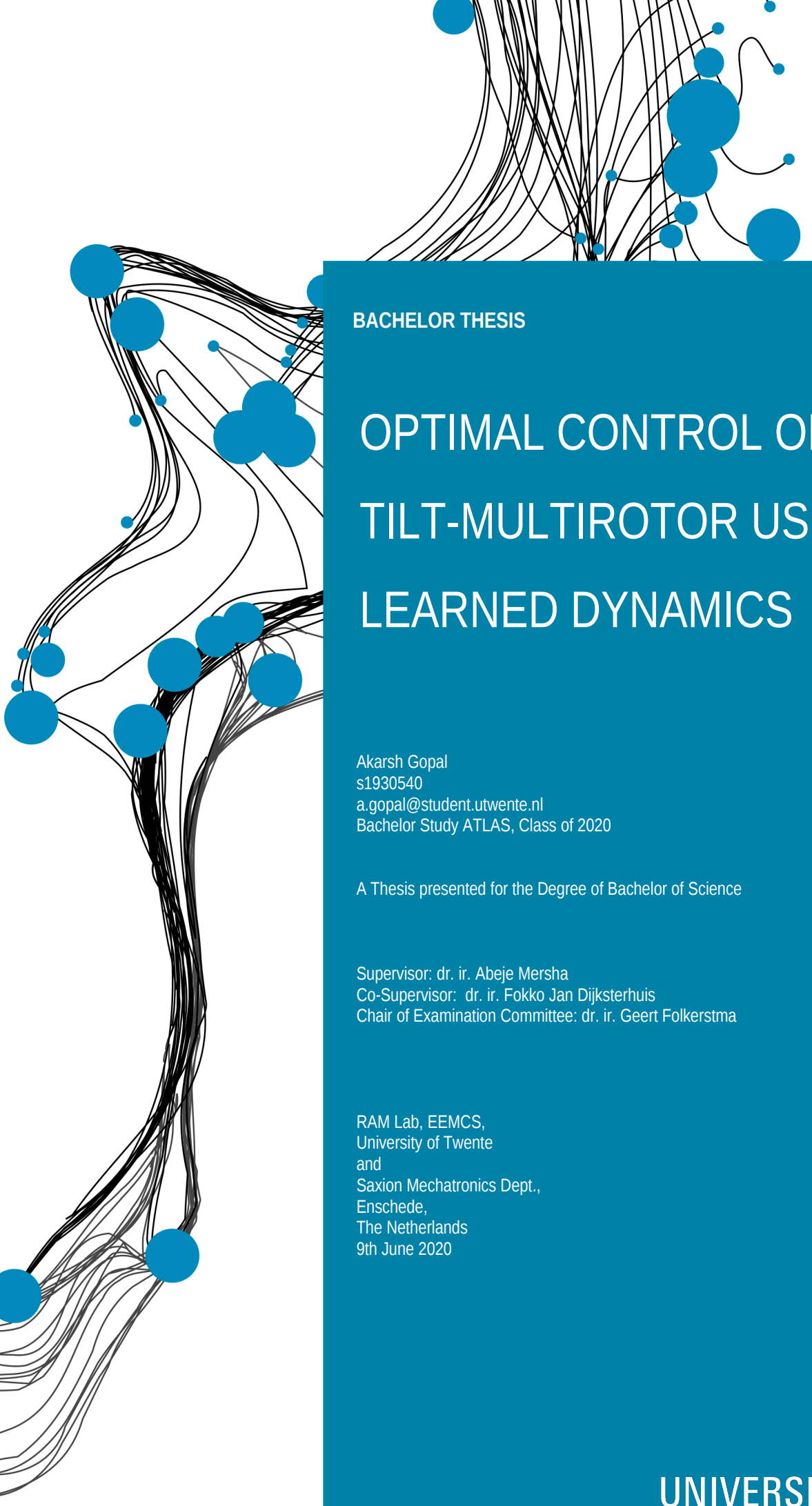


BACHELOR THESIS

OPTIMAL CONTROL OF TILT-MULTIROTOR USING LEARNED DYNAMICS

Akarsh Gopal
s1930540
a.gopal@student.utwente.nl
Bachelor Study ATLAS, Class of 2020

A Thesis presented for the Degree of Bachelor of Science

Supervisor: dr. ir. Abeje Mersha
Co-Supervisor: dr. ir. Fokko Jan Dijksterhuis
Chair of Examination Committee: dr. ir. Geert Folkerstma

RAM Lab, EEMCS,
University of Twente
and
Saxion Mechatronics Dept.,
Enschede,
The Netherlands
9th June 2020

UNIVERSITY OF TWENTE.

Abstract

Drones, particularly multirotor UAVs, are flying robots that have great potential in practical applications. However, using them in outdoor environments presents a challenge. Tilt-multirotor UAVs, where the forces and torques can be vectored have an advantage in maneuverability and maintaining their pose. Controlling them is more complicated, and accounting for all relevant dynamics is extremely difficult. Data-driven control approaches can alleviate these difficulties, by using machine learning for 'learning' the unknown dynamics. Particularly, NNs can be used in conjunction with MPC for this purpose. This research project investigates the development of a NN-based MPC for a quadrotor UAV with tilting propellers, in computer simulation using MATLAB.

The project consists of three main parts: modeling and simulation, controller development, and development of the NN. Once the nominal MPC is developed, it was run on 9 different reference trajectories, with 7 different disturbances for each, to collect training data for the NNs. Each disturbance had a NN trained for it. These NNs were then separately integrated into MPC to form an LBMPC for each disturbance, which were tested on a test trajectory to investigate the effects on performance. Additionally, the LBMPC with NNs were tested on one of the training trajectories to assess how well the LBMPC can perform on a previously trained trajectory.

Results indicate that for well-trained NNs and familiar trajectories the approach can significantly improve controller performance. However, they also indicate that for trajectories where the NN cannot accurately predict the error, the benefits of the approach are marginal and could also lead to failure if there is sufficient noise in the inputs to the NN. Additionally, several insights were gained through this research that will prove useful in implementing this approach on a physical system. There are also many avenues for further research identified based on these insights.

Contents

List of acronyms

MPC	Model Predictive Control
LBMPC	Learning Based Model Predictive Control
LQR	Linear Quadratic Regulator
NN	Neural Network
GP	Gaussian Process
ISO	International Standards Organisation
PID	Proportional Integral Derivative
SINDYc	Sparse Identification of Nonlinear Dynamics with control
UAV	Unmanned Aerial Vehicle
LTV	Linear Time Varying
DOF	Degree of Freedom
ReLU	Rectified Linear Units
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
IMU	Inertial Measurement Unit

Chapter 1

Introduction

UAVs, popularly called drones, are aerial robots that can carry sensors, manipulators, or packages for delivery. Drones open up several possibilities for applications that are being actively pursued in industry. However, they can be hard to control, especially in outdoor environments under the influence of wind and other disturbances. The particular design configuration of the drones can provide different possibilities for control methods, some easier and some harder.

This research project looks into the development of a hybrid of a data-based and analytical model-based controller for a UAV with tilting propellers. The project consists of three main parts: Modeling and Simulation, Controller Development, and Development of the Learning Module. The three are combined and tested in simulation to provide insight regarding the possible benefits of such an approach for a practical controller.

This chapter first provides the context of the project, followed by the framework under which it is conducted. Next, the goal of the project is stated, followed by the research questions to be answered. An overview of the report's structure is provided. Then, related works in the field are listed, providing further motivation for the approach used in this project. Finally, this project is positioned in contrast to the related works, and the reasoning behind the project's approach is provided.

1.1 Context

Drones are currently being deployed for inspection in various industries such as civil, petrochemical, off-shore, and mining [1] [2]. They enable observation of remotely located structures without the need to reach them in person. Many of these remote sensing applications are operational and economically successful. Human investigators would need to physically access the locations of interest making the inspections costly and potentially dangerous. Drones can also be used for applica-

tions that require interaction with structures that are hard to reach in person [3] [4]. However, interaction is more challenging technically since the drone is required to maintain a certain pose and exert a certain force or torque.

This project helps make progress towards better control and autonomy of UAVs, specifically tilt-multirotors, which are multirotors with tilting propellers. Autonomous operation of UAVs would enable various functionalities. They can be used to perform unmanned inspection and maintenance in hard-to-reach places in situations dangerous to humans, quick transport of goods such as vaccines, blood packages and diagnostic samples. All of these applications can help mitigate hazards and save lives. However, like most technologies, UAVs could also be used for nefarious purposes, such as delivery of contraband, invasion of privacy, and even violent crimes. It is important to consider potential misuse of the technology and develop safety and regulatory policies to minimise the possibility of such misuses.

Conventional drones are multirotors with a fixed orientation of thrust devices like propellers. Drones with tilting propellers called tilt-multirotors can be designed to be fully actuated as compared to the under-actuated conventional multi-rotors. This enables tilt-multirotors to follow arbitrary trajectories in state-space, which may allow them to perform more effectively under external disturbances than conventional multi-rotors. This allows the tilt-multirotors UAVs to also exert forces and torques onto external objects. However, due to the highly non-linear dynamics of these tilt-multirotor UAVs and several higher-order effects such as airflow interactions, it can be difficult to model their dynamics. Ensuring that they are controlled so as to minimize a certain cost function as done in Optimal control, requires a mapping of the system's dynamics and constraints to the cost function. Therefore, in order to solve the optimal control problem, the system's dynamics must be estimated well. Additionally, developing a cost function is not trivial. It is difficult to map the state of the system and the input effort to an objective such as performance and energy consumption.

As systems get more complicated in design, their modeling and control also become difficult. System identification is the process of determining the dynamics model of the system and Parameter estimation is the process of determining the parameters of the dynamics model of the system. Practically, these two processes are prone to inaccuracy for complex systems or can be capital- and time-intensive to perform. Additionally, disturbances and higher-order effects are difficult to model. Inaccuracies in system identification and parameter estimation can lead to plant-model mismatches which lead to loss of performance in model-based controllers and instability of the system in the worst case.

Modern control engineering involves the use of data-driven approaches to develop control systems. Machine learning methods have been used for system iden-

(a) (b)

Figure 1.1: (a) CAD model of the Saxion Tilt-rotor UAV. Courtesy: Saxion Mechatronica. Note that this is the quadrotor version. For this project, a quadrotor version of the system in the figure is used. (b)Tilting quadrotor UAV [5], similar to the configuration used in this project

tification, parameter estimation, and even controller design. There has been tremendous interest recently, in using machine learning approaches for control system design in complicated systems. Related works have used genetic algorithms, NNs, and Gaussian Process (GP) regression to estimate the dynamics models of systems like cars, robot arms, quadrotors, and autonomous helicopters. This project uses self-supervised learning to estimate the system dynamics of tilt-multirotors and uses optimal control approaches to develop a control system for a tilt-multirotor. Self-supervised learning is machine learning using data that is collected by the system itself, without explicit human effort as is done is supervised learning. This helps develop knowledge on how beneficial self-supervised learning is in estimating or 'learning' the dynamics of tilt-multirotors.

1.2 Framework

This project was conducted in association with Saxion Hogeschool Mechatronics under the MARS4Earth project. Saxion's tilt-multirotor platform (see figure1.1 a) was used as the reference for the system to be modeled and controlled. Particularly, for this project, this multirotor was considered in its **quadrotor configuration**. The motor-propeller sub-assemblies tilt independently (see figure 1.1 b), about the arm that supports them.

The initial goal was to conduct experiments on the physical system. However, due to physical restrictions during the project, the scope of the project was limited to numerical simulation.

1.3 Research Goal and Questions

Goal

To evaluate the benefits of using self-supervised learning to learn the dynamics model of a tilt-multirotor UAV and using the learned model to develop an optimal controller that can reject external disturbances.

Research Questions

- How can self-supervised learning be used to learn the model dynamics of the tilt-multirotor UAV?
- How beneficial would this learned model be in the synthesis of an optimal controller so as to reject external disturbance optimally?

1.4 Report organization

The the rest of the report is organised into the following chapters:

- Theoretical Background: Revises the theory behind the controller and learning module
- Methodology: Outlines the three main parts of the research and the corresponding work carried out.
  - Modelling and Simulation: The mathematical modeling of the system and the numerical simulation of the system are described.
  - Control: The development of the LBMPC controller is described.
  - Learning Module: The development of the NN learning module is described.
  - Simulation Test: The simulation test procedure is described.
- Results: The results of the simulation tests are presented, and observations are made on these results.
- Conclusions and Recommendations: The usefulness of the hybrid control approach is evaluated. The limitations of the research project are stated. Finally, in light of the results, recommendations for further research are made.
- Appendices

1.5 Related Works

There has been significant interest in the design and development of omnidirectional or over-actuated UAVs in the field recently: [6] and [7] are leading exam-

ples of such initiatives and present the challenges and promise of over-actuated UAVs. [8] presents a novel tilt-quadrotor design, define a mathematical model, and develop a controller for the same, using control allocation matrices, and hierarchical Proportional Integral Derivative (PID) control. The above works establish the superior maneuverability and actuation abilities of the tilt-multirotor UAVs. In the literature, optimal control is usually carried out at the trajectory planning level by having hierarchical control such as Linear Quadratic Regulator (LQR) for trajectory planning and PID control for lower-level controls [7]. While [7] develop optimal control for such a system, none of the above approaches look into higher-order dynamics of the systems and their performance under disturbances. This is mainly because these higher order effects are difficult to model and do not significantly affect the performance of the system under the operating conditions considered for these works. However, for UAVs operating in external environments it would be interesting to take these into consideration.

