**This document is downloaded from DR-NTU (https://dr.ntu.edu.sg)
Nanyang Technological University, Singapore.**

Optimization-based learning control of aerial robots operating in uncertain environments

Mehndiratta, Mohit

2020

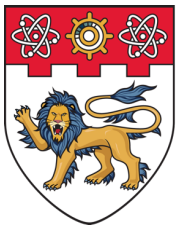
Mehndiratta, M. (2020). Optimization-based learning control of aerial robots operating in uncertain environments. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/144162

https://doi.org/10.32657/10356/144162

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 07 Nov 2021 21:33:09 SGT



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Optimization-based Learning Control of Aerial Robots Operating in Uncertain Environments

Mohit Mehndiratta

School of Mechanical and Aerospace Engineering

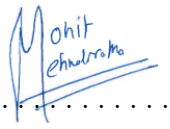
A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2020

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

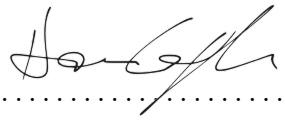
17 March 2020
.....
Date


.....
Mohit Mehndiratta

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

17 March 2020
.....
Date


.....
Prof. Domenico Campolo

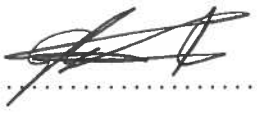
Co-Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

17 March 2020

.....

Date



.....

Prof. Erdal Kayacan

Authorship Attribution Statement

This thesis contains material from 5 paper(s) published/under-review in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

First case study in Chapter 3 is published as Mehndiratta, M. & Kayacan, E. Auton Robot (2019) 43: 2209. https://doi.org/10.1007/s10514-019-09875-y. (Quartile-2)

The contributions of the co-authors are as follows:

- I prepared the manuscript drafts. I designed the study, fabricated the robotic platform, and implemented the control algorithm. I performed all the flight test at the School of Mechanical and Aerospace Engineering and also analyzed the test results.
- Dr. Erdal Kayacan gave guidance regarding the test results and helped with the revision of the manuscript.

Second case study in Chapter 3 is published as M. Mehndiratta and E. Kayacan, "Online Learning-based Receding Horizon Control of Tilt-rotor Tricopter: A Cascade Implementation," 2018 Annual American Control Conference (ACC), Milwaukee, WI, 2018, pp. 6378-6383. doi: 10.23919/ACC.2018.8430814.

The contributions of the co-author are as follows:

- I prepared the manuscript drafts. I designed the study, implemented the control algorithm, and performed all the simulations at the School of Mechanical and Aerospace Engineering. I also interpreted the results.
- Dr. Erdal Kayacan helped with the revision of the manuscript.

Chapter 4 is published as Mohit Mehndiratta, Erkan Kayacan, Mahmut Reyhanoglu and Erdal Kayacan. "Robust Tracking Control of Aerial Robots Via a Simple Learning Strategy-Based Feedback Linearization," in IEEE Access, vol. 8, pp. 1653-1669, 2020, doi: 10.1109/ACCESS.2019.2962512. (Quartile-1)

The contributions of the co-authors are as follows:

- I prepared the manuscript drafts. I co-deduced the controller formulations and global minimum analysis along with Dr. Erkan Kayacan. I fabricated the aerial robot, implemented the proposed framework and performed all the flight test at the School of Mechanical and Aerospace Engineering. I also interpreted the experimental results.

- Dr. Erkan Kayacan provided the initial learning idea and wrote the stability proof along with Dr. Mahmut Reyhanoglu. He evaluated the proposed framework over a simulated example.
- Dr. Mahmut Reyhanoglu provided guidance regarding the theoretical concepts and edited the manuscript.
- Dr. Erdal Kayacan helped with the revision of the manuscript.

Chapter 5 is published as Mohit Mehndiratta and Erdal Kayacan. "Gaussian Process-based Learning Control of Aerial Robots for Precise Visualization of Geological Outcrops," 2020 European Control Conference (ECC), Saint Petersburg, Russia, 2020, pp. 10-16.

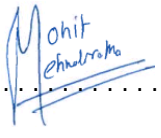
The contributions of the co-authors are as follows:

- I prepared the manuscript drafts. I designed the study, customized the robot and implemented the control algorithm. I performed the simulation tests at the School of Mechanical and Aerospace Engineering. With the help of a colleague Mr. Theo Morales, I conducted the real-world tests in Rødal chalk quarry, Aalborg, Denmark. I also interpreted the simulation and real flight results.
- Dr. Erdal Kayacan provided the initial project direction and helped with the revision of the manuscript.

Chapter 6 is submitted as Mohit Mehndiratta, Efe Camci and Erdal Kayacan. "Can Deep Models Help a Robot to Tune Its Controller? A Step Closer to Self-tuning Model Predictive Controller." In IEEE/ASME Transactions on Mechatronics.

The contributions of the co-authors are as follows:

- I proposed the idea for automated tuning and prepared the manuscript drafts. I co-designed the study with Mr. Efe Camci. I collected the flight data for the deep neural network model of the aerial robot and validated its performance. I performed the benchmark simulation tuning in Aarhus University, Denmark. I assembled the robot and conducted the real flight tuning as well as the trajectory tracking results in the Deep-Tech Lab of Aarhus University. I also interpreted the obtained results.
- Dr. Efe Camci provided the idea for active exploration strategy and designed the deep neural network model of the aerial robot. He helped with the interpretation of the benchmark study. He also conducted the user-based study in the School of Mechanical and Aerospace Engineering.
- Dr. Erdal Kayacan helped with the revision of the manuscript.

17 March 2020 .....  .....

Date Mohit Mehndiratta

Acknowledgements

This thesis would not be possible without the support of many people. I wish to thank all those who have inspired and motivated me throughout my Ph.D. journey.

First and foremost, I wish to express my utmost gratitude to my supervisor, Prof. Erdal Kayacan, for his invaluable advice, continuous support, and patience during my study. It has been a delightful experience to work under his supervision. His immense research experience has always helped me get through difficult times. I would also like to thanks my supervisor, Prof. Domenico Campolo, for accepting me as his student in the last year. His support and reciprocation towards thesis related issues have been overwhelming. This thesis is the result of many successful collaborations and discussions with Prof. Erkan Kayacan, Prof. Mahmut Reyhanoglu, Prof. Anna Prach, Prof. Girish Chowdhary, and Prof. Tufan Kumbasar. Many thanks to my colleagues, Nursultan Imanberdiyev, Efe Camci, Andriy Sarabakha, Basaran Bahadir Kocer, Yunus Govdeli, Karanjot Singh, and many more for the insightful discussions. Their moral support, especially after unsuccessful experiments, has been indispensable. I wish to express my gratitude to Siddharth Patel and Dogan Kircali for all the technical help that I needed during my experiments. I would also like to thanks my students, Lee Ying Jun Wilson, Ruddhi Kaustubh Gokhale, and Kenny Brian, for their help with the designs of my experimental platforms. My time at NTU has been a life-changing experience for me, and thus, I am sincerely grateful to all those people who contributed to it.

I also wish to express my profound gratitude to my family for their unconditional love and continuous support. To my grandmother, thank you for all the blessings. To my parents, without your inspiration and prayers, I would not have achieved this great goal in my life, and I will be forever grateful. To my sister, Charu Mehndiratta, thank you for being there whenever I needed someone by my side. To my girlfriend, Kavya Varma, thank you for the encouragement throughout my study.

Singapore *Mohit*
August 2020

"It is possible to fly without motors, But not without knowledge and skill."

—Wright, Wilbur

To my dear family

Contents

Abstract

Places that were hard to reach are now well accessible to the world with the help of aerial robots. As one of the biggest inventions of mankind in robotics, these robots place no risk on human lives because they are unmanned and remotely/autonomously operated in hostile situations. Together, these reasons make them the most promising candidate in numerous applications. Howbeit, their coupled and significantly nonlinear dynamics accompanied by open-loop instabilities lead to a complicated control problem. Although conventional control approaches such as proportional-integral-derivative (PID) and linear-quadratic-regulator (LQR), have been widely adopted, the underlying linearization leads to suboptimal performance during agile operations. Besides, there are environment-specific difficulties, like external disturbances during offshore operations, that result in an uncertain system model. Since the performance of model-based controllers is critically linked to the model accuracy, modeling uncertainties may significantly degrade their performance to the extent of instability. Therefore, rather than utilizing a sophisticated robot which is trained – and tuned – for a scenario in a specific environment perfectly, most people are interested in seeing them operating in unexplored conditions. In that vein, an aerial robot must learn from its own experiences and interactions with the environment for daily operations in real application scenarios. Moreover, realtime implementations of the control algorithms necessitate a tuning process that is arduous yet dangerous when performed directly on the real robot.

Taking inspiration and identifying an opportunity in these issues, this thesis throws light on the development of various learning algorithms to facilitate precise tracking control of multirotor aerial robots in uncertain environmental conditions. Firstly, to cater to the nonlinear dynamics, it implements two control algorithms, namely, the nonlinear model predictive controller (NMPC) and the feedback linearization control (FLC) method. Both the control approaches explicitly accommodate the nonlinear dynamics rather than linearizing the system. Their overall efficacy is demonstrated for the position and attitude tracking problems of the aerial robots.

Secondly, to accommodate the limited processing power that is available onboard aerial robots, this thesis employs fast solutions methodologies. Thanks to the efficient C++ scripts and the direct multiple shooting method along with the special

realtime iteration scheme adopted in automatic-control-and-dynamic-optimization (ACADO) toolkit; successful onboard implementation of the control algorithms is achieved for all the real-world tests. What is more, in the case of the NMPC-NMHE framework, a complete onboard implementation is realized on a low-cost embedded processor, named Raspberry Pi 3.

Thirdly, to tackle the uncertainties in the system model, this thesis proposes a few learning-based control approaches that are broadly categorized as: instantaneous learning control (InLC) and iterative learning control (ILC). In essence, the InLC technique utilizes an estimator to learn the model parameters, whereas the ILC scheme identifies the uncertain dynamics based on the experience from system repetitions. Besides, the learned system model is subsequently updated within the controller definition in both the approaches. In terms of the InLC scheme, two control frameworks are developed. The first incorporates a nonlinear moving horizon estimator (NMHE) to estimate the time-varying model parameters, thus making NMPC adaptive to the changing working conditions. The second framework constitutes a simple learning (SL) strategy to cater to the limitations of the traditional FLC method by updating controller gains and disturbance estimate within the feedback control law. In the ILC scheme, on the other hand, a Gaussian process (GP)-based regression technique models the disturbance forces that are encountered during the offshore visualization operation. Several simulations and real-world tests manifest that both InLC and ILC schemes have compelling abilities to substantially reduce the tracking error over their conventional counterparts throughout the operation.

Lastly, to circumvent the tedious tuning process, an active exploration approach is proposed to obtain the NMPC's weights. The auto-tuning framework extends the basic trial-and-error method to intelligently tune the weight sets. In essence, it benefits from the retrospective knowledge gained over previous trials, and thus, expedites the tuning procedure. Moreover, the safety of the robot is ensured by employing a deep neural network-based robot model. What is more, a seamless sim-to-real transition is exhibited via the direct deployment of the weight sets from simulation tuning for the real-world trajectory tracking application.

List of Figures

List of Tables

Symbols and Acronyms

Symbols

General

t	time (continuous) variable
$a \in \mathbb{R}$	scalar a with real value
$a \in \mathbb{R}^{+}$	scalar a with semi-positive real value, i.e. including zero
$a \in \mathbb{R}^{++}$	scalar a with positive real value, i.e. excluding zero
$\mathbf{a} \in \mathbb{R}^{n}$	vector (column) $\mathbf{a}$ of dimension n with real entries
$\mathbf{A} \in \mathbb{R}^{n \times m}$	matrix $\mathbf{A}$ of dimension n-by-m with real entries
$\mathbf{0}_n$	zero vector of n-dimensions
$\mathbf{a}_i$	i^{th} element of vector $\mathbf{a}$
$\mathbf{A}_i$	i^{th} column of matrix $\mathbf{A}$
$\text{diag}(\mathbf{a})$	diagonal matrix with elements $\mathbf{a}$
$\lvert a \rvert$	absolute value of a
$\lVert \mathbf{a} \rVert_2$	l_2 norm (Euclidean norm) of vector $\mathbf{a}$
$\lVert \mathbf{a} \rVert_{\mathbf{A}}^2$	square of weighted 2-norm of the vector $\mathbf{a}$, given as: $\mathbf{a}^T\mathbf{A}\mathbf{a}$
$\nabla f(\,\cdot\,)$	gradient of function f
$\mathbf{A} \succeq 0$	positive semi-definite matrix $\mathbf{A}$
$\mathbf{A} \succ 0$	positive definite matrix $\mathbf{A}$
$\mathbb{E}(\,\cdot\,)$	expected value function
$\text{cov}(\,\cdot\,)$	covariance function
$p(\,\cdot\,)$	probability density function
$\mathcal{O}(\,\cdot\,)$	order-of symbol

Convex Optimization

$\mathbf{x}$	decision variable
$f(\,\cdot\,)$	objective function
$h(\,\cdot\,)$	inequality constraints

$g(\cdot)$	equality constraints
$\mathtt{S}$	feasible set
$\mathtt{I}$	inequality constraints set
$\mathtt{E}$	equality constraints set
$\mathcal{L}(\cdot)$	Lagrange function
λ and ν	Lagrange dual variables
$\mathfrak{g}(\cdot)$	Lagrange dual function
$\mathbf{H}$	Hessian matrix
$\boldsymbol{\theta}$	parameter vector
α	positive step size
$\zeta(\cdot)$	residual function

Realtime Optimal Control

$L(\cdot)$	Lagrange term or stage cost
$E(\cdot)$	Mayer term or terminal cost
$\mathbf{x}(t)$	continuous-time state vector
$\mathbf{u}(t)$	continuous-time control vector
$\mathbf{h}(\cdot)$	path constraints
$\mathbf{r}(\cdot)$	terminal constraints
T	time horizon length
N	number of shooting intervals
T_s	grid length
$\mathbf{s}_k$	discrete-time state vector
$\mathbf{q}_k$	discrete-time control vector
$l(\cdot)$	discrete-time stage cost
X	discreatized state trajectory
U	discreatized control trajectory

Multirotor Modeling

$\mathcal{F}_E$	Earth-fixed (global) frame
$\mathcal{F}_B$	body frame
x, y, and z	respective positions in global x, y, and z axes
u, v, and w	respective velocities in body x, y, and z axes
p, q, and r	respective rotational velocities about body x, y, and z axes
ϕ, θ, and ψ	respective roll, pitch and yaw rotational attitude about global x, y, and z axes
F_x, F_y, and F_z	respective external forces along three body axes

τ_x, τ_y, and τ_z	respective external moments about three body axes
m	mass of the aerial robot
l	arm length of the aerial robot
I	moment of inertia about the corresponding body axis
g	acceleration due to gravity, i.e., $g = 9.81$ m/s^2
K_F	propeller's force coefficient
K_τ	propeller's drag-moment coefficient
Ω_i	rotational speed of i^{th} propeller
μ	back-rotor tilting angle in tricopter
$\mathbf{x}_k$	discrete state vector at k^{th} instant
$\mathbf{u}_k$	discrete control vector at k^{th} instant
$\mathbf{z}_k$	discrete output vector at k^{th} instant
$\boldsymbol{\nu}_k$	measurement noise vector at k^{th} instant
$\mathbf{p}$	parameter vector
$\mathbf{f}_{\mathtt{d}}(\cdot)$	discrete nominal model function
$\mathbf{g}_{\mathtt{d}}(\cdot)$	discrete unknown/uncertain function
F^{dist}	disturbance force along the corresponding body axis
z_{error}	absolute tracking error along z-axis

Nonlinear Model Predictive Controller

N_c	prediction horizon length
$\hat{\mathbf{x}}$	current estimate of state vector
$\mathbf{x}_k^{\text{r}}$	reference trajectory for state vector
$\mathbf{u}_k^{\text{r}}$	reference trajectory for control vector
$\mathbf{x}_{k,\min}$ and $\mathbf{x}_{k,\max}$	respective minimum and maximum state values
$\mathbf{u}_{k,\min}$ and $\mathbf{u}_{k,\max}$	respective minimum and maximum control values
$\mathbf{W}$	symmetric positive definite weighting matrix

Nonlinear Moving Horizon Estimator

M	estimation horizon length
$\hat{\mathbf{p}}$	current estimate of parameter vector
$\bar{\mathbf{x}}$	arrival cost data for state vector
$\bar{\mathbf{p}}$	arrival cost data for parameter vector
$\boldsymbol{\omega}_k$	process noise vector at k^{th} instant
$\mathbf{p}_{k,\min}$ and $\mathbf{p}_{k,\max}$	respective minimum and maximum parameter values
$\boldsymbol{\Sigma}$	noise covariance matrix

$\mathbf{P}_{\mathtt{L}}$ and $\mathbf{W}$	symmetric positive definite weighting matrices

Feedback Linearization Control

$\mathtt{u}$	scalar control variable
$\Delta(\,\cdot\,)$	modeling uncertainties
$\omega(t)$	time-varying external disturbance
$\mathbf{r}(t)$	reference trajectory vector
$\mathtt{u}_{\mathtt{b}}$	feedback control action
$\mathtt{u}_{\mathtt{f}}$	feedforward control action
$\mathbf{k}$	positive control gains vector
$\mathbf{e}$	tracking error vector
$d(t)$	lumped disturbance parameter

Simple Learning Strategy

$\hat{d}$	estimated disturbance
$c(\,\cdot\,)$	desired closed-loop error dynamics
$C(\,\cdot\,)$	cost function
α_i	positive learning rate for i^{th} controller gain
$\alpha_{\hat{d}}$	positive learning rate for disturbance estimate

Gaussian Process

$\mathbf{g} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	multivariate Gaussian distribution $\mathbf{g}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$g \sim \mathcal{GP}(m, k)$	scalar Gaussian process $g(\,\cdot\,)$ with mean function $m(\,\cdot\,)$ and covariance function $k(\,\cdot\,)$
$\mathbf{a}$	index vector of dimension D
$\widetilde{g}(\,\cdot\,)$	measure value of GP $g(\,\cdot\,)$
$\mathcal{D}$	input-output (combined) dataset of dimension N
$\boldsymbol{\Lambda}$	length scales vector
σ_g	process variance
σ_ν	measurement noise variance
$\boldsymbol{\theta}$	hyperparameter vector
GP^{dist}	designed GP along the corresponding body axis
$\boldsymbol{\xi}$	model mismatch vector

Automated Tuning

$\mathbb{W}$	weight set search space
e	position error
$\dot{e}$	position error derivative
j	jerk, acceleration derivative
e_{ss}	steady-state error
t_s	settling time
c	weight set assessing criteria
G_c	sub-grade of the corresponding criteria
G	total grade value
$\mathbf{W}^*$	maximum-graded (best) weight set
N_w	weight sets with positive grade in lookup table
ϵ	greed parameter for ϵ-greedy policy
λ	exploration neighborhood for weight set

Acronyms

BFGS	Broyden-Fletcher-Goldfarb-Shanno
BLr	Back-left Rotor
BRr	Back-right Rotor
BVP	Boundary Value Problem
DNN	Deep Neural Network
DO	Disturbance Observer
EKF	Extended Kalman Filter
FLC	Feedback Linearization Control
FLr	Front-left Rotor
FRr	Front-right Rotor
GGN	Generalized Gauss-Newton
GP	Gaussian Process
H-NMPC	High-level Nonlinear Model Predictive Controller
ILC	Iterative Learning Control
InLC	Instantaneous Learning Control
IVP	Initial Value Problem
LP	Linear Program
LQR	Linear-Quadratic-Regulator
LSTM	Long-Short Term Memory
L-NMPC	Low-level Nonlinear Model Predictive Controller
MOI	Moment of Inertia
NCP	Nominal Control Performance
NLP	Nonlinear Program
NMPC	Nonlinear Model Predictive Controller
NMHE	Nonlinear Moving Horizon Estimator
NN	Neural Network
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
QCQP	Quadratically Constrained Quadratic Program
PID	Proportional-Integral-Derivative
PSO	Particle Swarm Optimization
pLP	Parametric Linear Program
pQP	Parametric Quadratic Program
QP	Quadratic Program
QPr	Quadratic Programming
RL	Reinforcement Learning

ROS	Robot Operating System
RTI	Realtime Iteration
SISO	Single Input Single Output
SL	Simple Learning
SL-FLC	Simple Learning Strategy-based Feedback Linearization Control
SMC	Sliding Mode Controller
SOCP	Second-order cone program
SQP	Sequential Quadratic Program
SQPr	Sequential Quadratic Programming
SVR	Support Vector Regression

Chapter 1

Introduction

1.1 Motivation

The unprecedented growth of multirotor aerial robots in various industries underly the recent advances in sensor, actuation, and processing technologies as well as the availability of numerous low-cost (ARM-based) processors. They have been explored for a variety of applications, for instance, search and rescue at disaster sites [6, 7], air package delivery [8], damage evaluation after an earthquake [9], traffic monitoring [10], and exploration tasks in an unknown environment [11]. Moreover, researchers have also been working on developing algorithms and interfaces to support their safe yet efficient physical interaction and collaboration with human operators [12, 13].

To facilitate safe operation in urban environments, a precise path tracking performance is of extreme importance. Unfortunately, multirotor aerial robots are strongly coupled, open-loop unstable, and inherently nonlinear systems which render their control a challenging problem. Although conventional control approaches, e.g., proportional-integral-derivative (PID) and linear-quadratic-regulator (LQR) have been successfully utilized for their control [14], they involve linearization about a set of aforethought equilibrium points. As a result, their agile operation away from the linearization zone may lead to substandard performance or might even destabilize the system. Although robustness could be achieved by conservative tuning of these linearized controllers, the price to pay is performance. What is more, incorporating a decentralized single-input-single-output (SISO) PID design

for a severely coupled system further degrades the control performance. This is essentially due to the selfish nature of the individual PIDs that neglects the coupling and tries to minimize the error along their respective axes.

Over other model-based control techniques, the model predictive controller (MPC) owes its remarkable success to the unique ability of simultaneously handling constraints and optimize performance via recursive online optimization. It has been utilized for the control of various ground [15–17] as well as multirotor aerial robots [18–22]. In addition, its fault-tolerance and cooperative control capabilities have been demonstrated in [23] and [24], respectively.

Besides control challenges, another issue with the nonlinear systems is the difficulty that lies in their mathematical modeling. Precise modeling requires several experimental trials of the system which is arduous as well as time-consuming [25, 26], and in the end, the possibility of obtaining the accurate model is limited [27, 28]. In addition, there are certain operations involving (unknown) external disturbances that lead to an uncertain (time-varying) system model. Three of these working scenarios are depicted in Fig. 1.1. In the given figure, the first depicts an in-flight package delivery application that leads to mass variation, the second portrays an offshore visualization operation in presence of wind disturbance, and the third shows ceiling inspection of a tunnel that necessitates flight under the influence of ceiling effect[1].

Apart from all the benefits of the aforementioned model-based controllers, there is an associated down-side. That is, the optimal performance is only guaranteed with an accurate system model. This implies that for operations with external disturbances, as depicted in Fig. 1.1, the resulting plant-model mismatch would lead to degraded overall performance. Additionally, for rapidly growing uncertainties, they may even destabilize the system in the worst case. While a certain level of robustness to disturbance can be achieved by tuning for a specific scenario in a certain environment, yet there is no performance guarantee when they operate in an unknown environment full of internal and external uncertainties. For instance, vision-based localization while navigating through an unstructured environment induces additional uncertainties, which further degrades the performance of the perfectly tuned controllers. As a result, people in the robotics community are getting inclined towards controllers that can perform sufficiently well when exposed

[1] Reduction in the overall mass of the aerial robot when it flies in close proximity to the ceiling.

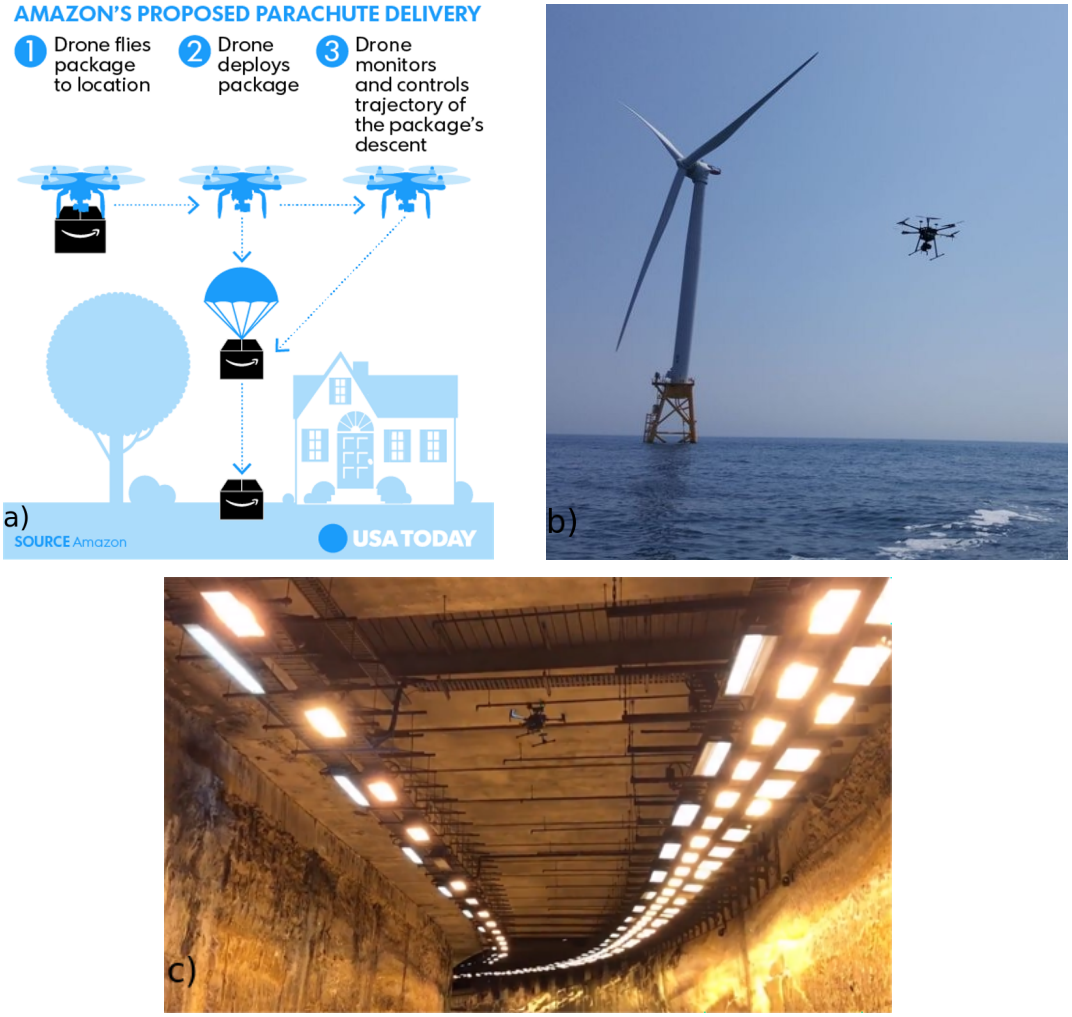


FIGURE 1.1: Disturbance operations: a) in-flight delivery by Amazon [1], b) offshore inspection of wind platforms [2], and c) tunnel inspection [3].

to unforeseen disturbances within an unknown environment. All these issues imply strong reasons to incorporate computationally efficient controllers that are accurate, reliable, and adaptive to unpredictable operating conditions.

In addition to the difficulties with the aerial robots, some challenges are associated with the implementation of the control algorithms. The foremost is the non-trivial controller tuning process which gets tedious and dangerous at the same time when performed directly on the aerial robot. This is also a major concern for the safe realtime implementation of the MPC, wherein some suitable weighting parameters are to be obtained for the underlying optimization problem.

1.2 Current Methodologies and Limitations

In order to deal with the uncertainties and/or disturbances in the system model, numerous control approaches have been proposed in the literature. To begin with, robust control algorithms that explicitly cater to (bounded) uncertainties have been widely utilized for different robotic platforms [29–36]. However, since these robust control methods are based on worst-case analysis, they sacrifice system performance in the absence of uncertainties, even though they exhibit impressive results in the presence of uncertainties. Another popular control approach for uncertain systems is the use of inherently robust controllers, for instance, a sliding mode controller (SMC) [37, 38]. While high controller gains are selected to be robust against the uncertainties based on the SMC theory, this high gain selection may cause problems such as chattering on the system response [39].

The adaptive controller is another type of control algorithm that is utilized to tackle the system uncertainties in aerial robots. Broadly, two approaches exist: the indirect and the direct. The indirect approach first identifies the system model and later updates the controller parameters based on the identified model. It is demonstrated for the tracking control of a ducted fan vertical takeoff and landing micro-aerial robot in [40]. On the contrary, the direct approach adapts the controller parameters such that the underlying system matches the performance of a (nominal) reference model. The tracking performance of the direct approach along with a combined approach (a combination of direct and indirect) is demonstrated in [41]. Another well-utilized technique is the adaptive-optimal control strategy which incorporates an adaptive controller for stability during the learning phase, followed by switching to the main model-based optimal controller that eventually optimizes the performance [42, 43]. Howbeit, since these adaptive controllers aim to match the performance of a given (or estimated for indirect method) reference model, they require accurate identification of the system. Other types of control approaches that facilitate the adaptation of the system to a certain extent are: gain scheduling control [44] and control based on switching between multiple models [45]. While computing the control gains for various flight conditions is tedious, the in-flight switching between multiple models may destabilize the closed-loop.

