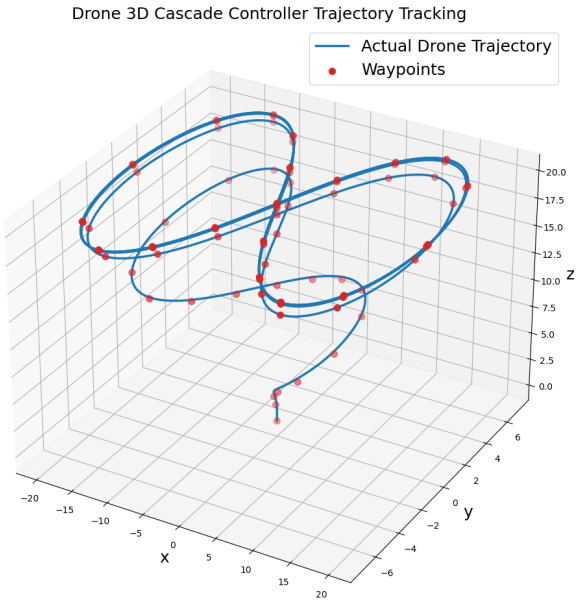

Figure 4.3: MPC Trajectory Tracking in 3D

Figure 4.3, shows the 3D trajectory of the drone with waypoints in red.

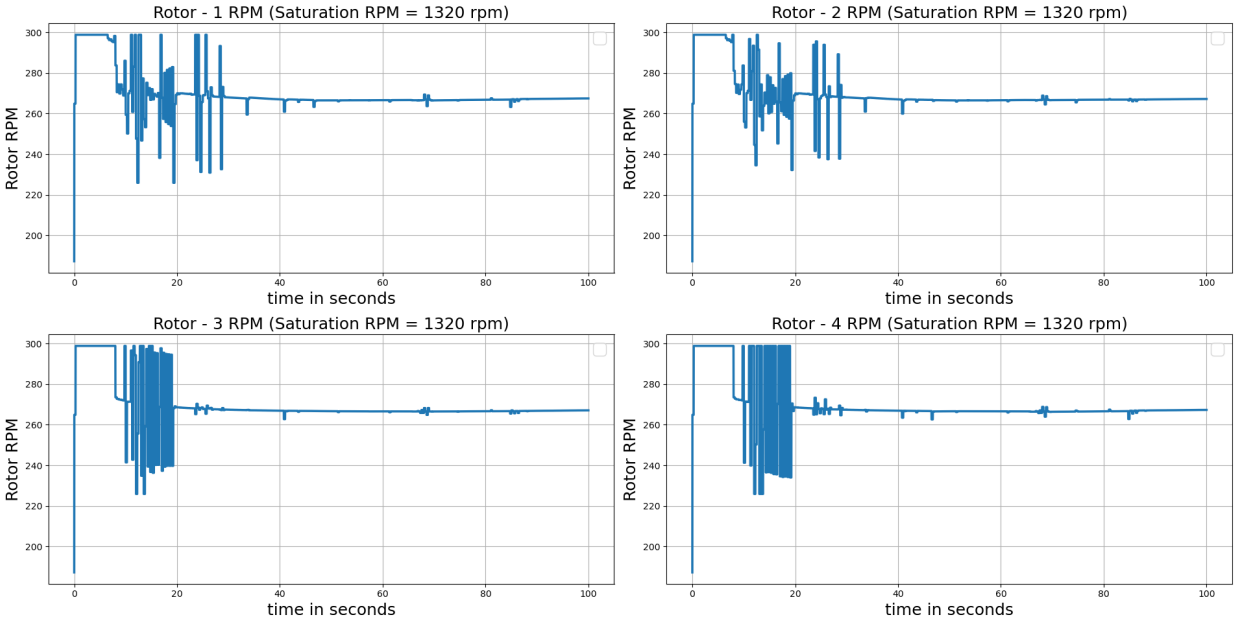

Figure 4.4: MPC Rotor RPM with Control Constraint at ≈ 300 RPM

Figure 4.4, shows the MPC Rotor RPM. We can see that the control effort stays within the chosen bounds. This ensures that the quadrotor's rotors do not exceed the saturation values and leading to failures and crashes.

Trajectory Coordinate	MPC	LQR
x	1.84855	6.59534
y	1.42350	4.59340
z	0.10685	7.40886

Table 4.1: Tracking Errors for MPC with Control Constaints only v/s LQR with Saturation

Table 4.1, compares the MSE tracking errors for MPC with control constraints only and Project Part A. We see a significant improvement in performance.

4.4.2 MPC with Control and State Constraints

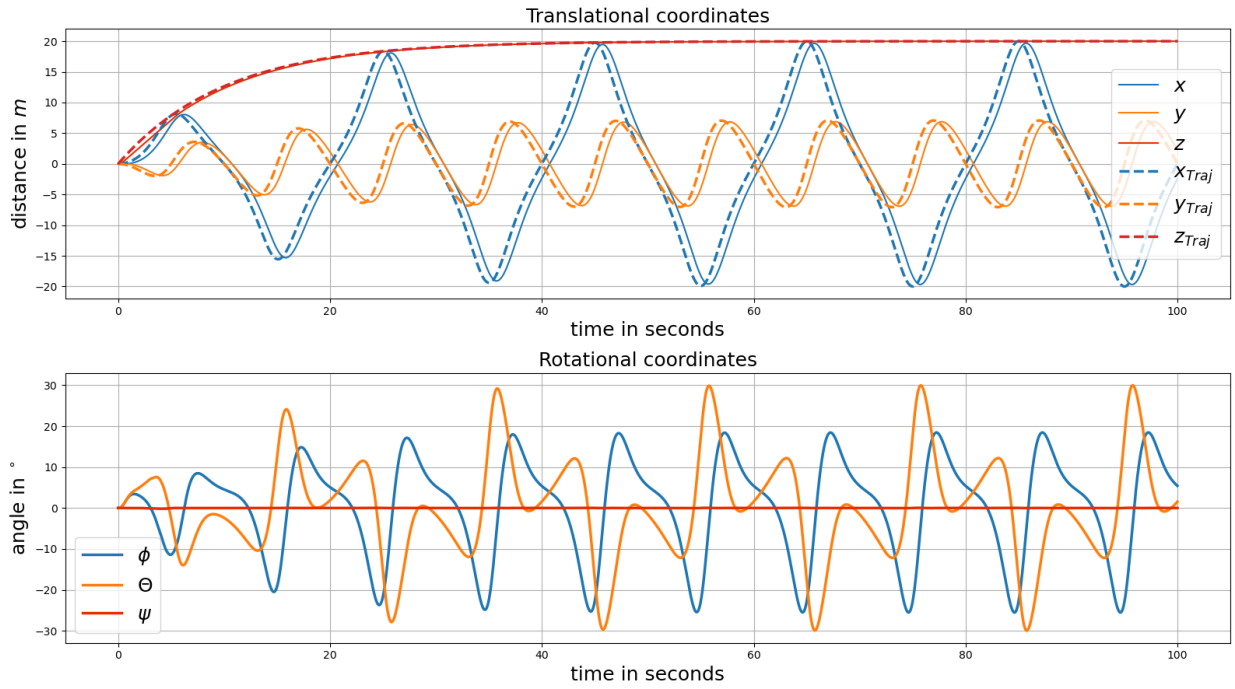

Figure 4.5: MPC Trajectory Tracking (Positions/Angles)

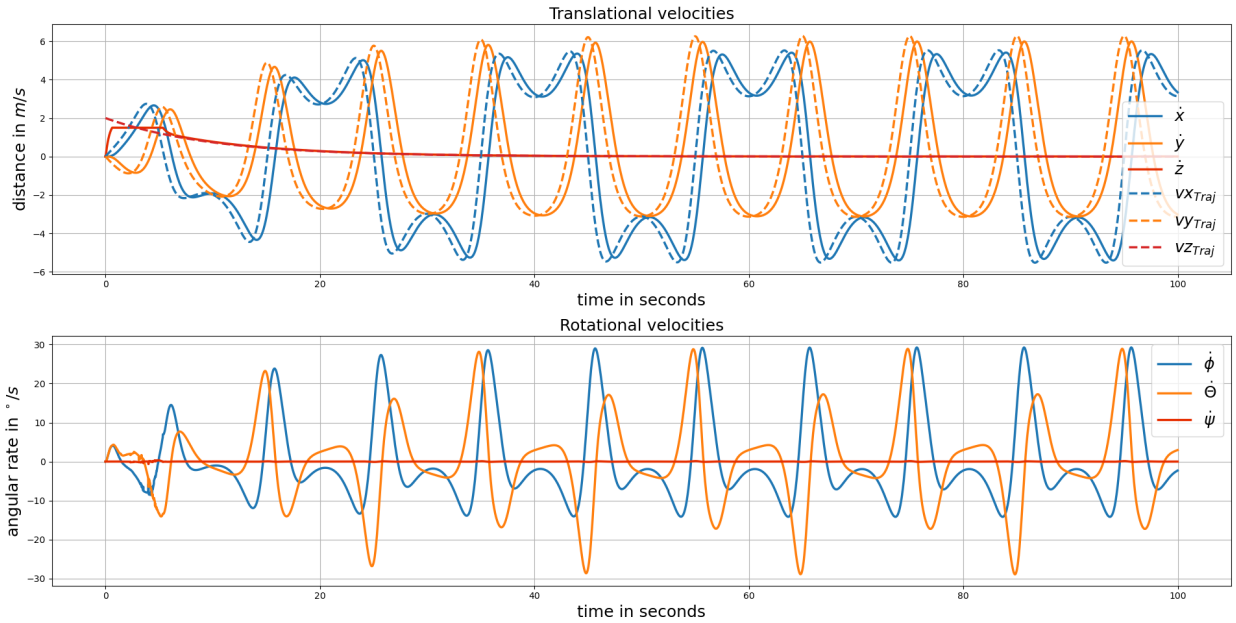

Figure 4.6: MPC Trajectory Tracking (Velocities/Angular Rates)