In studies on conventional quadrotors, optimal control is carried out in closed-loop using an MPC formulation of the controller [9]. This formulation is much more model-dependent and allows higher fidelity of control given its ability to predict the evolution of the system's dynamics. However, MPC is sensitive to the accuracy of the model and therefore degrades in performance when there is a mismatch between the plant and the model.

In [10], the authors prove mathematically that MPC using a combination of a nominal model and an oracle (learning module) for prediction, is safe. This approach is termed LBMPC. In and [11], the authors use LBMPC to offset the ground effect in quadrotors. In [12], LBMPC is used on a quadrotor using GP regression as the oracle. However, the main disadvantage of GP regression is the growth in inference time as the number of past data points being used. Additionally, in [12], the authors use a separate GP for each state variable, neglecting the relationships across the various state variables.

In [13], the authors use Sparse Identification of Nonlinear Dynamics with control (SINDYc) with MPC, where a dictionary of nonlinear terms is combined using sparse regression to effectively get the simplest non-linear function that describes the dynamics. The advantage of this approach is that the learning module can be trained roughly 100 times faster than NNs. The major disadvantage is that a dictionary of nonlinear terms needs to be known a priori. Since the main problem being addressed in this project is the learning of unknown dynamics, which means a prior dictionary of terms cannot be determined exhaustively, SINDYc is not used as the learning approach for this work.

In [14], the authors estimate the reward function for autonomously following acrobatic trajectories defined by model RC helicopter pilots. They also fit a dynamics

model to approximate the complex dynamics of the helicopter in such trajectories. They then use iterative LQR control in MPC formulation to perform acrobatic maneuvers on the RC helicopter autonomously. In [15], the authors implemented Model-Based Reinforcement Learning on a quadrotor by using MPC on a NN prediction of dynamics at the direct motor level. They do this by randomly sampling input sequences, using the NN to predict the evolution of the dynamics and selecting the sequence that minimizes the cost function for the optimal control problem. While these approaches are interesting, the former requires extensive and explicit human guidance and the latter requires a large number of trials and does not take advantage of existing models.

In [16], the helicopter dynamics are approximated by NNs and shown to perform well in approximating the dynamics. The authors specifically use Rectified Linear Units (ReLU) activation nodes in the NN and predict the state's time derivative given a history of state and input vectors. Similarly, in [17], the authors use two separate ReLU NN to predict the linear and rotational acceleration of the UAV respectively, using only the current state and control vectors as input. While the authors are successful in learning the dynamics of the UAV from scratch, they suggest it would be interesting to inspect the benefits of a hybrid approach: a combination of first-principles derived dynamics model enhanced by a data-driven neural network model. They postulate that this would be more practical and might yield similar if not better benefits. The above two works are the primary motivation for selecting ReLU NN for the learning module in this project since they are shown to be capable of fitting significantly nonlinear functions, particularly that of quadrotors.

1.6 This Work

This work focuses on the optimal control of a tilt-quadrotor system in the presence of external disturbances using learned dynamics. The approach used in this work is LBMPC, using a ReLU NN as the oracle, and training it in a self-supervised fashion.

The plant-model mismatch is created artificially by defining the plant with more fidelity than the controller's model. Therefore, the first step is to model and simulate the plant. Additionally, different disturbance forces are considered for testing the usefulness of the NNs in presence of unmodelled disturbances.A nominal MPC controller is implemented for the tilt-quadrotor system, and used to collect data for the training of the NN oracle. After integrating the NN oracle, (referred to as Learning Module) into the MPC controller, the complete LBMPC controller is simulated and the usefulness of the NN oracle is evaluated. Finally, recommendations for future work are made, and the steps required to transfer the approach to a physical system are discussed.

Chapter 2

Theoretical Background

This chapter provides the theoretical background for the concepts used in the project. First, optimal control and MPC are briefly explained. Next, the theory behind NNs is explained.

2.1 Optimal Control

In Optimal Control, the control problem is formulated as an optimization problem, specifically the minimization of a cost function. This cost function can contain terms that represent tracking error, input effort, and encode how the designer of the controller values the respective terms.

The discrete version of the optimal control problem, as used in this project, is as follows:

$$\mathrm{J}(\mathbf{x}_0, \mathbf{u}) = \sum_{t=0}^{T} L(\mathbf{x_t}, \mathbf{u_t}) \tag{2.1}$$

$$\begin{aligned} &\min_{u} J(\mathbf{x_0}, \mathbf{u}) \\ s.t.\ \ &\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_{tk}) \\ &b_i(\mathbf{x}_k, \mathbf{u}_k) \leq c_i \end{aligned} \tag{2.2}$$

where L is called the cost-to-go and is, in general, a non-linear function, f is the system state equations, b_i are system constraints, $\mathbf{x_0}$ is the initial state, and $\mathbf{u}$ is the control input sequence over the problem's horizon T. This means the optimization can be constrained by the system's dynamics and other input constraints. Therefore, the optimal control problem is that of constrained optimization.

The optimisation problem can be solved using various optimisation methods; several commercial [18] [19] and open-source solvers [20] [21] are available to this end. It could be solved in open-loop and the control sequence could be implemented. However, practically there are deviations in the system's state evolution as

compared to the predictions of the system dynamics model. It is therefore prudent to have feedback regarding the state of the system.

2.2 MPC

Feedback control is inherently reactive since it requires the system dynamics to update and provide the state information to the controller. Predictive control uses an internal model to predict how the system would evolve and effectively plans ahead. This ability to predict can help improve efficiency in control allocation by considering how the cost function varies over the horizon depending on the control.

MPC is an iterative predictive optimal control framework, which uses the following procedure:

1. Solve optimal control problem (equation 2.2) for a finite prediction horizon p
2. Implement first control step
3. Measure next state
4. Repeat 1 through 3 until termination

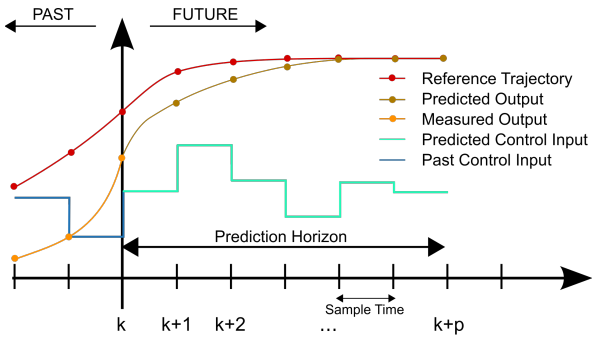


Figure 2.1: MPC scheme [22]

The procedure is visualized in figure 2.1. This effectively results in a combination of feedback and predictive control. The resulting control can be said to be locally optimal since the control sequence minimizes the cost over a relatively small horizon. The ability to include system constraints as part of the optimization problem is a major advantage of MPC. Note that MPC can be implemented for a nonlinear system, using a nonlinear cost function which results in a nonlinear optimization problem. While this is an advantage, it is also computationally demanding because the nonlinear optimization problem must be solved at every time step. The major disadvantages of MPC are the need for an accurate model of the system, and the online iterative optimization needed for control. However, efficient algorithms [23] exist that enable MPC on UAVs in the order of 100 Hz [9]. Additionally, the system's dynamics could be linearised at each step and formulated as a Linear Time Varying (LTV) system and the cost function could also be taken to be quadratic,

effectively making the optimization problem an LTV version of LQR, which is much simpler to solve computationally.

2.2.1 LBMPC

MPC and data-driven modeling can be combined to minimize one of the major disadvantages of MPC. As in [10], an oracle can be used to minimize plant-model mismatch and therefore improve controller performance.

The prediction model for LBMPC has the following form:

$$\mathbf{x_{k+1}} = f(\mathbf{x_k}, \mathbf{u_k}) + g(\mathbf{x_k}, \mathbf{u_k})$$

where f is the nominal model and g is referred to as the oracle, i.e the learned model. Note that the oracle could have different dependent variables apart from $\mathbf{x_k}, \mathbf{u_k}$. The LBMPC procedure is the same as that of MPC, but with the above prediction model.

2.3 Neural Networks

In this project, a NN is used as the oracle for LBMPC. NNs are essentially universal function approximators. They are parametrized compositions of alternating linear and nonlinear transforms of inputs such that an arbitrary nonlinear function can be approximated by appropriately setting the parameters. Per the universal approximation theorem [24], a wide enough neural network can approximate any given non-linear function.

The parameters of the NN are called weights and biases; these represent the slope and offset of the linear transformations of each input node. The ReLU activation function is

$$ReLU(x) = max(0, x)$$

A ReLU NN with input layer of dimension d, 1 hidden layer of n nodes, and a regression layer output with dimension m, can be mathematically represented as

$$\hat{\mathbf{y}} = W_2 \cdot max(0, W_1 \cdot \mathbf{x} + B_1) + B_2$$

where, W_1 and W_2 are weights, B_1 and B_2 are biases.

The net effect of the ReLU NN is to form a piecewise linear representation of the nonlinear function being approximated.

The learning is formulated as the minimisation of a cost or 'loss' function that represents the error between the NN's output and the training data's output values. For the regression learning problem, this is formulated as the Mean Squared Error

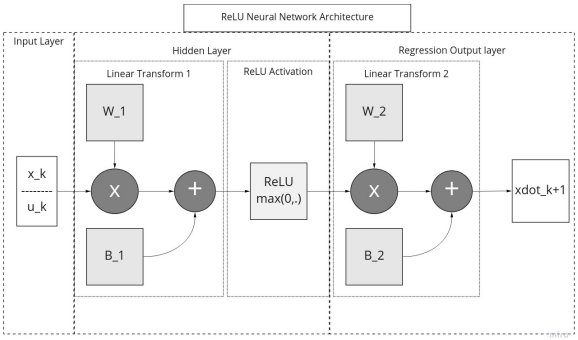


Figure 2.2: ReLU NN architecture as used in this project.

(MSE) of the ground truth $\mathbf{y}$ and predicted $\hat{\mathbf{y}}(\mathbf{x}, \theta)$ where θ is the set of parameters W_1, B_1, W_2, B_2

$$L(\mathbf{y}, \mathbf{x}, \theta) = \frac{1}{M} \sum_{i=1}^{M} (\mathbf{y} - \hat{\mathbf{y}}(\mathbf{x}, \theta))^2$$

where M is the number of training examples.

This loss function is minimized over the parameters θ within the parameter space Θ to find the 'right' parameters for the NN:

$$\theta = arg \min_{\theta \in \Theta} L(\mathbf{y}, \mathbf{x}, \theta)$$

The optimization problem is non-convex and is usually solved using gradient methods such as Stochastic Gradient Descent or extensions of the same like RMS Prop [25] or Adam [26]. In this project, the Adam optimizer algorithm is used because of its superior convergence properties [26].

The gradient method uses the gradient of the loss function with respect to each parameter to adjust the parameter to decrease the Loss function's value. Performing this process iteratively with appropriately sized steps results in the (local) minimization of the loss function. The general structure of the gradient method is:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\mathbf{x}, \mathbf{y}, \theta)$$

where α is the step size of the update, often called the learning rate. This updating step is applied iteratively until a convergence criterion for the loss function is reached.

NNs require more training examples than there are parameters. The number of parameters in the model roughly corresponds to the complexity of the non-linear function that can be approximated. If the number of training examples is lower than the number of parameters, the NN will over-fit the data, leading to poor generalization. This guides the requirements for the size of the training data.

Chapter 3

Methodology

In this chapter, the methodology of the project is described. First, an overview of the methodology is provided. Next, two sections elaborate on the modeling and simulation part of the project. This sets up the base of the simulation environment in which the controller and the learning module are developed and tested. The fourth section provides details about the MPC controller and its implementation. The final section provides details about the NN and its implementation.

3.1 Procedure

The overall procedure followed in the project is as follows:

1. Model the system mathematically and simulate it numerically
2. Implement MPC with nominal model
3. Create various trajectories for data collection and to test controller
4. Run MPC on the training trajectories and collect training data
5. Build the NN and train it on the training data
6. Integrate the NN oracle into MPC to form LBMPC
7. Run the LBMPC controller on the test trajectory
8. Analyse the results to evaluate the performance of LBMPC

3.2 Modeling

In this section, the tilt-rotor UAV's dynamics are modeled mathematically based on certain assumptions and simplifications. This mathematical model is simulated in MATLAB, as described in this chapter. This provides the simulation environment for the controller, as well as the learning module.