Another alternative approach is to accommodate some learning algorithm to learn

the uncertain model parameters throughout the operation. These learned parameters are then modified within the controller, in contrast to updating the controller parameters as done in adaptive control. Numerous learning approaches exist in the literature, whereby the widely utilized is the neural networks (NNs)-based learning. Its aerial robotic implementations include adaptive attitude tracking control of an aerial robot with unknown inertial matrix by utilizing NN-based compensation of unknown aerodynamic forces in [46], output feedback control of a quadrotor utilizing NN-based online learning of the complete robot dynamics as well as the uncertain nonlinear disturbances in [47], and adaptive NN-based stabilization control of quadrotor subjected to modeling error and wind disturbance in [48]. Other recent applications of these NN-based, so-called model-free learning approaches are demonstrated in [49–51]. While impressive performance is exhibited in most cases, still the biggest concern in such applications involving NN-based learning is the difficulty to realize the stability proof for the overall system. In addition, since several tuning parameters (e.g., weights) are randomly initialized, their repetitive success is difficult to guarantee while the overall performance is critically linked to their initial values.

1.3 Thesis Objectives and Solution Approach

In reference to the challenges associated with the operation of aerial robots in uncertain environments, the four main objectives of this thesis are as follows:

1. Design control algorithms that cater to the **nonlinear dynamics** of aerial robots without linearization.
2. Accommodate fast solution methodologies to facilitate the implementation of nonlinear MPC (NMPC) over **low-cost embedded processors**.
3. Develop reliable online learning algorithms to account for the **uncertain dynamics** due to operation in unknown environments.
4. Develop a technique to facilitate **safe automated tuning** of the incorporated control algorithm.

Besides, the three of the aforementioned objectives along with the proposed solution approaches are summarized in Fig. 1.2.

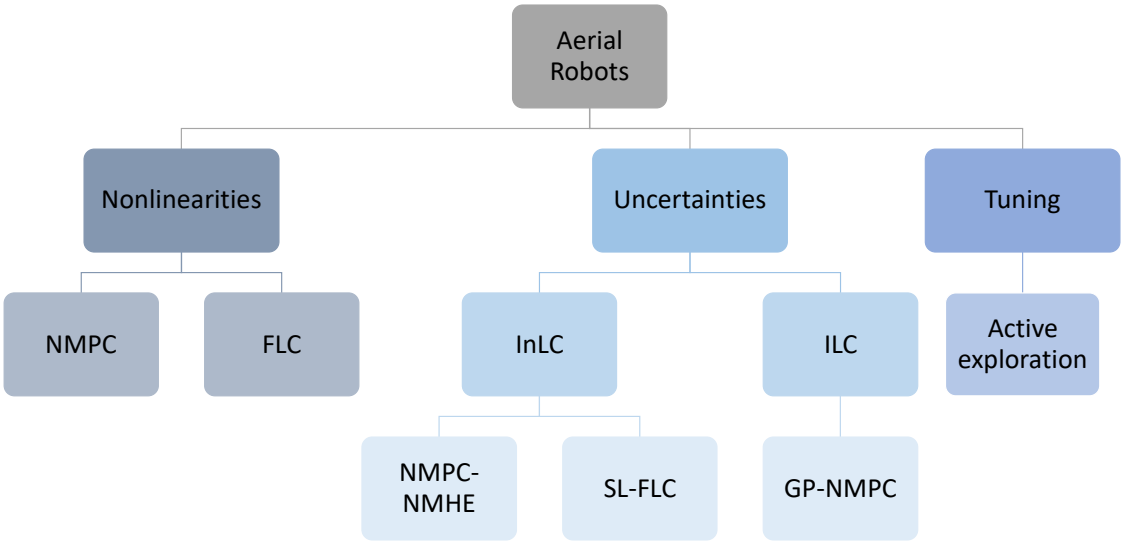


FIGURE 1.2: Overview of thesis objectives and proposed solution techniques.

To achieve the first objective, two control algorithms are adopted for the control of aerial robots. The first amongst them is the NMPC which explicitly accommodates the system nonlinearities within its optimization problem formulation. The second is the feedback linearization control (FLC) method that transforms the nonlinear system into its linear equivalent rather than linearizing it about a set of predetermined points. Consequently, the advanced linear control approaches can be freely utilized on the reduced system.

To facilitate the second objective, fast solution methods are adopted in this thesis. Amongst the two most prominent approaches, namely, indirect and direct methods, the latter is utilized on account of its several computational advantages over the former. Besides, the direct methods are further categorized whether being sequential or simultaneous, wherein a variant of the simultaneous technique, i.e., multiple shooting, is favored over the sequential technique (single shooting). The reason for this selection is essentially the initialization flexibility of the former which along with the parallelization ability results in faster convergence. Finally, the resulting sequential quadratic program (SQP) is solved incorporating the generalized Gauss-Newton (GGN) method along with a special realtime iteration (RTI) scheme that constrains the maximum Gauss-Newton iterations to one for realtime feasibility.

To realize the third objective, a few model learning control approaches are proposed in this thesis. Broadly, they are classified as: instantaneous learning control (InLC) and iterative learning control (ILC). While the InLC technique utilizes an

estimator (or disturbance observer) to learn the (time-varying) model parameters, the ILC scheme, on the other hand, utilizes experience from the system repetitions in order to identify the uncertain dynamics. Subsequently, the learned model parameters update the corresponding terms within the controller definition in both the approaches. Since the model-based controllers suffer from lack of modeling, parameter variations, and disturbances in their working environment, it is observed that both of the control methodologies have compelling abilities to significantly reduce the tracking error under different uncertainties throughout the operation.

In terms of the InLC scheme, two control frameworks are developed in this thesis. The first incorporates a nonlinear moving horizon estimator (NMHE) in conjunction with the NMPC, wherein the former estimates and updates the time-varying model parameters in an online manner. The constrained learning ability of the overall control framework is demonstrated for the position as well as attitude tracking control of an aerial robot. The second control framework comprises of a simple learning (SL) strategy in combination with the FLC method. The proposed SL strategy circumvents the limitations of the traditional FLC method by updating the controller gains along with the disturbance estimate in the feedback control law. Moreover, the test results exhibit that simple learning strategy-based FLC (SL-FLC) framework maintains the nominal control performance in the absence of modeling uncertainties and external disturbances and ensures robust control performance in their presence.

In terms of the ILC scheme, a Gaussian process (GP)-based regression technique learns the disturbance forces encountered in an offshore visualization operation of the aerial robot. These learned disturbance values then update the model definition within the GP-NMPC framework, thereby resulting in improved tracking performance over the conventional NMPC (without learning). Moreover, the long-short term memory (LSTM) feature of the developed GP model facilitates superior learning performance over its instantaneous counterpart (i.e., NMHE), as depicted from the simulation test results.

To realize the fourth objective, an automated weight set tuning framework for MPC is developed. The proposed active exploration approach extends the basic trial-and-error method and tunes the MPC weight sets more intelligently. It benefits from the retrospective knowledge gained in previous trials, resulting in a faster tuning procedure. In addition, to facilitate a safe tuning process, the MPC weights

are tuned over a deep neural network (DNN)-based model of the robot rather than tuning over the real robot. What is more, thanks to the high fidelity representation by the DNN model, a seamless sim-to-real transition is demonstrated via employing the weight sets obtained in simulation-based tuning directly on the real robot.

1.4 Contributions

The overall contributions of this thesis are distributed in two categories, namely, major and minor. The corresponding contributions in each category are summarized below.

Major Contributions:

- Instantaneous learning-based NMPC-NMHE framework to realize the precise position tracking of a custom-designed, 3D-printed tilt-rotor tricopter.
- Instantaneous learning-based cascade NMPC-NMHE framework for a precise position as well as attitude control of the tilt-rotor aerial robot.
- The SL strategy to update the controller gains and disturbance estimate within the traditional FLC method. Additionally, proof of stability and global optimality for the obtained gains and disturbance estimate.
- Iterative learning-based GP-NMPC framework for precise visualization of geological outcrops.
- Active exploration approach for safe automated tuning of the MPC weight sets.

Minor Contributions:

- Detailed mathematical modeling of two types of multirotor aerial robots, namely, quadrotor and tilt-rotor tricopter.
- Fast C++ NMPC and NMHE codes have facilitated an onboard implementation on a low-cost embedded processor, i.e., Raspberry Pi 3. To the best of

Authors' knowledge, this is the first (simultaneous) implementation of these methods on such a low-cost processor.

1.5 Outline of the Thesis

Chapter 2 lays the theoretical foundation for the concepts utilized in the thesis. It describes a generic convex optimization problem along with the underlying optimality conditions. Then, it illustrates standard convex problems, parametric programming problems, and the sequential quadratic programming approach to the solution of nonlinear programs. It also describes the direct multiple shooting method which reduces a continuous-time optimal control problem to a discrete-time sequential quadratic program. Besides, it presents various available software tools that are frequently utilized for realtime MPC implementations. Towards the end, it deduces the mathematical models for the utilized aerial robots.

Chapter 3 presents the instantaneous learning control technique. After a short literature overview, it discusses the problem formulations of NMPC and NMHE. Then, it exhibits two case studies for the NMPC-NMHE framework along with the implementation details. In the first study, the NMPC-NMHE framework is utilized for position tracking of the aerial robot in real flights. Whereas, in the second study, a cascade NMPC-NMHE design is incorporated for the position as well as attitude tracking of the robot in simulated flights.

Chapter 4 elaborates on another type of instantaneous learning technique, i.e., the SL strategy for the feedback linearization control algorithm. After a short overview of the other learning techniques that are commonly utilized with the FLC method, it provides the problem formulations of the traditional FLC and the FLC with integral action. Then, it illustrates the SL strategy and derives the update rules for controller gains and disturbance estimate. It also provides the stability proof and global minimum analysis for the updates. The efficacy of the SL-FLC framework is first validated in simulation over a second-order system, followed by the real-world validation over the aerial robot.

Chapter 5 presents the iterative learning control approach. It provides a short overview of the other works based on the iterative learning concept. Then, it

describes the aerial robot along with the onboard sensors utilized for outdoor navigation. It also discusses the fundamentals of the GP-based regression technique and provides the regression expressions for the deterministic inputs (one-step-ahead prediction) and the noisy inputs (multi-step ahead predictions). After illustrating the incorporated ILC framework in detail, it exhibits the efficacy of the control framework in simulation as well as real-world tests.

Chapter 6 manifests the solution to circumvent the non-trivial weight tuning process associated with the MPC design. After presenting the robotic platform and the underlying NMPC problem formulation, it illustrates the active exploration strategy that is utilized within the auto-tuning mechanism. It also provides the details for the obtained DNN model of the aerial robot. Then, it presents the implementation results of the proposed tuning methodology in the form of a benchmark study, wherein its performance is compared with the traditional trial-and-error approach. It also presents the fine-tuning results over the real robot. Towards the end, it includes a short user-based study, whereby the quality of weight sets obtained from five human users is compared to the best one obtained from simulation-based tuning.

Chapter 7 concludes the works presented in this thesis. It also describes some remaining challenges that could be adopted for future research.

Chapter 2

Theoretical Background

This chapter provides a theoretical background for the important concepts that are utilized throughout the thesis. In the beginning, Section 2.1 lays a foundation for the optimization theory by describing convex optimization and the underlying optimality conditions. Then, Section 2.2 illustrates how a generic continuous-time optimal control problem can be reduced to a discrete-time nonlinear program followed by its numerical solution utilizing some realtime techniques. In addition, it discusses various available software tools that are frequently utilized for realtime MPC implementations. Thereafter, the mathematical models of multirotor aerial robots are deduced in Section 2.3.

2.1 Convex Optimization

Convex optimization is a special class of mathematical optimization with some favorable features. Firstly, any locally optimal solution is also globally optimal [52]. Another appealing feature besides the well-established theoretical background is the availability of efficient and reliable numerical solution methods, as for most of the control and online estimation problems obtaining an analytical solution is not possible. Numerous books have been published on the optimization theory and few amongst them are [52–54].

A general convex optimization problem, also known as a nonlinear program (NLP), is of the form:

$$\min_{\mathbf{x}} f(\mathbf{x}) \tag{2.1a}$$

$$\text{s.t. } h_i(\mathbf{x}) \leq 0, \qquad i = 1, \cdots, n_i, \tag{2.1b}$$

$$g_e(\mathbf{x}) = 0, \qquad e = 1, \cdots, n_e, \tag{2.1c}$$

where $\mathbf{x} \in \mathtt{S} \subseteq \mathbb{R}^{n_\mathbf{x}}$ is the optimization (or decision) variable defined in a feasible set $\mathtt{S}$, $f(\mathbf{x})$: $\mathbb{R}^{n_\mathbf{x}} \mapsto \mathbb{R}$ is the scalar objective function, and $h_i(\mathbf{x})$: $\mathbb{R}^{n_\mathbf{x}} \mapsto \mathbb{R}$ and $g_e(\mathbf{x})$: $\mathbb{R}^{n_\mathbf{x}} \mapsto \mathbb{R}$ are the scalar inequality and equality constraint functions defined on sets $\mathtt{I} = \{1, \cdots, n_i\}$ and $\mathtt{E} = \{1, \cdots, n_e\}$, respectively. In addition, the functions f, h_i, and g_e are assumed convex in the decision variable $\mathbf{x}$, while the function g_e is further assumed to be affine. A solution to (2.1) would result in an optimal value for the objective as follows:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathtt{S}} \; f(\mathbf{x}) \tag{2.2}$$

where $\mathbf{x}^*$ is called an optimizer or a global optimizer (because of convexity) or an optimal solution. Moreover, for $f(\mathbf{x}^*) = -\infty$ the problem is called unbounded below, whereas, if the set $\mathtt{S}$ is empty then the problem is said to be infeasible and $f(\mathbf{x}^*) = \infty$ is set by convention [55].

2.1.1 Optimality Conditions

The primary optimization problem of 2.1 also called the primal problem can be converted to a supplementary Lagrange dual problem. The idea is to obtain the optimal solution by solving the dual problem instead of the primal, as the former is (mostly) easier to solve. The Lagrange dual problem is formulated by first obtaining a Lagrange dual function given as:

$$\mathcal{L}(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \sum_{i=1}^{n_i} \lambda_i h_i(\mathbf{x}) + \sum_{e=1}^{n_e} \nu_e g_e(\mathbf{x}), \tag{2.3}$$

where the scalars λ_i and ν_e are called Lagrange multipliers (or dual variables) associated with the inequality ($h_i(\mathbf{x}) \leq 0$ for $i \in \mathtt{I}$) and equality ($g_e(\mathbf{x}) = 0$ for $e \in \mathtt{E}$) constraints, respectively. This Lagrange dual function is subsequently maximized

over dual variables to obtain the 'best' lower bound to the primal problem:

$$\max_{\lambda,\nu} \mathfrak{g}(\lambda,\nu) \tag{2.4a}$$

$$\text{s.t. } \lambda_i \geq 0, \qquad i = 1, \cdots, n_i, \tag{2.4b}$$

wherein the expression for $\mathfrak{g}(\lambda,\nu)$ is as follows:

$$\mathfrak{g}(\lambda,\nu) = \min_{\mathbf{x}} \ \mathcal{L}(\mathbf{x},\lambda,\nu). \tag{2.5}$$

Remark 1. It is to be noted that $\mathfrak{g}(\lambda,\nu)$ is always a concave function as it is the pointwise maximum of a family of affine functions of (λ,ν) [55]. This validates the convexity of the dual problem – maximization of a concave function over a convex set –, even when the convexity of the primal may not be guaranteed. As a result, it is easier (in most cases) to solve the dual problem than the primal, as mentioned before.

Recall that the solution of the dual problem (denoted by d^*) is only a lower bound to the primal solution (denoted by p^*), i.e., $d^* \leq p^*$ and the difference $p^* - d^*$ is known as the duality gap. If $d^* = p^*$, i.e., the duality gap is zero, the condition of strong duality holds. This implies that the best obtained lower bound by solving the dual problem coincides with the optimal value of the primal (p^*). However, the strong duality does not hold even for convex primal problems in practice and some constraint qualifications conditions are imposed to imply the strong duality [55]. One of the commonly used constraint qualifications is the 'Slater's condition', which is defined in the following way.

Definition 2.1.1. Slater's condition: It states that for a strictly feasible primal problem for which there exists an $\mathbf{x}$, such that $h_i(\mathbf{x}) < 0 \ \forall \ i \in \mathtt{I}$ and $g_e(\mathbf{x}) = 0 \ \forall \ e \in \mathtt{E}$, then the strong duality holds.

In addition, below are the well-known Karush-Kuhn-Tucker (KKT) optimality conditions that provide the necessary and sufficient conditions for a globally optimal solution.

Theorem 2.1.1. KKT optimality conditions: Consider the (primal) optimization problem (2.1) and its dual (2.4). Let problem (2.1) be convex and $\mathbf{x}^*$ be a feasible point such that the objective function f, and the constraint functions

$h_i \ \forall \ i \in \mathtt{I}$ and $g_e \ \forall \ e \in \mathtt{E}$ are continuously differentiable at $\mathbf{x}^*$. If problem (2.1) satisfies Slater's condition then $\mathbf{x}^*$ is optimal if and only if there exist vectors λ^* and ν^* that satisfy the following conditions together with $\mathbf{x}^*$,

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^{n_i} \lambda_i^* \nabla h_i(\mathbf{x}^*) + \sum_{e=1}^{n_e} \nu_e^* \nabla g_e(\mathbf{x}^*) = 0, \tag{2.6a}$$

$$h_i(\mathbf{x}^*) \leq 0, \qquad i = 1, \cdots, n_i, \tag{2.6b}$$

$$g_e(\mathbf{x}^*) = 0, \qquad e = 1, \cdots, n_e, \tag{2.6c}$$

$$\lambda_i^* \geq 0, \qquad i = 1, \cdots, n_i, \tag{2.6d}$$

$$\lambda_i^* h_i(\mathbf{x}^*) = 0, \qquad i = 1, \cdots, n_i, \tag{2.6e}$$

Remark 2. In general, the KKT conditions can be justified to be the necessary conditions for any primal-dual optimal pair with strong duality and with differentiable objective and constraint functions. Whereas, in case of a convex primal problem, the KKT conditions become sufficient conditions such that a primal-dual pair $\mathbf{x}^*$, (λ^*,ν^*) which satisfies conditions (2.6) is the primal-dual optimal pair with zero duality gap [55].

2.1.2 Standard Convex Problems

Three widely encountered convex optimization problems are illustrated below in their increasing order of computational complexity. The formulation of many important practical problems into either of these categories resulted in their popularity.

Linear Program

Perhaps, the most commonly known problem class is the linear program (LP). For affine objective and constraint functions, it is defined in the following form,

$$\min_{\mathbf{x}} \ \mathbf{c}^T \mathbf{x} \tag{2.7a}$$

$$\text{s.t. } H_i^T \mathbf{x} \leq d_i, \qquad i = 1, \cdots, n_i, \tag{2.7b}$$

$$G_e^T \mathbf{x} = d'_e, \qquad e = 1, \cdots, n_e, \tag{2.7c}$$

where the corresponding vectors $\mathbf{c}$, H_i, and G_e are defined over $\mathbb{R}^{n_\mathbf{x}}$. Note that the constraints in the above equation result in a feasible set in the form of a polyhedron[1] over which a linear function is minimized [56]. The LPs are easiest to solve when compared to the other programs. One of the efficient solution methods is the simplex method that is specially designed for linear programs and can be considered as a specialized form of active set or interior point methods [56].

Quadratic Program

The next widely used problem class is the quadratic program (QP), which for a convex problem with affine objective and constraint functions is formulated as:

$$\min_{\mathbf{x}} \ \mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x} \tag{2.8a}$$

$$\text{s.t. } H_i^T\mathbf{x} \leq d_i, \qquad i = 1, \cdots, n_i, \tag{2.8b}$$

$$G_e^T\mathbf{x} = d'_e, \qquad e = 1, \cdots, n_e, \tag{2.8c}$$

where the Hessian matrix $\mathbf{H} \in \mathbb{R}^{n_\mathbf{x} \times n_\mathbf{x}}$ is assumed positive semidefinite ($\mathbf{H} \succeq 0$), yielding a convex optimization problem that is minimized over a polyhedron. In addition, for $\mathbf{H} \succ 0$, a strictly convex QP is obtained. Generally, the computational complexity in solving QPs is of a similar order as that for LPs. Moreover, most QPs can be solved using methods like active set, gradient or interior point, details of which is seen in [57].

Second-order Cone Program

Second-order cone programs (SOCP) are nonlinear convex problems that are the generalized versions of QPs. They involve the optimization over a second-order cone also known as the ice-cream cone. For good examples of SOCPs, one may

[1] A *polyhedron* is defined as the intersection of finite number of halfspaces, where a closed halfspace in $\mathbb{R}^n$ is a set of the form: $\mathcal{P} = \{\mathbf{x}' \in \mathbb{R}^n \mid \mathbf{a}^T\mathbf{x}' \leq \mathbf{b}\}$, for $\mathbf{a} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}$.

refer to [58, 59]. In a general form, an SOCP is represented as follows:

$$\min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} \tag{2.9a}$$

$$\text{s.t. } G_e^T\mathbf{x} = d'_e, \qquad e = 1, \cdots, n_e, \tag{2.9b}$$

$$\left\|H_i^T\mathbf{x} + D_i\right\|_2 \leq d_i^T + r_i, \qquad i = 1, \cdots, n_i, \tag{2.9c}$$

where (2.9c) represents the second-order cone constraints. Note that, if in the above equations, $H_i = 0 \; \forall \; i = 1, \cdots, n_i$, then the SOCP reduces to a general LP. Whereas, if $r_i = 0 \; \forall \; i = 1, \cdots, n_i$, the SOCP reduces to a quadratically constrained quadratic program (QCQP), which is a special case of QP with quadratic inequality constraints [56].

2.1.3 Parametric Programming

Besides having an explicit solution (in the form of numerical values) to an optimization problem (2.1), many practical scenarios require the dependence of the optimal solution on certain parameter values. This class of optimization problems is called parametric programs. Overall, these programs include the parametric dependent objective function and/or constraints, while the parameter is passed as an input to the optimization problem. Moreover, the programs with a scalar parameter are called parametric programs, whereas the programs having vector parameter are called multiparametric programs.

For a given convex optimization problem (2.1), a parametric programming problem in its general form can be written as:

$$\min_{\mathbf{x}} f(\mathbf{x}, \boldsymbol{\theta}) \tag{2.10a}$$

$$\text{s.t. } h_i(\mathbf{x}, \boldsymbol{\theta}) \leq 0, \qquad i = 1, \cdots, n_i, \tag{2.10b}$$

where the parameter $\boldsymbol{\theta}$ is considered to be a vector defined over $\mathbb{R}^{n_{\boldsymbol{\theta}}}$. The main aim in parametric programming is to solve the above equation for varying parameter $\boldsymbol{\theta}$, by obtaining explicit expressions for the objective function and the optimizer $\mathbf{x}^*(\boldsymbol{\theta})$, as functions of $\boldsymbol{\theta}$. Obtaining a parametric solution is only feasible in special cases of parametric linear programs (pLPs) and parametric quadratic programs (pQPs), which are significant for MPC applications (in particular explicit MPC).

Therefore, these two programs are illustrated below, whereas, for the details of other parametric programming problems, one may refer to [60]. Moreover, it is to be noted that for the sake of simplicity, only the case with inequality constraints is considered, however, the illustration can be easily extended for problems with equality constraints.

Parametric Linear Program

A pLP is considered as a linear program, wherein the parameter $\boldsymbol{\theta}$ enters the constraints linearly (c.f. [55, 61]):

$$J^*(\boldsymbol{\theta}) = \min_{\mathbf{x}} \mathbf{c}^T\mathbf{x} \tag{2.11a}$$

$$\text{s.t. } H_i^T\mathbf{x} \leq d_i + D_i^T\boldsymbol{\theta}_i, \qquad i = 1, \cdots, n_i, \tag{2.11b}$$

where the vector D_i is defined over $\mathbb{R}^{n_{\boldsymbol{\theta}}}$.

Theorem 2.1.2. Solution to pLP: Consider the parametric linear program of the form (2.11), wherein the set of feasible parameter $\boldsymbol{\theta}$ is defined by a closed polyhedron over $\mathbb{R}^{n_{\boldsymbol{\theta}}}$. The optimal value of the objective $J^*(\boldsymbol{\theta})$ is continuous, convex, and a piecewise affine function of $\boldsymbol{\theta}$. There exists an optimal solution $\mathbf{x}^*(\boldsymbol{\theta})$ which is continuous and a piecewise affine function of $\boldsymbol{\theta}$.

Amongst various problems in the pLP algorithms, the primal and dual degeneracy are most prevalent. While the primal degeneracy occurs when the basis describing the optimal solution is not unique, the dual degeneracy, on the other hand, results when there exist multiple optimizers. Moreover, lexicographic perturbation is utilized to resolve the issue of dual degeneracy [62].

Parametric Quadratic Program

In a similar way to pLP, a pQP is considered as a quadratic program, wherein the parameter $\boldsymbol{\theta}$ enters linearly in both the objective function and the constraints (c.f. [55] for a comprehensive description):

$$J^*(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{Y}\boldsymbol{\theta} \ \min_{\mathbf{x}} \mathbf{x}^T\mathbf{H}\mathbf{x} + \boldsymbol{\theta}^T\mathbf{P}\mathbf{x} \tag{2.12a}$$

$$\text{s.t. } H_i^T\mathbf{x} \leq d_i + D_i^T\boldsymbol{\theta}_i, \qquad i = 1, \cdots, n_i, \tag{2.12b}$$

where the additional matrices are defined as $\mathbf{Y} \in \mathbb{R}^{n_{\boldsymbol{\theta}} \times n_{\boldsymbol{\theta}}}$ and $\mathbf{P} \in \mathbb{R}^{n_{\mathbf{x}} \times n_{\mathbf{x}}}$. Theorem below states the important features of the parametric solution of a pQP.

Theorem 2.1.3. Solution to pQP: Consider the parametric quadratic program of the form (2.12) with $\mathbf{H} \succ 0$, $\begin{bmatrix} \mathbf{H} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{Y} \end{bmatrix} \succeq 0$, and the set of feasible parameter $\boldsymbol{\theta}$ defined by a closed polyhedron over $\mathbb{R}^{n_{\boldsymbol{\theta}}}$. The optimal value of the objective $J^*(\boldsymbol{\theta})$ is continuous, convex, and a piecewise quadratic function of $\boldsymbol{\theta}$. There exists an optimal solution $\mathbf{x}^*(\boldsymbol{\theta})$ which is unique, continuous, and a piecewise affine function of $\boldsymbol{\theta}$.

2.1.4 Sequential Quadratic Programming

Sequential quadratic programming (SQPr) is another approach to solve the NLPs with inequality constraints. In essence, the SQPr method solves in each iteration an inequality constrained QP which is obtained by linearizing the objective and constraint functions about a feasible point $\bar{\mathbf{x}}$:

$$\min_{\mathbf{x}} \ \frac{1}{2}\Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} + \nabla f(\bar{\mathbf{x}})^T\Delta\mathbf{x} \tag{2.13a}$$

$$\text{s.t. } h_i(\bar{\mathbf{x}}) + \nabla h_i(\bar{\mathbf{x}})\Delta\mathbf{x} \leq 0, \qquad i = 1, \cdots, n_i, \tag{2.13b}$$

$$g_e(\bar{\mathbf{x}}) + \nabla g_e(\bar{\mathbf{x}})\Delta\mathbf{x} = 0, \qquad e = 1, \cdots, n_e, \tag{2.13c}$$

where $\nabla h_i(\bar{\mathbf{x}})$ and $\nabla f_e(\bar{\mathbf{x}})$ are the constraint Jacobian matrices, while the matrix $\mathbf{H} \triangleq \nabla^2\mathcal{L}(\bar{\mathbf{x}}, \bar{\lambda}, \bar{\nu})$ represents the Hessian of the Lagrangian in an exact SQPr method. The primal and dual solution ($\mathbf{x}^*$,λ^*,ν^*) of the above QP are updated according to:

$$\begin{aligned} \bar{\mathbf{x}}^+ &= \bar{\mathbf{x}} + \alpha\Delta\mathbf{x}, \\ \bar{\lambda}^+ &= \bar{\lambda} + \alpha(\lambda^* - \bar{\lambda}), \\ \bar{\nu}^+ &= \bar{\nu} + \alpha(\nu^* - \bar{\nu}), \end{aligned} \tag{2.14}$$

where the term α represents the step size which is determined using some globalization strategy [57]. One may note that the convergence quality (linear or super-linear or quadratic) for the above QP depends on the Hessian approximation. Different techniques including Broyden-Fletcher-Goldfarb-Shanno (BFGS) and Generalized Gauss-Newton method exist and their details can be seen in [57]. The latter approach, however, is more beneficial for problems with a (nonlinear) least-square

objective of the form $f(\mathbf{x}) = \frac{1}{2}\|\zeta(\mathbf{x})\|_2^2$, that are mostly encountered in tracking or estimation applications. The function $\zeta(\,\cdot\,)$ typically represents an error or a misfit and is therefore called the residual function. Moreover, the GGN method approximates the Hessian of the Lagrangian as:

$$\mathbf{H}^{\mathrm{GGN}} = \nabla\zeta(\bar{\mathbf{x}})\nabla\zeta(\bar{\mathbf{x}})^T \approx \nabla^2\mathcal{L}(\bar{\mathbf{x}}, \bar{\lambda}, \bar{\nu}), \tag{2.15}$$

where $\nabla\zeta(\bar{\mathbf{x}})$ is the Jacobian of the residual function evaluated at $\bar{\mathbf{x}}$. Note that in the GGN method, the Hessian $\mathbf{H}^{\mathrm{GGN}}$ is independent of any dual variables, as evident from (2.15). This implies that the performance of this method is not linked with the availability of good initial guess for the Lagrange multipliers, which is of significant practical advantage. However, this method provides a good approximation to the exact Hessian as long as the residual $\|\zeta(\bar{\mathbf{x}})\|$ is small or the problem is moderately nonlinear [63].