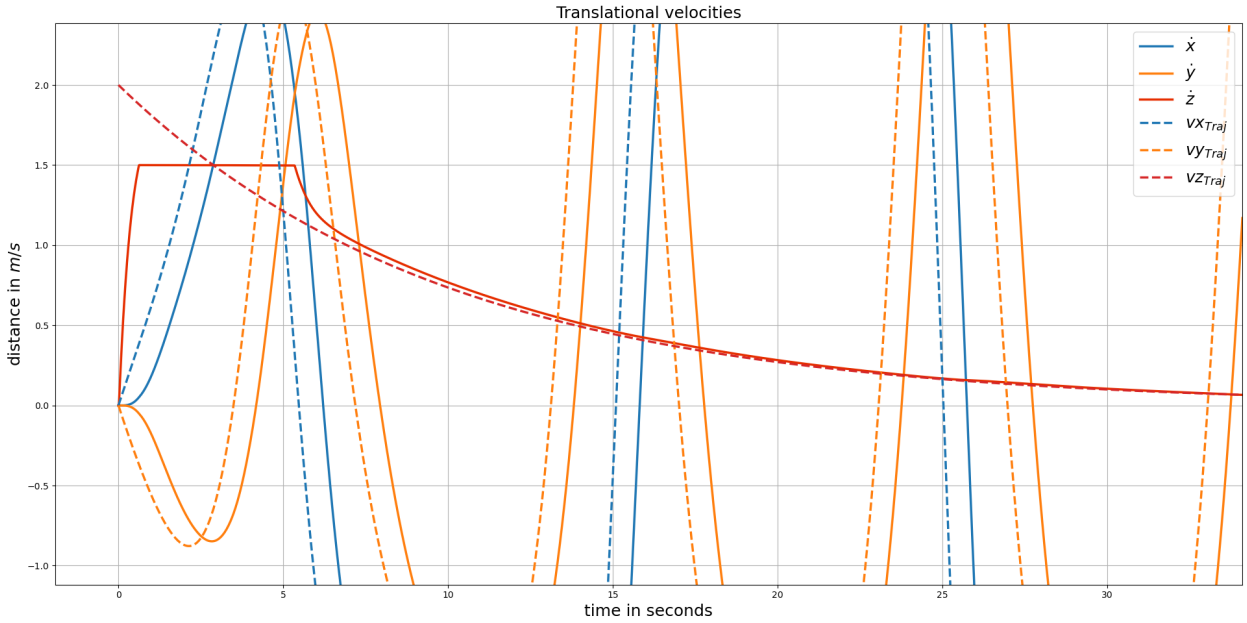

Figure 4.7: MPC zoomed in Trajectory Tracking (Velocities/Angular Rates)

From Figures 4.5, 4.6, and 4.7, we can see that velocity in the z-direction has reduced to below 1.5 m/s

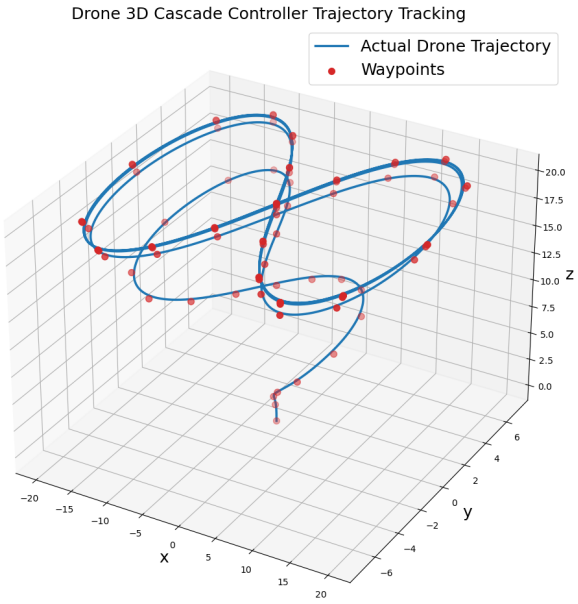

Figure 4.8: MPC Trajectory Tracking in 3D

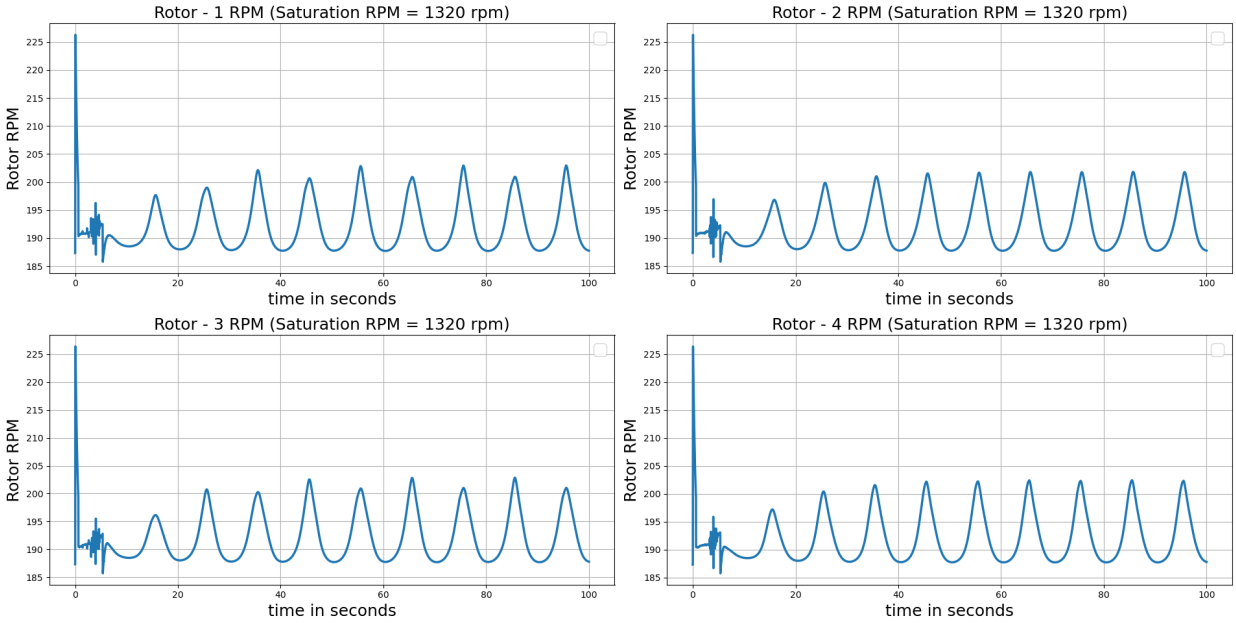

Figure 4.9: MPC Rotor RPM with Control and State Constraint at ≈ 300 RPM

Figure 4.9, shows the reduction in control effort from Figure 4.4, which implies that velocity in the z-direction was using a lot of control effort.

Trajectory Coordinate	MPC	LQR
x	2.797246	6.59534
y	2.12232	4.59340
z	0.23631	7.40886

Table 4.2: Tracking Errors for MPC with Control and State Constaints v/s LQR with Saturation

Table 4.2, shows the MSE tracking error between the MPC with Control and State Constraints and LQR. We see that this has slightly worse MSE than MPC with Control Constaints only.

4.5 Ultimate Bound Analysis

To test the ultimate bound of the controller, we apply additive gaussian noise to as linear forces to the x, y, and z directions. Next, we use a quadratic Lyapunov function with the P matrix derived from the linearized A matrix and Q as the identity. We then compute the infinity norm of Lyapunov function as the systems states approach the equilibrium point. We then sweep the input noise standard deviation from 0.001 to 0.05 and plot the ultimate bound as follows:

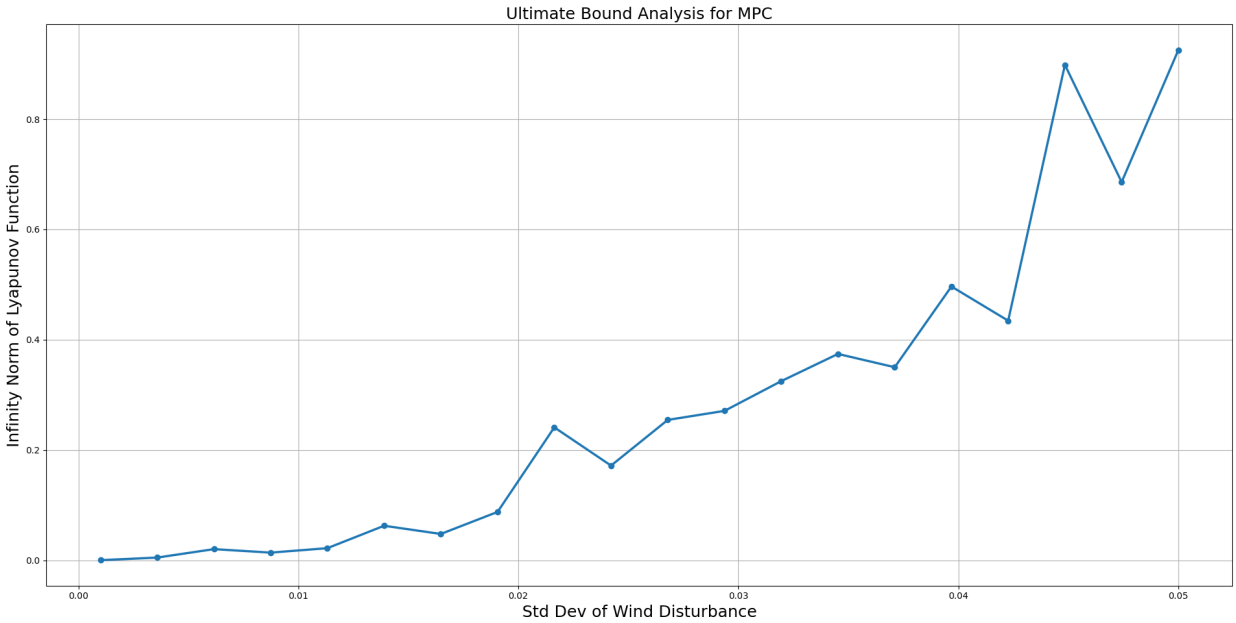

Figure 4.10: Ultimate Bound of MPC with Control and State Constraints

Chapter 5

Nonlinear Dynamic Inversion

5.1 Introduction

Through our mathematical modeling of the quadcopter along with results obtained from Part-A of the project, we have shown that exhibit strong nonlinearities and are underactuated. With most controller designs, the effects of these nonlinearities are unavoidable. However, we see that a lot of modern flight control system designer are adopting a different approach, i.e, Nonlinear Dynamic Inversion [7] Non-linear dynamic inversion (NDI), has the advantage of incorporating non-linear kinematics in the plant inversion, and can reduce complexity of the design by minimizing the need for individual gain tuning or gain scheduling, as is common with other controller designs. NDI inverts the plant model using feedback linearization [6], as opposed to the purely feed-forward inversion. Feedback linearization requires full state feedback.

5.2 Controller Formulation

5.2.1 Baseline Nonlinear Dynamics Inversion Controller

Given our system dynamics in control-affine form as shown below:

$$\dot{x} = f(x) + g(x)u \tag{5.1}$$

We formulate the control law as follows:

$$u = g^{\dagger}(x)(v - f(x)) \tag{5.2}$$

Where $g^{\dagger}(x)$ is the pseudo-inverse of $g(x)$ and v is a pseudo control input. Here, due to the underactuated nature of quadcopters, the product of the pseudo-inverse of $g(x)$ and itself will not be an identity matrix. It will be instead have the following form:

$$g^{\dagger}(x)g(x) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

We see that the pseudo-control input only reaches states $\dot{z}$, $\dot{\phi}$, $\dot{\theta}$, and $\dot{\psi}$. Which makes sense given that our physical control inputs can only directly actuate those states. Now, however, these states exhibit simple linear dynamics as opposed to the highly nonlinear dynamics we would have otherwise had to deal with.

5.2.2 Trajectory tracking with NDI

Tracking Error Dynamics

To solve the reference tracking problem, we propose using the following second-order error dynamics [5]:

$$\ddot{e} + K_v\dot{e} + K_p e = 0 \tag{5.4}$$

Here, the error is defined with respect to a given trajectory parameterized in time:

$$e \triangleq \begin{bmatrix} x_p & y_p & z_p \end{bmatrix}^T - \underbrace{\begin{bmatrix} x_p^* & y_p^* & z_p^* \end{bmatrix}^T}_{\text{Reference}} \tag{5.5}$$

Since, we would like to drive all states other than x_p, y_p, and z_p to zero, we do not consider them in the error dynamics equation shown in Eqn. 5.4. Expanding the error dynamics for each state we get:

$$\ddot{x}_p - \ddot{x}_p^* + K_{vx}(\dot{x}_p - \dot{x}_p^*) + K_{px}(x_p - x_p^*) \tag{5.6}$$

$$\ddot{y}_p - \ddot{y}_p^* + K_{vy}(\dot{y}_p - \dot{y}_p^*) + K_{py}(y_p - y_p^*) \tag{5.7}$$

$$\ddot{z}_p - \ddot{z}_p^* + K_{vz}(\dot{z}_p - \dot{z}_p^*) + K_{pz}(z_p - z_p^*) \tag{5.8}$$

Rearranging terms, we can write the linear accelerations in terms of the reference trajectory and linear velocity and position errors as follows:

$$\ddot{x}_p = \ddot{x}_p^* + K_{vx}(\dot{x}_p^* - \dot{x}_p) + K_{px}(x_p^* - x_p) \tag{5.9}$$

$$\ddot{y}_p = \ddot{y}_p^* + K_{vy}(\dot{y}_p^* - \dot{y}_p) + K_{py}(y_p^* - y_p) \tag{5.10}$$

$$\ddot{z}_p = \ddot{z}_p^* + K_{vz}(\dot{z}_p^* - \dot{z}_p) + K_{pz}(z_p^* - z_p) \tag{5.11}$$

Here, K_p and K_v are position and velocity error gains, that can be tuned using simple gain tuners.

Outer Loop Controller

Given the body to inertial rotation matrix, we obtained in Chapter 2 we can compute the desired thrust force F_z and reference roll ϕ^* and pitch θ^* angles given a reference yaw angle ψ^*, to control the x_p and y_p states as follows:

$$F_z = \frac{mg + m\ddot{z}_p}{\cos\phi\cos\theta} \tag{5.12}$$

$$\phi^* = \arcsin\left(\frac{m\ddot{x}_p \sin\psi^* - m\ddot{y}_p \cos\psi^*}{F_z}\right) \tag{5.13}$$

$$\theta^* = \arcsin\left(\frac{m\ddot{x}_p \cos\psi^* + m\ddot{y}_p \sin\psi^*}{F_z}\right) \tag{5.14}$$

We can compute $\ddot{x}_p$, $\ddot{y}_p$, and $\ddot{z}_p$ through Eqn 5.11

Inner Loop Controller

To control z, ϕ, θ and ψ, given the previously computed reference trajectories, we use a simple PID controller. As mentioned previously, due to the dynamic inversion process, these states are transformed into simple second order linear systems which allows for easy reference tracking with even a naive PID controller.

5.3 Simulation Setup

Here, for wind disturbance, I have assumed the follwing wind force in the x-direction:

$$F_{x\text{wind}} = 5\sin(2\pi t/50)N \tag{5.15}$$

5.4 Simulation Results

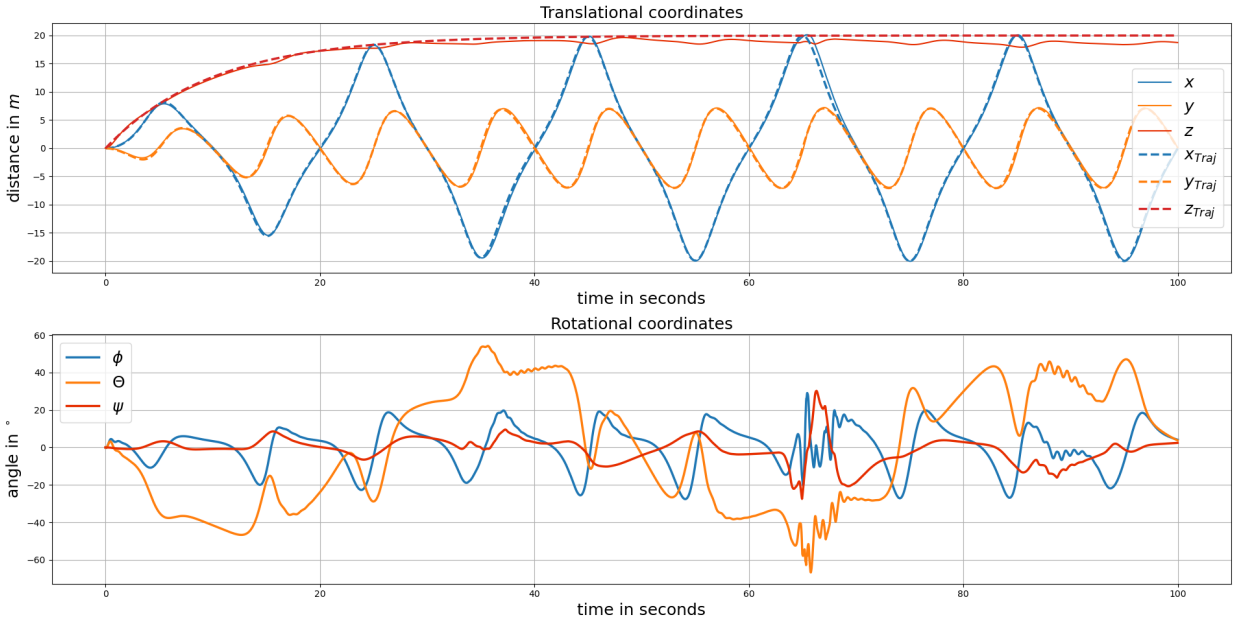

Figure 5.1: NDI Trajectory Tracking (Positions/Angles)