3.2.1 Convention and Notation

Any vector $[x_1, x_2, x_3 ... x_n]$ is written as $\mathbf{x}$. The time derivative of $\mathbf{x}$ is written as $\dot{\mathbf{x}}$. Two frames of reference are considered: An inertial, earth-fixed frame and a non-inertial, body-fixed frame. The coordinate axes are taken in the International Standards Organisation (ISO) convention: x-front, y-left, z-up. The right-hand system is considered for rotational axes and cross-products. This means counter-clockwise rotation about an axis is taken to be positive. $sin(\alpha)$ is written as s_α and $cos(\alpha)$ is written as c_α

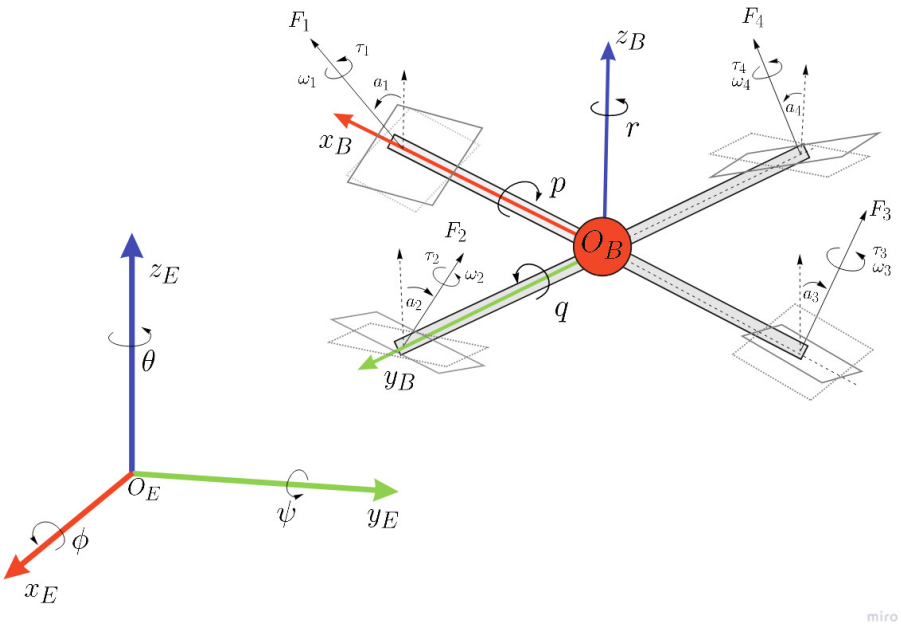


Figure 3.1: Reference frames and conventions. The forces and torques due to each propeller, along with the tilt of each propeller is illustrated.

The Euler angles (ϕ, θ, ψ) which correspond to roll about x axis, pitch about y axis, yaw about z axis are used. These angles are used in the z-y'-x'' rotation order for the formation of the rotation matrices and hence the rotational transformation of coordinates between the reference frames.

The rotation matrix R_{BE} is the matrix for transformation from the body-fixed frame to the earth-fixed frame. Per the properties of rotation matrices: $R^{-1} = R^T$, which means the transformation from earth-fixed frame to body-fixed frame is simply the transpose of R_{BE}, i.e $\mathrm{R}_{EB} = \mathrm{R}_{BE}^T$

$$\mathrm{R}_{\mathbf{BE}} = \begin{bmatrix} c_\phi c_\theta & c_\phi s_\theta s_\psi - s_\phi c_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix}$$

3.2.2 Assumptions and Simplifications

- The UAV is modeled as a rigid body, symmetric about the body-fixed xz and yz planes with arms of equal length, L.
- The center of mass of the system is assumed to coincide with the origin of the body-fixed frame.
- The UAV's principal axes are taken as the body-fixed x, y, and z axes. The inertia tensor is, therefore, a 3x3 diagonal matrix.
- The motors mounted on each arm can tilt about an axis passing from the center of mass to the motor. The tilting of these motors is limited to $[-\pi, \pi]$.
- The counter torque and gyroscopic moments due to the tilting of the motors are neglected.
- The motor-propeller combination is treated as a first-order system, since a DC motor may be approximated as a first order system, and the propeller can be taken to be rigidly attached to the motor.
- The tilting of the motors is also treated as a first-order system, since the servo motors performing the tilting are geared DC motors.
- Aerodynamic drag on the system is considered as $c_d v^2$ where c_d encompasses the entire product of coefficients in the drag equation: $0.5 C_d \rho A_f$. Relative wind is not considered for drag. Torque resultant of Aerodynamic drag as the UAV rotates is neglected.
- All external disturbances are considered as a point force and moment on the center of mass in the earth frame. However, these disturbances can be state dependant.
- The propeller aerodynamics are assumed to be only a function of the angular speed of the propeller.

3.2.3 UAV Dynamics Model

The UAV can move freely in 3D as well as rotate in 3D, therefore, it is said to have 6 degrees of freedom 6 Degree of Freedom (DOF). The dynamics of the UAV can be decomposed into sources of force, torque, and the equations of motion of 6DOF body. The net force and moment on the body are primarily results of the spinning of the propellers which cause aerodynamic forces and moments, but there could also be external influences such as wind, and gravity, etc. This force and moment cause linear and rotational acceleration respectively which give rise to differential equations in state-space of the UAV.

The tilt-rotor UAV's dynamics are modeled using the Newton-Euler equations, with angular rates considered in the body frame and linear velocity considered in the earth frame. This results in a system of differential equations with the following

structure:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{F_d}) \tag{3.1}$$

State and Input

The state-variables are position (x, y, z) and velocity $(\dot{x}, \dot{y}, \dot{z})$ in the earth frame, angular orientation (ϕ, θ, ψ) in the earth frame, and angular rates (p, q, r) in the body-fixed frame. Additionally, the rotational speeds of the motors $\omega_1, \omega_2, \omega_3, \omega_4$, and the tilt angles of the motors, $\alpha_1, \alpha_2, \alpha_3, \alpha_4$,, are considered for the state variables to account for the corresponding system dynamics. The latter

This is a minimal treatment for the state vector, since in reality there are complex interactions between the air-flows of the propellers, especially when tilted such that they interact directly. Additionally, the surrounding air in the proximity of the UAV can be considered for state variable, but this would be at the cost of complicating the model to an extent where online control becomes intractable due to extreme computational costs. Instead, these higher-order effects could be relegated to the learning-based control.

The state vector of the system is, therefore, a 12-dimensional vector:

$$\mathbf{x} = [x\ y\ z\ \theta\ \psi\ \phi\ \dot{x}\ \dot{y}\ \dot{z}\ p\ q\ r]^T$$

The inputs to the UAV consists of the angular rates of the 4 propellers: ω_{ref_1}, ω_{ref_2}, ω_{ref_3}, ω_{ref_4} and the tilt-angles α_{ref_1}, α_{ref_2}, α_{ref_3}, α_{ref_4} of the respective propellers.

The input vector to the system is therefore:

$$\mathbf{u} = [\omega_{ref_1}\ \omega_{ref_2}\ \omega_{ref_3}\ \omega_{ref_4}\ \alpha_{ref_1}\ \alpha_{ref_2}\ \alpha_{ref_3}\ \alpha_{ref_4}]^T$$

Propeller Thrust, Torque and Power

Each propeller is taken to produce a thrust force:

$$|\mathbf{F_i}| = f_i = k\omega_i^2\ , \mathrm{i} = 1, 2, 3, 4. \tag{3.2}$$

The components of the thrust in the body frame are:

$$\mathbf{F_1} = \begin{bmatrix} 0 \\ f_1 s_{\alpha_1} \\ -f_1 c_{\alpha_1} \end{bmatrix}, \quad \mathbf{F_2} = \begin{bmatrix} f_2 s_{\alpha_2} \\ 0 \\ -f_2 c_{\alpha_1} \end{bmatrix}, \quad \mathbf{F_3} = \begin{bmatrix} 0 \\ f_3 s_{\alpha_3} \\ -f_3 c_{\alpha_3} \end{bmatrix}, \quad \mathbf{F_4} = \begin{bmatrix} f_4 s_{\alpha_4} \\ 0 \\ -f_4 c_{\alpha_4} \end{bmatrix}$$

Each propeller is taken to produce a counter torque, opposite to the direction of angular velocity:

$$\tau_\mathbf{i} = -K|\omega_i^2|\hat{\omega_i}\ , \mathrm{i} = 1, 2, 3, 4. \tag{3.3}$$

Propellers 1 and 3 rotate counter-clockwise, and propellers 2 and 4 rotate clockwise. Additionally, each propeller produces a moment about the center of mass as:

$$\mathbf{L_i} \times \mathbf{F_i}, \mathrm{i} = 1, 2, 3, 4 \tag{3.4}$$

The net torques on the UAV due to each propeller in the body frame coordinates are:

$$\mathbf{M}_1 = \mathbf{L_1} \times \mathbf{F}_1 + \tau_1 = \begin{bmatrix} L \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ f_1 s_{\alpha_1} \\ f_1 c_{\alpha_1} \end{bmatrix} + \begin{bmatrix} 0 \\ \tau_1 s_{\alpha_1} \\ \tau_1 c_{\alpha_1} \end{bmatrix}$$

$$\mathbf{M}_2 = \mathbf{L_2} \times \mathbf{F}_2 + \tau_2 = \begin{bmatrix} 0 \\ L \\ 0 \end{bmatrix} \times \begin{bmatrix} -f_2 s_{\alpha_2} \\ 0 \\ f_2 c_{\alpha_2} \end{bmatrix} + \begin{bmatrix} -\tau_2 s_{\alpha_2} \\ 0 \\ -\tau_2 c_{\alpha_2} \end{bmatrix}$$

$$\mathbf{M}_3 = \mathbf{L_3} \times \mathbf{F}_3 + \tau_3 = \begin{bmatrix} -L \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ -f_3 s_{\alpha_3} \\ f_3 c_{\alpha_3} \end{bmatrix} + \begin{bmatrix} 0 \\ -\tau_3 s_{\alpha_3} \\ \tau_3 c_{\alpha_3} \end{bmatrix}$$

$$\mathbf{M}_4 = \mathbf{L_4} \times \mathbf{F}_4 + \tau_4 = \begin{bmatrix} 0 \\ -L \\ 0 \end{bmatrix} \times \begin{bmatrix} f_4 s_{\alpha_4} \\ 0 \\ f_4 c_{\alpha_4} \end{bmatrix} + \begin{bmatrix} \tau_4 s_{\alpha_4} \\ 0 \\ -\tau_4 c_{\alpha_4} \end{bmatrix}$$

Since the spinning of the propeller results in a thrust force as well as a net moment about the center of mass of the UAV, the dynamics are said to be coupled.

Newton-Euler equations

Formulating the net force vector on the body in the body-fixed frame excluding gravity: The net force on the body (in the body frame) due to the propellers is:

$$\mathbf{F}_B = \sum_{i=1}^{4} \mathbf{F}_i \tag{3.5}$$

The total force on the body (in the earth frame) is:

$$\mathbf{F}_E = \mathrm{R_{BE}} \mathbf{F}_B + \mathbf{g} \tag{3.6}$$

Using Newton's second law:

$$\dot{\mathbf{v}}_E = \frac{\mathbf{F}_E}{m} \tag{3.7}$$

The velocity in the earth frame is the time derivative of the UAV's position in the earth frame:

$$\mathbf{v}_E = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{3.8}$$

The total torque on the UAV (in the body frame) is;

$$\mathbf{M}_B = \sum_{i=1}^{4} \mathbf{M}_i \tag{3.9}$$

Using Euler's equation, the rate of change of angular momentum of the UAV in the body frame with the time derivative taken with respect to the earth frame is:

$$I\dot{\mathbf{\Omega}}_B = \mathbf{M}_B - \mathbf{\Omega}_B \times I\mathbf{\Omega}_B \tag{3.10}$$

The angular velocity of the UAV (in body frame) is:

$$\mathbf{\Omega}_B = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.11}$$

The body angular rates are converted to roll, pitch and yaw rates as:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi)sec(\theta) & cos(\phi)sec(\theta) \end{bmatrix} \cdot \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.12}$$

Extensions

The above equations provide the state equations for the UAV. Accounting for the motor dynamics and the tilting dynamics, 8 additional state variables need to be included:

$$\mathbf{x_{ext}} = [\alpha_1, \alpha_2, \alpha_3, \alpha_4, \omega_1, \omega_2, \omega_3, \omega_4]^T$$

This latter extension to the state is used in the plant simulation. On the controller's prediction model, this extension is neglected in the interest of creating a plant-model mismatch. This is also because it would be difficult to accurately model and determine the parameters for these systems in practice.

Additionally, the state equations for the extended state which includes the motor and tilting dynamics are:

Tilt Dynamics

The tilting of the motors are treated as a first order system [reference].

$$\dot{\omega}_i = \frac{1}{\tau_t}(\omega_{ref_i} - \omega_i) \tag{3.13}$$

where τ_t is the tilting mechanism's time constant.

Motor Dynamics

The mechanical dynamics of the motor-propeller sub-system is treated as a first-order system [reference]

$$\dot{\alpha_i} = \frac{1}{\tau_m}(\alpha_{ref_i} - \alpha_i) \tag{3.14}$$

where τ_t is the motors mechanical time constant.