2.2 Realtime Optimal Control

This section illustrates the optimal control problem (OCP) in a general form followed by a numerical scheme for its solution. The aim is to introduce a solution method that is commonly encountered in practical applications. In other words, a method that could be utilized for embedded NMPC applications which usually requires a solution to the problem in the milliseconds' range.

In the form of an ordinary differential equation (ODE), a continuous-time (parametric) OCP can be written as:

$$\min_{\mathbf{x},\mathbf{u}} \int_0^T L\big(\mathbf{x}(t), \mathbf{u}(t)\big)dt + E\big(\mathbf{x}(t)\big) \tag{2.16a}$$

$$\text{s.t. } \mathbf{x}(0) - \hat{\mathbf{x}}_0 = \mathbf{0}_{n_\mathbf{x}}, \tag{2.16b}$$

$$\dot{\mathbf{x}}(t) - \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big) = \mathbf{0}_{n_\mathbf{x}}, \qquad \forall t \in [0, T], \tag{2.16c}$$

$$\mathbf{h}\big(\mathbf{x}(t), \mathbf{u}(t)\big) \leq \mathbf{0}_{n_\mathbf{h}}, \qquad \forall t \in [0, T], \tag{2.16d}$$

$$\mathbf{r}\big(\mathbf{x}(T)\big) \leq \mathbf{0}_{n_\mathbf{r}}, \tag{2.16e}$$

where $\mathbf{x}(t) \in \mathbb{R}^{n_\mathbf{x}}$ and $\mathbf{u}(t) \in \mathbb{R}^{n_\mathbf{u}}$ are the respective state and input vectors at time t; Equations (2.16b) and (2.16e) represent the fixed initial and terminal constraints,

respectively, with the function definition $\mathbf{r}(\cdot) : \mathbb{R}^{n_\mathbf{x}} \mapsto \mathbb{R}^{n_\mathbf{r}}$; Equation (2.16c) gives the nonlinear system model, wherein the model function is given as $\mathbf{f}(\cdot,\cdot) : \mathbb{R}^{n_\mathbf{x}} \times \mathbb{R}^{n_\mathbf{u}} \mapsto \mathbb{R}^{n_\mathbf{x}}$; Inequalities in (2.16d) with the function definition $\mathbf{h}(\cdot,\cdot) : \mathbb{R}^{n_\mathbf{x}} \times \mathbb{R}^{n_\mathbf{u}} \mapsto \mathbb{R}^{n_\mathbf{h}}$, represent the path constraints imposed on the states and inputs. It is to be noted that the term $\mathbf{0}_{n_k}$ in the above equations represents a zero vector defined over $\mathbb{R}^{n_k}$. Moreover, the integral cost $L\big(\mathbf{x}(t), \mathbf{u}(t)\big)$ and the terminal cost $E\big(\mathbf{x}(t)\big)$ in the objective of (2.16a) are commonly known as the Lagrange term and the Mayer term, respectively. Note that in the following illustration, the time horizon T is considered constant. However, there exist numerous other applications considering T to be an optimization variable, thus leading to a time-optimal control problem.

The aim is to obtain the optimal solution to the problem (2.16) in the form: $\mathbf{u}^*(t, \hat{\mathbf{x}}_0)\ \forall\ t \in [0, T]$. For the considered tracking applications, this solution implies a (locally) optimal control trajectory as a function of time and the initial state. To solve the OCP (2.16), various numerical approaches are utilized in the literature. Broadly, they are classified into three main categories:-

1. Dynamic programming: these schemes are based on Bellman's principle of optimality [64], wherein they propagate a cost-to-go function backward in time. There exist methods to numerically compute the solution approximations, however, this approach suffers from the Bellman's curse of dimensionality, and thus, is only restricted to small state dimensions [63]. Nevertheless, unlike other methods, a globally optimal solution can be obtained.

2. Indirect methods: these methods are typically based on Pontryagin's maximum principle. They utilize 'first optimize, then discretize' approach in which a boundary value problem (BVP) to an ODE is formulated using the necessary condition of optimality for an infinite-dimensional OCP. Besides, the numerical solution to the obtained nonlinear multi-point BVP can be obtained using shooting techniques or the collocation method. Moreover, they suffer from two critical drawbacks, i.e., firstly, the underlying differential equations are difficult to solve due to strong nonlinearity and instability, and secondly, an entirely new problem formulation is required in most cases.

3. Direct methods: unlike previous, they are based on 'first discretize, then optimize' scheme, whereby a discrete-time system in the form of an NLP

is obtained out of the original infinite-dimensional, continuous-time OCP. Subsequently, the obtained NLP is solved using some (structure exploiting) optimization methods.

In many real-world applications involving constrained OCPs, direct methods are the most widely implemented techniques. This is due to their several advantages over the indirect methods, the most prominent being the ability to easily accommodate the inequality as well as the equality constraints. For this reason, they are adopted in this thesis and are further elaborated in the subsequent part. Additionally, one may refer to [63, 65] for a thorough overview and comparison of all the three approaches.

2.2.1 Direct Multiple Shooting Method

The direct multiple shooting method was originally proposed as a method to solve two-point BVPs [66] and was later extended by Bock and Plitt in order to solve the OCPs [67]. In essence, it computes a discrete-time approximation for the original OCP of type (2.16), utilizing finite-dimensional parameterization of the state and control trajectories. Different constituents of this multiple shooting parameterization are described below.

Multiple Shooting Grid

The shooting grid consists of $N+1$ fixed discretization points in the form:

$$0 = t_0 < t_1 < \cdots < t_N = T,$$

which are utilized to distribute the horizon into N shooting intervals. In addition, an equidistant grid defined over the horizon with time points t_i is considered, where the grid length T_s is selected as:

$$T_s = t_{i+1} - t_i = \frac{T}{N}, \quad \text{for} \quad i = 0, \cdots, N-1. \tag{2.17}$$

Note that an equidistant grid is considered only for the sake of simplicity, while a non-equidistant grid can also be utilized in a similar manner within the direct multiple shooting framework.

Control Parameterization

Next, the piecewise constant discretization of the control trajectory $\mathbf{u}(t)\ \forall\ t \in [0, T]$ is performed at the grid points with finitely many control parameters $[\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{N-1}]$. Usually, piecewise polynomial approximations are utilized for the control parameterization and the obtained piecewise constant controls are expressed as:

$$\mathbf{u}(t) = \mathbf{q}_i \quad \text{for} \quad t \in [t_i, t_{i+1}), \tag{2.18}$$

which can also be written in a combined form $\mathbf{u}(t; \mathbf{q})$ for simplicity.

State Parameterization

The defined shooting grid along with the parameterized controls imply a state trajectory which can be obtained by solving N coupled initial value problems (IVPs) of the form:

$$\dot{\mathbf{x}}_i(t) = \mathbf{f}\big(\mathbf{x}_i(t), \mathbf{q}_i\big), \quad \mathbf{x}_i(t_i) = \mathbf{s}_i, \quad t \in [t_i, t_{i+1}). \tag{2.19}$$

Each of the above IVPs has to be solved independently on the interval $[t_i, t_{i+1})$ for $i = 0, \cdots, N-1$, and with the initial value $\mathbf{s}_i$. The obtained state trajectory pieces are denoted by $\mathbf{x}_i(t; \mathbf{s}_i, \mathbf{q}_i)$, wherein the arguments $\mathbf{s}_i$ and $\mathbf{q}_i$ represent their dependence on the corresponding initial state value and control, respectively. Note that the additional state variables $\mathbf{s}_i \in \mathbb{R}^{n_\mathbf{x}}$ for $i = 0, \cdots, N$, are imported to couple the IVPs, while the following continuity constraints are added to restrict the additional degrees of freedom of the solution to some meaningful values,

$$\mathbf{s}_{i+1} = \mathbf{x}_i(t_{i+1}; \mathbf{s}_i, \mathbf{q}_i), \quad \text{for} \quad i = 0, \cdots, N-1. \tag{2.20}$$

Besides, the integral used to solve the IVPs within each interval of the shooting grid is (numerically) simultaneously computed as:

$$l_i(\mathbf{s}_i, \mathbf{q}_i) = \int_{t_i}^{t_{i+1}} L\big(\mathbf{x}_i(t_i; \mathbf{s}_i, \mathbf{q}_i), \mathbf{q}_i\big) dt, \quad \text{for} \quad i = 0, \cdots, N-1. \tag{2.21}$$

Remark 3. It is worth noting that when the initial condition for $\mathbf{x}_i(t)$ in (2.19) is replaced by the solution of the previous IVP, a sequential approach, also known as

the direct single shooting method, is obtained [68]. In essence, the single shooting method regards the state trajectory as an implicit function of the controls, whereby the forward simulation result at each instant affects the solution of the subsequent IVP. Generally, the simultaneous technique is preferred over the sequential technique essentially due to its flexibility in initializing the problem. Besides, it also facilitates the parallelization of the algorithm which results in faster convergence properties, especially for unstable systems [69].

Path Inequality Constraints Discretization

The need to impose the path inequality constraints in (2.16c) over the entire horizon $[0, T]$ results in a semi-infinite problem [70]. In order to avoid the computational intractability, the path constraints are also discretized in the sense that they are only imposed at the shooting grid points t_i for $i = 0, \cdots, N-1$. This further limits their violation in between the grid points. One may note that, although the same shooting grid for the discretization of the path constraints is considered, a finer grid can also be adopted without any problem.

Nonlinear Programming Formulation

A finite-dimensional NLP approximation of the original OCP (2.16) can be obtained by further approximating the continuous-time objective function by a discrete-time sum. The resulting NLP with a block sparse structure can be written as follows:

$$\min_{\mathbf{s},\mathbf{q}} \sum_{i=0}^{N-1} l_i(\mathbf{s}_i, \mathbf{q}_i) + E(\mathbf{s}_N) \tag{2.22a}$$

$$\text{s.t. } \mathbf{s}_0 - \hat{\mathbf{s}}_0 = \mathbf{0}_{n_\mathbf{x}}, \tag{2.22b}$$

$$\mathbf{s}_{i+1} - \mathbf{x}_i(t_{i+1}; \mathbf{s}_i, \mathbf{q}_i) = \mathbf{0}_{n_\mathbf{x}}, \qquad i = 0, \cdots, N-1, \tag{2.22c}$$

$$\mathbf{h}(\mathbf{s}_i, \mathbf{q}_i) \leq \mathbf{0}_{n_\mathbf{h}}, \qquad i = 0, \cdots, N, \tag{2.22d}$$

$$\mathbf{r}(\mathbf{s}_N) \leq \mathbf{0}_{n_\mathbf{r}}, \tag{2.22e}$$

where the combined state and control trajectories can be represented as $X = [\mathbf{s}_0^T, \cdots, \mathbf{s}_N^T]^T$ and $U = [\mathbf{q}_0^T, \cdots, \mathbf{q}_{N-1}^T]^T$, respectively. Note that when the above NLP is directly solved utilizing a nonlinear optimization framework, the X and

U represent a feasible state and control trajectory only at convergence [4]. This statement can be further visualized in Fig. 2.1, which simultaneously displays unconverged (Fig. 2.1a) and converged (Fig. 2.1b) multiple shooting trajectories for a scalar case, i.e., $s, q \in \mathbb{R}$.

Now, by applying the SQPr method (presented in Section 2.1.4) to the above NLP, the following structured QP subproblem is obtained which is solved at each sampling iteration [4]:

$$\min_{\Delta X, \Delta U} \sum_{i=0}^{N-1} \left\{ \frac{1}{2} \begin{bmatrix} \Delta \mathbf{s}_i \\ \Delta \mathbf{q}_i \end{bmatrix}^T \mathbf{H}_i \begin{bmatrix} \Delta \mathbf{s}_i \\ \Delta \mathbf{q}_i \end{bmatrix} + \mathbf{g}_i^T \begin{bmatrix} \Delta \mathbf{s}_i \\ \Delta \mathbf{q}_i \end{bmatrix} \right\} + \frac{1}{2} \Delta \mathbf{s}_N^T \mathbf{H}_N \Delta \mathbf{s}_N + \mathbf{g}_N^T \Delta \mathbf{s}_N \tag{2.23a}$$

$$\text{s.t. } \Delta \mathbf{s}_0 - \Delta \hat{\mathbf{s}}_0 = \mathbf{0}_{n_\mathbf{x}}, \tag{2.23b}$$

$$\mathbf{c}_i + \frac{\partial \mathbf{x}_i(t_i; \bar{\mathbf{s}}_i, \bar{\mathbf{q}}_i)}{\partial (\mathbf{s}_i, \mathbf{q}_i)} \begin{bmatrix} \Delta \mathbf{s}_i \\ \Delta \mathbf{q}_i \end{bmatrix} - \Delta \mathbf{s}_{i+1} = \mathbf{0}_{n_\mathbf{x}}, \qquad i = 0, \cdots, N-1, \tag{2.23c}$$

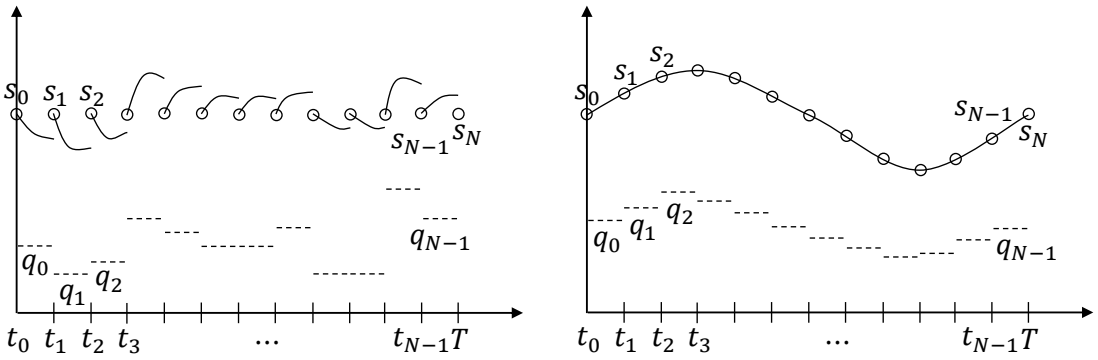
$$\mathbf{h}_i + \frac{\partial \mathbf{h}(\bar{\mathbf{s}}_i, \bar{\mathbf{q}}_i)}{\partial (\mathbf{s}_i, \mathbf{q}_i)} \begin{bmatrix} \Delta \mathbf{s}_i \\ \Delta \mathbf{q}_i \end{bmatrix} \leq \mathbf{0}_{n_\mathbf{h}}, \qquad i = 0, \cdots, N-1, \tag{2.23d}$$

$$\mathbf{r} + \frac{\partial \mathbf{r}(\bar{\mathbf{s}}_N)}{\partial \mathbf{s}_N} \Delta \mathbf{s}_N \leq \mathbf{0}_{n_\mathbf{r}}, \tag{2.23e}$$

where $\bar{X} = [\bar{\mathbf{s}}_0^T, \cdots, \bar{\mathbf{s}}_N^T]^T$ and $\bar{U} = [\bar{\mathbf{q}}_0^T, \cdots, \bar{\mathbf{q}}_{N-1}^T]^T$ are the current state and control trajectories that are selected as the linearization points for the problem functions. Also, the vectors $\mathbf{g}_i \triangleq \nabla l_i(\bar{\mathbf{s}}_i, \bar{\mathbf{q}}_i)$, $\mathbf{g}_N \triangleq \nabla E(\bar{\mathbf{s}}_N)$, $\Delta \hat{\mathbf{s}}_0 \triangleq \hat{\mathbf{s}}_0 - \bar{\mathbf{s}}_0$, $\mathbf{c}_i \triangleq \mathbf{x}_i(t_i; \bar{\mathbf{s}}_i, \bar{\mathbf{q}}_i) - \bar{\mathbf{s}}_{i+1}$, $\mathbf{h}_i \triangleq \mathbf{h}(\bar{\mathbf{s}}_i, \bar{\mathbf{q}}_i)$, and $\mathbf{r} \triangleq \mathbf{r}(\bar{\mathbf{s}}_N)$ are defined for notational convenience. The terminal cost matrix is taken as $\mathbf{H}_N \triangleq \nabla^2 E(\bar{\mathbf{s}}_N)$, while the state Hessian $\mathbf{H}_i$ for $i = 0, \cdots, N-1$, depends on the utilized approximation technique. In addition, the Lagrangian for the NLP (2.22) can be expressed as:

$$\begin{aligned} \mathcal{L}(X, U, \lambda, \nu) = &\sum_{i=0}^{N-1} l_i(\mathbf{s}_i, \mathbf{q}_i) + E(\mathbf{s}_N) + \lambda_{-1}^T (\mathbf{s}_0 - \hat{\mathbf{s}}_0) + \nu_N^T \mathbf{r}(\mathbf{s}_N) + \\ &\sum_{i=0}^{N-1} \lambda_i^T \big(\mathbf{x}_i(t_i; \mathbf{s}_i, \mathbf{q}_i) - \mathbf{s}_{i+1} \big) + \sum_{i=0}^{N-1} \nu_i^T \mathbf{h}(\mathbf{s}_i, \mathbf{q}_i), \end{aligned} \tag{2.24}$$

where λ_i for $i = 0, \cdots, N-1$ represent the multipliers for the continuity constraints in (2.22c); multiplier for the initial value constraint in (2.22b) is denoted by λ_{-1}; multipliers for the inequality constraints in (2.22d) and (2.22e) are denoted by ν_i



(A) Unconverged trajectory (B) Converged trajectory

FIGURE 2.1: Illustration of multiple shooting state and control trajectories for a scalar case.

for $i = 0, \cdots, N$. In case of the exact Hessian SQP method, $\mathbf{H}_i$ constitutes the following second-order derivatives,

$$\mathbf{H}_i \triangleq \nabla^2_{(\mathbf{s}_i, \mathbf{q}_i)} \mathcal{L}(\bar{X}, \bar{U}, \bar{\lambda}, \bar{\nu}). \tag{2.25}$$

Moreover, for a common least-squares type objective: $L(\mathbf{s}_i, \mathbf{q}_i) = \frac{1}{2} \left\| \zeta(\mathbf{s}_i, \mathbf{q}_i) \right\|_2^2$, the GGN Hessian approximation results as follows:

$$\mathbf{H}_i^{\text{GGN}} \triangleq \nabla \zeta(\bar{\mathbf{s}}_i, \bar{\mathbf{q}}_i) \nabla \zeta(\bar{\mathbf{s}}_i, \bar{\mathbf{q}}_i)^T, \tag{2.26}$$

where $\zeta(\,\cdot\,)$ represents the previously defined residual function.

2.2.2 Realtime Solution Approach

Knowing the fact that the optimization problems of (N)MPC and (nonlinear) moving horizon estimator (MHE) have similar forms, both are solved utilizing the same solution technique in this thesis. In summary, firstly, a continuous-time OCP reduces to a discretized NLP utilizing the direct multiple shooting method. Secondly, a sequential quadratic program is obtained out of the discretized NLP via linearization. Thereafter, with the help of the GGN method, the solution to the previously obtained SQP is computed. This whole process is repeated at sampling time with each new measurement, as also depicted in Fig. 2.2.

To solve the obtained SQP, a special realtime iteration scheme proposed in [71] is utilized. The RTI method is based on an SQPr-based algorithm that recursively

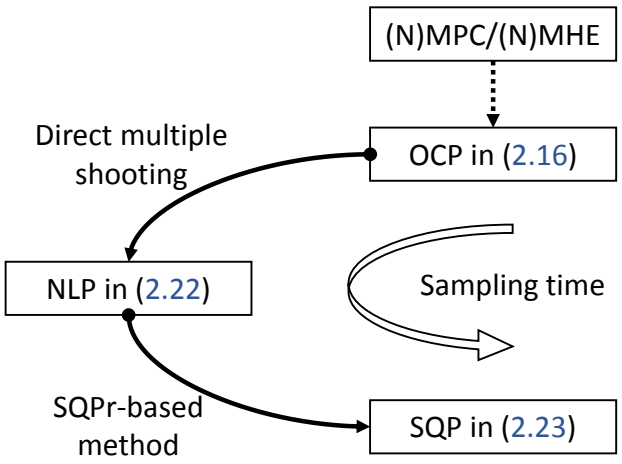


FIGURE 2.2: Overview of the numerical approach for realtime optimal control.

obtains a solution for the discretized NLP (2.22). For online feasibility, it directly performs iterations for the new problem – with current measurements –, rather than performing iterations for the old problem. This results in the execution of a single SQP type iteration per sampling time. Furthermore, each of these iteration is distributed into a *preparation* phase and a *feedback* phase, within the RTI scheme. The computationally intensive preparation phase is performed with the predicted state, without waiting for current measurement. As soon as measurement becomes available, the feedback phase immediately delivers a solution to the new problem by solving the already prepared convex SQP subproblem. Moreover, since linearization is performed with the previous instant's solution guess while incorporating current measurement as the initial value constraint, the RTI scheme realizes a combined predictor-corrector solution at the cost of only one QP solution [72].

Remark 4. It is to be noted that the adopted solution methodology, i.e., the direct multiple shooting technique along with a single iteration per sampling solution of the GGN method has a limitation of obtaining only the local solution in the vicinity of the previous guess. Hence, not much can be implied about the optimality (local/global) of the obtained solution without performing extensive analysis. Nevertheless, utilizing a high sampling rate for the closed-loop and long enough estimation window length positively contribute to the stability of the overall method. Another important point to note here is that the single iteration solution negatively impacts the convergence properties of these methods. That is, with every new reference value, it takes a few iterations before the solution could converge to the actual (sub)optimal point which eventually increases the response time for these methods.

2.2.3 Available Solution Tools and Methodologies

In general, the optimization problem of MPC (and MHE) can be solved by utilizing two methods, namely, explicit and implicit. Within the explicit methodology, the MPC implementation reduces to an offline optimization problem which is to be solved for all states in a feasible set `S` (or any of its subset). This eventually yields explicit expressions for the control law along with the objective function that are computed online utilizing the feedback of the current state. Several software packages exist that are specifically tailored for explicit MPC applications. They include `MPT3` [73] and `PAROC` [74], which are based on multiparametric programming to obtain explicit MPC solutions. While `MPT3` is a direct MATLAB based toolbox, `PAROC` is a standalone software associated with `POP` [74] which is yet another MATLAB based toolbox. Besides, `MATLAB Model Predictive Control` toolbox [75] based on quadratic programming (QPr) provides the design and analysis of explicit MPC along with the generation of C-codes for rapid implementation, as are also obtained with `MPT3`.

In stark contrast, implicit MPC methods result in an online optimization problem that is to be solved recursively for optimal state and control trajectory incorporating the feedback of the current controlled states. Amongst various available software packages, μ`AO-MPC` [76] and `qpOASES` [77] are the two widely used QPr-based tools with MATLAB/Simulink interface. While μ`AO-MPC` is a Python-based software that incorporates augmented Lagrangian method which makes it suitable for micro-controller applications, `qpOASES` is an open-source, C++-based implementation of the online active-set[2] strategy. `FORCES Pro` [79] is another software tool for solving complex optimization problems in milliseconds. It supports parametric programs and SQPs and provides interfaces for MATLAB/Simulink, C, and Python. Besides, few other MATLAB-based software packages targeting linear systems are `jMPC` toolbox [80], `fast_mpc` [81], and `PnPMPC` [82].

Efficient yet fast solution of QPs is mandatory for realtime application of (N)MPC. In that vein, GNU Octave-based `MPC Tools Package` [83] employs SQPr approach to provide a control and estimation tool for linear and nonlinear models. The `Automatic Control and Dynamic Optimization (ACADO)` toolkit [84] which is written in C/C++ is another software package that contributes a convenient tool

[2]They are amongst the widely utilized solution methods for solving constrained convex optimization problems. Interested readers are referred to [78] for a thorough discussion.

for nonlinear MPC with a range of algorithms for the direct optimal control scheme. In addition, C/C++ and Fortran-based `MUSCOD-II` [85] and MATLAB-based `OptCon` [86] are software packages that rely on multiple shooting method and provide real-time environments for application on nonlinear systems. Moreover, numerous other software packages are used to define and solve OCPs. They include `BLOM` [87] which is a software package that facilitates the modeling of nonlinear systems for MPC application and employs `IPOPT` [88] and MATLAB's `fmincon` solvers, `PROPT MATLAB Optimal Control Software` [89] which provides a platform for solving applied OCPs utilizing third-party NLP solvers, and `ICLOCS` [90] which is a MATLAB-based software that solves nonlinear problems incorporating path and boundary constraints.

On top of the MPC-specific solvers, several generic optimization software tools also contribute potential contenders for realtime implementation of MPC. The two most promising candidates that employ primal-dual interior-point[3] methods are `CVXGEN` [91] and `ECOS` [92]. The reason for their widespread utilization is the underlying custom code generation capability which is essential for embedded optimization. While `CVXGEN` is developed for treating small convex QPs, `ECOS` is written in ANSI C and primarily focuses on embedded applications. Furthermore, Table 2.1 summarizes the most promising solution tools amongst the aforementioned along with some additional information such as implementation type, the order of solution time, and availability.

Finally, amongst the aforementioned tools, the ACADO toolkit [93] is adopted as a solution platform in this thesis. This is essentially due to its incorporation of the direct multiple shooting method and the RTI approach altogether. In essence, the ACADO toolkit is an open-source C++ based software environment for automatic control and dynamic optimization applications. It provides an exhaustive code library that links the `qpOASES` solver to solve sequential quadratic subproblems that are obtained from the direct multiple shooting method. Besides, it has a code generation package that creates tailored (well structured to avoid redundant computations), self-contained C-codes based on a user-specified symbolic (N)MPC (or (N)MHE) problem formulation. Moreover, it is capable of running on embedded systems with limited processing power and can also be interfaced with other software platforms including `MATLAB/Simulink` [94]. Its usage can be summarized

[3] A type of classical interior point methods which form a second important solution class to solve convex QPs. Interested readers are referred to [57] for a thorough discussion.

TABLE 2.1: List of the available solution tools for realtime implementation of MPC [5], wherein μs and ms signify microseconds and milliseconds, respectively.

Tool	Implementation	Solution Time	Open Source
PAROC	Explicit	$< \mu$s	No
MPT3	Explicit	$< \mu$s	Yes
Matlab MPC Toolbox	Explicit	$< \mu$s	No
μAO-MPC	Implicit	μs - ms	Yes
qpOASES	Implicit	ms	Yes
fast_mpc	Implicit	ms	Yes
ACADO	Implicit	ms	Yes
MUSCOD-II	Implicit	ms	Yes
CVXGEN	Implicit	μs - ms	Yes
ECOS	Implicit	ms	Yes
FORCES Pro	Implicit	ms	No
MPC Tools Package	Implicit	ms	Yes
OptCon	Implicit	ms	No
BLOM	Implicit	ms	Yes

as: firstly, the optimization problems in terms of system equations and constraints are defined in a C++ environment and then, the self-contained C codes are obtained utilizing its code generation package. Later, these generated C codes can be imported to any computational platform for their final execution.

2.3 Dynamic Modeling of Aerial Robots

This section deduces the dynamic models of multirotor aerial robots. For modeling in general, a multirotor is considered to be a rigid body having multiple rotors depending on the configuration, for instance, a quadrotor has four rotors while a tilt-rotor tricopter has two stationary rotors and one tilting rotor. Both the quadrotor (with x-configuration) and tilt-rotor aerial robots are illustrated in Figs. 2.3 and 2.4, respectively, wherein FRr is the front-right rotor, FLr is the front-left rotor, BRr is the back-right rotor, BLr is the back-left rotor, and Br is back rotor.