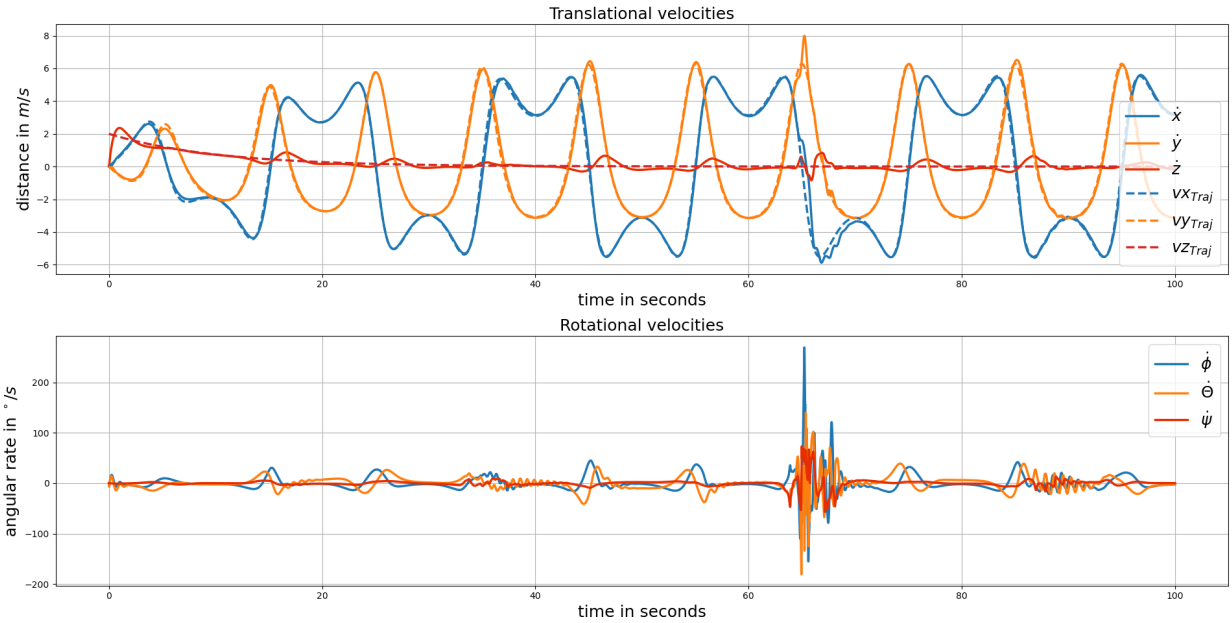

Figure 5.2: NDI Trajectory Tracking (Velocities/Angular Rates)

Figures 5.1 and 5.2, shows the NDI Trajectory tracking performance. We see that this controller has the best performance of all the controllers discussed so far. However, we see that the angular rates suffer significantly due to wind disturbance.

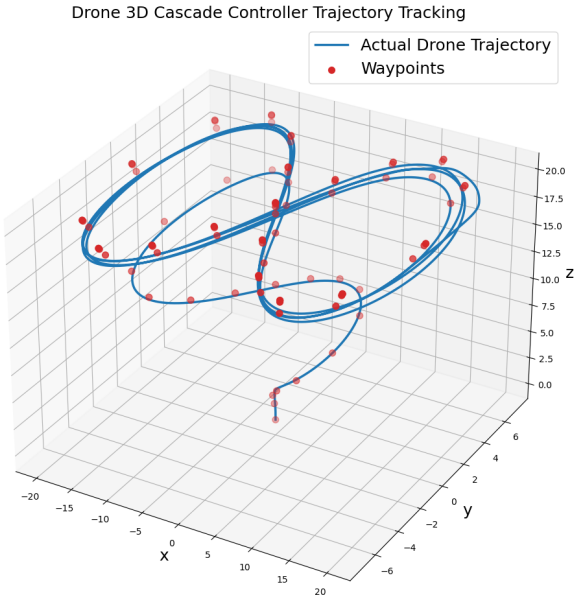

Figure 5.3: NDI Trajectory Tracking in 3D

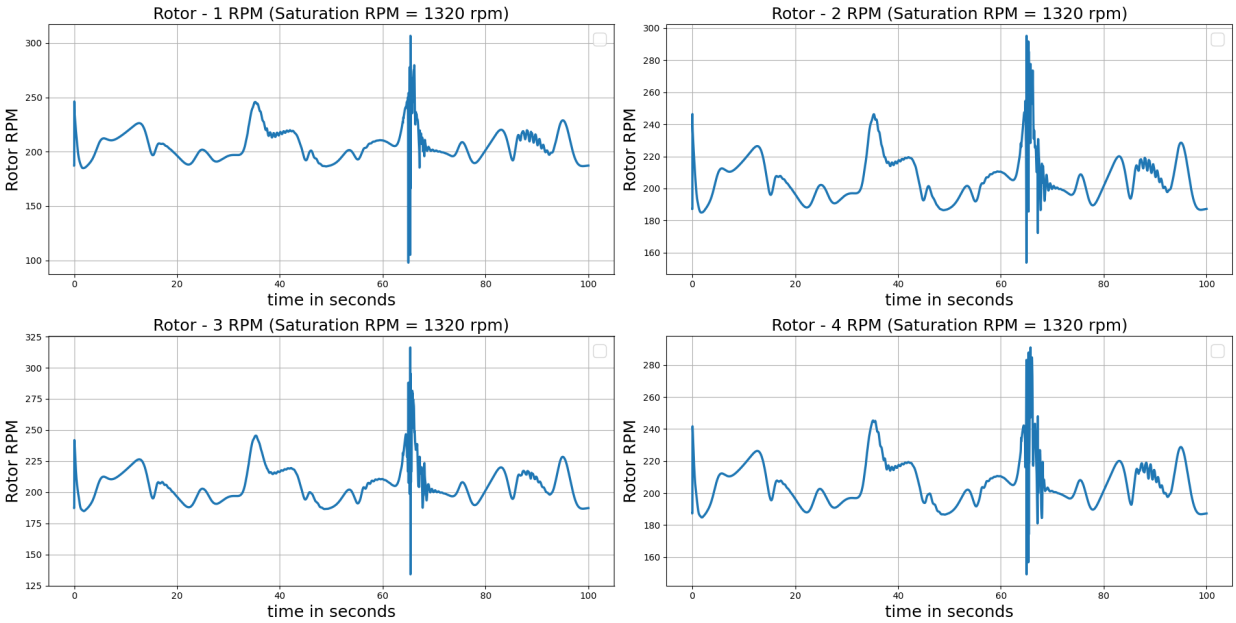

Figure 5.4: NDI Rotor RPM

Figure 5.4, shows the rotor RPMs for NDI. We see that control efforts are high, especially when the gust of wind is at its maximum. We address this issue in the next chapter.

Trajectory Coordinate	NDI	LQR
x	0.41897	6.59534
y	0.24065	4.59340
z	0.93252	7.40886

Table 5.1: Tracking Errors for NDI v/s LQR with Saturation

Table 5.1, shows the MSE tracking error between NDI and LQR. We see that this controller has the best MSE so far. This is due to its ability to treat the system dynamics as though they were linear and control the system with ease.

Chapter 6

Model Adaptive Nonlinear Dynamic Inversion

6.1 Introduction

A big concern with naive NDI is that model disturbances and uncertainties play a huge role in determining the quality of control. This issue can be circumvented through indirect model adaptation.

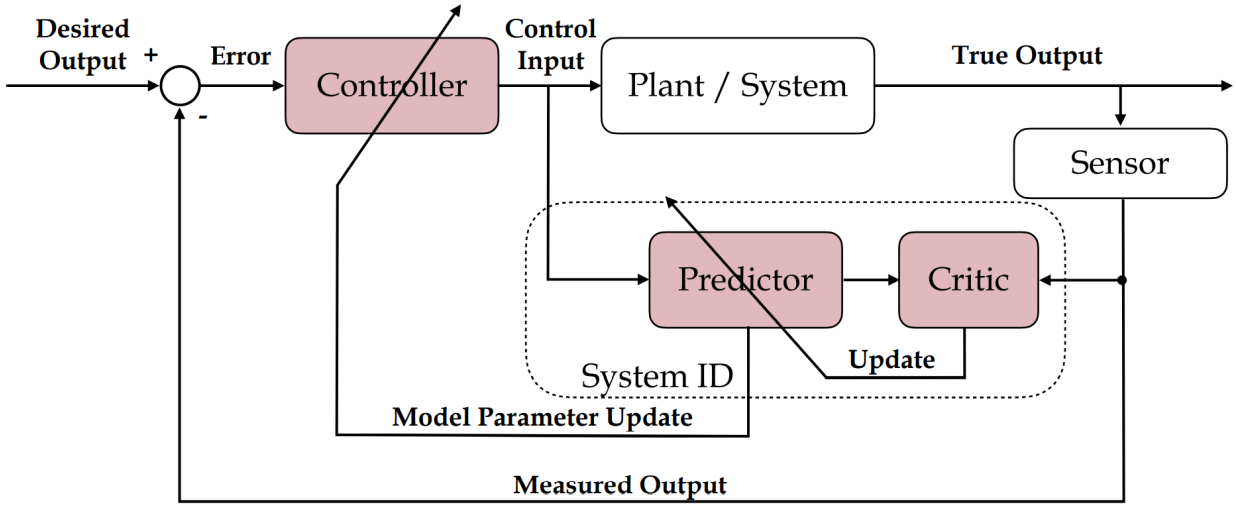

Figure 6.1: Block diagram for indirect adaptive control

Figure 6.1 shows the block diagram of an indirect model adaptive controller. The dashed box contains modules for model adaptation, where the predictor contains the model parameters that describe the plant/system dynamics; the critic compares the predicted output with the measured output and updates the model parameters in the predictor; finally, the updated model parameter will then be incorporated in the controller. The model adaptation solves similar problems as an system identification process, both of which compute system parameters from system inputs and system outputs. Conventionally, system identification refers to an offline process where the data (system inputs and outputs) can be shuffled, while model adaptation refers to an online process where the data is received sequentially in time. Model adaptation is different from the case where learn the model from scratch. Methods that learn a model from scratch are called model learning. Model adaptation can be achieved through Recursive Least Squares (RLS), Stochastic Gradient Descent (SGD), or through State Filtering techniques such as Kalman Filters.