Aerodynamic Drag

Aerodynamic drag is modeled as a function of velocity in the earth frame

$$\mathbf{F_a} = \begin{bmatrix} c_{dx}\dot{x}^2 & c_{dy}\dot{y}^2 & c_{dz}\dot{z}^2 \end{bmatrix} \tag{3.15}$$

3.2.4 Disturbances

To examine how the NN can help adapt to various disturbance forces, different forces, and 'disturbances' are formulated. There are seven variations of disturbances used in this project:

1. No disturbance
   The first variation is the case with no disturbance. Therefore,

   $$\mathbf{F_D} = [0\ 0\ 0]^T N$$

2. Constant
   This disturbance is used to gauge how well the NN can represent constant errors. The NN should be able to learn to predict this disturbance the most accurately among all the disturbances. Additionally, this disturbance represents a constant wind-like force in the xy plane.

   $$\mathbf{F_D} = [5\ 5\ 0]^T N$$

3. Ground effect 1
   This disturbance represents the ground effect based on a modified version of the helicopter ground effect as presented in [27]. The ground is taken as $z_k = -7\ m$.

   $$F_{D_z} = F_{thrust}(\frac{1}{1 - 3(\frac{R}{z})^2} - 1)$$

   where $R = 0.45m$ is the diameter of the propeller, $z = 7 + z_k\ m$ is the height of the UAV from the ground

   $$\mathbf{F_D} = [0\ 0\ F_{D_z}]^T N$$

4. Ground effect 2
   This disturbance represents the ground effect based on the modified helicopter's ground effect model as in [28] which is an extended version of [27] and better resembles ground effect for quadrotors. The disturbance is a function of the UAV's thrust and z-position.

$$F_{D_z} = F_{thrust}\left(\frac{1}{1 - (\frac{R}{4z})^2 - R^2\frac{z}{\sqrt{(d^2+4z)^3}} - \frac{R^2}{2}\frac{z}{\sqrt{(2d^2+4z^2)^3}} - 2R^2K\frac{z}{\sqrt{(d^2+4z^2)^3}}} - 1\right)$$

   where $R = 0.45m$ is the diameter of the propeller, $z = 7 + z_k\ m$ is the height of the UAV from the ground, $d = 0.55m$ is the arm length of the UAV and $K = 2$ is an empirical constant (same as used in [28])

$$\mathbf{F_D} = [0\ 0\ F_{D_z}]^T N$$

5. Sigmoid
   This disturbance is meant to represent an extreme case of a ground effect like disturbance, where there is a force-field that saturates at negative z values compared to the origin. This is formulated as a sigmoid multiple of a quarter of the thrust of the UAV.

$$F_{D_z} = \frac{F_{thrust}}{4}(1 - \frac{1}{1 + e^{-z_k}})$$
$$\mathbf{F_D} = [0\ 0\ F_{D_z}]^T N$$

6. Model Parameter Error
   In reality, it is often the case that the estimation of the parameters of the system being controlled can be erroneous. To examine how the LBMPC controller functions in the presence of such model parameter errors, small errors in model parameters are introduced as listed in table 3.2 in the next section. These errors in parameters are within 5% of the plant's parameters. Note that the disturbance force, $\mathbf{F_D}$ itself is set to 0.
7. Noisy State Estimate
   The last variation of disturbance attempts to represent practical state estimation systems where the state estimates can be noisy. The standard deviation of the noise was maximized until the LBMPC controller failed, to examine how much noise can be handled by the controller. Additionally, to represent the worst-case scenario while control is still feasible, the model parameter error is also included in this 'disturbance'. However, the disturbance force, $\mathbf{F_D}$ itself is set to 0. Noise is modelled as a normally distributed random noise $N(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}})$ on the state estimation.

$$\tilde{\mathbf{x}_k} = \mathbf{x}_k + N(0, 0.01)$$

Net Force and Moment

Therefore, the net force on the body in the earth frame and moment in the body frame are:

$$\mathbf{F_E} = \mathrm{R_{BE}}\mathbf{F_B} + \mathbf{g} + \mathbf{F_D} - \mathbf{F_a} \tag{3.16}$$

$$\mathbf{M_B} = \mathbf{M_B} \tag{3.17}$$

3.3 Simulation

The above model is simulated using MATLAB, by numerically solving the state differential equations with the ode45 solver available in MATLAB. The Simulation setup is described in this section.

3.3.1 MATLAB setup

The MATLAB simulation environment is illustrated in figure 3.2. The simulation consists of a loop that steps through the duration of the simulation in increments of sample time, and the control is determined at each iteration. For this to make sense practically, the control input must be determined at a frequency greater than or equal to the sampling frequency. The outer loop also appends the relevant information of each iteration such as state, input, reference trajectory, and state derivative, to collect data regarding the simulation and for the training of the NN.

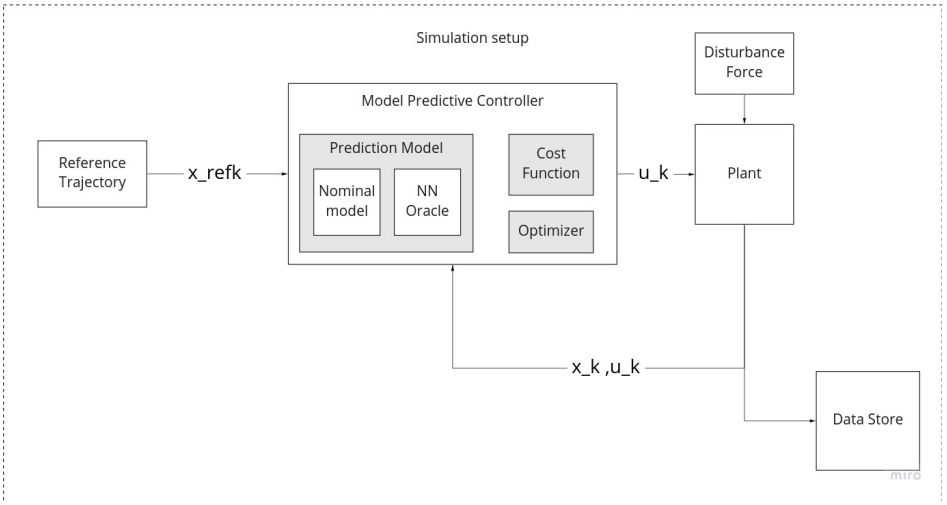


Figure 3.2: Simulation setup

Simulation Process

The main procedure followed in the simulation is as follows:

1. Start
2. For duration:
   2.1. Get reference trajectory $\mathbf{x}_{refk}$
   2.2. Get control input $\mathbf{u}_k$ from controller
   2.3. Use current state $\mathbf{x}_k$ and control input $\mathbf{u}_k$ to simulate state evolution up to next step $\mathbf{x}_{k+1}$
   2.4. Log all relevant information
3. End

Plant

The simulation model of the tilt-rotor UAV with maximum fidelity is referred to as the 'plant'. This represents the actual physical system as closely as possible. The plant is simulated by using MATLAB's ode45 differential equation solver. The initial conditions are described as the current state and control inputs from the controller. Then, the system equations are solved up to the next time step in increments of sampling time T_s. The model used for prediction by the controller is referred to as the controller model.

Controller Model

The prediction model used by the controller does not have as much fidelity as the plant. The plant contains the full extent of the above dynamics model, while the controller model will exclude the aerodynamic drag, tilting dynamics and motor dynamics. The controller model also does not account for the disturbances. This causes plant-model mismatch and is meant to represent practical mismatches. The prediction model is fed to the controller in a discrete form as explained further in section 3.4.

3.3.2 Parameters

The parameters used for the plant model and the controller model are provided in tables 3.1, 3.2. The mass and moments of inertia were estimated based on the CAD model. The CAD model reference had 0.55 m arms and therefore L was taken to be 0.55 m.

The propeller thrust coefficient k, was calculated from commercial off-the-shelf datasheets [29] of propellers of the same size as in the CAD model. Considering

Parameter		Value
Mass	m	6 kg
Moments of Inertia	I_{ii}	[1.2,1.2,2.3] kgm^2
Arm length	L	0.55 m
Propeller thrust coeff	k	8x10^{-5} Ns^2/rad^2
Propeller drag coeff	K	4x10^{-6} Nms^2/rad^2
Coefficients of drag	c_{di}	[0.01,0.01,0.05] Ns^2/m^2
Motor time constant	τ_ω	0.01 s
Tilt time constant	τ_α	0.2 s

Table 3.1: Plant model parameters. First 6 parameters are identical for the controller model

Parameter		Value
Mass	m	6 kg
Moments of Inertia	I_{ii}	[1.25,1.15,2.2] kgm^2
Arm length	L	0.55 m
Propeller thrust coeff	k	8.1x10^{-5} Ns^2/rad^2
Propeller drag coeff	K	3.9x10^{-6} Nms^2/rad^2

Table 3.2: Controller model erroneous parameters

propeller drag coefficient $K = \frac{k}{20}$ is reasonable for a practical system. The motor time constant was set based on commercially available brush-less direct current motor specifications for mechanical time constants [30]. The tilt time constant was set based on commercially available servo motor specifications for 60 degree rotation at 1 kg-cm torque [31]. The aerodynamic drag coefficients are based on reasonable assumptions regarding profile area, constant air density, and coefficients of drag.

Model Verification

The dynamics model was verified by passing constant inputs and initial conditions to check whether the simulated model operates as expected.

3.4 Control

The MPC controller is implemented using MATLAB's Nonlinear MPC toolbox's nlmpc [32] block. The main aspects of the implementation of the controller involve defining the prediction model, cost function, and implementing the tool in closed-loop simulation.

MPC Control Procedure (1 step):

1. Start
2. Get reference trajectory $\mathbf{x}_{refk}$, current state $\mathbf{x}_k$, previous input $\mathbf{u}_{k-1}$, prediction model 3.18
3. Minimise the cost function 3.20 over the sequence of inputs $\mathbf{u}_k, \mathbf{u}_{k+1}, ...\mathbf{u}_{k+p}$ for horizon p
4. Output the first control step $\mathbf{u}_k$
5. End

3.4.1 MPC

MATLAB's nlmpc block performs nonlinear MPC control given a dynamics model and a cost function. The nlmpc tool can take in a continuous-time prediction model or a discrete-time prediction model. The nlmpc tool internally integrates the continuous-time model using trapezoidal discretization. However, since the nlmpc tool needs to include the NN model later, it was used with a discrete prediction model.

Prediction model

The discrete prediction model used is of the LBMPC form:

$$\mathbf{x}_{k+1} = x_k + T_s(f(\mathbf{x}, \mathbf{u}) + g(\mathbf{x}, \mathbf{u}, \mathbf{F_d})) \tag{3.18}$$

Note that for the nominal MPC controller only f is used and g is included for LBMPC controller. Here, f and g are continuous state equations, with f being the nominal model and g being the prediction of the NN. This is a 1-step forward Euler integration with a discretization time $T_s = 0.1\ s$, the same as the sample time of the controller. This discretization time was found to provide sufficient accuracy. Smaller discretization times were tried such as $0.01\ s$, but these were found not to offer much better performance.

Equation 3.18 follows from the continuous version of the prediction model equation:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) + g(\mathbf{x}, \mathbf{u}, \theta) \tag{3.19}$$

$f(\mathbf{x}, \mathbf{u})$ is the nominal model, same as in section 3.2. $g(\mathbf{x}, \mathbf{u}, \theta)$ is the oracle, which is to be learned.

The nominal MPC controller is also referred to as the baseline controller since its performance is the baseline with which the NN based LBMPC controller will be compared.

nlmpc

nlmpc solves the nonlinear constrained optimization problem each time step using the Sequential Quadratic Program algorithm and the 'fmincon' optimization solver. The precise functioning of these algorithms are beyond the scope of this project. Initial tests with nlmpc showed that computing a control step would take about 1 second, which is not useful for controlling a UAV at the level of the motors and tilts. This was found to be significantly expensive computationally and therefore is impractical.

LTV Formulation

Formulating the nonlinear MPC problem using a LTV description of the system allowed the optimization problem to be formulated as a Linear Quadratic Tracking problem and speeds up control step computation time to the order of tens of milliseconds, which is roughly 100 times faster than the full nonlinear MPC formulation. In the MATLAB simulation environment, the LTV MPC was running at 40 Hz on average, but worst-case computation time was 15Hz. Therefore, to provide a sufficient margin to the controller, a sample time and a control time $T_s = 0.1$ s, is chosen, and the nlmpc controller was run in LTV mode.

Cost Function

The cost function is to be constructed such that it represents the requirements of the application in terms of tracking performance and energy efficiency. The MPC formulation of control requires a discrete control and a discrete cost function. The discrete cost function used for this project is the default cost function of nlmpc. This is a quadratic cost function as in equation 3.20 and only the weights Q, R, S, and T were tuned. This is because there was no basis for an alternative cost function, and the required terms for control were represented in the cost function. Tuning the weights of these terms would therefore suffice for the purpose of this project. It remains however, to investigate if there are other cost functions that could improve the performance of the LBMPC approach.

$$J(\mathbf{x}^*, \mathbf{u}^*) = \sum_{0}^{T} Q(\mathbf{x_t} - \mathbf{x}_t^*)^2 + R(\mathbf{u_t} - \mathbf{u}_t^*)^2 + S\dot{\mathbf{x}_t}^2 + T\dot{\mathbf{u}_t}^2 \tag{3.20}$$

$$s.t. \quad equation\ 3.18\ ;$$

$$0 \leq \omega_i \leq 1000, \quad -\pi \leq \alpha_i \leq \pi$$

The specific values for Q, R, S, and T are listed in table 3.3. These weights affect the aggressiveness of control, the tradeoff between trajectory tracking and

input effort, and the cost of the rate of change in the variables. The input effort costs for the motors are taken to be the propeller thrust coefficient since this acts as a proxy for the power consumption (power would be a function of the third power of the motor's angular rate), the cost of tilting is set at 0.1 as this was found to provide a good balance between rapid tilting and hesitation to tilt (values from 0.01 to 1 were tried). The cost weights for the state variables were set at 10 for each of the first six variables since these are the only variables considered for tracking for this project. The value 10 was found to provide the most stable performance for both the MPC and LBMPC controllers. (values between 1 to 20 were tried).