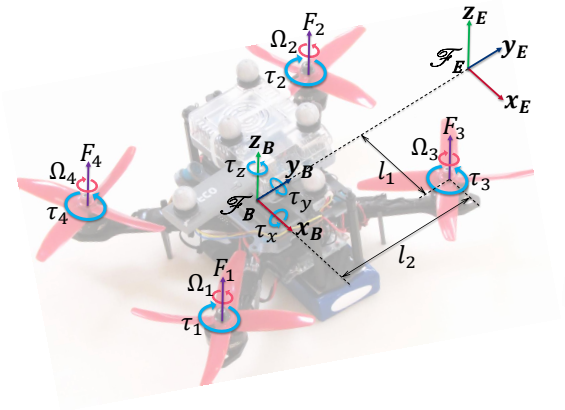


FIGURE 2.3: Quadrotor aerial robot, wherein FRr and BLr are rotating counter-clockwise and FLr and BRr are rotating clockwise.

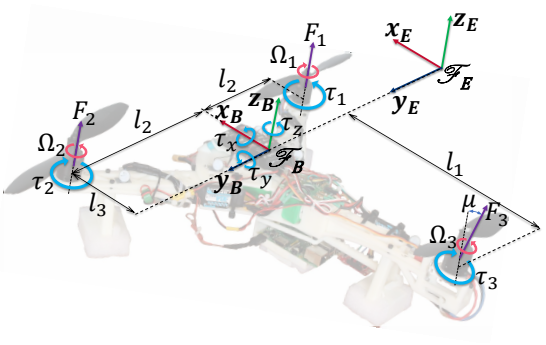


FIGURE 2.4: Tilt-rotor tricopter aerial robot, wherein FLr and Br (tilting rotor) are rotating counter-clockwise and FRr is rotating clockwise.

2.3.1 Kinematic Equations

They are obtained by applying coordinate transformations between Earth-fixed frame $\mathcal{F}_E$ and body frame $\mathcal{F}_B$:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \mathbf{R}_{EB} \begin{bmatrix} u \\ v \\ w \end{bmatrix}; \quad \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \mathbf{T}_{EB} \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \tag{2.27}$$

where x, y, z and ϕ, θ, ψ are the translational position and rotational attitude, respectively, that are defined in frame $\mathcal{F}_E$; u, v, w and p, q, r are the translational and rotational velocities that are defined in frame $\mathcal{F}_B$; $\mathbf{R}_{EB}$ is the translation transformation matrix between frames $\mathcal{F}_E$ and $\mathcal{F}_B$, whereas $\mathbf{T}_{EB}$ maps the rotational velocity component from $\mathcal{F}_B$ to $\mathcal{F}_E$ ($c : cos$, $s : sin$, $t : tan$):

$$\mathbf{R}_{EB} = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - s\psi c\phi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\psi c\phi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix}, \quad \mathbf{T}_{EB} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & \frac{s\phi}{c\theta} & \frac{c\phi}{c\theta} \end{bmatrix}.$$

2.3.1.1 Rigid-body Equations

The rigid-body dynamic equations are derived based on Newton-Euler formulation in the body coordinate system (cf. [95]). Within these equations, the multirotor is assumed to be a point mass, wherein all the forces act at the center of gravity. In

general, they are categorized as:

Force equations

$$\dot{u} = rv - qw + gsin(\theta) + \frac{1}{m}F_x, \tag{2.28a}$$

$$\dot{v} = pw - ru - gsin(\phi)cos(\theta) + \frac{1}{m}F_y, \tag{2.28b}$$

$$\dot{w} = qu - pv - gcos(\phi)cos(\theta) + \frac{1}{m}F_z, \tag{2.28c}$$

Moment equations

$$\dot{p} = \left(\frac{1}{I_{xx}I_{zz} - I_{xz}^2}\right)\Big[\Big\{ - pq(I_{xz}) + qr(I_{yy} - I_{zz})\Big\}I_{zz} - \Big\{qr(I_{xz}) + pq(I_{xx} - I_{yy})\Big\}I_{xz} + \tau_x(I_{zz}) - \tau_z(I_{xz})\Big], \tag{2.29a}$$

$$\dot{q} = pr\left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right) - (r^2 - p^2)\left(\frac{I_{xz}}{I_{yy}}\right) + \tau_y\left(\frac{1}{I_{yy}}\right), \tag{2.29b}$$

$$\dot{r} = \left(\frac{1}{I_{xx}I_{zz} - I_{xz}^2}\right)\Big[\Big\{qr(I_{xz}) + pq(I_{xx} - I_{yy})\Big\}I_{xx} - \Big\{ - pq(I_{xz}) + qr(I_{yy} - I_{zz})\Big\}I_{xz} + \tau_z(I_{xx}) - \tau_x(I_{xz})\Big], \tag{2.29c}$$

where F_x, F_y, F_z are the total external forces and τ_x, τ_y, τ_z are the total external moments acting in frame $\mathcal{F}_B$. In addition, I_{xx}, I_{yy}, I_{zz}, and I_{xz} represent the moments of inertia of the multirotor along axes $\mathcal{F}_{B_x}$, $\mathcal{F}_{B_y}$, $\mathcal{F}_{B_z}$, and $\mathcal{F}_{B_{xz}}$, respectively. Recall that a quadrotor has a dual plane of symmetry, thus $I_{xz} = I_{yz} = 0$, whereas a tilt-rotor tricopter only has a single plane of symmetry, hence $I_{yz} = 0$ but $I_{xz} \neq 0$.

2.3.1.2 External Forces and Moments

Using the momentum theory, steady-state thrust and drag-moment generated by a hovering rotor can be modeled as:

$$F_i = K_F\Omega_i^2, \quad \tau_i = K_\tau\Omega_i^2, \tag{2.30}$$

where F_i and τ_i are the force and drag-moment generated by i^{th} rotor with Ω_i angular velocity, and K_F and K_τ are the respective force and drag-moment coefficients. While the expressions for total external force and moment acting in body

frame for a quadrotor (Fig. 2.3) are:

$$\mathbf{F}^{ext} = \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}, \quad \boldsymbol{\tau}^{ext} = \begin{bmatrix} \{F_2 + F_3 - (F_1 + F_4)\} l_2 \\ \{F_2 + F_4 - (F_1 + F_3)\} l_1 \\ \tau_3 + \tau_4 - (\tau_1 + \tau_2) \end{bmatrix}, \quad (2.31)$$

the corresponding expressions for a tilt-rotor tricopter (Fig. 2.4) are as follows ($c : cos$, $s : sin$):

$$\mathbf{F}^{ext} = \begin{bmatrix} 0 \\ -F_3 s(\mu) \\ F_1 + F_2 + F_3 c(\mu) \end{bmatrix}, \quad \boldsymbol{\tau}^{ext} = \begin{bmatrix} (F_2 - F_1) l_2 \\ (F_3 c(\mu)) l_1 - (F_1 + F_2) l_3 + \tau_3 s(\mu) \\ \tau_1 - \tau_2 - \tau_3 c(\mu) + (F_3 s(\mu)) l_1 \end{bmatrix}, \quad (2.32)$$

wherein μ represents the back-rotor tilting angle for the tricopter, and $\mathbf{F}^{ext} = [F_x, F_y, F_z]^T$ and $\boldsymbol{\tau}_{ext} = [\tau_x, \tau_y, \tau_z]^T$.

For most of the applications in this thesis, the above dynamic models are discretized based on direct multiple shooting method, utilizing a shooting grid size of $T_s = 0.01s$. Additionally, the explicit Runge-Kutta 4^{th} order integrator, with 2-steps per shooting interval is incorporated. Finally, the overall model in a discrete-time form can be written as:

$$\mathbf{x}_{k+1} = \mathbf{f}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{g}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k), \quad \mathbf{z}_k = \mathbf{x}_k + \boldsymbol{\nu}_k, \quad (2.33)$$

where the state vector $\mathbf{x} \in \mathbb{R}^{12} = [x, y, z, u, v, w, \phi, \theta, \psi, p, q, r]^T$, and the control vector $\mathbf{u} \in \mathbb{R}^4$ for a quadrotor is $[\Omega_1, \Omega_2, \Omega_3, \Omega_4]^T$ and for a tilt-rotor tricopter is $[\Omega_1, \Omega_2, \Omega_3, \mu]^T$. The term $\mathbf{z} \in \mathbb{R}^{12}$ represents the measurement vector that includes zero-mean white Gaussian noise $\boldsymbol{\nu} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{\nu}})$. Recall that precise identification of the true dynamics is challenging due to the uncertainties or time-varying disturbances acting on the system. Therefore, the overall model is considered to be comprised of an *a priori* known nominal function $\mathbf{f}_{\mathtt{d}}(\cdot, \cdot) : \mathbb{R}^{12} \times \mathbb{R}^4 \mapsto \mathbb{R}^{12}$ and an *uncertain/unknown* function $\mathbf{g}_{\mathtt{d}}(\cdot, \cdot) : \mathbb{R}^{12} \times \mathbb{R}^4 \mapsto \mathbb{R}^{12}$.

Chapter 3

Instantaneous Learning-based Nonlinear Model Predictive Control

This chapter discusses the instantaneous learning control technique. To recall, the InLC scheme utilizes an estimation algorithm in conjunction with the controller in order to identify the uncertain/time-varying model parameters. Subsequently, these learned parameters update the incorporated model within the controller. Although the extended Kalman filter (EKF) is a popular choice, the suboptimal performance often underlies the linearization of the nonlinear dynamics. On the contrary, the NMHE solves an optimization problem over a past window to obtain the current estimate. In this chapter, the NMHE-based learning is preferred due to the two main reasons, firstly, its ability to incorporate constraints and secondly, it utilizes a range of data for estimating the current parameter, unlike EKF which utilizes just the data at the previous instant.

The proposed learning-based NMPC is tested for the trajectory tracking control of an aerial robot, operating in a disturbed environment. The enticing feature of this control framework is the learning part that makes it adaptive to the changing working conditions. To thoroughly investigate the performance of NMHE in learning the disturbances, two case studies are illustrated in this chapter:

1. Position control in the presence of external disturbances: the tracking performance of learning-based NMPC – high-level NMPC along with NMHE that learns the external disturbances on the system – is investigated for three different disturbance scenarios.

2. Cascade control for the uncertain system model: the tracking performance of a cascade NMPC structure – high-level NMPC (H-NMPC) for position control, low-level NMPC (L-NMPC) for attitude control, and NMHE to learn two crucial aerodynamic parameters – is investigated for a square-shaped trajectory.

Within both the aforementioned case studies, a significant change in the robot's model is expected, thus justifies the use of learning-based NMPC.

The outline of this chapter is as follows: Section 3.1 presents other instantaneous learning-based control schemes along with the related recent research. The problem formulations of NMPC and NMHE are described in Sections 3.2 and 3.3, respectively. Thereafter, two case studies for the implementation of the NMPC-NMHE framework are illustrated in Section 3.4. Lastly, some conclusions are drawn in Section 3.5.

3.1 Literature Overview

Numerous implementations have been demonstrated for the instantaneous learning-based scheme, incorporating various estimators and/or disturbance observers (DOs). A polytropic nonlinear DO is implemented with unknown inputs for the landing application of an aerial robot on a moving platform in [96]. In [97], a state-observation and control problem for a bi-tethered aerial system is considered, wherein a nonlinear state estimator based on the high gain- and Luenberger-observers are utilized. A nonlinear DO is adopted for resilient control of a quadrotor utilizing a backstepping controller in [98]. A hierarchical controller employing a new DO with finite time convergence is proposed for a path tracking control of a coaxial aerial robot in [99]. A DO-based control with anti-windup of a small fixed-wing aerial robot is demonstrated for disturbance rejection in [100]. In [101], a decoupling controller based on dynamic surface control for quadrotors is proposed, wherein a second-order sliding mode-type DO is utilized to restrain the influence of system uncertainties and external disturbances.

Other applications of the instantaneous learning-based scheme include an onboard control of a quadrotor using modified EKF for estimation in [102], centralized control of an autonomous trailer-tractor system utilizing NMHE-based estimation of

slip parameters in [16], and centralized control of quadrotor incorporating NMHE-based estimation of the ceiling effect in [103]. Besides, [104] depicts the Authors' previous work, wherein control of a 3 degree of freedom helicopter setup utilizing NMHE-based estimation of aerodynamic coefficients is achieved. Moreover, it compares the performance of NMHE-based estimation over the EKF-based estimation and validates the supremacy of the former over the latter.

Furthermore, a simple learning strategy to improve the tracking performance of the traditional feedback linearization controller has been proposed in [105, 106]. In essence, the proposed strategy instantaneously updates the controller gains and disturbance estimate within the feedback control law via a gradient descent technique. The overall control framework is implemented for the tracking control of a tilt-rotor tricopter aerial robot and is illustrated to outperform the traditional controller.

3.2 Nonlinear Model Predictive Controller

The NMPC is an advanced, dynamic optimization-based strategy that computes an optimal control action by optimizing the model's behavior over a finite window, often recalled as prediction horizon (N_c). To elaborate, this optimal system forecast results from an open-loop optimization problem which is represented in the form of a constrained, finite horizon OCP. The solution to this OCP for the current state results in an optimal sequence of control actions (over the horizon), the first term of which is regarded as the optimal control action [5]. Moreover, the finite prediction window recedes forward in time, which results in a *receding horizon control* technique, as illustrated in Fig. 3.1.

For the considered tracking applications, the parametric OCP incorporates a least-square form that penalizes the deviations of predicted state and control trajectories

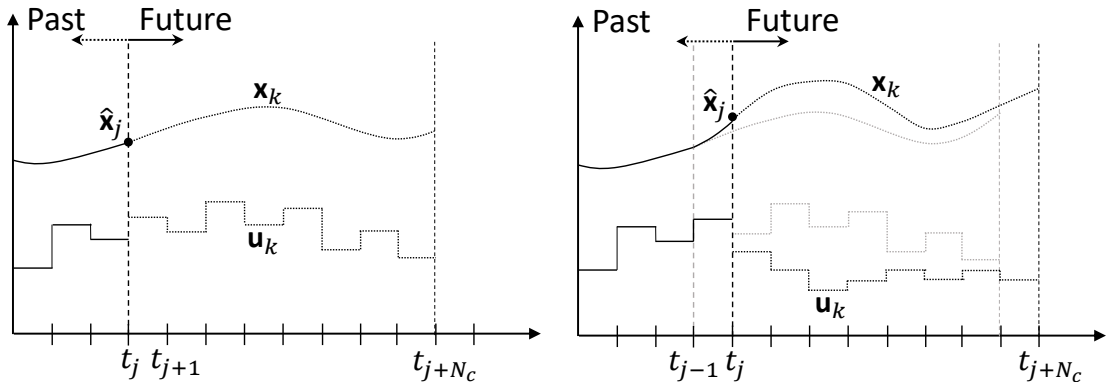


FIGURE 3.1: Illustration of receding horizon control principle for two successive instants.

from their specified references over the given prediction horizon ($[t_j, t_{j+N_c}]$). Therefore, the underlying discrete-time optimization problem is of the form:

$$\min_{\mathbf{x}_k, \mathbf{u}_k} \frac{1}{2} \left\{ \sum_{k=j}^{j+N_c-1} \left(\|\mathbf{x}_k - \mathbf{x}_k^{\mathrm{r}}\|^2_{\mathbf{W_x}} + \|\mathbf{u}_k - \mathbf{u}_k^{\mathrm{r}}\|^2_{\mathbf{W_u}} \right) + \left\|\mathbf{x}_{N_c} - \mathbf{x}_{N_c}^{\mathrm{r}}\right\|^2_{\mathbf{W}_{N_c}} \right\} \tag{3.1a}$$

$$\text{s.t. } \mathbf{x}_j = \hat{\mathbf{x}}_j, \tag{3.1b}$$

$$\mathbf{x}_{k+1} = \mathbf{f}_{\mathrm{d}}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{g}_{\mathrm{d}}(\mathbf{x}_k, \mathbf{u}_k), \qquad k = j, \cdots, j + N_c - 1, \tag{3.1c}$$

$$\mathbf{x}_{k,\min} \leq \mathbf{x}_k \leq \mathbf{x}_{k,\max}, \qquad k = j, \cdots, j + N_c, \tag{3.1d}$$

$$\mathbf{u}_{k,\min} \leq \mathbf{u}_k \leq \mathbf{u}_{k,\max}, \qquad k = j, \cdots, j + N_c - 1, \tag{3.1e}$$

where $\mathbf{x}_k \in \mathbb{R}^{n_\mathbf{x}}$ is the differential state, $\mathbf{u}_k \in \mathbb{R}^{n_\mathbf{u}}$ is the control output, and $\hat{\mathbf{x}}_j \in \mathbb{R}^{n_\mathbf{x}}$ is the current state estimate; time-varying state and control references are denoted by $\mathbf{x}_k^{\mathrm{r}}$ and $\mathbf{u}_k^{\mathrm{r}}$, respectively; terminal state reference is represented by $\mathbf{x}_{N_c}^{\mathrm{r}}$; $\mathbf{W_x} \in \mathbb{R}^{n_\mathbf{x} \times n_\mathbf{x}}$, $\mathbf{W_u} \in \mathbb{R}^{n_\mathbf{u} \times n_\mathbf{u}}$, and $\mathbf{W}_{N_c} \in \mathbb{R}^{n_\mathbf{x} \times n_\mathbf{x}}$ are the corresponding positive (semi)-definite weight matrices that are assumed constant in this thesis. However, their time-varying formulations can also be included in a similar manner. Furthermore, $\mathbf{x}_{k,\min} \leq \mathbf{x}_{k,\max} \in \mathbb{R}^{n_\mathbf{x}}$ and $\mathbf{u}_{k,\min} \leq \mathbf{u}_{k,\max} \in \mathbb{R}^{n_\mathbf{u}}$, specify the lower and upper bounds on the states and control outputs, respectively.

The last expression in (3.1a) is the terminal penalty term which signifies the cost incurred due to the finite nature of the prediction horizon. This term is included in the problem formulation to retain the advantages of the infinite horizon approximation. Consequently, the stability, robustness, and recursive feasibility is ensured [107]. In essence, the system states stay in an initial feasible set, wherein the admissible controls ensure the reachability through the prediction horizon. Moreover,

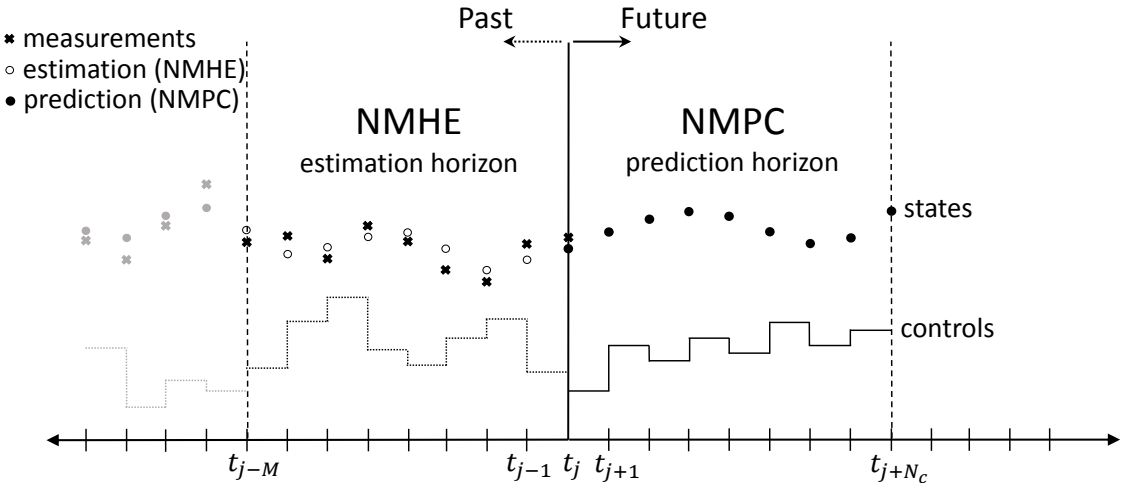


FIGURE 3.2: Illustration of estimation horizon for NMHE and prediction horizon for NMPC at a given time instant t_j [4].

selection of a wider terminal set and/or longer prediction horizon facilitates a larger initial feasible set. Hence, the stability guarantee of the closed loop can be realized by imposing terminal constraints and/or a corresponding terminal cost [108]. Another way of ensuring stability is via a problem formulation with a sufficiently long horizon [109]. Besides, a locally stabilizing control law that acts as an additional prediction horizon also retains the stability [110]. One may note that the stability and optimality proofs of NMPC are not explicitly included in this thesis, however, the interested readers are referred to [111, 112].

3.3 Nonlinear Moving Horizon Estimator

Typically, the NMHE is designed as a dual problem to NMPC, thereby utilizing the same optimization problem structure. However, the three main differences in the OCP formulation of NMHE over NMPC include: (i) instead of future predictions, it utilizes the past measurements over estimation horizon (M), (ii) there is no initial state constraint like (3.1b), and (iii) the optimization variables are the states and unknown system parameters only, excluding the control outputs. Moreover, the correlation between NMPC and NMHE is illustrated in Fig. 3.2 that simultaneously displays the estimation horizon (of length M) and prediction horizon (of length N_c) for the current time t_j.

Hence, similar to NMPC, the NMHE is formulated in a least-square form which penalizes the deviation of estimated outputs from measurements. Note that $\mathsf{L} \triangleq j-M+1$ is denoted in the following expressions for notational convenience. Additionally, to accommodate model mismatch in the form of process noise, a suitable component (arrival cost) is included in the optimization problem formulation of NMHE. Finally, the discrete-time dynamic optimization problem to estimate the constrained states ($\hat{\mathbf{x}}$) and the unknown parameter ($\hat{\mathbf{p}}$) at time t_j using the process model, measurement model, and available measurements within the horizon, is of the form:

$$\min_{\hat{\mathbf{x}}_k, \hat{\mathbf{p}}} \left\{ \left\| \begin{matrix} \hat{\mathbf{x}}_{\mathsf{L}} - \bar{\mathbf{x}}_{\mathsf{L}} \\ \hat{\mathbf{p}} - \bar{\mathbf{p}}_{\mathsf{L}} \end{matrix} \right\|^2_{\mathbf{P}_{\mathsf{L}}} + \sum_{k=\mathsf{L}}^{j} \|\mathbf{z}_k - \hat{\mathbf{x}}_k\|^2_{\mathbf{W}_{\boldsymbol{\nu}}} + \sum_{k=\mathsf{L}}^{j-1} \|\boldsymbol{\omega}_k\|^2_{\mathbf{W}_{\boldsymbol{\omega}}} \right\} \tag{3.2a}$$

$$\text{s.t. } \hat{\mathbf{x}}_{k+1} = \mathbf{f}_{\mathsf{d}}(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{p}) + \mathbf{g}_{\mathsf{d}}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}) + \boldsymbol{\omega}_k, \qquad k = \mathsf{L}, \cdots, j-1, \tag{3.2b}$$

$$\hat{\mathbf{x}}_{k,\min} \leq \hat{\mathbf{x}}_k \leq \hat{\mathbf{x}}_{k,\max}, \qquad k = \mathsf{L}, \cdots, j, \tag{3.2c}$$

$$\hat{\mathbf{p}}_{\min} \leq \hat{\mathbf{p}} \leq \hat{\mathbf{p}}_{\max}, \tag{3.2d}$$

where $\boldsymbol{\omega}_k$ represents the added process noise; $\hat{\mathbf{x}}_{k,\min} \leq \hat{\mathbf{x}}_{k,\max}$ and $\hat{\mathbf{p}}_{\min} \leq \hat{\mathbf{p}}_{\max}$ specify the lower and upper bounds on the estimated states and parameters, respectively. $\mathbf{P}_{\mathsf{L}}$, $\mathbf{W}_{\boldsymbol{\nu}}$, and $\mathbf{W}_{\boldsymbol{\omega}}$ are the respective weight matrices that are taken as the inverse of the corresponding covariance matrices:

$$\mathbf{P}_{\mathsf{L}} = \boldsymbol{\Sigma}_{\boldsymbol{\omega},0}^{-\frac{1}{2}}, \quad \mathbf{W}_{\boldsymbol{\nu}} = \boldsymbol{\Sigma}_{\boldsymbol{\nu}}^{-\frac{1}{2}}, \quad \mathbf{W}_{\boldsymbol{\omega}} = \boldsymbol{\Sigma}_{\boldsymbol{\omega}}^{-\frac{1}{2}}, \tag{3.3}$$

where $\boldsymbol{\Sigma}_{\boldsymbol{\omega},0}$ is the initial process noise covariance matrix (incorporating state and parameter, both), $\boldsymbol{\Sigma}_{\boldsymbol{\nu}}$ is the measurement noise covariance matrix, and $\boldsymbol{\Sigma}_{\boldsymbol{\omega}}$ is the process noise covariance matrix. In addition, $\bar{\mathbf{x}}_{\mathsf{L}}$ and $\bar{\mathbf{p}}_{\mathsf{L}}$ denote the estimated state and parameter values (arrival cost data) at the start of the estimation horizon. One may note that the stability proofs of NMHE are also not explicitly included in this thesis. However, for the adopted least-square–type problem formulation, one may refer to [113].

Remark 5. Note that the rate of learning (or convergence) performance of NMHE is linked with the estimation window length which is mainly problem-specific. To elaborate, for fast dynamical systems like aerial robots, window length cannot be too high due to limited onboard computational resources. Another point to

take note is that the estimation accuracy may not always increase with the window length. This is essentially the case in those applications that involve highly stochastic uncertainties, while the selection of longer window length may eventually result in a deteriorated overall estimation quality.

3.4 Case Study

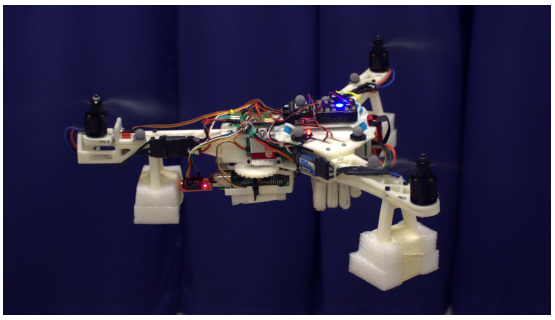
In this section, the two case studies are illustrated. For each case, first, a brief description of the problem statement is included followed by the NMPC and NMHE design. Thereafter, the obtained test results are presented along with their interpretation.

3.4.1 Position Control in the Presence of External Disturbances.

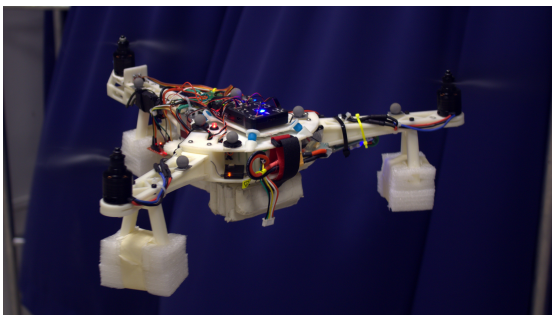
For the first case study, the NMPC is employed as a high-level position controller, while NMHE is designed to learn the time-varying robot model. To replicate the situation of an (unknown) outdoor testing environment, the tracking performance of the learning-based NMPC is investigated – and simultaneously compared with conventional NMPC – for three test case scenarios, incorporating different types of disturbances: (i) sudden mass variations encountered in package delivery applications, (ii) ground effect by flying in close proximity to the ground, and (iii) wind gust via two industrial fans.

3.4.1.1 Prototyped Aerial Robot

The aerial robot used in this application is a 3D-printed tilt-rotor tricopter. It is a fully custom-designed platform which is inspired by commercial Talon tricopter available in the market. All the required electronics are integrated into the frame design to provide a compact, stable, and lightweight system. Besides, the payload dropping mechanism mounted at the base comprises of two plates. Amongst them, one houses the servomotor, while the other holds the payload blocks to be dropped. These payload blocks are designed with hooks, which are positioned within slots



(A) Side view.



(B) Isometric view.

FIGURE 3.3: Prototyped 3D-printed tilt-rotor tricopter.

of the latter plate. In terms of operation, a circular gear attached to the servomotor drives a linear gear that results in a translational motion of a push rod. This translational motion eventually opens the interlocking for the blocks, thus allows them to be dropped sequentially. Moreover, the tricopter houses a Pixhawk flight controller for low-level stabilization along with an onboard embedded processor (Raspberry Pi 3) which executes all the control codes as well as controls the servomotor for the dropping mechanism. The two views of the assembled system in flight are presented in Fig. 3.3. As desired, it is a compact robot with span and length of 42.4 cm and 37.6 cm, respectively. Its overall weight is around 1.018 kg including all electronics except the battery.

The constant intrinsic parameters for the custom-designed tricopter are listed in Table 3.1. While parameters such as mass (m) and the arm lengths (l_1,l_2,l_3) are directly measured, the thrust and drag-moment coefficients are evaluated based on simple experimentation. It involves measuring thrust and drag-moment generated by the propeller for various rotor RPMs and later plotting them together in order to compute the slope of the plot. Regarding the details of this experiment, one may refer [114].