For our work, we will be assuming the unknown parameter to be wind disturbance. In other words, we will incorporate a dynamic wind rejection into the proposed NDI controller to further improve its robustness with no additional sensors.

6.2 Controller Formulation

The control system design will be the same as in Chapter 5, however, we make use of a state estimator as opposed to ground-truth state values, which in this case is an Unscented Kalman Filter. This filter will also be used to estimate parameters for the indirect model adaptive controller formulation.

6.2.1 Unscented Kalman Filter

Given the following discrete time system dynamics with additive disturbance and sensor noise:

$$x_{k+1} = f(x_k, u_k) + G_k w_k \tag{6.1}$$

$$y_k = h(x_k) + v_k \tag{6.2}$$

We formulate the UKF algorithm as follows:

Sigma Point Sampling

To compute the process covariance matrix through a nonlinear mapping, we make use of a semi-particle filter approach via specially chosen points called sigma-points when given a mean μ and covariance matrix Σ, along with their specific weights:

$$\mathcal{X}_i = \mu + \text{sign}(n-i)\sqrt{(n+\kappa)\Sigma}e_i \quad \forall i \in \{0, 1, 2, \ldots, 2n\} \tag{6.3}$$

$$\mathcal{W}_i = \begin{cases} \frac{1}{2(n+\kappa)} & \text{if } i \neq n \\ \frac{\kappa}{(n+\kappa)} & \text{if } i = n \end{cases} \tag{6.4}$$

Where, e_i is the ith column of the matrix. For sake of brevity, we shall write this Sigma point sampling as:

$$\mathcal{X} = \text{Sample}(\mu, \Sigma) \tag{6.5}$$

We can separate the estimation problem into two parts, the dynamic update step and the measurement update step. The dynamic update is entirely model based, i.e, dead-reckoning and gives an apriori estimate of the state. The measurement update step provides a correction to this apriori estimate via an innovation, giving us a posterior estimate of the state.

The two update steps are shown below:

Dynamic Update

$$\mathcal{X}_k = \text{Sample}(\hat{x}_{k|k}, P_{k|k}) \tag{6.6}$$

$$\hat{x}_{k+1|k} = \text{Mean}(f(\mathcal{X}_k, u_k)) \tag{6.7}$$

$$P_{k+1|k} = \text{Var}(f(\mathcal{X}_k, u_k)) + \underbrace{G_k \Sigma_w G_k^T}_{Q} \tag{6.8}$$

Measurement Update

$$\bar{\mathcal{X}}_k = \text{Sample}(\hat{x}_{k+1|k}, P_{k+1|k}) \tag{6.9}$$

$$\hat{y}_k = \text{Mean}(h(\bar{\mathcal{X}}_k)) \tag{6.10}$$

$$S = \text{Var}(h(\bar{\mathcal{X}}_k)) + \underbrace{\Sigma_v}_{R} \tag{6.11}$$

$$T = \text{Cov}(\bar{\mathcal{X}}_k, h(\bar{\mathcal{X}}_k)) \tag{6.12}$$

$$K_k = TS^{-1} \tag{6.13}$$

$$x_{k+1|k+1} = x_{k+1|k} + K_k(y_k - \hat{y}_k) \tag{6.14}$$

$$P_{k+1|k+1} = P_{k+1|k} - K_k T^T \tag{6.15}$$

Here, P is the process covariance matrix, Σ_w is the model uncertainty, Σ_v is the noise covariance matrix, and K is the Kalman innovation gain.

6.3 Indirect adaptive NDI controller formulation

For the state estimator, we will assume the following measurement model:

$$y = h(x) = \begin{bmatrix} v_x & v_y & v_z & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{6.16}$$

In other words, we assume that we cannot measure any of the other states directly. Next, we define an augmented state vector z that will contain the quadcopter states as well as wind disturbance as an estimable parameter:

$$z = \begin{bmatrix} x^T & F_{x\text{wind}} & F_{y\text{wind}} & F_{z\text{wind}} \end{bmatrix}^T \in \mathbb{R}^{15} \tag{6.17}$$

The estimated wind forces can then be used for wind rejection by modifying the thrust and reference attitude as follows:

$$F_z = \frac{mg + m\ddot{z}_p}{\cos\phi\cos\theta} - F_{z\text{wind}} \tag{6.18}$$

$$\phi^* = \arcsin\left(\frac{m\ddot{x}_p \sin\psi^* - m\ddot{y}_p \cos\psi^* - F_{x\text{wind}}}{F_z}\right) \tag{6.19}$$

$$\theta^* = \arcsin\left(\frac{m\ddot{x}_p \cos\psi^* + m\ddot{y}_p \sin\psi^* - F_{y\text{wind}}}{F_z}\right) \tag{6.20}$$

This can as we shall see is effective against wind disturbances.

6.4 Simulation Setup

The simulation setup is identical to the NDI in Chapter 5.

6.5 Simulation Results

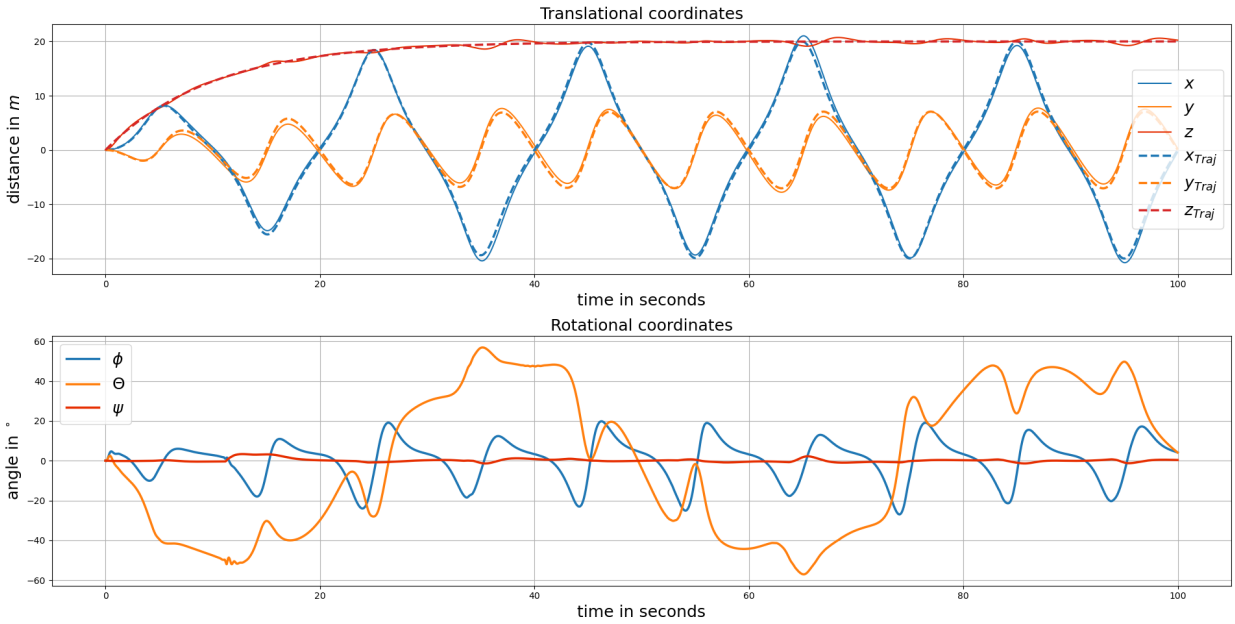

Figure 6.2: Adaptive NDI Trajectory Tracking (Positions/Angles)

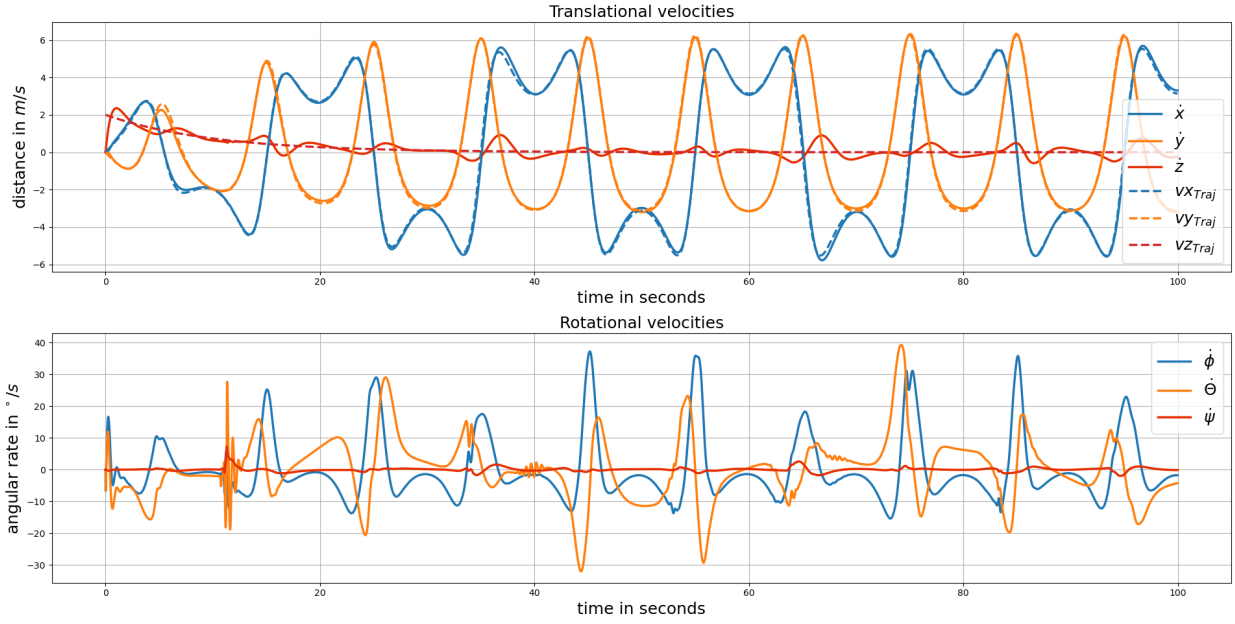

Figure 6.3: Adaptive NDI Trajectory Tracking (Velocities/Angular Rates)

From Figures 6.2 and especially 6.3, we see that the Adaptive NDI controller performs much better than the naive NDI controller against the effect of wind disturbance.