3.4.2 Controller setup

The parameters of the nlmpc controller are provided in table 3.3. A prediction horizon of 15 steps was used to collect training data. However, with LBMPC it was found that 15 steps was too long a prediction horizon because the NN based predictions would become poorly conditioned and accumulate significant error leading to destabilizing input sequences. A prediction horizon of 10 steps was found to be stable for both LBMPC and nominal MPC. Therefore, this was used for all subsequent simulations which required comparisons with LBMPC performance.

Parameter	**Value**
Sample time T_s	0.1 s
Prediction discretization time	0.1 s
Prediction Horizon data collection	15 steps
Prediction Horizon experiments	10 steps
Control horizon	1 step
Initial condition x_0	[0 0 0 0 0 0 0 0 0 0 0 0]
Initial condition u_0	[400 400 400 400 0 0 0 0]
Actuator limits u_{min}	[0 0 0 0 $-\pi$ $-\pi$ $-\pi$ $-\pi$]
Actuator limits u_{max}	[1000 1000 1000 1000 π π π π]
State cost weights Q	[10 10 10 10 10 10 0 0 0 0 0 0]
Input cost weights R	[8x10^{-5} 8x10^{-5} 8x10^{-5} 8x10^{-5} 0.1 0.1 0.1 0.1]
State derivative cost weights S	[0 0 0 0 0 0 0 0 0 0 0 0]
Input rate cost weights T	[0.01 0.01 0.01 0.01 0.5 0.5 0.5 0.5]

Table 3.3: Baseline and (LB)MPC controller parameters

The control reference, $\mathbf{u}_{ref0}$ is initially set to [400 400 400 400 0 0 0 0] since this sets the initial motor speed reference to close to what is required for hover. Note that this is not exactly the motor speed required for hover. 400 is selected for no

particular reason apart from being sufficiently close to the required speed for hover. Much lower speeds or much higher speeds were found to destabilize the system (the motor dynamics seem to be the cause for this since it causes a lag in dynamics and sufficient to make the controller take extreme actions). Except for the initial input reference, for the remainder of the simulation, the input reference is set to zero for all inputs.

3.5 Learning Module

Given the plant model mismatch (artificially created in simulation), it is expected that the nominal prediction model has errors in the prediction of the state evolution. This section describes the learning problem, the data collection, the NN, and the integration into the LBMPC controller.

3.5.1 Learning Problem

The goal of the learning module is to predict the error in the model's prediction of linear acceleration. The learning problem is limited to the linear acceleration because the prediction of velocity or position is not ultimately useful since these follow from the integration of acceleration only. The prediction of rotational dynamics was seen to be much more sensitive to error and would require significant tuning of the learning module. Therefore, as a first step, the learning module only focuses on linear acceleration.

Additionally, only the current state and control input vectors are used as input for the NN because the performance of the prediction did not change significantly when additional past data was included, and to keep the modifications to the controller minimal.

The learning objective for the NN is formulated as the minimization of MSE in the prediction of linear acceleration.

Learning Objective

$$\min_{\theta} \sum_{i=1}^{M} (\mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{F_D}) - \tilde{g}(\mathbf{x}, \mathbf{u}, \theta))^2 \tag{3.21}$$

Where

$$\mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{F_D}) = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} - f(\mathbf{x}, \mathbf{u})$$

Assumptions

- Full state estimate is available.
  The input to the NN is the current full state and control input vectors. To examine how much noise in state information can be tolerated by the LBMPC, noisy state estimates were used as a form of disturbance. The results are listed in the next chapter.
- Linear acceleration ground truth can be recorded for training.
  This is a reasonable assumption since Inertial Measurement Unit (IMU)s have accelerometers that can be used to capture ground truth of linear acceleration, albeit with some sensor noise which can be filtered out.

3.5.2 Data collection

Training data for the NN was collected by running the nominal MPC controller on 9 training trajectories that were created to cover as much of the system's dynamics as possible for better prediction performance. To do so, trajectories with variations in only one of the 6DOF, as well as trajectories with variations in multiple DOF simultaneously, were crafted. Additionally, 1 test trajectory was created. In total 9 training trajectories were created, and 1 test trajectory. The trajectories' names, the duration for which the controller was run on them to collect data, and the generating functions for each trajectory are given in table A.2. The duration of each trajectory was based on the time-period of the generating function as well as how much of the dynamics the trajectory is intended to explore. This can be seen as a kind of weightage of data for the NN to train on.

Test Trajectory

Finally, a trajectory was crafted for use as test trajectory (trajectory 10 ('Loop') in table A.2). This trajectory has a combination of sinusoidal variations in the position but not orientation. The NN was not trained on this trajectory and would only be tested on it as explained in the next chapter. The performance of the LBMPC on the test trajectory will indicate the generalization ability of the NN and the LBMPC controller.

Remark: Roty was not covered since formulation of dynamics is prone to singularity at $\theta = \pm \frac{\pi}{2}$. This singularity is avoidable by formulating the dynamics using quaternion (using 1 extra state variable) or rotation matrix entries (using 6 extra state variables). This is not addressed in the project in the interest of simplicity and since the focus is on first establishing the usefulness of the NN based LBMPC approach.

Remark: The first five steps of each recorded trajectory was omitted from the training data since large tracking errors initially result in the nominal controller producing significant spikes. This was found to cause issues downstream with the training of the NNs due to the large errors in the prediction by the nominal model.

Remark: In practice, data collection would be done using the physical system and would be much more complicated. Practical trajectories are hard to hand design and therefore some form of direct high-level control by humans might be necessary for data collection to avoid having to craft trajectories. Also, it is important to note that most applications require relatively simple trajectories that involve slow movement or long periods of hover, which will make it hard to collect sufficient data on the dynamics.

Data Collection Process

1. Start
2. For each disturbance 1 through 6 (section 3.2.4)
   2.1. For each trajectory 1 through 9 (table A.2)
      2.1.1. Run MPC for duration of trajectory per table A.2
      2.1.2. Collect and store $\mathbf{x}$,$\mathbf{u}$,$\dot{\mathbf{x}}$,$\dot{\mathbf{x}}_{pred}$
3. End

3.5.3 NN model

The NN oracle is a ReLU NN model, which is based on the architecture in [16] [17]. A separate NN model was trained for each disturbance 1 through 6, and a different number of nodes were tried. The best performance for each disturbance was uniformly provided by 150 nodes. Having more layers or nodes results in overfitting as was empirically observed. Having fewer resulted in poorer performance. Therefore, for each disturbance 1 through 6, a corresponding NN was trained with the data collected on that disturbance.

Remark: The NN used for disturbance 7 is the same as the one trained on disturbance 6. This is because the point of disturbance 7 is to check the vulnerability of the NN to noisy state estimates. Since disturbance 7 is just disturbance 6 with noisy state estimates, the same NN was used. (See figure A.1 for an Overview of trajectories, disturbances and NNs)

The number of nodes in the hidden layer needed for this project was found to be much lower (150 nodes) than in [16] (2000 nodes). This is mostly because the learning problem is focused on the error in prediction whereas in [16] it is to predict the complete dynamics. After the brief architecture search, the following architecture

for the NN was selected: 20 input features, 1 hidden layer with 150 nodes of ReLU activation, 3 outputs. This is the same architecture as in figure 2.2.

Since the values of the various input features to the NN are of different magnitudes, they are scaled and re-centered using z-score normalization. The mean and standard deviation are computed from the training data and carried over for the testing as well. The normalization for any variable x is:

$$\bar{x} = \frac{x - \mu_x}{\sigma_x}$$

Therefore, with the normalised inputs, the NN is:

$$\hat{g} = W_2\Phi(W_1[\bar{\mathbf{x}_k}, \bar{\mathbf{u}_k}]^T + B_1) + B_2 \tag{3.22}$$

Where,

$$\Phi(x) = \max(0, x)$$

The NN was built using MATLAB's Deep learning toolbox [33] which allows a high-level design of NNs while providing a fair amount of customization.

3.5.4 Training

Overall, 24473 timesteps of data were collected, of which 16473 are randomly sampled and used for training the NN. The remainder 8000 examples are used as validation dataset and used for tuning the hyperparameters. MATLAB's training NN tool was used with the Adam optimizer [26]. The hyperparameters found to provide the least validation Root Mean Squared Error (RMSE) are presented in the Appendix A.1. Hyperparameter search was not conducted extensively, and there may be better sets of hyperparameters for NN corresponding to each disturbance. The main hyperparameters tuned were the number of neurons and the number of training epochs. The NN is trained for 400 epochs as this was found to provide the minimum validation RMSE before starting to overfit (validation RMSE begins to increase).

3.5.5 LBMPC with NN

The predicted state derivative, $\dot{\mathbf{x}}_{pred}$ is used for the discrete model with the same forward Euler integration scheme as in equation 3.18. The trained NN is used to determine $g(\mathbf{x}, \mathbf{u}, \theta)$ in equation 3.18.

There are two ways to use the trained NNs to predict g:

1. By directly computing the NN's output using equation 3.19.
2. Using the predict function, which is MATLAB's Deep Learning toolbox's internal tool for NNs, and is relatively slow.

The first method was found to be roughly 100 times faster (at 0.0001 s per prediction), and would practically help retain the ability to perform 10Hz control. While this shows NN can be used at a fast enough rate, there is a discrepancy in method (1) and method (2) due to precision loss as a result of the internal handling of calculation. This resulted in a significant difference in controller performance, and method (1) was found to fail occasionally as a result. Therefore, for consistency, method (2), predict, was used for all experiments.

Remark: While the NN has not seen the test trajectory itself in any form, in general, due to the very fact that the NN's inclusion causes changes in the control inputs, the dynamics will vary for any trajectory from the baseline trajectories and therefore will not be identical to the training trajectories. However, the test trajectory ensures that the dynamics have no states common with the training data set, and therefore test the generalization capability of the NN.

3.6 Simulation Tests

The MPC baseline controller is simulated on the test trajectory (trajectory 10 in table A.2) for 20 seconds for each disturbance (1 through 7 in section 3.2.4). The simulation tests last only 20 seconds for each disturbance because one complete cycle of the test trajectory is completed by 20 seconds and a longer duration would simply repeat the same loop. The LBMPC controller is simulated on the test trajectory for 20 seconds for each disturbance using the predict function for the corresponding NN. The cost over the trajectory, the prediction error, and the tracking error are recorded.

The prediction error is the error in prediction of linear acceleration between the nominal model with NN and the actual linear acceleration at that time step. The baseline which this is compared to is simply the error in prediction of linear acceleration between the nominal model without NN and the actual linear acceleration at that time step. Both of these errors are collected using the LBMPC controller. This is because the trajectory itself will be different for MPC and LBMPC, so it is difficult to compare the predictions of the baseline model and the LBMPC model.

The tracking error is the error in tracking between the baseline controller and the LBMPC controller. The cost used as results is the sum of the cost function's (3.20) value per step over the entire duration for each of the baseline and LBMPC controllers, for each disturbance.

Simulation Test Procedure

1. Start

2. For each disturbance 1 through 7
   2.1. Run nominal MPC on the test trajectory for 20 s.
   2.2. Collect $\mathbf{x}_{ref}$, $\mathbf{x}$,$\mathbf{u}$, cost, $\dot{\mathbf{x}}$, $\dot{\mathbf{x}}_{\mathbf{pred}}$
3. End
4. Start
5. For each disturbance
   5.1. Run LBMPC on the test trajectory for 20 s using NN trained on respective disturbance
   5.2. Collect $\mathbf{x}_{ref}$, $\mathbf{x}$,$\mathbf{u}$, cost, $\dot{\mathbf{x}}$, $\dot{\mathbf{x}}_{\mathbf{pred}}$
6. End

In the next chapter, the results of the simulation tests are summarised using MSEs of prediction error, tracking error, and the total costs for each disturbance. Additionally, the trajectory tracking and control inputs are plotted.

Chapter 4

Results

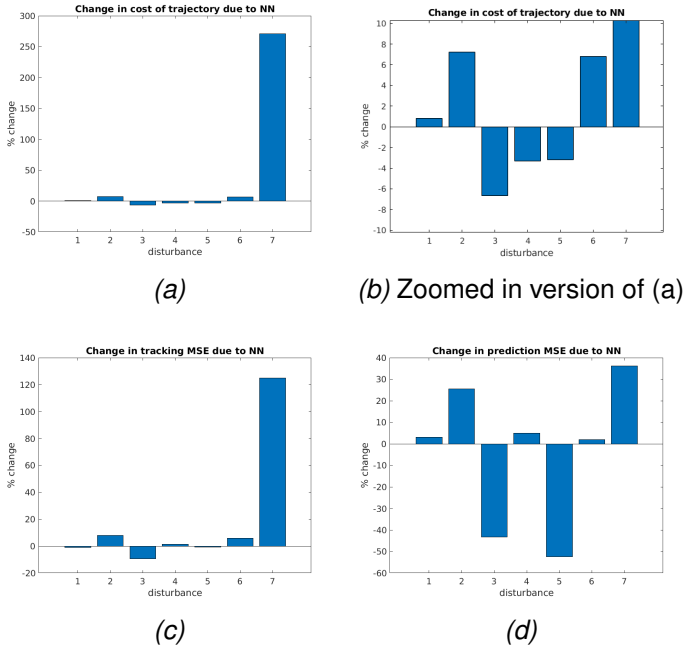
This chapter presents the results of the simulation tests conducted as described in the previous chapter. Further, this chapter makes observations based on these results, presents the failure modes of the LBMPC controller, and the possible reasons for the same.