The incorporated discrete-time nonlinear model for the considered tricopter (shown in Fig. 2.4) at high-level is of the form:

$$\mathbf{x}_{k+1} = \mathbf{f}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}) + \mathbf{g}, \quad \mathbf{z}_k = \mathbf{x}_k + \boldsymbol{\nu}_k, \tag{3.4}$$

where the state vector $\mathbf{x} \in \mathbb{R}^6 = [x, y, z, u, v, w]^T$ and the control vector $\mathbf{u} \in \mathbb{R}^4 = [\phi, \theta, \psi, F_z]^T$. Note that the above expression is a modification of (2.33), wherein the term $\mathbf{g} \in \mathbb{R}^6$ represents a disturbance vector instead of the disturbance function

TABLE 3.1: Intrinsic parameters for the 3D-printed tilt-rotor tricopter.

Parameter	Description	Value
m	Mass of tricopter	1.442 kg
l_1	Moment arm	0.284 m
l_2	Moment arm	0.212 m
l_3	Moment arm	0.092 m
I_{xx}	MOI about $\mathcal{F}_{B_x}$	0.016053 kg-m^2
I_{yy}	MOI about $\mathcal{F}_{B_y}$	0.028158 kg-m^2
I_{zz}	MOI about $\mathcal{F}_{B_z}$	0.032752 kg-m^2
I_{xz}	MOI about $\mathcal{F}_{B_{xz}}$	0.029763 kg-m^2
J_P	MOI of propeller	6×10^{-5} kg-m^2
K_F	Force coefficient	3.76×10^{-5} N-s^2
K_τ	Drag-moment coefficient	2.56×10^{-6} Nm-s^2

which is given as: $\mathbf{g} = [0, 0, 0, F^{x_{dist}}, F^{y_{dist}}, F^{z_{dist}}]^T$, and $\mathbf{p} \in \mathbb{R}^3$ is the parameter vector. The terms $F^{x_{dist}}$, $F^{y_{dist}}$, $F^{z_{dist}}$ are the respective disturbance forces along the three directions. Also, the nominal function is $\mathbf{f_d}(\,\cdot\,,\,\cdot\,,\,\cdot\,) : \mathbb{R}^6 \times \mathbb{R}^4 \times \mathbb{R}^3 \mapsto \mathbb{R}^6$.

3.4.1.2 Instantaneous Learning-based NMPC

The optimal control outputs for the position tracking NMPC are computed utilizing the current feedback of the states. Then, these optimal inputs are given to the low-level controller as its desired setpoints. The low-level controller in Pixhawk has a P-PID architecture, which employs a P controller for the attitude and a PID controller for the angular rate. The following control gains are utilized in the design:

$$\begin{aligned}
&\mathrm{P}_{\text{roll}} = 6.5, \qquad \mathrm{P}_{\text{roll-rate}} = 0.1,\ \mathrm{I}_{\text{roll-rate}} = 0.05,\ \mathrm{D}_{\text{roll-rate}} = 0.002,\\
&\mathrm{P}_{\text{pitch}} = 6.5, \qquad \mathrm{P}_{\text{pitch-rate}} = 0.1,\ \mathrm{I}_{\text{pitch-rate}} = 0.05,\ \mathrm{D}_{\text{pitch-rate}} = 0.002,\\
&\mathrm{P}_{\text{yaw}} = 4.8, \qquad \mathrm{P}_{\text{yaw-rate}} = 0.15,\ \mathrm{I}_{\text{yaw-rate}} = 0.1,\ \mathrm{D}_{\text{yaw-rate}} = 0.
\end{aligned}$$

This low-level control architecture is utilized individually for each of the three axes (roll, pitch, and yaw), as their dynamics are decoupled in a standard multirotor aerial robot. Finally, the control outputs from the low-level controller are the desired moments (τ_x, τ_y, τ_z) in the respective directions. These moment commands along with the thrust command from NMPC are mapped to the actuator commands

for the tilt-rotor tricopter ($\mathbf{u}_{\text{P-PID}} = [\Omega_1, \Omega_2, \Omega_3, \mu]^T$) as follows:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 cos(\mu) \\ \Omega_3^2 sin(\mu) \end{bmatrix} = \begin{bmatrix} K_F & K_F & K_F & 0 \\ -K_F l_2 & K_F l_2 & 0 & 0 \\ -K_F l_3 & -K_F l_3 & K_F l_1 & K_\tau \\ K_\tau & -K_\tau & -K_\tau & K_F l_1 \end{bmatrix}^{-1} \begin{bmatrix} F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}, \tag{3.5}$$

Note that the above equation is obtained incorporating the external moment dynamics for tricopter, as given in (2.30) and (2.32). Moreover, the overall control scheme is summarized in Fig. 3.4. In the subsequent part, the designs of NMPC and NMHE for the presented framework are individually illustrated.

NMPC Design

For the trajectory tracking NMPC, the state vector $\mathbf{x}_{\text{NMPC}} \in \mathbb{R}^6$, control vector $\mathbf{u}_{\text{NMPC}} \in \mathbb{R}^4$, measurement vector $\mathbf{z}_{\text{NMPC}} \in \mathbb{R}^6$, and parameter vector $\mathbf{p}_{\text{NMPC}} \in \mathbb{R}^3$ are composed of:

$$\begin{aligned} \mathbf{x}_{\text{NMPC}} &= [x, y, z, u, v, w]^T, & \mathbf{u}_{\text{NMPC}} &= [\phi, \theta, \psi, F_z]^T, \\ \mathbf{z}_{\text{NMPC}} &= \mathbf{x}_{\text{NMPC}}, & \mathbf{p}_{\text{NMPC}} &= [m, F^{x_{dist}}, F^{y_{dist}}]^T. \end{aligned}$$

Note that in the parameter vector, only the disturbance forces along x and y directions, i.e., $F^{x_{dist}}$ and $F^{y_{dist}}$ are considered. This is because the effect of $F^{z_{dist}}$ along z-direction already appears in the mass (m) of the robot. In addition, the following state and control trajectories, obtained by the trial-and-error method, are given as references for the optimization problem of NMPC:

$$\mathbf{x}^{\text{r}} = \mathbf{x}^{\text{r}}_{N_c} = [x^{\text{r}}, y^{\text{r}}, z^{\text{r}}, u^{\text{r}}, v^{\text{r}}, w^{\text{r}}]^T, \quad \mathbf{u}^{\text{r}} = [-0.0414, 0, 0, mg]^T,$$

where x^{r}, y^{r}, z^{r} and u^{r}, v^{r}, w^{r} are the position and linear velocity references, respectively, and m and g are mass and gravitational constant. It is to be noted that for the OCP formulation of NMPC, the parametrization of the nonlinear model (in translation) is also done with respect to p, q, and r. Hence, the three rotational rates are fed to NMPC along with the other states, as also illustrated in Fig. 3.4.

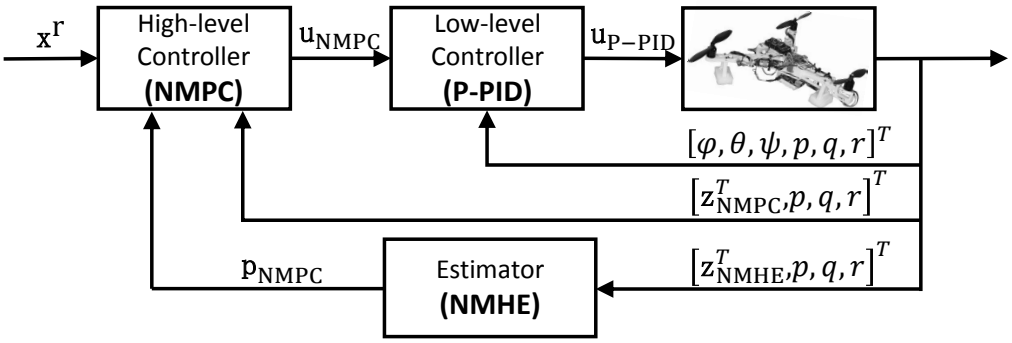


FIGURE 3.4: Closed-loop control diagram of the proposed instantaneous learning scheme.

In order to realize a stable behavior from the low-level controller, the following constraints are defined in the optimization problem of NMPC:

$$0.5mg \text{ (N)} \leq F_z \leq 1.7mg \text{ (N)}, \quad 20 \text{ (°)} \leq \phi, \theta \leq 20 \text{ (°)}, \tag{3.6}$$

where $m = 1.9$ kg is the maximum takeoff mass (including payload). Also, the following weight matrices are obtained by the trial-and-error method:

$$\mathbf{W_x} = \text{diag}(22, 25, 20, 1.3, 1.3, 2.0), \quad \mathbf{W_u} = \text{diag}(23, 24, 80, 0.012),$$
$$\mathbf{W}_{N_c} = \text{diag}(40, 40, 40, 2, 2, 2).$$

Amongst the above matrices, the terminal weight matrix ($\mathbf{W}_{N_c}$) is weighted more in comparison to the weight matrix for states ($\mathbf{W_x}$). This is a typical way of defining them, where the reason being is to assure the stability of the OCP. Furthermore, the prediction window $N_c = 30$ is selected to facilitate the realtime applicability of the control framework.

Remark 6. It is to be noted that the above weight matrices do not imply tuning of the aerial robot for a particular takeoff mass in a specific scenario. Rather, they are obtained to achieve a smooth response from the robot for a range of takeoff mass and all the considered case scenarios in this study.

NMHE Design

Recall that the main task of NMHE is to estimate online the parameter vector $\mathbf{p}$ that comprises of the robot mass and two disturbance forces. This parameter vector

varies with time under the influence of artificial disturbances that are induced in different scenarios. Hence, by referring to Fig. 3.4, the NMHE is designed with the following state $\mathbf{x}_{\text{NMHE}} \in \mathbb{R}^3$, control $\mathbf{u}_{\text{NMHE}} \in \mathbb{R}^3$, measurement $\mathbf{z}_{\text{NMHE}} \in \mathbb{R}^6$, and parameter $\mathbf{p}_{\text{NMHE}} \in \mathbb{R}^3$ vectors:

$$\begin{aligned} \mathbf{x}_{\text{NMHE}} &= [u, v, w]^T, & \mathbf{u}_{\text{NMHE}} &= [\phi, \theta, F_z]^T, \\ \mathbf{z}_{\text{NMHE}} &= [\mathbf{x}_{\text{NMHE}}^T, \mathbf{u}_{\text{NMHE}}^T]^T, & \mathbf{p}_{\text{NMHE}} &= [m, F^{x_{dist}}, F^{y_{dist}}]^T \end{aligned}$$

For the selected tricopter model, the weight matrices $\mathbf{P}_{\text{L}}$, $\mathbf{W}_{\boldsymbol{\nu}}$ and $\mathbf{W}_{\boldsymbol{\omega}}$, are chosen to be:

$$\begin{aligned} \mathbf{P}_{\text{L}} &= \text{diag}(0.1^2, 0.1^2, 0.1^2, 0.0316^2, 0.1^2, 0.1^2)^{-1/2}, \\ \mathbf{W}_{\boldsymbol{\nu}} &= \text{diag}(22.36^2, 22.36^2, 22.36^2, 31.62^2, 31.62^2, 14.14^2)^{-1/2}, \\ \mathbf{W}_{\boldsymbol{\omega}} &= \text{diag}(0.633^2, 0.633^2, 0.633^2, 0.316^2, 0.316^2, 0.316^2)^{-1/2}. \end{aligned}$$

The above values are decided based on calibration experience with the onboard (and external) sensors, incorporating the definitions in (3.3). In addition, the arrival cost is initialized with the following state and parameter vectors:

$$\bar{\mathbf{x}}_L = [0, 0, 0]^T, \quad \bar{\mathbf{p}}_L = [1.85, 0.1, 0.1]^T,$$

while the constraints that are imposed to achieve a constrained estimation of the parameter vector are as follows:

$$1.35 \text{ (kg)} \leq m \leq 2.0 \text{ (kg)}, \quad -4 \text{ (N)} \leq F^{x_{dist}}, F^{y_{dist}} \leq 4 \text{ (N)}, \tag{3.7}$$

where the minimum and maximum limits for the disturbance forces are obtained based on experience with the utilized fans. Furthermore, the estimation window length $M = 40$ is selected to be more than $N_c = 30$ to realize a comparatively slower learning from NMHE.

Remark 7. In the beginning, even longer estimation horizon ($M = 50$) was selected. However, due to the limited computational power of the onboard computer, the sampling frequency for NMHE came down to almost 18-Hz. Hence, for maintaining the frequency to 30-Hz, the estimation horizon was brought down to 40.

3.4.1.3 Experimental Results

In terms of the realtime execution of NMPC and NMHE, the `ACADO` generated C codes are directly imported onboard the Raspberry Pi 3. Although it has low computational power, the optimized nature of the generated C codes along with efficient ROS (robot operating system) scripts facilitate a sampling frequency of about 50-Hz for NMPC and about 30-Hz for NMHE. An OptiTrack motion capture system, consisting of eight 240 FPS (frames per second) cameras, is utilized which provides the position and attitude feedback over a local area network. This position and attitude information is given to an estimator (EKF), which subsequently computes the other states including translation and rotational velocities of the robot. Then, the full-state information along with the reference trajectory command ($\mathbf{x}^{\mathrm{r}}$) is provided to NMPC via ROS topics, over a wireless network. Subsequently, the NMPC computes the optimal control commands that are passed to the low-level controller (Pixhawk) via serial communication. Finally, the obtained actuator commands are given to the rotors and tilting servo, as can be visualized from the block diagram in Fig. 3.5.

Next, the position tracking results for the 3D-printed aerial robot incorporating three different disturbance sources are presented. The indoor experimental setup including the cameras of the utilized motion capture system is depicted in Fig. 3.6.

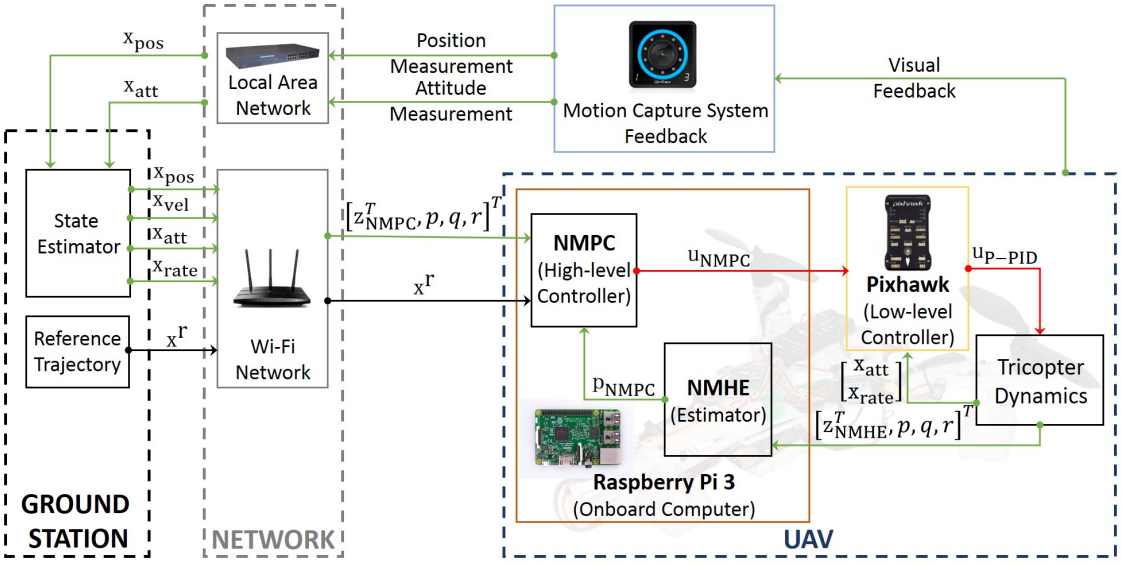


FIGURE 3.5: Schematic block-diagram for the realtime implementation on the aerial robot.
Notation: $\mathbf{x}_{\mathrm{pos}} = [x, y, z]^T$; $\mathbf{x}_{\mathrm{vel}} = [u, v, w]^T$; $\mathbf{x}_{\mathrm{att}} = [\phi, \theta, \psi]^T$; $\mathbf{x}_{\mathrm{rate}} = [p, q, r]^T$.

In all the experiments, the results are obtained for two cases: (i) NMPC without learning (conventional NMPC), and (ii) NMPC with learning (NMPC-NMHE framework). Firstly, a description of the experiment is provided. Thereafter, the implementation details are included which is followed by our interpretation of the obtained results. Different trajectories are selected for these experiments to depict the robustness of the framework. A demonstration video depicting the experiments can be found at: https://youtu.be/B9I4rPONa44.

Case Scenario 1: Payload Dropping

The first experiment tackles a package delivery scenario, wherein the robot tracks a reference trajectory while dropping packages to their time-based designated locations, as depicted in Fig. 3.7. The robot has a total takeoff mass of 1.442 kg with a payload of 457g, which constitutes 32% of its empty mass (without payload). The drop of this payload implies a massive change in the model parameter (m) that has to be handled by the controller, and hence, justify the use of learning-based framework for this application.

In this application, the robot with initial state: $\mathbf{x}(0) = [0, 0, 1.5, 0, 0, 0]^T$, tracks a time-based circular trajectory of radius 1.5m at a flight speed of 1 m/s. For both the aforementioned cases (without and with learning), first a complete circular trajectory is tracked with maximum takeoff mass and thereafter, the four blocks are dropped at fixed time-intervals in the sequence: 86g $\rightarrow$ 114g $\rightarrow$ 114g $\rightarrow$ 143g. Note that in this application, since the disturbance by weight dropping is only

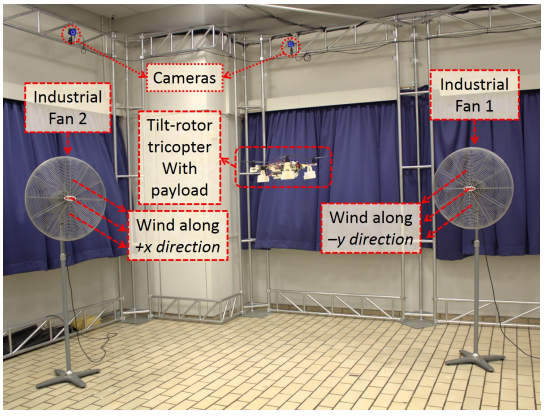


FIGURE 3.6: Indoor testing environment for the realtime experiments.

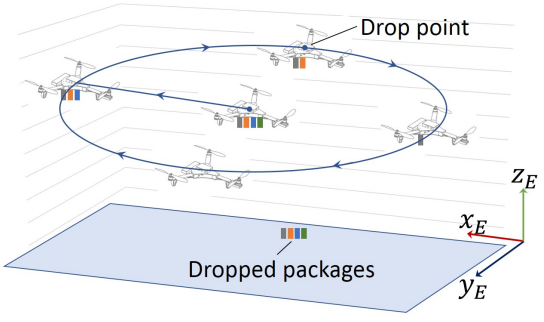


FIGURE 3.7: Payload drop experiment in Case scenario 1.

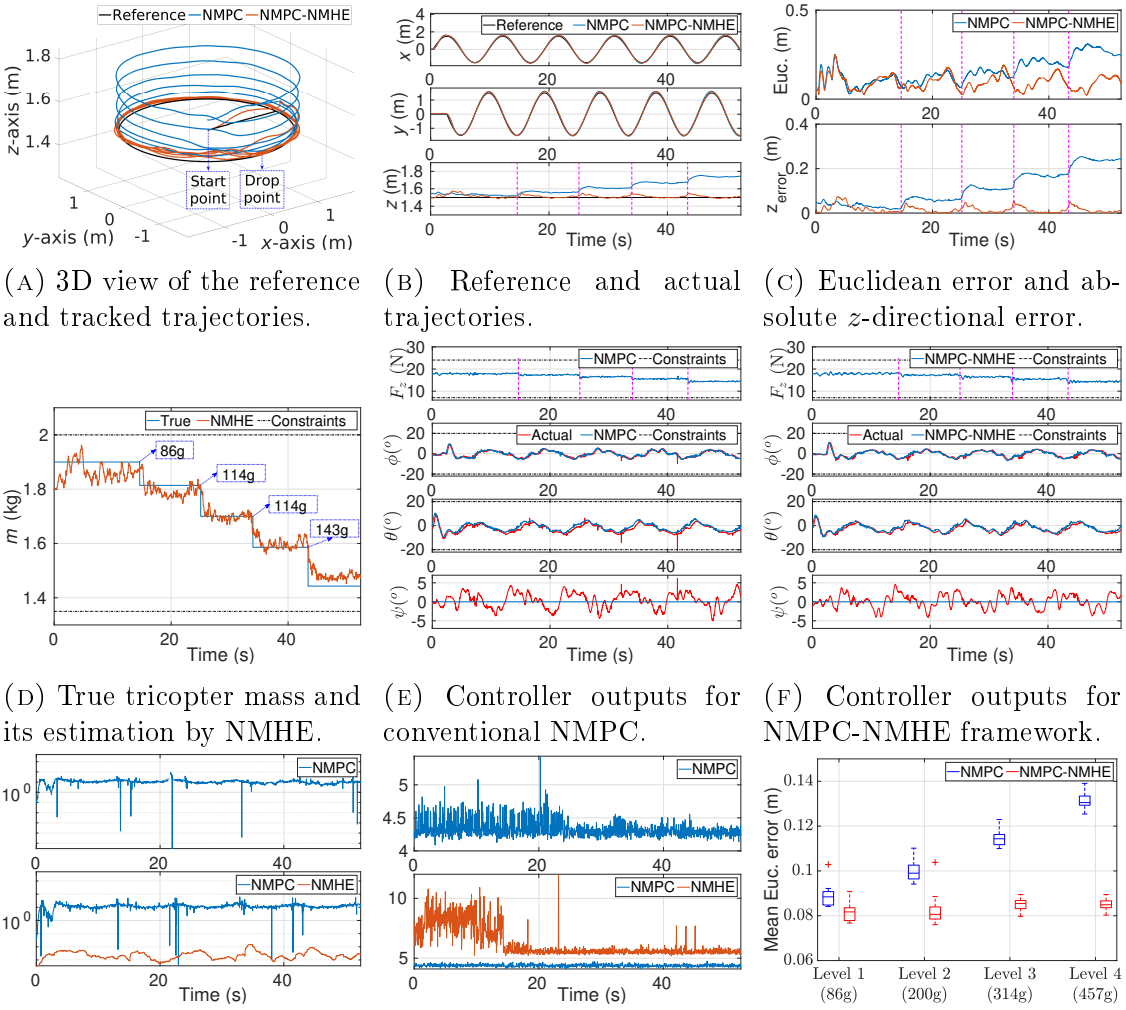
introduced along the z-direction, the parameter vectors for NMPC and NMHE are modified to be $\mathtt{p}_{\text{NMPC}} = \mathtt{p}_{\text{NMHE}} = m$.

The overall position tracking performance for both the controllers is presented in Figs. 3.8a and 3.8b. As visualized, the z-direction tracking of the NMPC-NMHE framework is much precise without any error accumulation with weight drops, in comparison to the conventional NMPC. This effect is anticipated as NMHE in the learning case helps NMPC to adapt itself to the changing dynamics (robot's mass) and hence, the plant-model mismatch diminishes with time. Moreover, the position tracking Euclidean error and the absolute error along z-direction (z_{error}) for both are shown in Fig. 3.8c. While the mean values of Euclidean error and z_{error} for NMPC without learning are 0.1598m and 0.1113m, respectively, the learning helps to lower them down as the mean Euclidean error and z_{error} for NMPC-NMHE framework are 0.0962m and 0.0159m, respectively. In essence, learning improves the tracking performance by 6.36 cm in terms of Euclidean error and 9.54 cm in terms of z_{error}.

Remark 8. It is to be noted that in this application, the tracking error is considered to be the distance between the current position and the corresponding closest point on the trajectory.

The control outputs commanded by the controller within conventional NMPC and NMPC-NMHE framework are presented in Figs. 3.8e and 3.8f, respectively. From Figs. 3.8e and 3.8f, it is visualized that the NMPC commands for both the cases are well within the constraints specified in (3.6). Moreover, one may notice a comparatively poor tracking performance in terms of the yaw angle (ψ) response for both the cases. It is emphasized that this is due to the tuning of PIDs along the yaw channel in the low-level controller i.e., within Pixhawk.

Figure 3.8d depicts the performance of NMHE in estimating the robot mass, along with its true value. As can be seen, the mass estimation always stays within the bounds specified in (3.7), which eventually illustrates the bounded learning capability of NMHE. This is an important aspect as the stability of NMPC is typically guaranteed for an accurate system model; whereas an unbounded (diverging) estimation of time-varying parameters may eventually destabilize the closed-loop. Note that, although m is a physical parameter, its estimation by NMHE also accommodates the modeling and operational uncertainties (for instance, discharging



(A) 3D view of the reference and tracked trajectories.

(B) Reference and actual trajectories.

(C) Euclidean error and absolute z-directional error.

(D) True tricopter mass and its estimation by NMHE.

(E) Controller outputs for conventional NMPC.

(F) Controller outputs for NMPC-NMHE framework.

(G) KKT tolerance for the underlying OCPs.

(H) Execution time (ms) for solving the OCPs.

(I) Mean Euclidean error for four levels of weight drops.

FIGURE 3.8: Case Scenario 1: trajectory tracking control performance of a tilt-rotor tricopter with varying mass, where the vertical magenta lines represent the instants of payload drop. In (D), it is observed that the estimated tricopter mass converges to the true value after a transition time of a few seconds. Moreover, none of the constraints ever get violated, which implies constrained learning. Furthermore, in (I), tests for each level are repeated 10 times.

of the battery) induced during operation. Primarily, it is deduced as an adaptive parameter that enables NMPC to realize an offset-free tracking along the z-direction.

The single Gauss-Newton iteration per sampling instant that is performed to solve the optimization problems of NMPC and NMHE would result in a suboptimal solution. Therefore, it becomes necessary to check the optimality of the control framework. In that vein, Karush-Kuhn-Tucker (KKT) tolerances are obtained for

the respective NMPC and NMHE in both, learning and without learning cases, as shown in Fig. 3.8g. In the case of a linear system with a least-square objective, these values reduce to zero, whereas, for the tilt-rotor tricopter, higher values are anticipated due to the underlying nonlinearities and inter-couplings. Nevertheless, their low and not too drastically varying magnitudes still signify well-defined optimization problems.

The execution times at each sampling instant are shown in Fig. 3.8h. While the average execution time for NMPC in without learning case is 4.3 ms, the average execution times for NMPC and NMHE in the case with learning are 4.4 ms and 6.2 ms, respectively. Recall that difference in computation times of NMPC and NMHE within the NMPC-NMHE framework is due to the choice of a longer estimation horizon ($M = 40$) over the prediction horizon ($N_c = 30$).

Remark 9. For a shorter horizon, mass learning is fast which eventually makes NMPC aggressive towards the change. On the other hand, for a longer horizon, the NMHE gradually learns the mass parameter and hence, a smooth response is obtained from NMPC. Therefore, selecting an appropriate estimation horizon on Raspberry Pi is a trade-off between the desired rate of learning and the available computational power. In general, the latter is more important due to the minimum sampling frequency required for a sustained closed-loop.

Additionally, to statistically support the claim that learning-based NMPC improves tracking performance in the presence of abrupt mass disturbance, experiments with four different levels of payload drop are performed. For each level, the tests are repeated 10 times and the resulting box plot with the mean Euclidean error values is presented in Fig. 3.8i. As can be seen, the mean Euclidean error for conventional NMPC increases with the increasing weight drop level, while the mean Euclidean error rise for NMPC-NMHE framework is negligible. This is also according to our expectation because the learning helps NMPC to adapt itself to the changing conditions, hence results in a minimal error rise even with the increasing magnitude of disturbance.

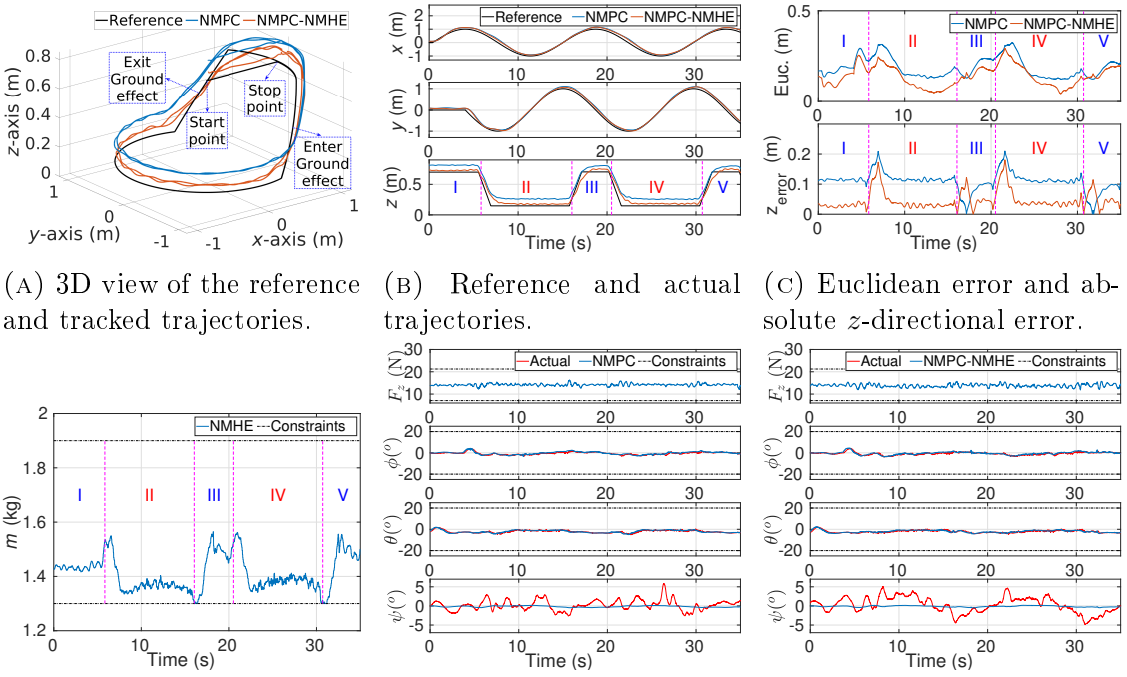
Case Scenario 2: Ground Effect

The second experiment analyzes another disturbance type which is commonly encountered when an aerial robot flies in proximity to the ground, i.e., 'ground effect'.