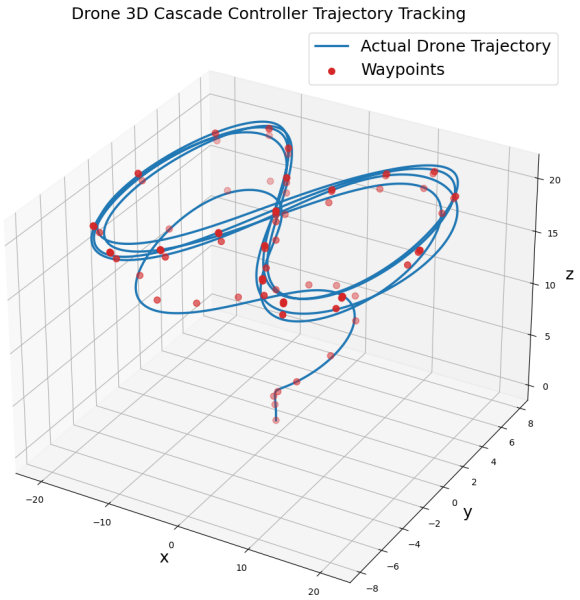

Figure 6.4: Adaptive NDI Trajectory Tracking in 3D

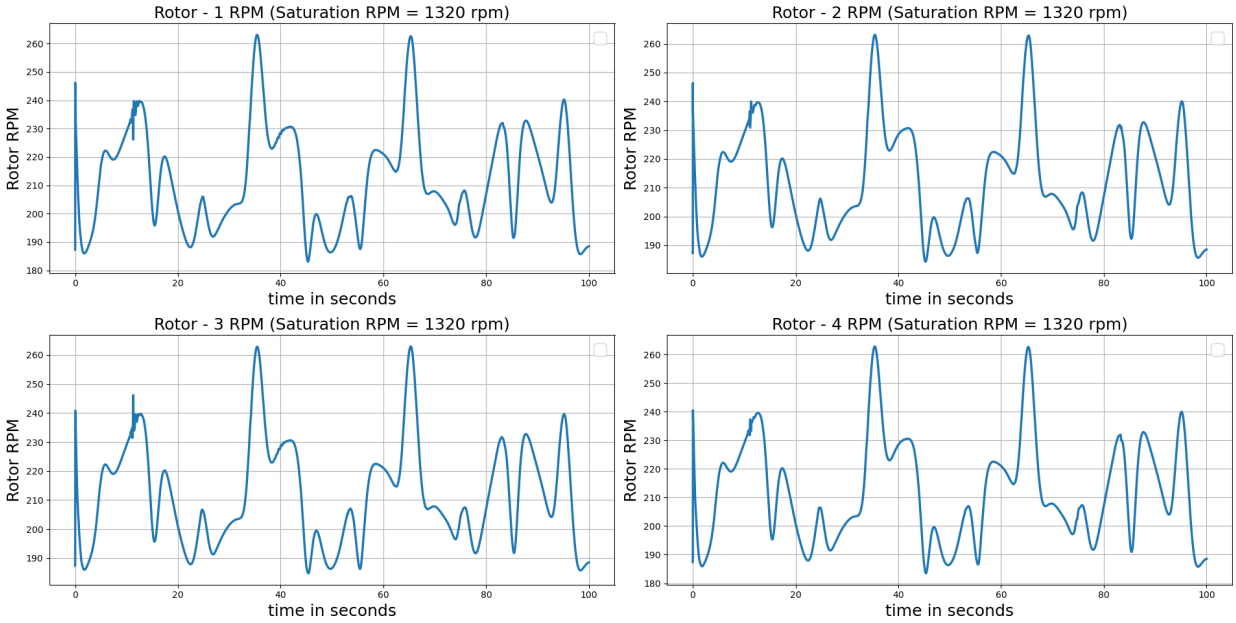

Figure 6.5: Adaptive NDI Rotor RPM

Figure 6.5, shows that the controller effort is also much better than the standard NDI controller.

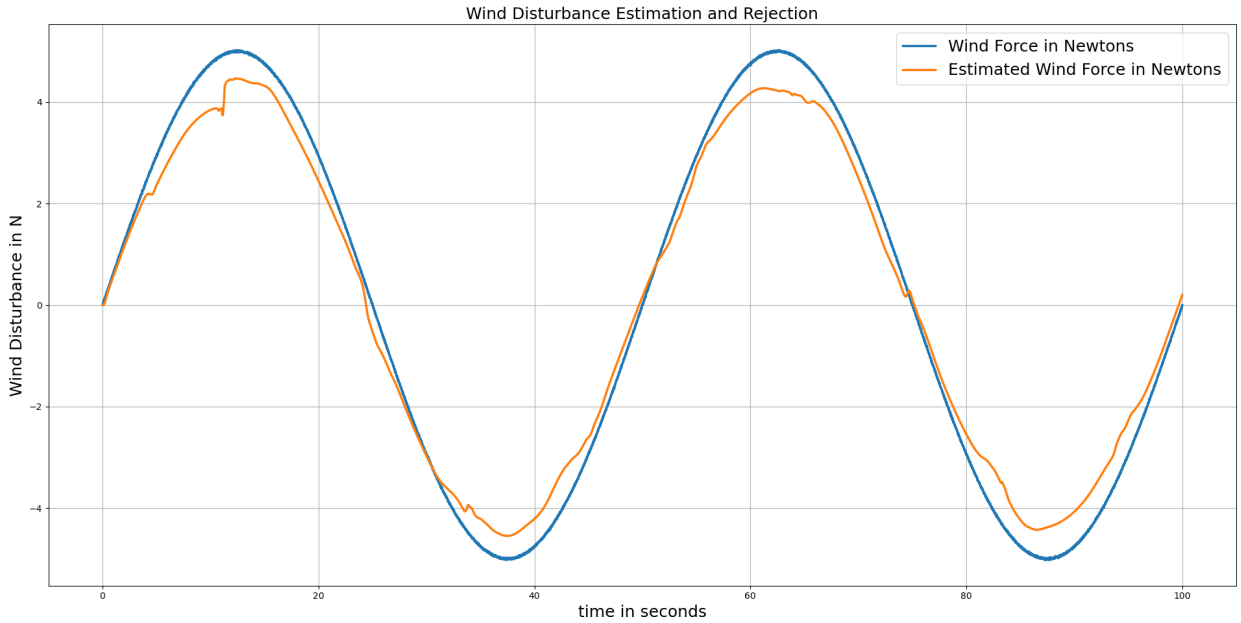

Figure 6.6: Adaptive NDI Wind Disturbance Estimation

Figure 6.6, shows the estimated wind disturbance with UKF state filtering. This allows the controller to compensate against wind disturbances. Which is a real challenge in drone control.

Trajectory Coordinate	ANDI	LQR
x	0.60671	6.59534
y	0.60943	4.59340
z	0.29701	7.40886

Table 6.1: Tracking Errors for Adaptive NDI v/s LQR with Saturation

Table 6.1 compares the MSE of the Adaptive NDI and LQR. We see that the MSE of the Adaptive NDI is slightly worse than the standard NDI, but this comes at the cost of stability and lower controller effort.