4.1 Simulation Results

The results, as collected through the simulation tests (see section3.6), are quantified through total costs, MSEs of the state tracking and prediction errors. The percentage changes of these metrics relative to baseline, for the test trajectory are plotted in figure 4.1. The seven disturbances (see section 3.2.4) are numbered 1 through 7 and represented on the x-axis of each plot. The y-axis represents the percentage change in the metric. Negative changes in cost, tracking MSE, and prediction MSE indicate better performance of the controller compared to the baseline.

In figure 4.1(a), it is seen that the cost increases extremely for disturbance 7 (noisy state estimate). Zooming into the figure, it can be seen that costs are decreased only for disturbances 3, 4 and 5. Looking at the tracking MSE and prediction MSE figures, it can be seen that once again, the direction of changes is not consistent for the different disturbances.

Clearly, per figure 4.1, the LBMPC does not uniformly decrease cost for the disturbances, and performs worse for disturbances 1, 2, 6 and 7. Note that the performance on the test trajectory primarily indicates the generalization ability of the NN and therefore the LBMPC controller to trajectories it was not trained on. However, it could be expected that the NN improves prediction, and therefore tracking and costs on at least disturbances 1 through 6. It is interesting that performance with disturbance 2 is worse, while performance on more complicated disturbances such as disturbance 3, 4, and 5 show improved performance. This could imply that the NNs



(a) *(b)* Zoomed in version of (a)

(c) *(d)*

Figure 4.1: The summary of results for the simulation tests on the test trajectory. (a) Total cost over the test trajectory for each disturbance with NN, plotted as a percentage change compared to baseline. (b) A zoomed in version of (a) is presented since disturbance 7 causes significant increase in cost such that the other values are not clear. (c)Tracking MSE. (d) Prediction MSE.

trained on disturbance 1, 2 and 6 data respectively are overfitting, while the NNs trained on disturbances 3, 4 and 5 respectively are able to generalize the prediction sufficiently to improve performance.

In order to gain insight into what is happening for disturbance 7 on the test trajectory, the tracking MSE, prediction MSE, trajectory tracking, and control inputs are plotted in figure 4.2. Figure 4.3 shows the same plots for disturbance 1. Clearly, the controller fails on the test trajectory with disturbance 7.

The x-axes of Figures 4.2 (a) and (b) are as follows: For subfigure (a), x, y, z directions are numbered 1, 2, 3 respectively. For subfigure (b), the state variabes x, y, z, ϕ, θ, ψ are numbered 1, 2, 3, 4, 5, 6 respectively.

As seen in figure 4.1, the costs, tracking error, and prediction errors are not consistent in terms of indicating improvement in the controller's performance on the test trajectory due to LBMPC. The priority for determining the ultimate usefulness of the oracle in LBMPC is the cost of the trajectory since this encodes all the requirements of the control design. The next priority is tracking error since this is the highest weighted component in the cost. The last priority among the three criteria is the prediction error. However, as seen in figures 4.3(e) and (f), small changes in prediction can result in significantly different control inputs, and therefore affect costs disproportionately.

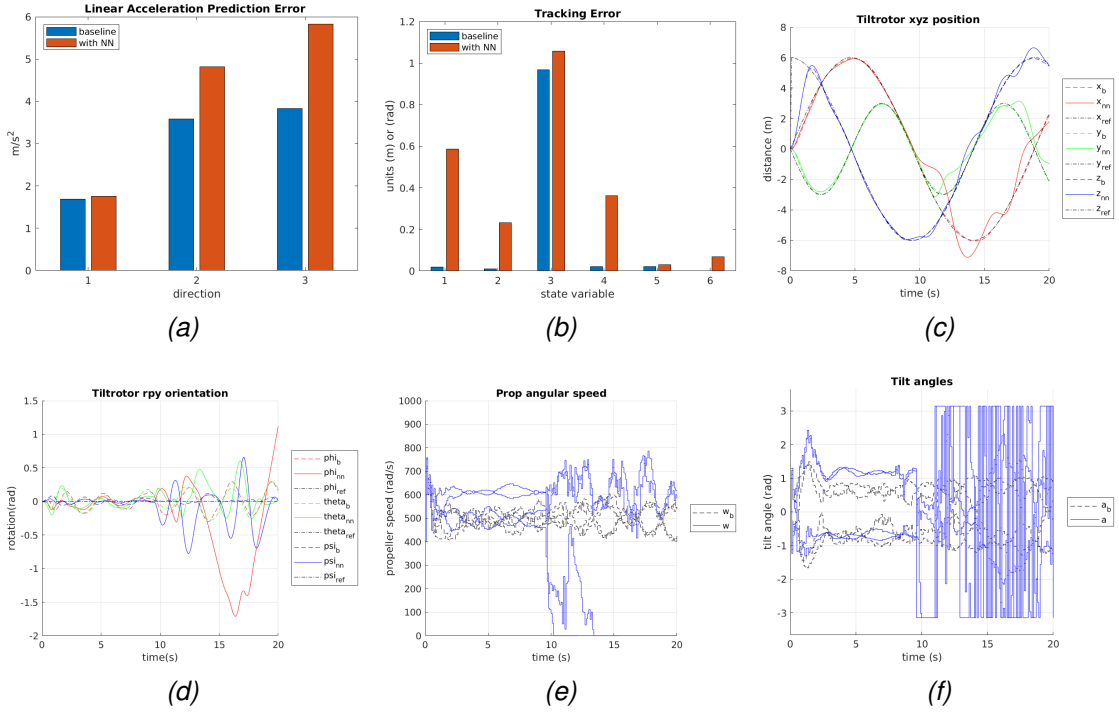


Figure 4.2: Performance of LBMPC controller on test trajectory for disturbance 7. Note the beginning at 10 seconds. Also observe the wild oscillations in the tilt angle commands in (f) and the complete shutdown of motors in (e). It can be said that the noisy state estimates cause failure of the LBMPC controller in this trajectory, whereas the baseline controller tracks the reference.

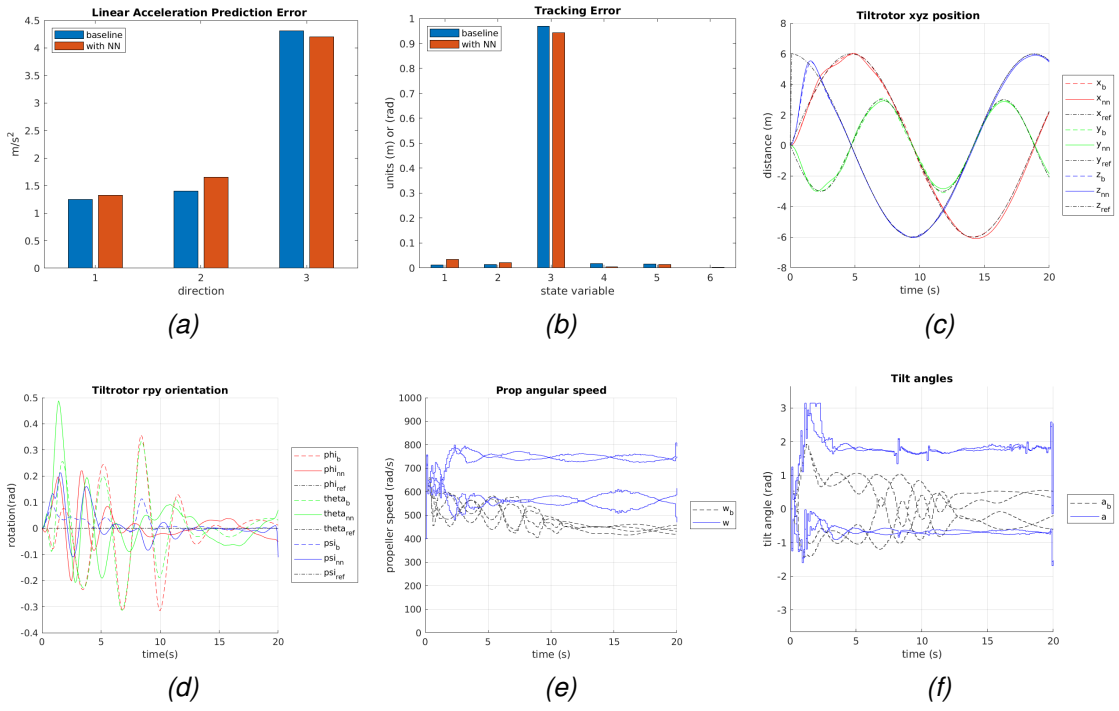


Figure 4.3: Performance of LBMPC controller on test trajectory for disturbance 1. Note the stark difference in inputs between baseline and LBMPC even though prediction error and tracking error do not change much.

Training trajectories

Additionally, to gain insight into how well the LBMPC controller performs on the training trajectories and to find out whether the NNs are indeed overfitting, the simulation procedure 3.6 was also run on trajectory 8 ('Complex' in table A.2, since it has the most variations in state variables). The results are plotted in figures 4.4 and 4.5. From these results, it is evident that there is a significant improvement with LBMPC for the training trajectory, but a large discrepancy between the results of this trajectory and the test trajectory. This further points towards overfitting and poor generalization. There are two ways to alleviate the overfitting: simpler NN with fewer parameters, or more training data. The former would result in marginal gains, but the latter could result in improvement of performance.

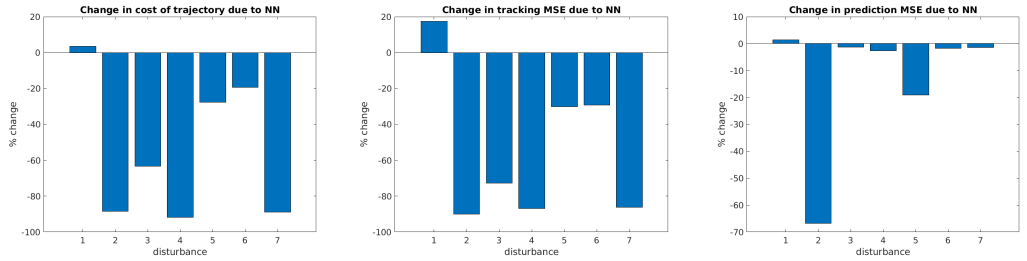
However, these results also show that the NN based LBMPC controller can significantly improve performance if trained well. As expected, the NN helps decrease prediction error for disturbance 2 the most since it is a simple, constant function. Next, disturbance 5 has significantly lower prediction error as well since it was an exaggerated, relatively simple ground effect-like function. The prediction error is marginally better for disturbances 3, 4, 6, and 7, and marginally worse for disturbance 1. This could be attributed to the relatively low magnitude of disturbance or deviations from baseline prediction for disturbances 3, 4, 6, and 7, and the lack of any deviation from baseline prediction for disturbance 1. The NN is marginally useful for the former disturbances, while it is detrimental for the latter, introducing noisy prediction. Additionally, when the NN is sufficiently trained on a trajectory, noisy inputs do not seem to cause failure, whereas for previously unseen trajectories noisy inputs seem to cause failure.

Therefore, it can be qualitatively stated that the NN oracle adds the most value when sufficiently trained, and when there is sufficient error in the baseline model prediction.

The plots of the other disturbances for both test trajectory and training trajectory are presented in Appendix B.

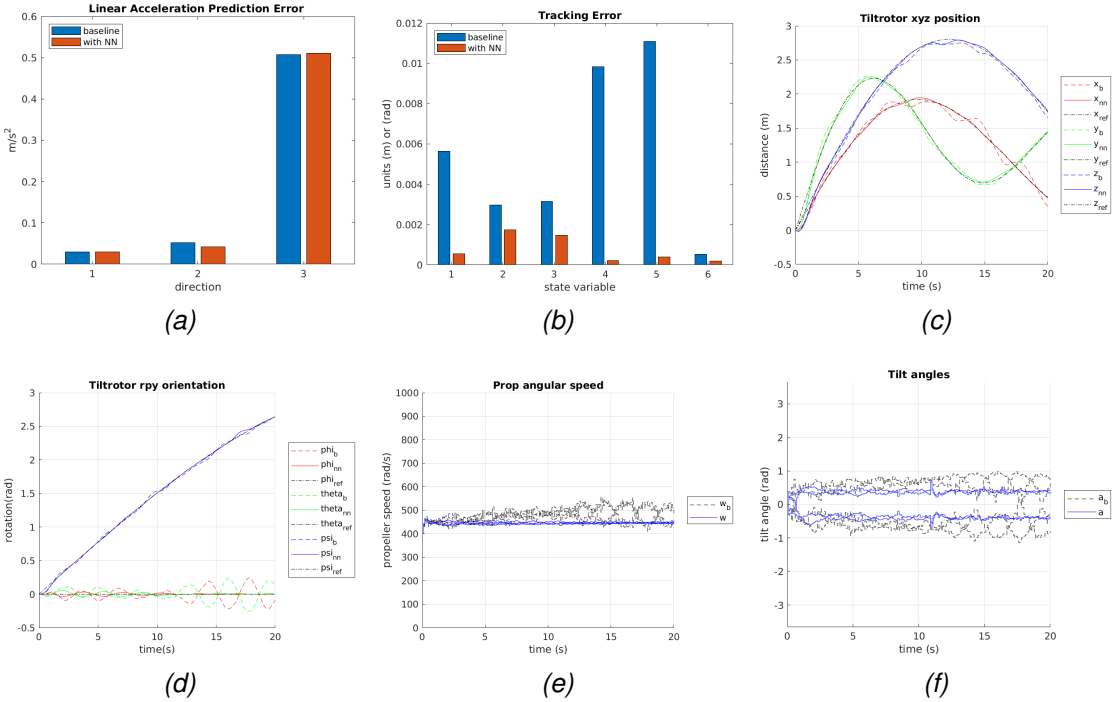
Failure modes

1. Longer prediction horizon (e.g 15 steps) leads to failure of the LBMPC controller. This can be attributed to poor conditioning and accumulation of prediction errors due to NN when the dynamics are sufficiently out of distribution relative to what the NN was trained on.
2. Using the predict function and using the direct NN output computation does not lead to the same output (differs in the 6th decimal), but this causes sufficient differences in the rollout of the trajectory that the latter fails for certain



(a) (b) (c)

Figure 4.4: The summary of results for the simulation tests on one of the training trajectories: 'Complex' trajectory. Clearly, the improvement in costs and tracking MSE is significant. The prediction MSE is also reflective of how well NN can predict the respective disturbances. In (c), disturbance 2, which is the constant disturbance has the greatest improvement in prediction error. Note that there is no instability due to noisy state estimates (disturbance 7). In fact, the performance with noisy state estimates (disturbance 7) is better than that with perfect state estimates (disturbance 6). It is clear that the greatest improvement is on disturbance 2, where the prediction MSE is also minimized the most.