For this scenario, the tricopter tracks a circular trajectory of radius 1m, involving two different heights, as also depicted in Fig. 3.9a. While the tricopter begins to follow the trajectory at $z = 0.7$m with negligible influence of ground effect, it descends to $z = 0.15$m after-a-while wherein the ground effect gets dominating. Subsequently, it climbs back to $z = 0.7$m and stays at this height, before going for another round. In this application also, the influence of disturbance forces ($F^{x_{dist}}$, $F^{y_{dist}}$) is expected to be negligible, hence the corresponding NMPC and NMHE parameter vectors are: $\mathtt{p}_{\text{NMPC}} = \mathtt{p}_{\text{NMHE}} = m$.

Position tracking performance for both the cases in the presence of ground effect can be seen in Figs. 3.9a and 3.9b. As expected, the learning-based NMPC dominates the conventional NMPC in terms of z-direction tracking. The improvement is better visualized from Fig. 3.9c, where the Euclidean error and z_{error} are plotted. As observed in Fig. 3.9c (particularly z_{error} plot), there lies an offset along the z-direction for conventional NMPC, due to which its z_{error} is more than that for the NMPC-NMHE framework. Although there are two instances (around 15s and 30s) where z_{error} for the NMPC-NMHE framework becomes more, this is during climbing and hence, can be associated with the transient response. In other words, at the instant when tricopter starts to climbs, its estimated (effective) mass within the NMPC-NMHE framework is less, and hence, the tracking error instantaneously rises until the new true mass is estimated by NMHE. The mean values for Euclidean error and z_{error} in conventional NMPC's case are 0.1852m and 0.1091m, respectively, while for NMPC-NMHE framework the mean error values are 0.1380m and 0.0459m, respectively. Hence, the learning resulted in an improvement of 4.72 cm in terms of Euclidean error and 6.32 cm in terms of z_{error}.

Next, the control outputs for conventional NMPC and NMPC-NMHE framework are presented in Figs. 3.9e and 3.9f, respectively. From these figures, it is evident that NMPC commands for both the cases are well within the specified constraints. The mass estimation performance of NMHE in the NMPC-NMHE framework is depicted in Fig. 3.9d. The estimated value always stays restricted within the specified bounds that again showcases the bounded learning capability of NMHE. In addition, the estimated mass value varies in an intuitive pattern along the trajectory. That is when the tricopter is flying under the influence of ground effect, the effective mass is expected to decrease as also seen in sections II and IV of Fig. 3.9d, and vice versa when it is flying away from the ground. Finally, the average execution



(A) 3D view of the reference and tracked trajectories.

(B) Reference and actual trajectories.

(C) Euclidean error and absolute z-directional error.

(D) Estimated tricopter mass by NMHE.

(E) Controller outputs for conventional NMPC.

(F) Controller outputs for NMPC-NMHE framework.

FIGURE 3.9: Case Scenario 2: trajectory tracking control performance of a tilt-rotor tricopter in the presence of ground effect, where the vertical magenta lines in (C) and (D) represent sections with the same z-position, such that sections I, III and V are with $z = 0.7$m, and sections II and IV are with $z = 0.15$m. Additionally, in (D), it is observed that the tricopter mass decreases as it hovers close to the ground which is also intuitively consistent.

time for NMPC in without learning case is 4.21 ms, while the average execution times for NMPC and NMHE in the case with learning are 4.33 ms and 7.591 ms, respectively.

Case Scenario 3: Wind Gust Disturbance

The third experiment mimics one of the most commonly occurring disturbances during outdoor flights, i.e., wind gust. Although all our experiments are performed indoors, recall that the main goal of this work is to bring the robot from an engineered (well-known) indoor environment to a less known outdoor world. In that vein, artificial wind gust disturbances are introduced with the help of two industrial fans that are placed perpendicular to each other at a distance of 2.5m from the origin. Overall, they generate disturbances of magnitude 3.3-3.7 m/s (each fan) along $+x$ and $-y$ directions, as can be visualized from Fig. 3.6. For a

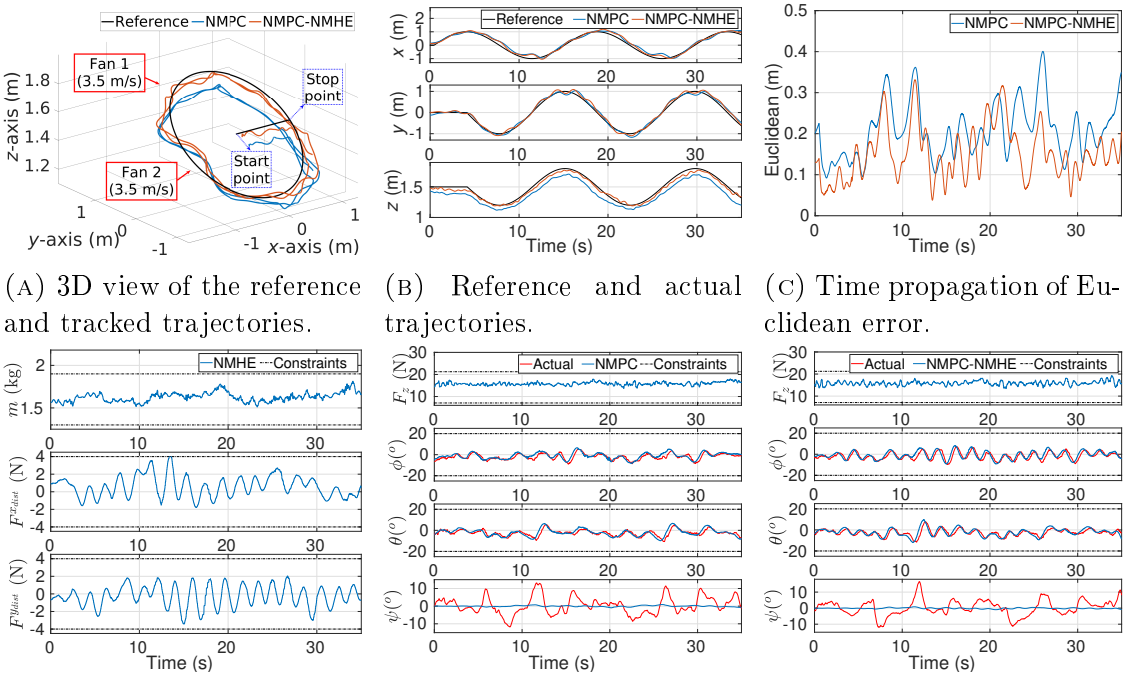
better judgment on the learning ability of the NMPC-NMHE framework, the experiments are conducted with two trajectories, namely, slanted circle, and hover at the origin with sinusoidally varying z-position. Moreover, in this experiment, since the disturbances are explicitly introduced along x and y directions, the magnitudes of disturbance forces ($F^{x_{dist}}$, $F^{y_{dist}}$) are also estimated along with the mass: $\mathbf{p}_{\text{NMPC}} = \mathbf{p}_{\text{NMHE}} = [m, F^{x_{dist}}, F^{x_{dist}}]^T$. Next, the trajectory tracking results for slanted circle and hover trajectories are sequentially presented.

Slanted Circular Trajectory

Position tracking performance of both the controllers for a slanted circular trajectory of radius 1m can be seen in Figs. 3.10a and 3.10b. To quantitatively analyze the tracking accuracy, the Euclidean errors are also plotted in Fig. 3.10c. From the figure, it is evident that Euclidean error for the NMPC-NMHE framework is always lower in comparison to the conventional NMPC. Besides, the mean Euclidean error values for without and with learning cases are 0.2078m and 0.1515m, respectively, which eventually implies an improvement of 5.63 cm due to learning.

Remark 10. Looking at these error values, one may question the magnitude of improvement that a learning-based framework brings to this application. Recall that when a robot encounters wind gust disturbance, estimated disturbance forces are fed to NMPC with some time-lapse. Consequently, before NMPC could perform the necessary corrections, tricopter proceeds further on its course, making the estimated values obsolete for the current corrections. In other words, there lies a time-lag between the estimation and the corrective action taken by the controller, thereby making learning less effective within the scope of this trajectory. Nevertheless, to validate the proficiency of learning, these experiments are performed again for a hover with varying z-position trajectory, the results of which are discussed later.

The control outputs for conventional NMPC and NMPC-NMHE framework for a slanted circular trajectory are presented in Figs. 3.10e and 3.10f, respectively. From these figures, it is again evident that the NMPC commands for both the cases are well within the specified bounds. Figure 3.10d depicts the performance of NMHE in estimating the tricopter mass along with the introduced disturbance forces. The constrained estimation ability of NMHE is again validated here, as none of the specified bounds are ever violated, even when the estimated values for disturbance forces reached the bounds.



(A) 3D view of the reference and tracked trajectories.

(B) Reference and actual trajectories.

(C) Time propagation of Euclidean error.

(D) Estimated mass and disturbance forces by NMHE.

(E) Controller outputs for conventional NMPC.

(F) Controller outputs for NMPC-NMHE framework.

FIGURE 3.10: Case Scenario 3a: trajectory tracking control performance of a tilt-rotor tricopter in the presence of wind gust disturbance for a slanted circular reference. In (D), it is observed that none of the constraints are violated that again validates the bounded learning capability of NMHE.

Remark 11. Since the disturbance forces are induced via same wind speed for both the fans, the estimator is expected to produce less oscillating estimation values. However, it is pointed out that due to the circular motion of the robot, the effective disturbance forces vary with time. This assertion is validated by the estimation results of the hover trajectory that are presented later in this section.

Furthermore, the average execution time for NMPC in without learning case is 4.212 ms, while the average execution times for NMPC and NMHE in the case with learning are 4.92 ms and 6.694 ms, respectively.

Hover Trajectory

As asserted before, due to the time-varying nature of the wind gust disturbance while flying a circular trajectory, NMHE does not have enough time to learn the true disturbance values. As a result, only a limited performance improvement is observed. Therefore, to validate the efficacy of learning in dealing with the induced wind gust disturbances, these experiments are performed again for a hover

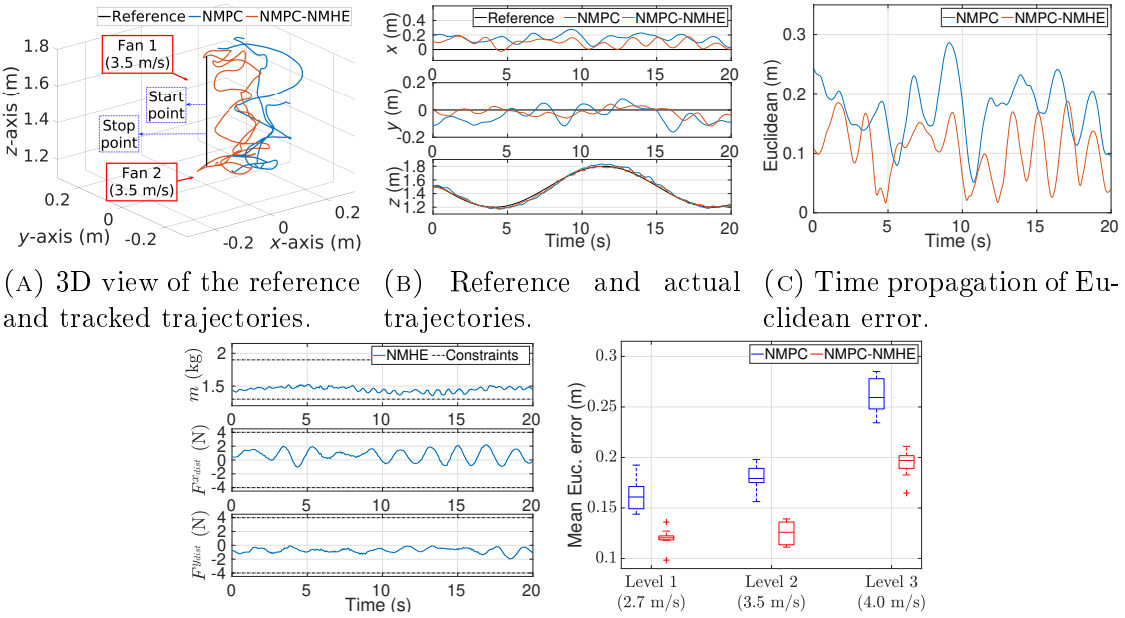
trajectory, involving a motion only along the z-direction. While selecting the hover trajectory, it is expected that the stationary (along x and y directions) nature of this trajectory would provide ample time to NMHE for learning the disturbance forces, such that, a significant improvement in tracking would be observed.

Position tracking results for a hover with changing z-position reference by both types of NMPCs can be seen in Figs. 3.11a and 3.11b. As expected, there lies a steady offset along $+x$ and $-y$ directions for conventional NMPC, while the NMPC within NMPC-NMHE framework compensates for it. Moreover, the corrective actions of learning-based NMPC are more visible in Fig. 3.11c, where the Euclidean errors for both without and with learning NMPCs are presented. In addition, the mean Euclidean error values for without and with learning cases are 0.1760m and 0.1047m, respectively, which eventually showcases an improvement of 7.13 cm brought by learning for this scenario. Furthermore, the estimation results of NMHE within the NMPC-NMHE framework are shown in Fig. 3.11d. One may observe a comparatively less oscillating estimation of disturbance forces for this hover trajectory, which is in contrast with the estimation results for the previous trajectory. As a summary, the mean Euclidean and absolute z-position error values for all the aforementioned test case scenarios, obtained with both types of NMPCs are listed in Table 3.2.

Finally, to further support the claim that learning-based NMPC improves the tracking performance in presence of wind gust disturbance, 10 experiments each with three different levels of wind speeds are performed and their respective mean Euclidean error values are illustrated in the form of a box plot in Fig. 3.11e. As expected, the mean Euclidean error for both type of NMPCs increase with the increasing wind speed. However, the error rise for the NMPC-NMHE framework with increasing wind speed is minimal compared to the one for conventional NMPC. This is in accordance with our intuition as the learning helps NMPC to adapt according

TABLE 3.2: Mean Euclidean and absolute z-position errors for the three case scenarios. (N.A.: not applicable)

	Euclidean error (cm)		**z-position error (cm)**	
Case study	NMPC	NMPC-NMHE	NMPC	NMPC-NMHE
1	17.72	15.93	5.61	1.09
2	18.52	13.80	10.91	4.59
3a	20.78	15.15	N.A.	N.A.
3b	17.70	10.47	N.A.	N.A.



(A) 3D view of the reference and tracked trajectories.

(B) Reference and actual trajectories.

(C) Time propagation of Euclidean error.

(D) Estimated mass and disturbance forces by NMHE.

(E) Mean Euclidean error for three wind levels.

FIGURE 3.11: Case Scenario 3b: trajectory tracking control performance of a tilt-rotor tricopter in the presence of wind gust disturbance for a hover with varying z-position reference. It is to be noted that in (D), smoother estimation performance is observed in contrast with Fig. 3.10d. Furthermore, in (E), 10 tests for each speed level are performed, wherein the wind speed ranges for each level are, Level 1: $2.4 - 3.0$ m/s; Level 2: $3.3 - 3.7$ m/s; Level 3: $3.8 - 4.2$ m/s.

to the environment, hence results in a lower error rise even with the increasing magnitude of disturbances.

Remark 12. One may think that the wind speeds which are utilized for testing may not be too strong. However, it is emphasized that these speeds are strong enough to employ a significant gust-force on the considered aerial robot. This is due to the high ratio of the applied wind gust disturbance magnitude in comparison to the tricopter size/mass. Moreover, it is to be noted that the closed-loop framework should be able to handle any wind gust magnitude, provided that the actuators (rotors) can handle the disturbance force and the NMHE is designed with the appropriate constraint values.

3.4.2 Cascade Control for the Uncertain System Model

The second case study investigates the effect of uncertainty in the two intrinsic parameters, namely, thrust and drag-moment coefficients, thereby resulting in a

time-varying model. Note that it is important to estimate these parameters because apart from system identification errors, they can vary in-flight due to wind disturbance or propeller degradation. In that vein, a cascade NMPC structure is employed, wherein the two NMPCs (H-NMPC and L-NMPC) perform position and attitude tracking, while NMHE learns online the two uncertain parameters.

Remark 13. Recall the advantages of utilizing a cascade structure over a full-state controller: (i) ability to distribute the overall control task that facilitates their execution on (cheap) individual processors, and (ii) flexibility to run the two loops at different frequencies. The second aspect is crucial for realtime control of aerial robots as the stability-ensuring low-level controller has to run at a higher rate.

3.4.2.1 Problem Statement

A Talon tilt-rotor tricopter model is considered in this study. For the configuration shown in Fig. 2.4, the overall system model in discrete form, similar to (2.33), is written as:

$$\mathbf{x}_{k+1} = \mathbf{f}_{\mathrm{d}}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}) + \boldsymbol{\omega}_k, \quad \mathbf{z}_k = \mathbf{x}_k + \boldsymbol{\nu}_k, \tag{3.8}$$

wherein all the uncertainties are expressed together in the form of zero-mean, Gaussian process noise $\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{\omega}})$. Also, the corresponding state vector is $\mathbf{x} \in \mathbb{R}^{12} = [x, y, z, u, v, w, \phi, \theta, \psi, p, q, r]^T$, the control vector is $\mathbf{u} \in \mathbb{R}^4 = [\Omega_1, \Omega_2, \Omega_3, \mu]^T$, and the unknown parameter vector is $\mathbf{p} = [K_F, K_\tau]^T$. Accordingly, the nominal function is $\mathbf{f}_{\mathrm{d}}(\,\cdot\,,\,\cdot\,,\,\cdot\,) : \mathbb{R}^{12} \times \mathbb{R}^4 \times \mathbb{R}^2 \mapsto \mathbb{R}^{12}$. Moreover, the intrinsic model parameters utilized in this study are listed in Table 3.3.

3.4.2.2 Receding Horizon Control and Estimation

In order to handle the additional nonlinearities of the tilt-rotor tricopter, a cascade NMPC structure is utilized in this application, as depicted in Fig. 3.12. The H-NMPC that tracks a given position reference, computes the optimized control inputs in terms of the attitude angles and the total required thrust. The desired attitude commands are subsequently passed to the L-NMPC that solves an attitude

TABLE 3.3: Tilt-rotor tricopter intrinsic parameters.

Parameter	Description	Value
m	Mass of tricopter	1.412 kg
l_1	Moment arm	0.357 m
l_2	Moment arm	0.2845 m
l_3	Moment arm	0.149 m
I_{xx}	MOI about $\mathcal{F}_{B_x}$	0.0149 kg-m^2
I_{yy}	MOI about $\mathcal{F}_{B_y}$	0.0219 kg-m^2
I_{zz}	MOI about $\mathcal{F}_{B_z}$	0.0344 kg-m^2
I_{xz}	MOI about $\mathcal{F}_{B_{xz}}$	0.0202 kg-m^2
J_P	MOI of propeller	6×10^{-5} kg-m^2
K_F	Force coefficient	6.85×10^{-6} N-s^2
K_τ	Drag-moment coefficient	3.35×10^{-7} Nm-s^2

tracking problem. Therefore, the control outputs of L-NMPC are the actuator commands that are given to the tricopter after a summation with the thrust command of H-NMPC, as also visualized in Fig. 3.12.

High-level NMPC Design

Similar to the previous case study, the position tracking H-NMPC is designed with the following state, control, and measurement vectors:

$$\mathbf{x}_{\text{H-NMPC}} = [x, y, z, u, v, w]^T, \quad \mathbf{u}_{\text{H-NMPC}} = [\phi^*, \theta^*, \psi^*, F_z^*]^T,$$
$$\mathbf{z}_{\text{H-NMPC}} = \mathbf{x}_{\text{H-NMPC}},$$

while the following state and control references are utilized in the OCP:

$$\mathbf{x}^{\text{r}}_{\text{H-NMPC}} = \mathbf{x}^{\text{r}}_{N_c,\text{H-NMPC}} = [x^{\text{r}}, y^{\text{r}}, z^{\text{r}}, 0, 0, 0]^T, \quad \mathbf{u}^{\text{r}}_{\text{H-NMPC}} = [-0.0414, 0, 0, 0.5mg]^T.$$

Note that the H-NMPC's control outputs are denoted with a superscript $(\cdot)^*$ to differentiate them from the actual response of the tricopter. Also, to realize a stable response from L-NMPC, the following constraints are defined:

$$0.01mg \text{ (N)} \leq F_z^* \leq mg \text{ (N)}, \quad -45(^\circ) \leq \phi^*, \theta^* \leq 45(^\circ), \tag{3.9}$$

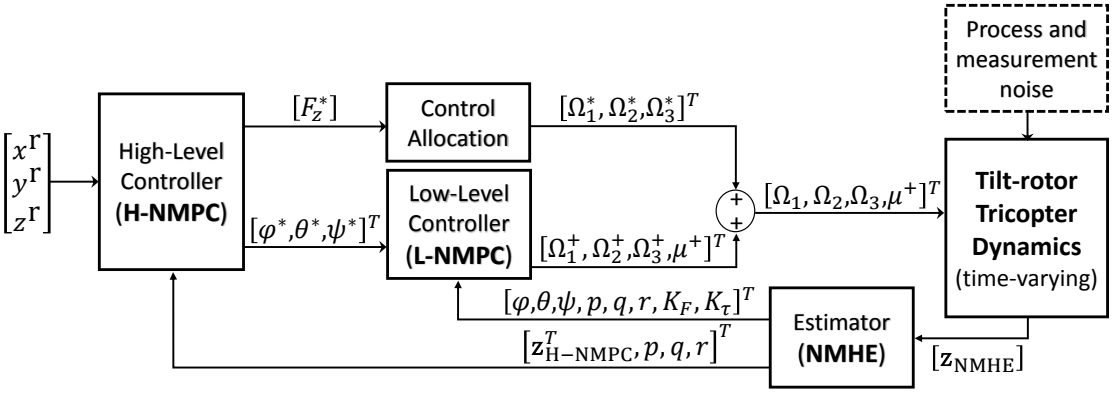


FIGURE 3.12: Closed-loop control diagram for the learning-based cascade NMPC framework.

along with the weight matrices that are obtained via the trial-and-error method as follows:

$$\mathbf{W}_{\mathbf{x},\text{H-NMPC}} = \text{diag}(35, 35, 20, 3.5, 5, 3), \quad \mathbf{W}_{\mathbf{u},\text{H-NMPC}} = \text{diag}(30, 30, 50, 0.005),$$
$$\mathbf{W}_{N_c,\text{H-NMPC}} = \text{diag}(50, 50, 30, 4, 6, 4).$$

Furthermore, the prediction window $N_c = 30$, is selected for the H-NMPC.

Remark 14. One may notice a control allocation block after the H-NMPC in Fig. 3.12. It computes the rotor velocities that are required to climb/descent, based upon the total thrust commanded by the H-NMPC:

$$\Omega_1^* = \sqrt{\frac{F_z^*}{3K_F}}, \quad \Omega_2^* = \sqrt{\frac{F_z^*}{3K_F}}, \quad \Omega_3^* = \sqrt{\frac{F_z^*}{3K_F cos(\mu)}}, \tag{3.10}$$

where μ is the current tilt angle for the back rotor. Note that the above expressions are computed utilizing the external force definitions in (2.30) and (2.32).

Low-level NMPC Design

The state, control and parameter vectors for attitude tracking L-NMPC are composed of:

$$\mathbf{x}_{\text{L-NMPC}} = [\phi, \theta, \psi, p, q, r]^T, \quad \mathbf{u}_{\text{L-NMPC}} = [\Omega_1^+, \Omega_2^+, \Omega_3^+, \mu^+]^T,$$
$$\mathbf{z}_{\text{L-NMPC}} = \mathbf{x}_{\text{L-NMPC}}, \quad \mathbf{p}_{\text{L-NMPC}} = [K_F, K_\tau]^T,$$

where Ω's are in rad/s and μ is in radian. The state and control reference trajectories selected for L-NMPC are as follows:

$$\mathbf{x}^{\mathrm{r}}_{\text{L-NMPC}} = \mathbf{x}^{\mathrm{r}}_{N_c,\text{L-NMPC}} = [\phi^*, \theta^*, \psi^*, 0, 0, 0]^T, \quad \mathbf{u}^{\mathrm{r}}_{\text{L-NMPC}} = [595, 605, 544, 0.173]^T.$$

Additionally, the following control constraints are imposed in L-NMPC such that even after a full actuation command from the L-NMPC, the tricopter still has some room to follow H-NMPC's climb/descent command without saturating the rotors:

$$0(\text{rad/s}) \leq \Omega_1^+, \Omega_2^+, \Omega_3^+ \leq 700(\text{rad/s}), \quad -25(^\circ) \leq \mu^+ \leq 25(^\circ). \tag{3.11}$$

Also, the L-NMPC is designed with the weight matrices:

$$\mathbf{W}_{\mathbf{x},\text{L-NMPC}} = \text{diag}(25, 35, 22, 0.2, 0.2, 0.2), \quad \mathbf{W}_{N_c,\text{L-NMPC}} = \text{diag}(30, 40, 25, 1, 1, 1),$$
$$\mathbf{W}_{\mathbf{u},\text{L-NMPC}} = \text{diag}(1 \times 10^{-4}, 1.3 \times 10^{-4}, 1.3 \times 10^{-5}, 20).$$

which are obtained by the trial-and-error method. Moreover, the same prediction window length as H-NMPC ($N_c = 30$) is utilized for L-NMPC.

NMHE Design

For the considered tricopter model, NMHE is designed to perform simultaneously state and parameter estimation with the same state, control, and parameter vectors as specified for (3.8). Besides, the following weight matrices are selected for the optimization problem of NMHE:

$$\begin{aligned}
\mathbf{P}_{\mathsf{L}} &= \text{diag}([0.1^2]_{12}, [0.032^2]_2)^{-\frac{1}{2}}, \\
\mathbf{W}_{\boldsymbol{\nu}} &= \text{diag}([0.032^2]_3, [0.045^2]_3, [0.022^2]_3, [0.224^2]_3, [0.003^2]_3, 0.002^2)^{-\frac{1}{2}}, \\
\mathbf{W}_{\boldsymbol{\omega}} &= \text{diag}([0.1^2]_2, [0.14^2]_7, [0.17^2]_3, [0.89^2]_2)^{-\frac{1}{2}},
\end{aligned}$$

wherein vector $[\,\cdot\,]_i \in \mathbb{R}^i$. Note that the above values of weight matrices are decided based upon the definition in (3.3) while exploiting the availability of the true values of noise covariance matrices ($\boldsymbol{\Sigma}_{\boldsymbol{\omega}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\nu}}$). Also, the arrival cost is initialized with the state vector $\bar{\mathbf{x}}_L = \mathbf{0}_{10} \in \mathbb{R}^{10}$ and the parameter vector $\bar{\mathbf{p}}_L = [5 \times 10^{-6}, 3 \times 10^{-7}]^T$. Moreover, the following constraints are imposed for achieving a constrained

estimation of the unknown parameters:

$$\begin{aligned} 4.11 \times 10^{-6}(\text{N-s}^2) \leq K_F \leq 9.59 \times 10^{-6}(\text{N-s}^2), \\ 2.47 \times 10^{-7}(\text{Nm-s}^2) \leq K_\tau \leq 5.75 \times 10^{-7}(\text{Nm-s}^2). \end{aligned} \tag{3.12}$$

Furthermore, the estimation window length M is selected to be 30 for computational tractability, thus facilitating the realtime applicability of the framework.