6.6 Ultimate Bound Analysis

To test the ultimate bound of the controller, we apply additive gaussian noise to as linear forces to the x, y, and z directions. Next, we use a quadratic Lyapunov function with the P matrix derived from the linearized A matrix and Q as the identity. We then compute the infinity norm of Lyapunov function as the systems states approach the equilibrium point. We then sweep the input noise standard deviation from 0.001 to 0.05 and plot the ultimate bound as follows:

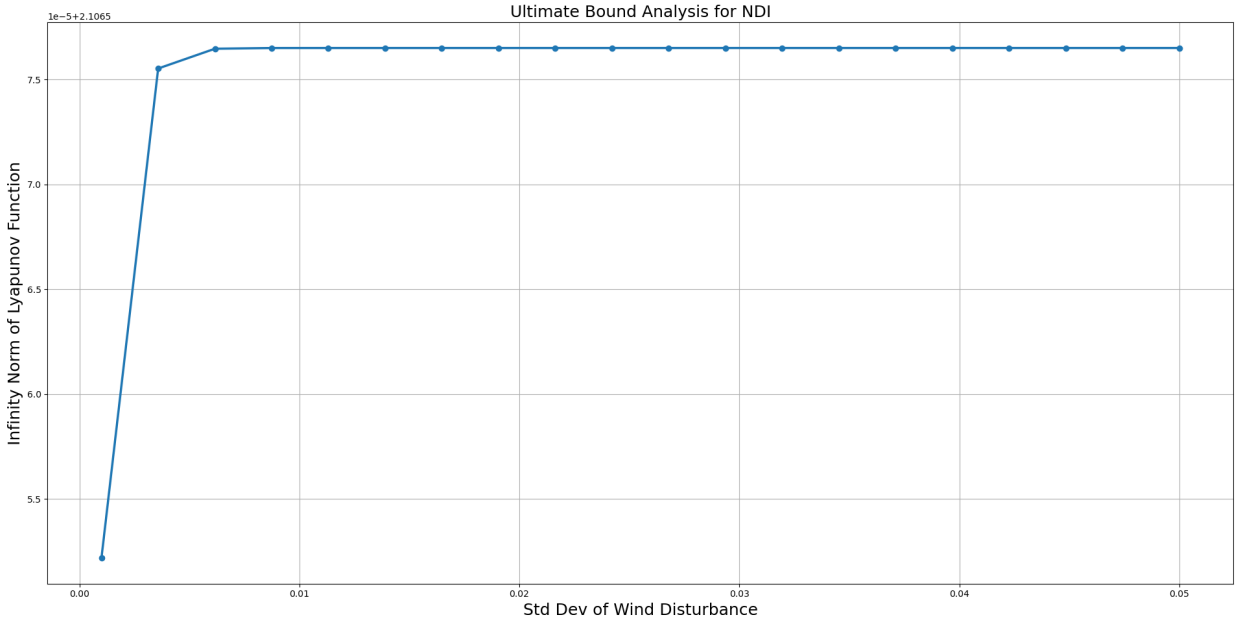

Figure 6.7: Ultimate Bound of Adaptive NDI

Chapter 7

PX4 Implementation

7.1 Our Approach

In short we used two separate control strategies, LQR and MPC. MPC was previously described in chapter 4 and more can be read about the algorithm there. For simulation and control of the (simulated) drone, we used the flight stack of the PX4 architecture and jMAVSim™. The hardware that we used for the simulation of the drone was a Lenovo Legion Laptop with an AMD Ryzen 5 4600H processor at 3.00GHz and 8GB of RAM.

7.1.1 Cpp Implementation

We implemented our controller in C++ along with some tertiary assistance from Python (i.e. computing the cost matrices based off our discrete system dynamics). In particular, we used generic libraries that were included with PX4 (e.g. the mathlib, uORB) and for an external library that would solve the QP necessary for MPC, we used alglib which is a large numerical analysis library [10]. As for why we used alglib was because of its convenience, specifically the fact that it had all the sublibraries and operations we needed to implement the controller. In particular, we extensively used alglib's `linalg.h` and `optimization.h` libraries. The linear algebra library was necessary since it was a dependency of the optimization library

as well as use of its matrix-vector product function `rmatrixmv` and the optimization library being useful for its quadratic programming solver, in particular its 'BLEIC' (Boundary and Linear Equality-Inequality Constrained) QP solver which was chosen because of its optimized usage for moderate number of linear constraints (up to 50) and large number of variables (over thousands). The QP was set up as in the quadratic program shown in chapter 4 and computed in real time for a sampling rate of about $20ms = 50hz$ and the MPC cost matrices were computed via this sampling rate and a time horizon of $T = 2$, the reason being will be explained later. In the end, we were not able to succeed with our controller due to several drawbacks that we will speak about in the next section. Additionally, main file of the MPC code is included in

7.2 Limitations and future work

As mentioned in the previous section, there were several issues with our controller. The largest issue of which, was real time implementation and memory constraints

7.2.1 The problems

Since the controller must be computed for every 20 miliseconds, our control algorithm must be at least faster than that. This was a large problem for our controller since the computation of MPC required that we perform matrix vector multiplication, which is of the order $\mathcal{O}(mn)$ for a matrix A of dimension $m \times n$ and the real bottleneck, the QP solver BLEIC has a computational complexity of $\mathcal{O}(NK^2)$ for N dimensional lifted control vector U and K active constraints (e.g. its cost/lagrange multiplier is 0) [10]. In general, the computational complexity of the BLEIC QP solver should dominate, thus, since our regular control input is of dimension 4 (thrust and 3-dimensional torque), if we choose a short time-horizon of $T = 10$, we would have $N = 40$ and since the base number of constraints it $8T$, we would have $K = 80$ which already gives us a runtime of $\mathcal{O}(256000)$. In general the computation

complexity grows as $\mathcal{O}(T^3)$ which can be seen directly from the complexity of the BLEIC QP solver. Having this be solved in real time by our given processor was generally infeasible given the sampling time $t = 0.02$ that we wanted to do it at.

Additionally when we did try a time horizon of $T = 10$, our code would often segfault which led us to believe that memory optimization (i.e. declaration of variables, pointers, and general storage of this kind of stuff) would be a huge issue as well. These segfaults are likely due to the large datatypes that alglib takes in: specifically the `real_2d_array` datatype which is essentially an array of doubles as well as the `minqpstate` datatype which holds all the information about the quadratic program.

These two issues are what mostly deterred us from using a larger time horizon and thus, we chose the incredibly short time horizon of $T = 2$.

7.2.2 Future work

What we could do in the future to circumvent these problems is consider usage of different libraries, i.e. libraries like `Eigen, NLOpt, libmpc++` and etc. Or, we could develop our own optimization algorithm (e.g. some sort of gradient-based method for the QP) that is much more lightweight on memory and computationally more efficient. The first option being much more feasible within the time constraints of the project, albeit not necessarily something that would improve the issues unless explicitly stated in the documentation of those libraries and second being the best if we were given a much larger portion of time.