(a) (b) (c)

(d) (e) (f)

Figure 4.5: Performance of LBMPC controller on training trajectory ('Complex'). Note that the baseline controller tracks poorly after 10 seconds, while the LBMPC controller tracks substantially better. This could be attributed to the fact that the NN can sufficiently offset prediction error due to erroneous model parameters, such that even in the presence of noisy state estimates, the controller can function well. However, observe the contrast with respect to the performance for disturbance 7 on the test trajectory. the LBMPC controller ultimately fails on the test trajectory but not on the trained trajectory. It is also important to consider the variation in orientation for this trajectory, while there is no variation in orientation in the test trajectory.

trajectories.

3. It was found that for the same NN, but only different initialization, the controller could fail. This indicates that the initialization procedure needs to be looked into more deeply. For this project, the initialization was repeated until the LBMPC controller would not fail. This needs further inspection and points to the vulnerability of the LBMPC controller to the initialization of the NN.

Chapter 5

Conclusions and Recommendations

This chapter presents conclusions based on the results presented in the previous chapter. Further, recommendations are made for the improvement of the research as well as avenues for future research.

5.1 Conclusions

This project has extended the prior literature and looked into using NNs to minimize plant-model mismatch by predicting the nominal model's prediction error. Having tested NN based LBMPC with different disturbances and found that while it does not definitively improve performance on a previously untrained trajectory, it can significantly improve the tilt-multirotor's performance on previously trained trajectories under external disturbances. It can be concluded that there is promise in using NN in conjunction with MPC controllers using the LBMPC formulation.

The following insights were developed through this research:

- Noisy inputs to the NN prediction model can cause failure if it cannot generalize well, but for previously trained trajectories, noisy inputs do not cause failure.
- The NN based predictive model tends to accumulate error over the prediction horizon, and hence it is important to use a suitably small prediction horizon.
- Small differences in prediction can result in significantly different control inputs, thereby affecting the cost accordingly. This makes the cost difference sensitive to both prediction error and cost weights.
- Erroneous prediction early on in the trajectory can compound costs and result in an overall higher cost even if the overall prediction error over trajectory is low.
- There are regions in state-space where the prediction by NN seems to be more erroneous than in other regions. This needs quantification so that the NN could be selectively used or its prediction could be weighed against this measure of

uncertainty in prediction when computing the control.

The results, while not definitively conclusive, do indicate the usefulness of NNs in minimizing plant model mismatch. However, this comes with a requirement for sufficient data gathering such that generalization is achieved. Results also show that it is possible that the NN oracle makes undisturbed flight perform worse while making certain disturbed flights perform better. The benefits vary depending upon the type of disturbance, how well the oracle is tuned to the trajectory and disturbance, and the ability of the oracle to generalize.

The limitations and failure modes discovered in this project also point to paths for further research.

5.2 Limitations

1. A primary limitation of this project is the contrived nature of the problem.
   1.1. The disturbances used are fabricated and do not correspond to real disturbances as is. The two ground effects are based on empirical studies and attempt to model realistic disturbances, but are limited in their representation of reality.
   1.2. The trajectories used for training data collection are simplistic since hand-designing realistic complex trajectories is extremely difficult.
2. The restriction to simulation resulted in a restriction on data collection and the decision to consider only current state and control input as inputs for NN might have resulted in lower performance than would be possible when using the prior history of state and control as well.
3. The control frequency of 10 Hz used for this project would not be very useful for practical systems since it is relatively slow compared to practical disturbance and noise effects, requiring control at 50 Hz or higher frequencies.
4. The cost function used is the quadratic cost function, whereas a more tailored cost function could be used that might improve the LBMPC performance.
5. NN hyperparameter tuning and more sophisticated architectures could be used to improve performance. [16] introduce novel initialization and sparse activation that could prove beneficial for oracle performance. This could also address the vulnerability of the LBMPC controller to NN initialization.
6. The NNs were trained only once prior to deployment. They could be fine-tuned iteratively as more data was collected using LBMPC.

5.3 Recommendations

The insights generated through this project lead to several recommendations regarding how to improve this research were it to be repeated, as well as avenues for future research.

The first recommendation to improve the presented research would be to perform data collection using a real system. This would eliminate the contrived nature and any limitations due to hand-crafted data collection trajectories and artificial plant-model mismatch. The second recommendation would be to collect much more data for training since the data are collected from the real system. The data could be super-sampled during simulation since the plant's state evolution can be accessed at a higher temporal resolution than the control intervals. The third recommendation would be to perform more extensive NN architecture and hyperparameter search.

Considerations for implementation on a practical system

1. The collection of data is a major problem that needs to be solved. A nominal MPC controller could be used for control, but the trajectories will still need to be determined such that the system dynamics are adequately represented in the training data for the NN.
2. The current 10Hz control frequency is not very practical and will most likely need to be increased for implementation on a physical system. More efficient optimization solvers would need to be used for faster control. Additionally, the entire control algorithm needs to be ported to an embedded system ultimately.
3. The current approach assumed perfect full state knowledge at each time step. When a noisy version of state estimate was tested on the tested trajectory, the controller failed, but the LBMPC controller performed well on a previously seen training trajectory. This means the NN needs to be well trained and should be able to generalize to the trajectory being flown. A high accuracy state estimation system including systems such as ViCon might need to be used for initial testing on physical systems.
4. The NN based LBMPC approach could be tested in a phased manner, starting with a high fidelity physics simulation with the embedded software in the loop. For instance, ROS Gazebo could be used for simulation, and MATLAB could be used to generate the embedded code for the target hardware, which could be emulated. Next, the setup could be simulated with the embedded hardware in the loop. Finally, actual flight tests could be conducted.

The first recommendation for future research is the inclusion of past state and input data as inputs for NN prediction. This could lead to better prediction performance and would need modification of the MPC controller to accommodate such a prediction model.

The second recommendation is to use a measure of uncertainty in NN prediction to determine whether to and how much to weigh the predictions. The error in NN predictions could also be collected and used for further training iteratively. The control could then be switched online between nominal MPC and LBMPC based on the prediction uncertainty of the NN. In doing so, it is important to consider the possibility of large deviations in control due to minor differences in prediction as observed in the results.

A more interesting direction for research would be to eliminate the need for online optimization with MPC and directly use a NN policy. However, the primary challenge here would be to perform better than LBMPC in tracking since this would effectively be imitation learning. Another interesting avenue for research would be to speed up MPC computation by learning the cost function prediction like in Value iteration.

Overall, using self-supervised learning, particularly with NNs, shows promise in improving optimal control performance for nonlinear systems with unknown or uncertain dynamics, even under external disturbances. This project has investigated the LBMPC approach using NNs and shown that it could indeed be beneficial for practical systems if trained and implemented well.

Chapter 6

Acknowledgement

I sincerely thank my supervisor dr. ir. Abeje Mersha for having encouraged, consistently guided, and supported me in this project. I thank ATLAS and the University of Twente for providing me with the opportunity to work on such an interesting topic for my Bachelor Thesis. I thank Saxion's Mechatronics department for supporting and providing advice regarding the project. I thank my supervisors dr. ir. Geert Folkertsma and dr. ir. Fokko Jan Dijksterhuis for their guidance and advice. I thank Gabriel and Ajinkya for their encouragement and providing their feedback on the report. I thank my friends and parents for their constant support throughout the project.

This project is a combination of three fields: Machine Learning, Control Engineering and Robotics. The intersection of these fields is extremely interesting to me personally, and shows great promise for the future of robotics. Although there is a fair amount of literature on the intersection of these fields, recent advances in computation, data collection, sensors, and machine learning show that there is a lot yet to be achieved.

The code repository for the project, along with the training data, NN parameters and simulation experiment results data are available at: `https://github.com/akarshgopal/bachelor_thesis`

Bibliography

[1] F. SA, “Flyability — drones for indoor inspection and confined space,” 2020. [Online]. Available: https://www.flyability.com/

[2] “Harrisaerial,” 2020. [Online]. Available: https://www.harrisaerial.com/remote-sensing/

[3] “Skygauge,” 2020. [Online]. Available: https://skygauge.co/

[4] “Voliro,” 2020. [Online]. Available: https://www.voliro.com/

[5] D. Invernizzi, M. Giurato, P. Gattazzo, and M. Lovera, “Full pose tracking for a tilt-arm quadrotor uav,” 08 2018, pp. 159–164.

[6] M. Kamel, S. Verling, O. Elkhatib, C. Sprecher, P. Wulkop, Z. Taylor, R. Siegwart, and I. Gilitschenski, “The voliro omniorientational hexacopter: An agile and maneuverable tiltable-rotor aerial vehicle,” *IEEE Robotics Automation Magazine*, vol. 25, no. 4, pp. 34–44, 2018.

[7] M. Allenspach, K. Bodie, M. Brunner, L. Rinsoz, Z. Taylor, M. Kamel, R. Siegwart, and J. Nieto, “Design and optimal control of a tiltrotor micro aerial vehicle for efficient omnidirectional flight,” *arXiv preprint arXiv:2003.09512*, 2020.

[8] R. Falconi and C. Melchiorri, “Dynamic model and control of an over-actuated quadrotor uav,” *IFAC Proceedings Volumes*, vol. 45, no. 22, pp. 192–197, 2012.

[9] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart, “Fast nonlinear model predictive control for multicopter attitude tracking on so(3),” in *2015 IEEE Conference on Control Applications (CCA)*, 2015, pp. 1160–1166.

[10] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, “Provably safe and robust learning-based model predictive control,” *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[11] A. Aswani, P. Bouffard, and C. Tomlin, “Extensions of learning-based model predictive control for real-time application to a quadrotor helicopter,” in *2012 American Control Conference (ACC)*, 2012, pp. 4661–4666.

[12] F. Berkenkamp and A. P. Schoellig, “Safe and robust learning control with gaussian processes,” in *2015 European Control Conference (ECC)*. IEEE, 2015, pp. 2496–2501.

[13] E. Kaiser, J. N. Kutz, and S. L. Brunton, “Sparse identification of nonlinear dynamics for model predictive control in the low-data limit,” *Proceedings of the Royal Society A*, vol. 474, no. 2219, p. 20180335, 2018.

[14] P. Abbeel, A. Coates, and A. Y. Ng, “Autonomous helicopter aerobatics through apprenticeship learning,” *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[15] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister, “Low-level control of a quadrotor with deep model-based reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[16] A. Punjani and P. Abbeel, “Deep learning helicopter dynamics models,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3223–3230.

[17] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, “Learning quadrotor dynamics using neural network for flight control,” in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 4653–4660.

[18] “Propt: A matlab optimal control solver,” 2020. [Online]. Available: http://tomdyn.com/

[19] “Forces-pro: Real time decision making software,” 2020. [Online]. Available: https://www.embotech.com/forces-pro/

[20] “Acado: Toolkit for automatic control and dynamic optimization,” 2020. [Online]. Available: https://acado.github.io/

[21] “Casadi: Open source nonlinear optimization and automatic differentiation,” 2020. [Online]. Available: https://web.casadi.org/

[22] “Mpc scheme basic.” [Online]. Available: https://en.wikipedia.org/wiki/Model_predictive_control#/media/File:MPC_scheme_basic.svg

[23] B. Houska, H. J. Ferreau, and M. Diehl, “An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range,” *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011.

[24] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001.

[25] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] P. Wei, S. N. Chan, S. Lee, and Z. Kong, "Mitigating ground effect on mini quadcopters with model reference adaptive control," *International Journal of Intelligent Robotics and Applications*, vol. 3, no. 3, pp. 283–297, 2019.

[28] P. Sanchez-Cuevas, G. Heredia, and A. Ollero, "Characterization of the aerodynamic ground effect and its influence in multirotor control," *International Journal of Aerospace Engineering*, vol. 2017, 2017.