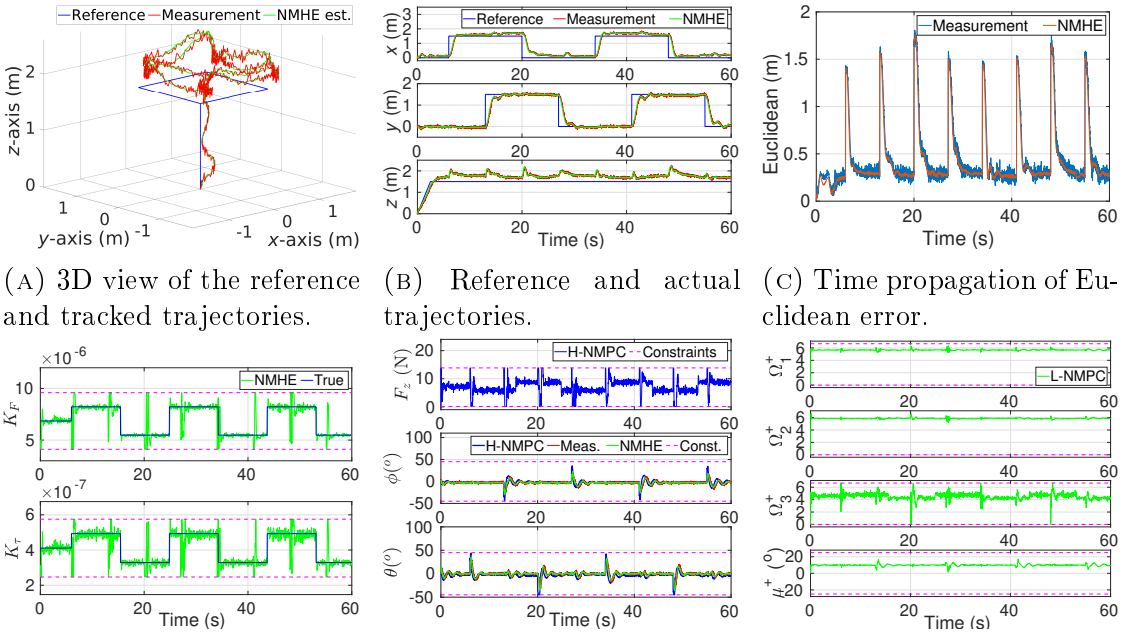
3.4.2.3 Numerical Results

Next, the simulation results are presented for the proposed learning-based cascade NMPC which is employed for trajectory tracking of a tilt-rotor tricopter. In this application, the `ACADO` generated C codes are utilized to create MATLAB/Simulink based S-functions. Moreover, all the simulations are performed with a fixed sampling time of $\Delta t = 0.01$s in MATLAB/Simulink installed on hardware consisting of 3.4-GHz Intel Core i7 processor and 8 GB RAM. For initializing the simulations, the tricopter is considered to be on the ground with zero state and control vectors, and with the parameters given in Table 3.3.

The subsequent part provides position tracking results for a square-shaped reference trajectory of 1.5m length, as shown in Fig. 3.13. The motivation behind incorporating this aggressive trajectory is to further excite the existing nonlinear and highly coupled dynamics of the tilt-rotor tricopter and thus, examine the ability of cascade NMPC in handling it. In addition, NMHE is employed to learn the change in tricopter dynamics, which is explicitly induced by varying the aerodynamic parameters K_F and K_τ by $\pm 20\%$ from their true values (listed in Table 3.3).

The tracking performance of cascade NMPC for the square-shaped trajectory can be visualized in Figs. 3.13a and 3.13b. The incurred absolute tracking errors along x, y, and z directions are 0.2204m, 0.1202m, and 0.2639m, respectively. In addition, the Euclidean error plot is shown in Fig. 3.13c, wherein the mean value over the entire trajectory is 0.4278m. Note that these error values are computed considering the NMHE's estimates.

The control outputs for the high- and low-level NMPCs are shown in Figs. 3.13e and 3.13f, respectively. As evident, none of the constraints are violated by both H-NMPC and L-NMPC, which essentially supports the realtime applicability of



(A) 3D view of the reference and tracked trajectories.

(B) Reference and actual trajectories.

(C) Time propagation of Euclidean error.

(D) True values of aerodynamic parameters and their estimation utilizing NMHE.

(E) Controller outputs for H-NMPC.

(F) Controller outputs for L-NMPC, wherein $\Omega^{+}_{1,2,3}$ are $\times 1000$ (RPM).

FIGURE 3.13: Trajectory tracking control performance of the tilt-rotor tricopter with aerodynamic parameters K_F and K_τ varying by $\pm 20\%$ from their true values.

the presented control framework. It is to be noted that ψ response is omitted in Fig. 3.13e for a concise representation. Additionally, the mean absolute error values for L-NMPC in following the ϕ^*, θ^* and ψ^* commands of H-NMPC are 2.32°, 4.05° and 1.34°, respectively.

The results of parameter estimation by NMHE for square-like varying parameters are shown in Fig. 3.13d. Owing to the learning ability of NMHE, both the parameters are well assessed, without any constraint violation, even when they are aggressively varying. Besides, one may appreciate the noise filtering ability of NMHE in Figs. 3.13b and 3.13c, as it successfully caters to the present measurement noise.

Lastly, the average execution times taken by NMHE, H-NMPC and L-NMPC are 2.0 ms, 0.6 ms and 0.7 ms, respectively. Moreover, the combined average execution time of the closed-loop is approximately 3.3 ms, which is substantially less than the adopted sampling time of 10 ms, and thus, validates the realtime feasibility of the closed-loop framework.

3.5 Conclusions

An instantaneous learning-based NMPC employed for the trajectory tracking of a tricopter aerial robot without violating the system constraints has been illustrated in this chapter. The proposed learning-based framework has been evaluated for two case studies. wherein the effect of three disturbance sources is individually analyzed in the first, while the effect of an uncertain model due to time-varying parameters is illustrated in the second. In the first case study, the high-level NMPC adapts itself to the changing working conditions owing to the learning of disturbance parameters by NMHE. As a result, z-directional tracking improvements of 85.7% and 57.9% have been realized for scenarios 1 and 2, respectively, whereas in scenario 3, 40.5% improvement in terms of the Euclidean error has been obtained for the hover with varying z-position reference. Moreover, owing to the fast C++ based execution of NMPC and NMHE codes, a complete onboard implementation of the closed-loop has been realized on a low-cost embedded processor. Similarly, in the second case study, the simulation results have manifested a sufficient tracking improvement due to learning. Furthermore, thanks to the RTI scheme of `ACADO` toolkit, a combined execution time below 4 milliseconds has exhibited the realtime feasibility within the second case study.

Chapter 4

Simple Learning Strategy for Feedback Linearization Control Method

Recall that for the safe operation of aerial robots in urban environments, a precise path tracking is of extreme importance. While the imprecise modeling or varying operational uncertainties degrade the performance of model-based controllers, learning algorithms are incorporated to make the former adaptive to the changing environment. In line with this motivation, another type of instantaneous learning technique, i.e., simple learning for feedback linearization control algorithm is presented in this chapter.

In the proposed simple learning-based FLC framework, the controller gains and the disturbance estimate are updated in the feedback control law by minimizing a cost function which is defined based on the closed-loop error dynamics of the nominal system. As a result, the SL-FLC framework maintains the nominal control performance in the absence of uncertainties and disturbances, while exhibiting robust control performance in their presence. Besides, it also ensures the desired closed-loop error dynamics in the presence of uncertainties and disturbances. The stability of the proposed approach is proven for a second-order uncertain nonlinear system. Also, it is illustrated that the SL strategy can find the global optimum point, and the controller gains and disturbance estimate converge to a finite value, thus resulting in a bounded control signal at the steady-state.

The outline of this chapter is as follows: Section 4.1 presents a brief literature overview discussing the widely utilized control techniques for uncertain nonlinear systems along with the learning algorithms that are commonly incorporated within the FLC scheme. Then, Section 4.2 describes the FLC's problem formulation followed by the deduction of the proposed learning strategy in Section 4.3. Thereafter, the simulation and experimental validations are presented in Sections 4.4 and 4.5, respectively. Finally, some conclusions are drawn in Section 4.6.

4.1 Literature Overview

Recall that the performance of the traditional FLC method is sensitive to uncertainties and disturbances in the system dynamics such that the closed-loop error dynamics of the system cannot converge to zero [115, 116]. To circumvent this, FLC with an integral action (FLC-I) scheme has been proposed in the literature to ensure robust control performance. While it can only handle time-invariant uncertainties, it also deteriorates nominal control performance in their absence [115].

To overcome the limitations of the traditional FLC method, several learning approaches have been utilized within the feedback linearization framework. In [117], an artificial NN with a nonlinear autoregressive-moving average model has been developed to learn the feedback linearized inputs for a nonlinear plant. In [118], the Gaussian process method has been used to identify a model for the FLC scheme with limited prior knowledge of the system. Also, a dynamic linearization method has been developed for discrete-time nonlinear systems in the literature [119–123]. To summarize, this method builds a linearized model that is equivalent to the real nonlinear system at each operation point of the closed-loop system dynamics. Since robots usually encounter varying working conditions, the aforementioned techniques may not be good candidates. This is because these techniques utilize the data which is generated in advance, which may no longer fully represent the system during operation. For instance, the total mass of an aerial robot can vary so that the system model changes over time.

Another nonlinear control approach using dynamic NN-based input-output feedback linearization has been proposed in [124]. In this approach, the controller generates a control signal that can eliminate the system's nonlinearities utilizing

the trained dynamic NN. The controller gains and the weights of the dynamic NN are selected by using particle swarm optimization (PSO) algorithm. Whereas PSO requires powerful computing platforms to obtain the optimal controller gains in a realtime operation, the proposed simple learning strategy can obtain the optimal controller gains in less than a millisecond on an inexpensive computing platform. This aspect is crucial for fast robotic applications, wherein there is a tendency to use cost-effective processors for which computationally efficient algorithms are needed.

Besides, both direct [41] and indirect [40] adaptive control approaches have been implemented for the control of aerial robots under uncertain conditions. However, an accurate system model is required as they aim to match the performance of a given (or estimated for indirect method) reference model. To overcome this requirement, some nonlinear adaptive methodologies have been proposed. For instance, [125] compares an adaptive SMC with the traditional FLC method for tracking control of a quadrotor. While the adaptive SMC shows improved tracking performance over the traditional FLC method, it assumes infinite switching times in the inputs which are unrealizable in reality, and thus, may lead to chattering effects. Another approach has been proposed in [126], wherein an adaptive backstepping technique is utilized for tracking control of a quadrotor with uncertain mass. Nevertheless, the proposed approach, like the indirect method, only corrects the parameter errors which may not be sufficient to bring down the tracking error in many applications.

4.2 Feedback Linearization Control

The considered second-order uncertain nonlinear system is of the following form:

$$\dot{x}_1 = x_2, \tag{4.1a}$$

$$\dot{x}_2 = f(\mathbf{x}, \mathtt{u}) + \Delta(\mathbf{x}) + \omega(t), \tag{4.1b}$$

where $\mathbf{x} = [x_1, x_2]^T \in \mathbb{R}^2$ is the state vector, $\mathtt{u} \in \mathbb{R}$ is the control input, and $\omega(t)$ is the time-varying external disturbance. Also, $f(\mathbf{x}, \mathtt{u}) \in \mathbb{R}$ and $\Delta(\mathbf{x}) \in \mathbb{R}$ are smooth, continuous differentiable and nonlinear functions, wherein $\Delta(\mathbf{x}) \in \mathbb{R}$ represents the modeling uncertainties. The objective is to find a control action $\mathtt{u}$ such that the system tracks the reference trajectory $\mathbf{r}(t) = [r_1(t), r_2(t)]^T$, where

$r_2 = \dot{r}_1$, with an acceptable accuracy while all the states and the control remain bounded.

4.2.1 Traditional FLC Method

To facilitate the tracking of the given reference trajectory, the traditional FLC method results in the following control law:

$$\mathtt{u} = \beta(\mathbf{x}, \mathtt{u}_\mathtt{b}^{\mathrm{FLC}}, \mathtt{u}_\mathtt{f}), \tag{4.2}$$

where $\mathtt{u}_\mathtt{b}^{\mathrm{FLC}}$ and $\mathtt{u}_\mathtt{f}$ are the feedback and feedforward control actions that are expressed as follows:

$$\mathtt{u}_\mathtt{b}^{\mathrm{FLC}} = \mathbf{ke} = k_1 e_1 + k_2 e_2, \tag{4.3}$$

$$\mathtt{u}_\mathtt{f} = \dot{r}_2. \tag{4.4}$$

In the above equations, the tracking error vector is represented by $\mathbf{e}(t) = [e_1(t), e_2(t)]^T$ wherein $e_1 = r_1 - x_1$ and $e_2 = r_2 - x_2$, and $\mathbf{k} = [k_1, k_2]$, $k_i > 0$ for $i = 1, 2$, represents the control gain vector. The feedback function $\beta(\mathbf{x}, \mathtt{u}_\mathtt{b}^{\mathrm{FLC}}, \mathtt{u}_\mathtt{f})$ is chosen such that:

$$f(\mathbf{x}, \beta(\mathbf{x}, \mathtt{u}_\mathtt{b}^{\mathrm{FLC}}, \mathtt{u}_\mathtt{f})) = \mathtt{u}_\mathtt{b}^{\mathrm{FLC}} + \mathtt{u}_\mathtt{f}, \tag{4.5}$$

and consequently, the closed-loop error dynamics take the following form:

$$\dot{e}_2 + k_2 e_2 + k_1 e_1 = -d(t). \tag{4.6}$$

Here, the term $d(t)$ represents a lumped disturbance parameter that comprises of modeling uncertainties and external disturbance in the closed-loop error dynamics and is expressed as:

$$d(t) = \Delta(\mathbf{x}) + \omega(\mathbf{t}). \tag{4.7}$$

Assumption 1. The lumped disturbance parameter in (4.6) is bounded such that $d^* = \sup_{t>0} |d(t)|$ exists.

Recall that, since $k_i > 0$ for $i = 1, 2$, the closed-loop error dynamics for $d(t) = 0$ are globally exponentially stable at $e_1 = e_2 = 0$.

Remark 15. It is to be noted that the current chapter assumes that the disturbances consist of a constant and vanishing perturbation terms. The cases where the disturbance includes non-vanishing perturbation terms are left as future work.

Remark 16. Equation (4.6) demonstrates that it is not possible to drive the error dynamics to the desired equilibrium point utilizing the FLC law proposed in (4.2). The reason being is the nonzero right-hand side in (4.6) that is due to the presence of modeling uncertainties in the system model and/or external disturbances. This illustrates why the traditional FLC method is sensitive to disturbances.

4.2.2 FLC with Integral Action

To make the FLC robust against modeling uncertainties as well as disturbances, an integral action can be added to the control law of the traditional FLC method. The resulting control law for FLC-I method would look similar to (4.2) except that the feedback control expression is modified to include the effect of the integral action:

$$\mathtt{u}_{\mathtt{b}}^{\text{FLC-I}} = \mathbf{ke} = k_1 e_1 + k_2 e_2 + k_{\text{int}} \int_0^t e_1 dt, \tag{4.8}$$

where $\mathbf{e} = [e_1, e_2, e_1]^T$ and $\mathbf{k} = [k_1, k_2, k_{\text{int}}]$ are the updated error and (positive) control gain vectors, respectively. Additionally, it is assumed that all the assumptions utilized in the traditional FLC's case are valid.

Applying the control law for FLC-I method to the uncertain nonlinear system in (4.1), the closed-loop error dynamics can be obtained as follows:

$$\ddot{e}_2 + k_2 \dot{e}_2 + k_1 \dot{e}_1 + k_{\text{int}} e_1 = -\dot{d}(t), \tag{4.9}$$

The above equation implies that if the modeling uncertainties and external disturbances have a steady-state value, i.e., $\dot{d}(t) = 0$, then the state can be driven to the desired equilibrium point.

Remark 17. Equation (4.9) shows that if the modeling uncertainties and external disturbances have a steady-state value, i.e, they are time-invariant, then the FLC-I method would successfully eliminate the steady-state error in the system. However, note that adding an integral action may result in a degraded performance

in comparison to the nominal performance as achieved with the traditional FLC method in the absence of uncertainties [115].

4.3 Simple Learning Strategy

The SL strategy for the FLC method is designed according to the desired closed-loop error dynamics of the nonlinear system. To make the FLC method adaptive, the SL strategy updates the controller gains and disturbance term within the FLC formulation by minimizing a cost function which is defined as the square of the closed-loop error dynamics. Consequently, the updated feedback control law is written as follows:

$$\mathtt{u}_{\mathtt{b}}^{\text{SL-FLC}} = \mathbf{ke} - \hat{d} = k_1 e_1 + k_2 e_2 - \hat{d}, \tag{4.10}$$

where $\hat{d}$ represents the estimated disturbance term. The closed-loop error dynamics can be written as:

$$\dot{e}_1 = e_2, \tag{4.11a}$$

$$\dot{e}_2 = -k_1 e_1 - k_2 e_2 - d + \hat{d}. \tag{4.11b}$$

4.3.1 Update Rules

The requirement to realize robust control performance by the FLC method is that the desired closed-loop error dynamics, defined as:

$$c(\mathbf{e}, \mathbf{k}^{\text{des}}) = \dot{e}_2 + k_2^{\text{des}} e_2 + k_1^{\text{des}} e_1, \tag{4.12}$$

should converge to zero. Since $\dot{e}_2 = -k_1 e_1 - k_2 e_2 - d + \hat{d}$, $c(\mathbf{e}, \mathbf{k}^{\text{des}})$ depends on k_1, k_2, $\hat{d}$, e_1, e_2, and d, i.e., the controller gains, disturbance estimate, error variables, and the disturbance. This forms the working principle within the SL-FLC algorithm i.e., to minimize a cost function that is defined based on the closed-loop error dynamics such that the system error and disturbance estimation error can instantaneously converge to zero. In that vein, the following closed-loop error

function (or cost function) is minimized which is the square of the desired closed-loop error dynamics defined in (4.12):

$$C(\mathbf{e}, \mathbf{k}^{\text{des}}) = \frac{1}{2}\Big(c(\mathbf{e}, \mathbf{k}^{\text{des}})\Big)^2. \tag{4.13}$$

The above equation implies that if the closed-loop error function $C(\,\cdot\,,\,\cdot\,)$ converges to zero, then the robust control performance condition, i.e., $c(\,\cdot\,,\,\cdot\,) = 0$, is satisfied such that the error will converge to zero.

In this chapter, a first-order iterative optimization algorithm, i.e., gradient descent is favored to minimize the closed-loop error function $C(\,\cdot\,,\,\cdot\,)$ due to its lower computational requirements. Recall that in a gradient descent approach, steps are taken proportional to the negative of the gradient of the closed-loop error function ($\nabla C(\,\cdot\,,\,\cdot\,)$), to find the minimum. Since the expression for the proposed closed-loop error function $C(\,\cdot\,,\,\cdot\,)$ comprises of the errors, controller gains, and disturbance estimates, its partial derivative is required for gradient computation. Hence, the following rule is utilized for updating the controller gains of the feedback linearization controller:

$$\dot{k}_i = -\alpha_i \frac{\partial C(\mathbf{e}, \mathbf{k}^{\text{des}})}{\partial k_i}, \tag{4.14}$$

where $\alpha_i > 0$ is the learning rate for the i^{th} controller gain. Utilizing the chain rule in (4.14), the expression can be rewritten as follows:

$$\dot{k}_i = -\alpha_i c(\mathbf{e}, \mathbf{k}^{\text{des}}) \frac{\partial c(\mathbf{e}, \mathbf{k}^{\text{des}})}{\partial k_i}. \tag{4.15}$$

It is implicit from the system definition in (4.1) that the controller gains appear only with $\dot{e}_2$ within the expression for the desired closed-loop error dynamics $c(\mathbf{e}, \mathbf{k}^{\text{des}})$. Thus, (4.15) can be rewritten as:

$$\dot{k}_i = -\alpha_i c(\mathbf{e}, \mathbf{k}^{\text{des}}) \frac{\partial \dot{e}_2}{\partial k_i}, \tag{4.16}$$

which eventually reduces to the following form. utilizing (4.11b):

$$\dot{k}_i = \alpha_i c(\mathbf{e}, \mathbf{k}^{\text{des}}) e_i. \tag{4.17}$$

Similarly, incorporating the gradient descent approach, the update rule for the disturbance estimate is written as:

$$\dot{\hat{d}} = -\alpha_{\hat{d}} \frac{\partial C(\mathbf{e}, \mathbf{k}^{\text{des}})}{\partial \hat{d}}, \tag{4.18}$$

where $\alpha_{\hat{d}} > 0$ is the learning rate for the disturbance estimate $\hat{d}$. Again, after using the chain rule and subsequently performing some algebraic manipulations, one can compute the final expression for the update rule as:

$$\dot{\hat{d}} = -\alpha_{\hat{d}} c(\mathbf{e}, \mathbf{k}^{\text{des}}). \tag{4.19}$$

It is emphasized that within (4.17) and (4.19), the controller gains and disturbance estimate are updated until the condition $c(\mathbf{e}, \mathbf{k}^{\text{des}}) = 0$ is fulfilled.

Remark 18. It is possible to design a nonlinear disturbance observer to estimate the lumped disturbance $d(t)$ following the methodology in [127]. The disturbance estimator design in this chapter is based on a gradient descent rule while, in general, nonlinear disturbance observers are designed based on Lyapunov techniques. In essence, both approaches follow the same control design strategy: (i) design a controller to achieve stability and other performance specifications (e.g., asymptotic tracking) assuming that the disturbance is measurable and available for feedback, (ii) design a disturbance estimator, and (iii) use the disturbance estimate in place of the disturbance in the control law. Furthermore, unlike any (nonlinear) disturbance observer-based control paradigm, the proposed control framework updates the controller gains of the underlying FLC method which further exhibits robust control performance in the presence of disturbances.

4.3.2 Stability Proof

In this chapter, a Routh-Hurwitz criterion-based stability analysis is utilized to prove the closed-loop system stability with the SL algorithm. In that vein, the closed-loop dynamics can be rewritten utilizing (4.11a) and (4.11b) as follows:

$$\ddot{e}_1 + k_2 \dot{e}_1 + k_1 e_1 - \hat{d} + d = 0. \tag{4.20}$$

Time differentiating the above equation and using the fact that $\dot{d} = 0$, the following expression is obtained,

$$\dddot{e}_1 + k_2\ddot{e}_1 + k_1\dot{e}_1 + \dot{k}_2\dot{e}_1 + \dot{k}_1 e_1 - \dot{\hat{d}} = 0. \tag{4.21}$$

Utilizing (4.12), (4.17), and (4.19), the expressions for $\dot{k}_i$ and $\dot{\hat{d}}$ can be derived as:

$$\dot{k}_i = \alpha_i(\ddot{e}_1 + k_2^{\text{des}}\dot{e}_1 + k_1^{\text{des}}e_1)e_i, \tag{4.22}$$

$$\dot{\hat{d}} = -\alpha_{\hat{d}}(\ddot{e}_1 + k_2^{\text{des}}\dot{e}_1 + k_1^{\text{des}}e_1). \tag{4.23}$$

Plugging the above expressions into the closed-loop dynamics and assuming $\boldsymbol{\eta} = [e_1\ \dot{e}_1\ \ddot{e}_1]^T$ denotes the state, yields the following:

$$\dddot{e}_1 + b_1(\boldsymbol{\eta})\ddot{e}_1 + b_2(\boldsymbol{\eta})\dot{e}_1 + b_3(\boldsymbol{\eta})e_1 = 0, \tag{4.24}$$

where,

$$\begin{aligned} b_1(\boldsymbol{\eta}) &= k_2 + \alpha_{\hat{d}} + \beta(\boldsymbol{\eta}), \quad b_2(\boldsymbol{\eta}) = k_1 + k_2^{\text{des}}(\alpha_{\hat{d}} + \beta(\boldsymbol{\eta})), \\ b_3(\boldsymbol{\eta}) &= k_1^{\text{des}}(\alpha_{\hat{d}} + \beta(\boldsymbol{\eta})), \quad \beta(\boldsymbol{\eta}) = \alpha_1 e_1^2 + \alpha_2 \dot{e}_1^2. \end{aligned} \tag{4.25}$$

The closed-loop dynamics (4.21) can now be expressed in a pseudo-linear form [128] as:

$$\dot{\boldsymbol{\eta}} = A(\boldsymbol{\eta})\boldsymbol{\eta}, \tag{4.26}$$

where,

$$A(\boldsymbol{\eta}) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -b_3(\boldsymbol{\eta}) & -b_2(\boldsymbol{\eta}) & -b_1(\boldsymbol{\eta}) \end{bmatrix}. \tag{4.27}$$

Clearly, $b_i(\boldsymbol{\eta}) > 0$, for $i = 1, 2, 3$, and $\forall\boldsymbol{\eta}$. By applying Routh-Hurwitz criterion, it can be shown that the eigenvalues of $A(0)$ are in the open left half-plane (i.e., the closed-loop system is asymptotically stable) if $b_1(0)b_2(0) > b_3(0)$, where,

$$b_1(0) = k_2 + \alpha_{\hat{d}}, \quad b_2(0) = k_1 + k_2^{\text{des}}\alpha_{\hat{d}}, \quad b_3(0) = k_1^{\text{des}}\alpha_{\hat{d}}, \tag{4.28}$$

or equivalently if:

$$k_2^{\text{des}}\alpha_{\hat{d}}^2 + (k_1 + k_2 k_2^{\text{des}} - k_1^{\text{des}})\alpha_{\hat{d}} + k_1 k_2 > 0. \tag{4.29}$$

In what follows, k_i, k_i^{des}, and $\alpha_{\hat{d}}$ are chosen such that this stability condition is satisfied.

Remark 19. In this chapter, it is assumed that the average rate of change of the induced disturbances is much lower than the states e_1 and e_2.

4.3.3 Global Minimum

The most important concern in SL strategy is that the closed-loop error dynamics may attain some local minima. Hence, this subsection provides an analytical proof to show that there exist no local minima for the considered formulation of the SL strategy. In particular, it is shown that the second derivatives of the cost function with respect to the variables k_i and $\hat{d}$ have the same sign. That is, the cost function does not have a change in the curvature sign through the variables implying that there is no local minima through these variables.

Utilizing the method in previous subsection, the second derivative of the cost function C with respect to k_i is obtained as follows:

$$\frac{\partial^2 C(\mathbf{e}, \mathbf{k}^{\text{des}})}{\partial k_i^2} = -e_i \underbrace{\frac{\partial (c(\mathbf{e}, \mathbf{k}^{\text{des}})}{\partial k_i}}_{-e_i} = (e_i)^2. \tag{4.30}$$

Similarly, the second derivative of the cost function with respect to the disturbance estimate $\hat{d}$ is computed as follows:

$$\frac{\partial^2 C(\mathbf{e}, \mathbf{k}^{\text{des}})}{\partial \hat{d}^2} = \underbrace{\frac{\partial (c(\mathbf{e}, \mathbf{k}^{\text{des}})}{\partial \hat{d}}}_{1} = 1. \tag{4.31}$$

Equations (4.30) and (4.31) show that the sign of the curvature of the cost function for the controller gains and the disturbance estimate is always positive; thus, there exist no local minima such that the closed-loop error dynamics reach to the global minimum. After reaching the global minimum, since α_i and α_d are constant and positive, the controller gains update law in (4.17) and the disturbance estimate update law in (4.19) converge to a finite value. Furthermore, a finite value for the coefficients in steady-state results in a bounded control action.

4.4 Simulation Study

This section illustrates the implementation of the proposed SL-FLC framework together with the traditional FLC and FLC-I methodologies on a simulation example. The main purpose of this simulation study is to demonstrate that the SL-FLC framework can ensure the desired closed-loop error dynamics in the presence of model uncertainties and external disturbances. Moreover, the second-order dynamical system considered for the simulation study is of the form (4.1), wherein the model function is of the following form:

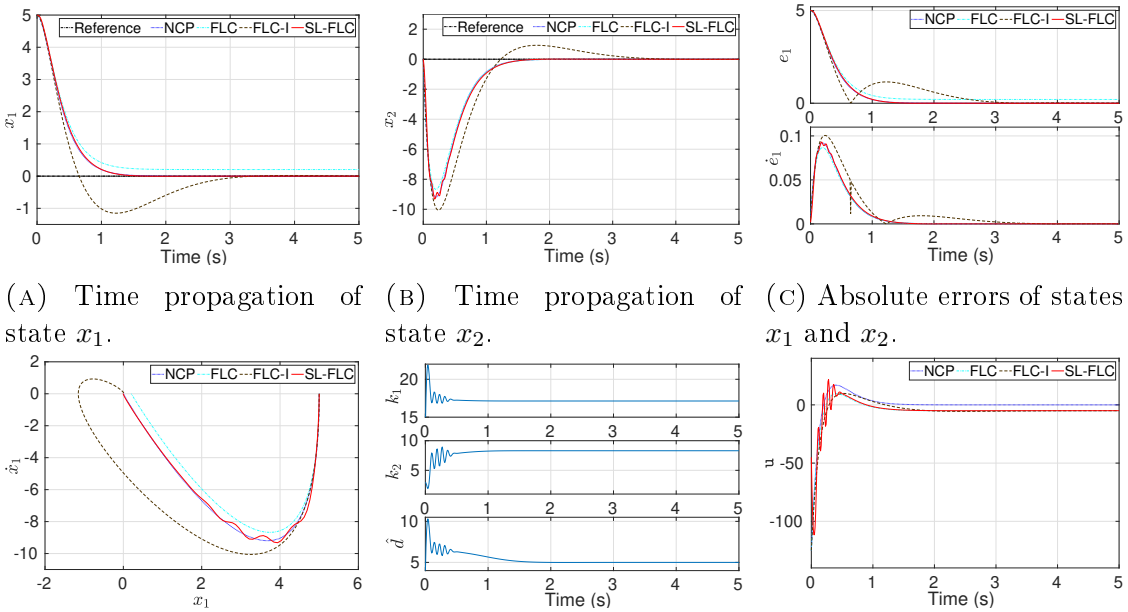
$$f(\mathbf{x}, \mathtt{u}) = x_2^3 + \mathtt{u}. \tag{4.32}$$

The entire simulation runs for a total of 5s with a sampling time of 0.01s. The initial conditions on the states are $\mathbf{x} = [5, 0]^T$, whereas the reference signal $\mathbf{r}$ is set to zero throughout the simulation time. The controller gains for the desired closed-loop error dynamics ($c = \ddot{e}_1 + k_2^{\text{des}}\dot{e}_1 + k_1^{\text{des}}e_1$) are selected as $\mathbf{k}^{\text{des}} = [k_1^{\text{des}}, k_2^{\text{des}}]^T = [25, 10]^T$. Moreover, the initial conditions for the controller gains are selected as $\mathbf{k}(0) = [9, 3]^T$, while the initial condition for the disturbance estimate is taken as $\hat{d}(0) = 0$. Furthermore, the learning rates α_i and α_d are set to 0.75 and 3, respectively.