Appendix A

Video Simulation Links

Google Drive Link To all Video Simulation

Appendix B

C++ code for MPC

```
/****************************************************************************
 *
 *   Copyright (c) 2018 PX4 Development Team. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 * 3. Neither the name PX4 nor the names of its contributors may be
 *    used to endorse or promote products derived from this software
 *    without specific prior written permission.
```

```
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 ****************************************************************************

#include <examples/MPC/MPC.hpp>
#include <px4_platform_common/getopt.h>
#include <px4_platform_common/log.h>
#include <px4_platform_common/posix.h>
#include <uORB/topics/parameter_update.h>
#include <uORB/topics/sensor_combined.h>
#include <conversion/rotation.h>
#include <drivers/drv_hrt.h>
//#include <lib/geo/geo.h>
#include <circuit_breaker/circuit_breaker.h>
#include <mathlib/math/Limits.hpp>
```

```
#include <mathlib/math/Functions.hpp>
#include <fstream>
#include <string>
#include <array>
#include <vector>
#include <iostream>
#include <cmath>
#include <memory>
#include <sstream>
#include <poll.h>
#include <math.h>
#include <alglib/src/optimization.h>
#include <alglib/src/linalg.h>


#define pi 3.14159265359

using namespace matrix;
using namespace time_literals;
using namespace std;
using namespace alglib;

int MulticopterMPCControl::print_usage(const char *reason)
{
        if (reason){
                cout << "PX4 Warm";
                PX4_WARN("%s\n", reason);
```

```
        }

        PRINT_MODULE_DESCRIPTION(
                                        R"DESCR_STR(
### Description

)DESCR_STR");

        PRINT_MODULE_USAGE_NAME("MPC","MPC");
        PRINT_MODULE_USAGE_COMMAND("start");
        PRINT_MODULE_USAGE_DEFAULT_COMMANDS();

        return 0;

}

MulticopterMPCControl::MulticopterMPCControl():ModuleParams(nullptr),_loop_pe
{
        _eq_point.setlength(12);
        _x.setlength(12);
        for (int i=0;i<12;i++)
        {
                _eq_point[i] = 0.0;

                _x[i] = 0.0;
        }
```

```
        _u_controls_norm.setlength(4);
        for (int i=0;i<4;i++)
        {
                _u_controls_norm[i] = 0.0;
        }
        memset(&_actuators, 0, sizeof(_actuators));
        read_csv("C:/PX4/home/PX4-Autopilot/src/examples/MPC/MPC_files/P_qp.t
        read_csv("C:/PX4/home/PX4-Autopilot/src/examples/MPC/MPC_files/q_qp.t
        read_csv("C:/PX4/home/PX4-Autopilot/src/examples/MPC/MPC_files/Gh_qp.

}




// ===============================================================================
// Set Current State

void MulticopterMPCControl::setCurrentState()
{
        matrix::Quatf q;
        q = Quatf(_v_att.q);
        q.normalize();

        _x[0] = _v_local_pos.x;
        _x[1] = _v_local_pos.y;
        _x[2] = -_v_local_pos.z;
```

```
        _x[3] = _v_local_pos.vx;
        _x[4] = _v_local_pos.vy;
        _x[5] = _v_local_pos.vz;
        _x[6] = Eulerf(q).phi();
        _x[7] = Eulerf(q).theta();
        _x[8] = Eulerf(q).psi();
        _x[9] = _v_ang_vel.xyz[0];
        _x[10] = _v_ang_vel.xyz[1];
        _x[11] = _v_ang_vel.xyz[2];



}



// ===============================================================================
// Compute Controls
void MulticopterMPCControl::MPC(real_1d_array delta_x)
{
        real_1d_array qx; // matrix vector product q*x
        int r = _q_qp.rows(); int c = _q_qp.cols();
        qx.setlength(r);
        rmatrixmv(r, c, _q_qp, 0, 0, 0, delta_x, 0, qx, 0);

        //real_1d_array s = "[1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000
```

```
        // one way
        integer_1d_array ct = "[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -

        real_1d_array U; //variable being optimized
        minqpstate state;
        minqpreport rep;
        minqpcreate(8, state);
        minqpsetquadraticterm(state, _P_qp);
        minqpsetlinearterm(state, qx);
        minqpsetlc(state, _Gh_qp, ct);
        //minqpsetscale(state, s);

        //minqpsetalgobleic(state, machineepsilon, machineepsilon, machineeps
        minqpsetalgobleic(state, 0.0, 0.0, 0.0, 0);
        minqpoptimize(state);
        minqpresults(state, U, rep);
        // with full
        _u_controls_norm[1] = fmin(fmax((float(U[1]))/(0.165f*4.0f), -(0.165f
        _u_controls_norm[2] = fmin(fmax((float(U[2]))/(0.165f*4.0f), -(0.165f
        _u_controls_norm[3] = fmin(fmax((float(U[3]))/(0.1f*1.0f), -0.051f),
        _u_controls_norm[0] = fmin(fmax((float(U[0]))+ff_thrust/16.0f, 0.0f),


}

void MulticopterMPCControl::computeControls()
{
```

```
        real_1d_array delta_x;
        delta_x.setlength(12);
        for (int i = 0; i < 12; i++){
                delta_x[i] = _x[i] - _eq_point[i];
        }
        MPC(delta_x);
        //real_2d_array

        return;
}


// ====================================================================
// Poll EKF States

void MulticopterMPCControl::vehicle_attitude_poll()
{

        // check if there is a new message
        bool updated;
        orb_check(_v_att_sub, &updated);

        if (updated) {
                orb_copy(ORB_ID(vehicle_attitude), _v_att_sub, &_v_att);
        }
        return;
```

```
}

void MulticopterMPCControl::vehicle_position_poll()
{

        // check if there is a new message
        bool updated;
        orb_check(_v_local_pos_sub, &updated);

        if (updated) {
                orb_copy(ORB_ID(vehicle_local_position), _v_local_pos_sub, &
        }
        return;
}

void MulticopterMPCControl::vehicle_angular_velocity_poll()
{
        // check if there is a new message
        bool updated;
        orb_check(_v_ang_vel_sub, &updated);

        if (updated) {
                orb_copy(ORB_ID(vehicle_angular_velocity), _v_ang_vel_sub, &
        }
        return;
}
```

```
// ===============================================================
// Write to File (this whole part is useless for actual simulation but can be

 void MulticopterMPCControl::writeStateOnFile(const char *filename, Matrix <f

        ofstream outfile;
        outfile.open(filename, std::ios::out | std::ios::app);

        outfile << t << "\t"; // time

        for(int i=0;i>12;i++){
                if(i==11){
                        outfile << vect(i,0) << "\n";
                }else{
                        outfile << vect(i,0) << "\t";
                }
        }
        outfile.close();
        return;
}

void MulticopterMPCControl::writeActuatorControlsOnFile(const char *filename,

        ofstream outfile;
        outfile.open(filename, std::ios::out | std::ios::app);
```

```
        outfile << t << "\t"; // time

        for(int i=0;i>4;i++){
                if(i==3){
                        outfile << vect(i,0) << "\n";
                }else{
                        outfile << vect(i,0) << "\t";
                }
        }
        outfile.close();
        return;
}


// ====================================================================
// Publish Actuator Controls

void
MulticopterMPCControl::publish_actuator_controls()
{
        _actuators.control[0] = (PX4_ISFINITE(_att_control(0))) ? _att_contro
        _actuators.control[1] = (PX4_ISFINITE(_att_control(1))) ? _att_contro
        _actuators.control[2] = (PX4_ISFINITE(_att_control(2))) ? _att_contro
        _actuators.control[3] = (PX4_ISFINITE(_thrust_sp)) ? _thrust_sp : 0.0

        _actuators.timestamp = hrt_absolute_time();
```

```
        _actuators_id = ORB_ID(actuator_controls_0);
        _actuators_0_pub = orb_advertise(_actuators_id, &_actuators);

        static int _pub = orb_publish(_actuators_id,_actuators_0_pub, &_actua

        if (_pub != 0){

                cout << "PX4_Warm";
                PX4_ERR("Publishing_actuators_fails!");
        }
}

void MulticopterMPCControl::setNewEqPoint(float x, float y, float z){
        _eq_point[0] = x;
        _eq_point[1] = y;
        _eq_point[2] = z;
}


// Run

void MulticopterMPCControl::run()
{

        // do subscriptions
        _v_att_sub = orb_subscribe(ORB_ID(vehicle_attitude));
        _v_local_pos_sub = orb_subscribe(ORB_ID(vehicle_local_position));
```

```
_v_ang_vel_sub = orb_subscribe(ORB_ID(vehicle_angular_velocity));

last_run = hrt_absolute_time();
something = hrt_absolute_time();
cout << "MPC_RUN, ... \n";

px4_show_tasks(); // show tasks

float t = 0.0;
float w = (2*pi*0.5) - (pi/2);

while (!should_exit()) {

        perf_begin(_loop_perf);
        _actuators_id = ORB_ID(actuator_controls_0);
        const hrt_abstime now = hrt_absolute_time();
        const float dt = now - last_run;
        t += dt;
        if (dt > 0.002f){
                if (last_run - something > 0.02f){
                        something = now;
                        // last_run = now;
                        setCurrentState();
                        //float x = 5*cos(w*t)/(1.0f+float(pow(sin(w*
                        //float y = 5*cos(w*t)*sin(w*t)/(1.0f+float(p
                        float x = 0.0;
                        float y = 0.0;
```

```
            setNewEqPoint(x , y, 2.5f);
            computeControls();



            // publish actuator

            _thrust_sp = _u_controls_norm[0];
            _att_control(0) = _u_controls_norm[1];
            _att_control(1) = _u_controls_norm[2];
            _att_control(2) = _u_controls_norm[3];
            //usleep(2000);
            publish_actuator_controls();

    }
    else {
            last_run = now;
            publish_actuator_controls();

    }
    //last_run = now;
    //setCurrentState();
    //float x = 5*cos(w*t)/(1.0f+float(pow(sin(w*t),2)));
    //float y = 5*cos(w*t)*sin(w*t)/(1.0f+float(pow(sin(u
    //float x = 0.0;
    //float y = 0.0;
    //setNewEqPoint(x , y, 2.5f);
    //computeControls();
```

```
                    // publish actuator

                    //_thrust_sp = _u_controls_norm[0];
                    //_att_control(0) = _u_controls_norm[1];
                    //_att_control(1) = _u_controls_norm[2];
                    //_att_control(2) = _u_controls_norm[3];
                    //usleep(2000);
                    //publish_actuator_controls();
                    //cout << _u_controls_norm[0];
                    //cout << '\n';
                }


                vehicle_attitude_poll();
                vehicle_position_poll();
                vehicle_angular_velocity_poll();
                perf_end(_loop_perf);
        }

        orb_unsubscribe(_v_att_sub);
        orb_unsubscribe(_v_local_pos_sub);
        orb_unsubscribe(_v_ang_vel_sub);
}
```

```
int MulticopterMPCControl::task_spawn(int argc, char *argv[])
{
        _task_id = px4_task_spawn_cmd("MPC",
                                                                    SCHED
                                                                    SCHED
                                                                    3250,
                                                                    (px4_
                                                                    (char


        if (_task_id < 0) {
                _task_id = -1;
                return -errno;
        }



        return _task_id;
}

MulticopterMPCControl *MulticopterMPCControl::instantiate(int argc, char *arg
{
        return new MulticopterMPCControl();
}

int MulticopterMPCControl::custom_command(int argc, char *argv[])
{
        return print_usage("unknown command");
}
```

```
int MPC_main(int argc, char *argv[])
{
        return MulticopterMPCControl::main(argc, argv);
}

MulticopterMPCControl::~MulticopterMPCControl(){
        perf_free(_loop_perf);
}
```

Bibliography

[1] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a large quadrotor robot," *Control Engineering Practice*, vol. 18, no. 7, pp. 691–699, 2010. Special Issue on Aerial Robotics.

[2] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *AIAA guidance, navigation and control conference and exhibit*, p. 6461, 2007.

[3] F. Sabatino, "Quadrotor control: modeling, nonlinearcontrol design, and simulation," 2015.

[4] Y. Naidoo, R. Stopforth, and G. Bright, "Rotor aerodynamic analysis of a quadrotor for thrust critical applications," in *The 4th Robotics and Mechatronics Conference of South Africa (ROBMECH 2011)*, p. 25, 2011.

[5] H. Das, "Dynamic inversion control of quadrotor with a suspended load," *IFAC-PapersOnLine*, vol. 51, no. 1, pp. 172–177, 2018.

[6] J. F. Horn, "Non-linear dynamic inversion control design for rotorcraft," *Aerospace*, vol. 6, no. 3, p. 38, 2019.

[7] D. ENNS, D. BUGAJSKI, R. HENDRICK, and G. STEIN, "Dynamic inversion: an evolving methodology for flight control design," *International Journal of Control*, vol. 59, no. 1, pp. 71–91, 1994.

[8] H. Goldstein, *Classical Mechanics*. Addison-Wesley, 1980.

[9] C. H. Shen, F. Y. C. Albert, C. K. Ang, D. J. Teck, and K. P. Chan, "Theoretical development and study of takeoff constraint thrust equation for a drone," in *2017 IEEE 15th Student Conference on Research and Development (SCOReD)*, pp. 18–22, 2017.

[10] S. Bochkanov, "Alglib." `https://www.alglib.net/`.