[29] "Propeller datasheet tmotor," 2020. [Online]. Available: http://store-en.tmotor.com/goods.php?id=347

[30] "Bldc motor datasheet," 2020. [Online]. Available: http://uav-en.tmotor.com/html/uav/html/2019/AT_0429/246.html

[31] "Hitec servo datasheet," 2020. [Online]. Available: https://hitecrcd.com/products/servos/digital/micro-mini-wing/d85mg/product

[32] "nlmpc: Nonlinear mpc on matlab," 2020. [Online]. Available: https://www.mathworks.com/help/mpc/ref/nlmpc.html

[33] "Matlab deep learning toolbox," 2020. [Online]. Available: https://www.mathworks.com/products/deep-learning.html

Appendix A

Table A.1: NN Hyperparameters

Hyperparameter	Value
Training Epochs T_s	400
Initial Learning Rate	0.01
Batch Size	32
Normalization	z-score
Weights Initialisation	glorot
Bias Initialisation	0
Validation frequency	every 30 epochs
Shuffling	once every epoch

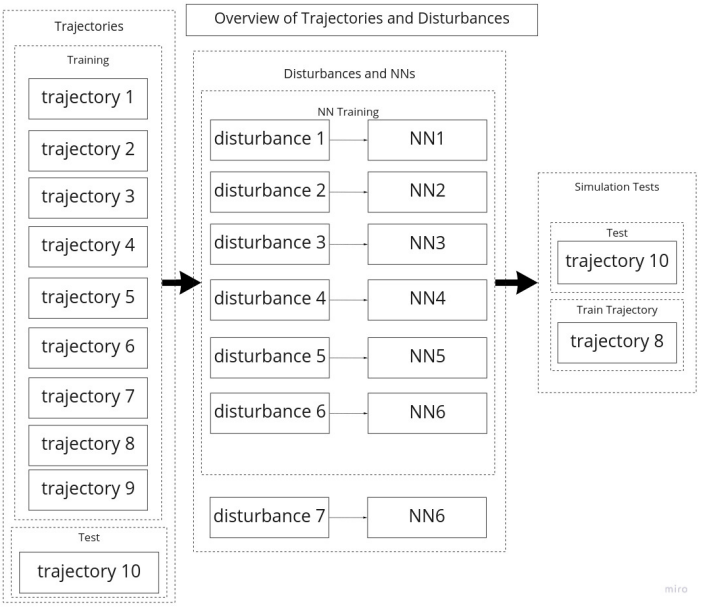
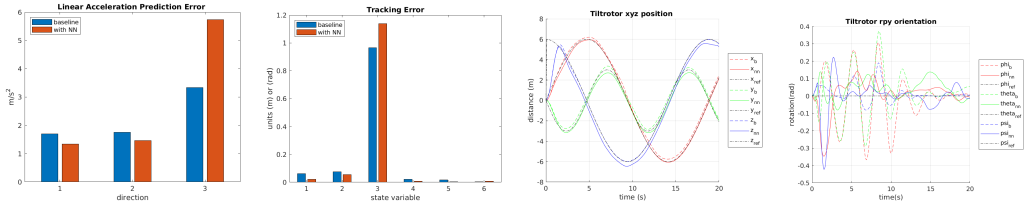


Figure A.1: Overview of Trajectories, Disturbances, and NNs.

Table A.2: Trajectories

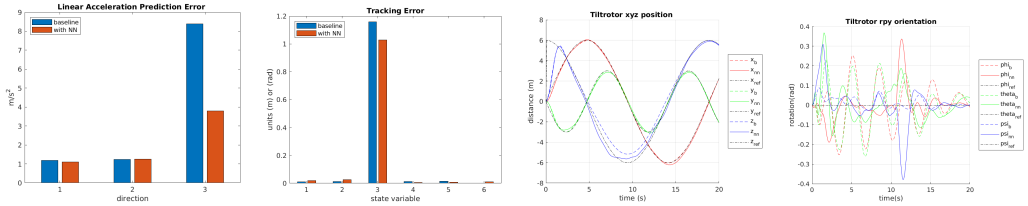
Trajectory	**Duration (s)**	**Generating function**
1. Stationary	50	$\mathbf{x}_{ref} = 0$
2. Sinx	200	$x(t) = 6sin(0.5t)$
3. Siny	200	$y(t) = 6sin(0.5t)$
4. Sinz	200	$z(t) = 6sin(0.5t)$
5. Rotx	200	$\phi(t) = \pi sin(\frac{t}{20})$
6. Rotz	200	$\psi(t) = \pi sin(\frac{t}{20})$
7. Data	200	$x(t) = sin(\frac{t}{10}) + sin(\frac{t}{8}) + sin(\frac{t}{5})$ $y(t) = sin(\frac{t}{15}) + sin(\frac{t}{6}) + sin(\frac{t}{3})$ $z(t) = sin(\frac{t}{12}) + sin(\frac{t}{9}) + sin(\frac{t}{6})$
8. Complex	1000	$x(t) = sin(\frac{t}{10}) + sin(\frac{t}{8})sin(\frac{t}{1000}) + sin(\frac{t}{5}) + sin(\frac{t}{100}) + 3sin(\frac{t}{500})$ $y(t) = sin(\frac{t}{15})cos(\frac{t}{1000}) + sin(\frac{t}{6}) + sin(\frac{t}{3}) + sin(\frac{t}{120}) + 3sin(\frac{t}{500})$ $z(t) = sin(\frac{t}{12}) + sin(\frac{t}{9}) + sin(\frac{t}{6}) + sin(\frac{t}{150})$ $\psi(t) = \pi sin(\frac{t}{20})$
9. Helix	200	$x(t) = 2cos(0.943t)$ $y(t) = 2sin(0.943t)$ $z(t) = 0.2t$
10. Loop (Test)	20	$x(t) = 6sin(\frac{t}{3})$ $y(t) = -6sin(\frac{t}{3})cos(\frac{t}{3})$ $z(t) = 6cos(\frac{t}{3})$

Appendix B: Result Plots



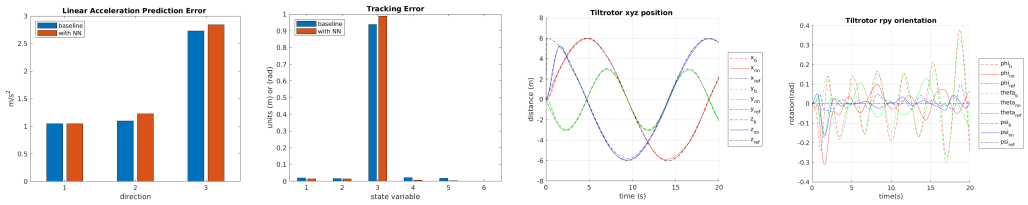
(a) (b) (c) (d)

Figure B.1: Controller performance on test trajectory 10 for disturbance 2



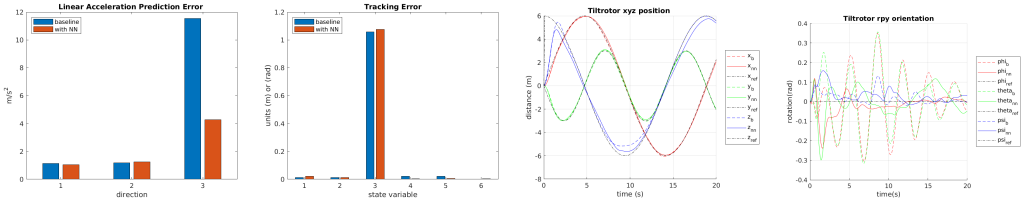
(a) (b) (c) (d)

Figure B.2: Controller performance on test trajectory 10 for disturbance 3



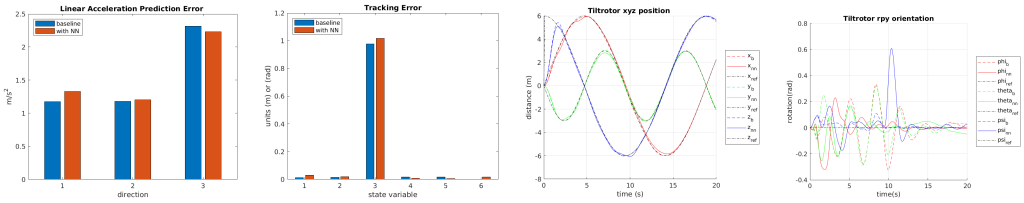
(a) (b) (c) (d)

Figure B.3: Controller performance on test trajectory 10 for disturbance 4



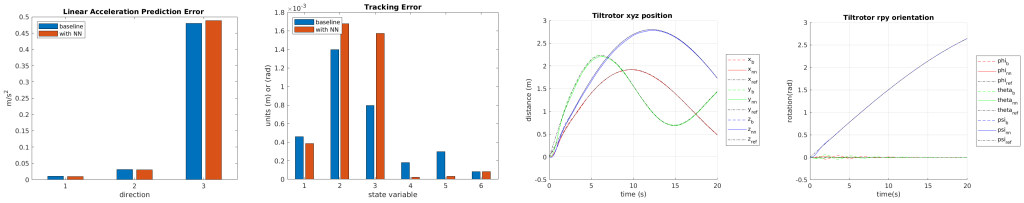
(a) (b) (c) (d)

Figure B.4: Controller performance on test trajectory 10 for disturbance 5



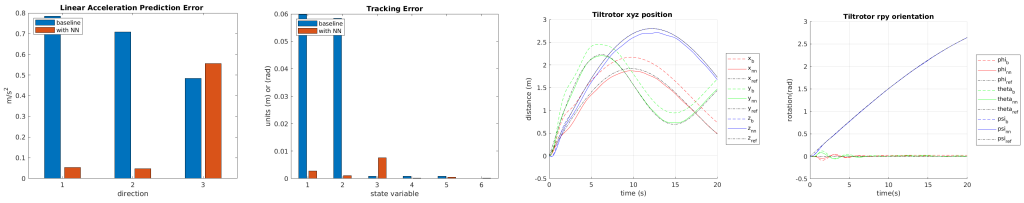
(a) (b) (c) (d)

Figure B.5: Controller performance on test trajectory 10 for disturbance 6



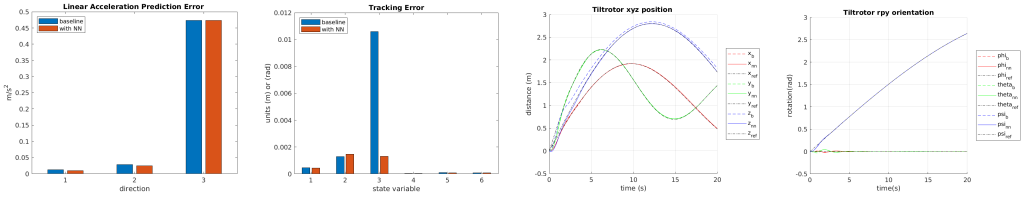
(a) (b) (c) (d)

Figure B.6: Controller performance on training trajectory 8 for disturbance 1



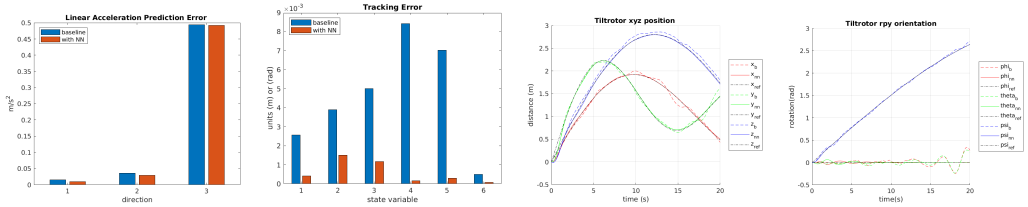
(a) (b) (c) (d)

Figure B.7: Controller performance on training trajectory 8 for disturbance 2



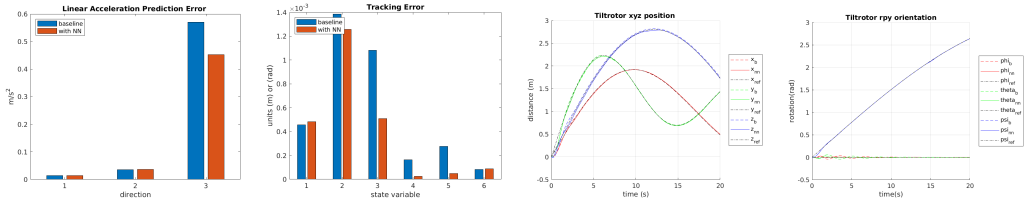
(a) (b) (c) (d)

Figure B.8: Controller performance on training trajectory 8 for disturbance 3



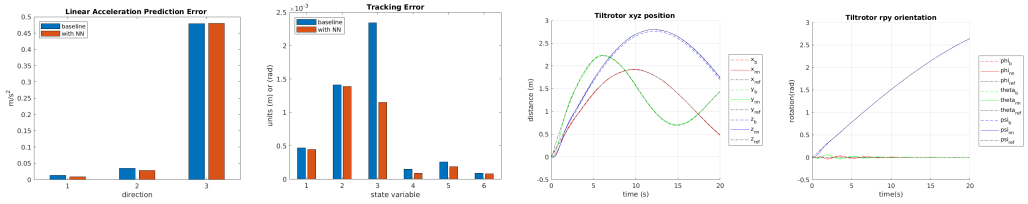
(a) (b) (c) (d)

Figure B.9: Controller performance on training trajectory 8 for disturbance 4



(a) (b) (c) (d)

Figure B.10: Controller performance on training trajectory 8 for disturbance 5



(a) (b) (c) (d)

Figure B.11: Controller performance on training trajectory 8 for disturbance 6