To illustrate the desired closed-loop error dynamics ensuring-capability of the SL-FLC framework, i.e., realizing the nominal control performance (NCP)[1], the SL-FLC framework controls the aforementioned system in the presence of model uncertainties and external disturbance. Besides, its control performance is compared with the traditional FLC and FLC-I methodologies. In terms of the uncertainties, an external force $\omega(t) = 5$, is imposed as the external disturbance while there exist modeling uncertainties in the form: $\Delta(\mathbf{x}) = (x_1)^2$ on the system. Additionally, utilizing the lumped disturbance definition in (4.7), the disturbance imposed on the system is $d(t) = 5$, after the states converge to zero.

The performance of all the controller are shown in Figs. 4.1a and 4.1b. Note that NCP depicts the nominal control performance in the absence of any uncertainty and disturbance, while the FLC method, FLC-I method, and SL-FLC framework show

[1]Note that NCP refers to the control performance of the system controlled by traditional FLC method in the absence of model uncertainties and external disturbance



(A) Time propagation of state x_1.

(B) Time propagation of state x_2.

(C) Absolute errors of states x_1 and x_2.

(D) Phase portraits of states x_1 and x_2.

(E) Adaptations of controller gains and disturbance estimate.

(F) Control signals from different controllers.

FIGURE 4.1: Tracking performance of NCP, traditional FLC method, FLC-I method, and SL-FLC framework for a second-order dynamical system in the presence of modeling uncertainties and external disturbance. From the results, it is evident that even in the presence of disturbance, the SL-FLC framework can match the nominal control performance.

the control performance in the presence of model uncertainties and external disturbance. As visualized from the figures, while the FLC and FLC-I methodologies are unable to ensure the nominal control performance (i.e., the desired closed-loop error dynamics), the SL-FLC algorithm ensures it in the presence of disturbances. Moreover, the FLC method is not robust to disturbances as stated in Remark 16 while the FLC-I method provides robust control performance against disturbance. However, the FLC-I method causes undesired effects such as overshoots and large settling time as stated in Remark 17.

The absolute values of the errors and error rates are shown in Fig. 4.1c. The SL-FLC framework realizes a similar performance as that of the NCP which validates that it ensures the nominal control performance in the presence of model uncertainties and external disturbance. Moreover, the error of the system controlled by the FLC method cannot converge to zero, whereas the error of the system controlled by the FLC-I method can converge to zero with a large settling time. In addition, the phase portraits for all the controllers are shown in Fig. 4.1d. It is visualized

that the phase portrait for the SL-FLC framework converges to the nominal phase portrait (i.e., NCP) in the presence disturbances. This again implies that the SL-FLC framework is capable of ensuring the desired closed-loop error dynamics of the system in the presence of modeling uncertainties and external disturbance.

The time updates of the controller gains and disturbance estimate for the SL-FLC framework are presented in Fig. 4.1e. As observed, the controller gains converge to constant values while the disturbance estimate reaches the true value $d(t) = 5$, in the steady-state. Furthermore, the control inputs for the NCP, FLC method, FLC-I method, and SL-FLC framework are shown in Fig. 4.1f. Note that, in steady-state response, the control signal for the NCP is equal to zero since there exists no disturbance, whereas for the others the control signals are nonzero due to the external disturbance imposed on the system.

4.5 Experimental Validation

This section exhibits the experimental validation of the proposed SL-FLC framework on an aerial robot. First, it briefly describes the considered robot along with its model. Then, it illustrates the design of the SL-FLC framework as a high-level controller for position tracking. Thereafter, it presents the tracking results of the SL-FLC framework and the traditional FLC method for the three disturbance scenarios.

4.5.1 Aerial Robot

To test the efficacy of the proposed algorithm, the same 3D printed tilt-rotor tricopter shown in Fig. 2.4 is incorporated, as utilized in Section 3.4.1.1. Moreover, the combined discrete-time nonlinear model at high-level is of the form:

$$\mathbf{x}_{k+1} = \mathbf{f}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k) + \boldsymbol{\omega}_k, \quad \mathbf{z}_k = \mathbf{x}_k + \boldsymbol{\nu}_k, \tag{4.33}$$

wherein all the uncertainties are expressed together in the form of process noise $\boldsymbol{\omega}$. Also, the state vector $\mathbf{x} \in \mathbb{R}^6 = [x, y, z, u, v, w]^T$, the control vector $\mathbf{u} \in \mathbb{R}^3 = [\phi, \theta, F_z]^T$, and the nominal function $\mathbf{f}_{\mathtt{d}}(\,\cdot\,,\,\cdot\,) : \mathbb{R}^6 \times \mathbb{R}^3 \mapsto \mathbb{R}^6$.

4.5.2 SL-FLC Framework Design

Three SL-FLC frameworks are independently employed to achieve a precise tracking performance by the tricopter. The overall control framework is illustrated in Fig. 4.2. In principle, they simultaneously perform position as well as velocity control along the three x, y, and z axes. Once the position reference ($\mathbf{x}^{\mathrm{r}}_{\mathrm{pos}}$) is given to the controller, the desired velocities ($\mathbf{x}^{\mathrm{r}}_{\mathrm{vel}} = [u^{\mathrm{r}}, v^{\mathrm{r}}, w^{\mathrm{r}}]^T$) are computed as the rate of change of the position setpoints, i.e., $\mathbf{x}^{\mathrm{r}}_{\mathrm{vel}} = \dot{\mathbf{x}}^{\mathrm{r}}_{\mathrm{pos}}$[2]. Subsequently, these velocity references along with the position reference are utilized in computing the final attitude angles and throttle commands ($\mathbf{u}_{\text{SL-FLC}} = [\phi^{\mathrm{des}}, \theta^{\mathrm{des}}, F_z^{\mathrm{des}}]^T$) that are given to the low-level controller. Note that the low-level controller is designed with the similar P-PID architecture as illustrated in Section 3.4.1.2. Moreover, using the force equations given in (2.28), the update rules for the three feedback linearization controllers are obtained as follows:

$$\theta^{\mathrm{des}} = sin^{-1}\left[\frac{1}{g}\left\{qw - rv + \dot{u}^{\mathrm{r}} + k_u(u^{\mathrm{r}} - u) + k_x(x^{\mathrm{r}} - x) - \hat{d}_u\right\}\right], \tag{4.34}$$

$$\phi^{\mathrm{des}} = sin^{-1}\left[\frac{1}{gcos(\theta^{\mathrm{des}})}\left\{pw - ru - \dot{v}^{\mathrm{r}} - k_v(v^{\mathrm{r}} - v) - k_y(y^{\mathrm{r}} - y) + \hat{d}_v\right\}\right], \tag{4.35}$$

$$F_z^{\mathrm{des}} = m\left\{pv - qu + gcos(\phi^{\mathrm{des}})cos(\theta^{\mathrm{des}}) + \dot{w}^{\mathrm{r}} + k_w(w^{\mathrm{r}} - w) + k_z(z^{\mathrm{r}} - z) - \hat{d}_w\right\}, \tag{4.36}$$

where k_x, k_y, k_z and k_u, k_v, k_w are the controller gains for position and velocity, respectively. In addition, the terms $\hat{d}_u$, $\hat{d}_v$, $\hat{d}_w$ are the disturbance estimates along the three axes.

Recall that the proposed SL strategy makes the FLC method adaptive to the changing operational conditions. It updates the controller gains and the disturbance estimates according to the expressions in (4.17) and (4.19), respectively, while utilizing the position error, and the velocity errors along with their first derivatives. Moreover, for this application, the expressions for the desired closed-loop error dynamics along the three axes reduce to:

$$c_x(\mathbf{e}, \mathbf{k}^{\mathrm{des}}) = \ddot{e}_x + k_u^{\mathrm{des}}\dot{e}_x + k_x^{\mathrm{des}}e_x, \tag{4.37}$$

$$c_y(\mathbf{e}, \mathbf{k}^{\mathrm{des}}) = \ddot{e}_y + k_v^{\mathrm{des}}\dot{e}_y + k_y^{\mathrm{des}}e_y, \tag{4.38}$$

$$c_z(\mathbf{e}, \mathbf{k}^{\mathrm{des}}) = \ddot{e}_z + k_w^{\mathrm{des}}\dot{e}_z + k_z^{\mathrm{des}}e_z, \tag{4.39}$$

[2]It is to be noted that the position derivatives are filtered to ensure smooth velocity references.

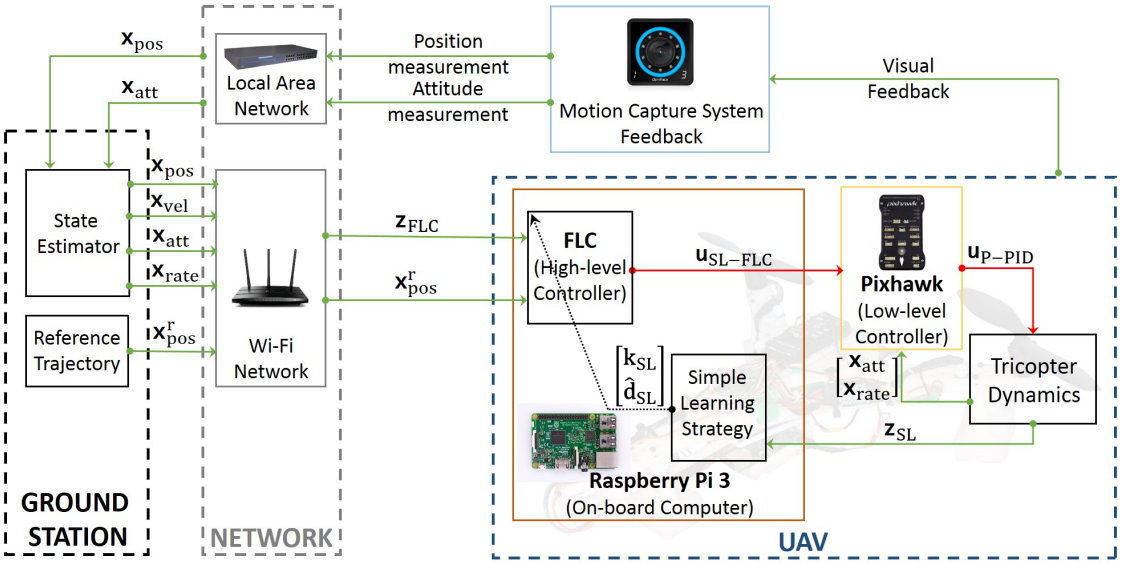


FIGURE 4.2: Realtime implementation: block diagram of the overall implementation, wherein the block naming 'Raspberry Pi 3' represents the on-board computer which executes the proposed SL-FLC framework. Also, the control output of the low-level controller in Pixhawk is: $\mathbf{u}_{\text{P-PID}} = [\Omega_1, \Omega_2, \Omega_3, \mu]^T$. Notation: $\mathbf{z}_{\text{FLC}} = [x, y, z, u, v, w, p, q, r]^T$; $\mathbf{z}_{\text{SL}} = [x, y, z, u, v, w]^T$; $\mathbf{k}_{\text{SL}} = [k_x, k_y, k_z, k_u, k_v, k_w]^T$; $\hat{\mathbf{d}}_{\text{SL}} = [\hat{d}_u, \hat{d}_v, \hat{d}_w]^T$.

where e_x, e_y, e_z are the translational position error, $\dot{e}_x$, $\dot{e}_y$, $\dot{e}_z$ are the translational velocity error and $\ddot{e}_x$, $\ddot{e}_y$, $\ddot{e}_z$ are the translational acceleration error. Also, the terms k_x^{des}, k_y^{des}, k_z^{des} and k_u^{des}, k_v^{des}, k_w^{des} are the desired FLC gains for position and velocity, respectively.

Remark 20. To obtain the above update rules, the external forces F_x and F_y are taken to be zero within the force equations (2.28).

Remark 21. Due to a time-varying trajectory, a nonzero tracking error always occurs. Therefore, to avoid the continuous update of the controller gain values for insignificant errors, a dead-zone (or saturation zone) is implemented. That is, below a certain threshold, the tracking errors (and their derivatives) are taken to be zero within the cost function. In practice, the threshold must be selected to be less than the desired accuracy but more than the noise on the measurements.

Remark 22. In this thesis, two InLC-type frameworks are illustrated, namely, NMPC-NMHE and SL-FLC. The former incorporates NMHE to estimate the model parameters which subsequently update the controller definition. On the other hand, the latter is similar to a direct-type adaptive controller, wherein the controller parameters along with the disturbance estimate are updated to achieve the desired

closed-loop error dynamics. Both have their own merits and demerits. For instance, since NMHE solves a dynamic optimization problem over a finite horizon, it is computationally expensive yet it can accommodate system constants. Whereas, the SL strategy solves a static optimization problem via gradient descent approach, which makes it computationally less demanding. However, according to the current formulation, the constants cannot be incorporated systematically. Besides, since it updates the controller parameters based on the trajectory error, implementation of a dead-zone is mandatory, as also discussed in Remark 21.

4.5.3 Tracking Results

Next, the realtime tracking results of the proposed SL-FLC framework are illustrated. In addition to the tracking performance, its robustness to varying uncertainties is also analyzed that are explicitly induced in the form of: (i) mass variation, (ii) ground effect, and (iii) wind gust. To appreciate the importance of learning, all the experiments are performed with two types of controllers, namely, the traditional FLC method (without learning) and the FLC method incorporating the SL algorithm (i.e. SL-FLC framework).

All the control codes are written in C++ for their efficient execution on-board the Raspberry Pi 3 in a ROS environment. The average sampling frequency achieved for the controllers throughout each experiment is about 100-Hz. Additionally, the same OptiTrack motion capture system is utilized for indoor experiments, as also depicted in Fig. 4.3. Moreover, a modified version of the block diagram as shown previously in 3.5, illustrating the overall realtime implementation can be visualized in Fig. 4.2.

The following gains for the desired closed-loop error dynamics are obtained by the trial-and-error procedure which are directly used within the traditional FLC method:

$$\begin{aligned} k_x^{\text{des}} &= 4.5, \quad k_y^{\text{des}} = 4.7, \quad k_z^{\text{des}} = 8.0, \\ k_u^{\text{des}} &= 4.0, \quad k_v^{\text{des}} = 4.1, \quad k_w^{\text{des}} = 3.8. \end{aligned} \tag{4.40}$$

On the other hand, the controller gains and learning rates utilized for initializing the SL-FLC framework are as follows:

$$\begin{aligned} &k_x(0) = 4.3, \quad k_y(0) = 4.5, \quad k_z(0) = 7.7, \\ &k_u(0) = 3.8, \quad k_v(0) = 3.9, \quad k_w(0) = 3.6, \end{aligned} \tag{4.41}$$

$$\begin{aligned} &\alpha_x = 0.02, \quad \alpha_y = 0.02, \quad \alpha_z = 0.03, \\ &\alpha_u = 0.02, \quad \alpha_v = 0.02, \quad \alpha_w = 0.03. \end{aligned} \tag{4.42}$$

The above control gains and learning rates are also obtained via the trial-and-error procedure such that a stable response is obtained. In the subsequent part, each of the three experiments is discussed in detail. It is emphasized that different trajectories are utilized for all experiments in order to exhibit the robustness of the control framework towards various trajectories.

Case Scenario I - Mass Variation

The first experiment discusses a package delivery problem as depicted in Fig. 4.4. Unlike the previous payload dropping experiment, presented in case 1 of Section 3.4.1.3, a time-based 8-shaped trajectory of length 2.6m along x-direction and 1.3m along y-direction is incorporated, while the tricopter is initialized with $\mathbf{x}(0) = [0, 0, 1.5, 0, 0, 0]^T$. Moreover, for both traditional FLC method and SL-FLC framework, first a complete 8-shaped trajectory is tracked by the tricopter with a full payload, followed by the dropping of each payload block once per lap at fixed time intervals in the increasing order of their mass: $86g \rightarrow 114g \rightarrow 114g \rightarrow 143g$. In addition to the controller tuning based (4.40) - (4.42), the initial disturbance vector and the corresponding learning rates are selected to be:

$$\begin{aligned} &\hat{d}_u(0) = 0.15, \quad \hat{d}_v(0) = -0.02, \quad \hat{d}_u(0) = -0.3, \\ &\alpha_{\hat{d}_u} = 0.2, \quad \alpha_{\hat{d}_v} = 0.2, \quad \alpha_{\hat{d}_w} = 0.25. \end{aligned}$$

Note that the above initial disturbance values result in an offset-free hover tracking of the tricopter by the traditional FLC method.

Position tracking performance of the traditional FLC method and the SL-FLC

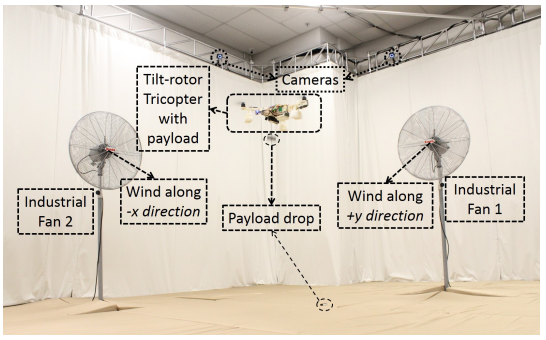


FIGURE 4.3: Experimental setup: 3D-printed tilt-rotor tricopter in the motion capture system lab along with two industrial fans.

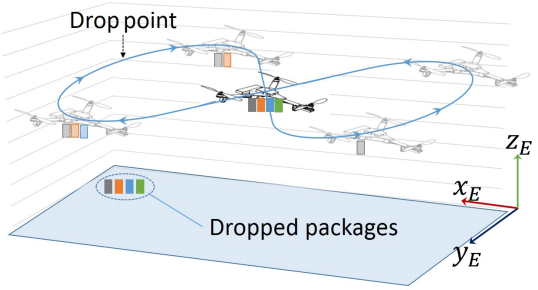
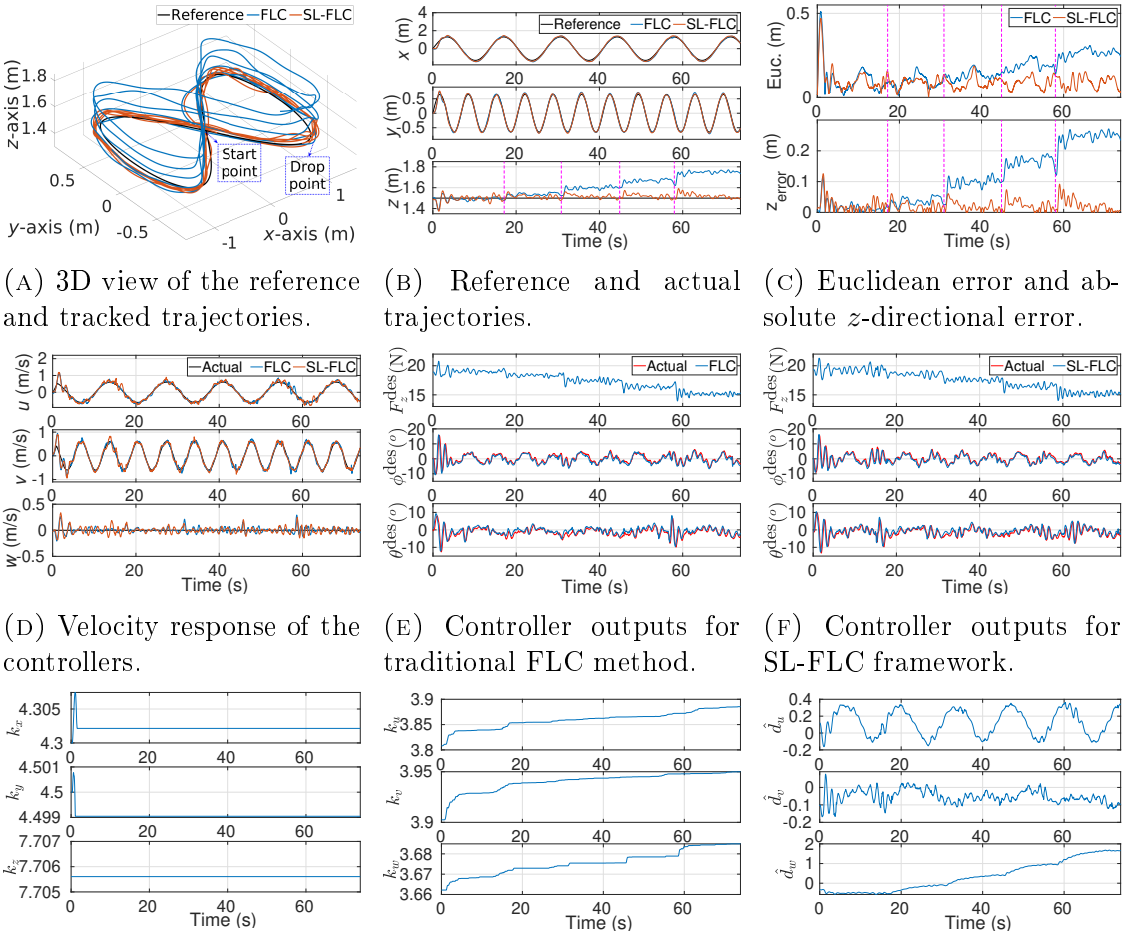


FIGURE 4.4: Package delivery problem in case scenario I.

framework can be seen in Figs. 4.5a and 4.5b. As visualized, tracking along z-direction for the SL-FLC framework is much precise when compared to the traditional FLC method. This is in accordance with our anticipation as the SL algorithm appropriately updates the control and disturbance parameters to compensate for the disturbances due to sequential payload drops. As a result, the plant-model uncertainties that arise at the dropping instant diminish with time which eventually results in a precise tracking performance by the SL-FLC framework. Apart from the qualitative comparison, the Euclidean error and the absolute error along z-direction (z_{error}) are depicted in Fig. 4.5c for both the controllers. While the mean values for Euclidean error and z_{error} in case of the traditional FLC method are 0.1573m and 0.1103m, respectively, the corresponding mean values for the SL-FLC framework are 0.0923m and 0.0220m. This implies that the SL algorithm helped the FLC method to reduce the Euclidean error by 6.5 cm and z_{error} by 8.83 cm, over the traditional FLC method.

The velocity tracking performance is presented in Fig. 4.5d. It is seen that both the controllers resulted in a similar performance. Mostly, the responses are overlapping but a slight more oscillatory behavior is observed with the SL-FLC framework. This oscillatory response of the SL-FLC framework can be explained due to the transition (or learning) phase. Besides, the control outputs for both the controllers are presented in Figs. 4.5e and 4.5f, wherein a similar performance by both the controllers is observed. Another point to take note in Figs. 4.5e and 4.5f is that the low-level controller of Pixhawk is well-tuned such that it is able to accurately follow the references from high-level FLC method. Finally, the variation of the control gains for position and velocity within the SL-FLC framework are plotted in



(A) 3D view of the reference and tracked trajectories.

(B) Reference and actual trajectories.

(C) Euclidean error and absolute z-directional error.

(D) Velocity response of the controllers.

(E) Controller outputs for traditional FLC method.

(F) Controller outputs for SL-FLC framework.

(G) Controller gains for position.

(H) Controller gains for velocity.

(I) Disturbance estimates from SL algorithm.

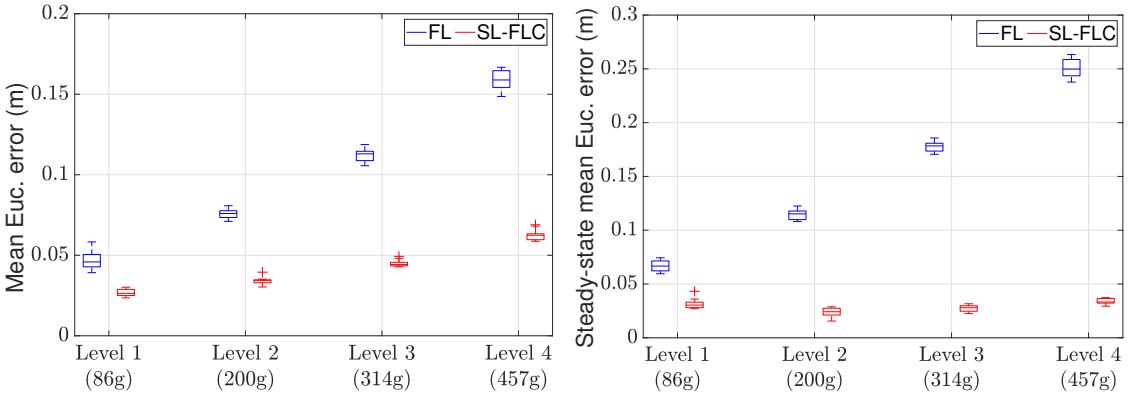
FIGURE 4.5: Case Scenario I: trajectory tracking performance of the tilt-rotor tricopter with varying mass for the 8-shaped reference, where the vertical magenta lines represent the instants of payload drop.

Figs. 4.5g and 4.5h, respectively. While there is no substantial change for position gains (except an abrupt change in the beginning), there is a significant increase in the velocity gains, especially k_w, with each payload drop. Similarly, the disturbance estimates that are presented in Fig. 4.5i, get accurately estimated throughout the experiment such that the SL-FLC framework results in a precise trajectory tracking performance.

Remark 23. It is to be noted that the effect of mass disturbance is not visible in the controller gains for position. The reason for this behavior is that the induced force disturbance is only acting along the force equations (given in (2.28)), which govern the velocity response of the system.

In order to illustrate that the viability of the proposed SL-FLC framework is not only limited to small weight variations, additional statistical tests are performed. Within each test, four different levels of weight drops, with each successive drop larger than the previous, are executed while the tricopter is hovering at a stationary point ($\mathbf{x}^{\mathrm{r}}_{\mathrm{pos}} = [0, 0, 1.5]^T$). Moreover, for each weight drop level, the experiments are repeated ten times while the mean Euclidean error values for both the controllers are recorded in a box plot which is presented in Fig. 4.6a. It is evident that the mean Euclidean error for the traditional FLC method increases with the rising weight drop level, whereas the error rise for the SL-FLC framework remains comparably negligible. This again is per expectation as the learning helps the FLC method to adapt according to changing environment even when the magnitude of change is abruptly varying.

Remark 24. In Fig. 4.6a, one may expect to see a constant mean Euclidean error with the SL-FLC framework for all levels of weight drop. However, it is to be noted that rising mean Euclidean error for the SL-FLC framework is essentially due to the error at the dropping instant. That is, for heavier weight drops, the instantaneous error at the dropping instant gets higher which subsequently diminishes over time. Therefore, the overall mean Euclidean error increases by a small amount with the increasing weight drop level. This argument is further validated in the box plot for steady-state mean Euclidean error presented in Fig. 4.6b.



(A) Mean Euclidean error for the entire flight.

(B) Mean Euclidean error at the steady-state.

FIGURE 4.6: Statistical results for four levels of abrupt weight drops, wherein ten tests for each weight level are performed.

Case Scenario II - Ground Effect

The second experiment analyzes the ground effect disturbance that is commonly encountered during takeoff, landing and ground proximity flying. Similar to the case 2 of Section 3.4.1.3, the tricopter tracks a circular trajectory of 1m radius incorporating two height levels, namely, 0.8m and 0.125m, as can be visualized in Fig. 4.7a. In the beginning, tricopter starts to follow the circular trajectory with state vector $\mathbf{x}(0) = [0, 0, 0.8, 0, 0, 0]^T$ while under negligible influence of the ground effect. Then, on the circular path, it gradually descends to $z = 0.125$m where the ground effect gets dominating. After completing about half-a-circle, it climbs back to $z = 0.8$m and subsequently, repeats the lap for another time.

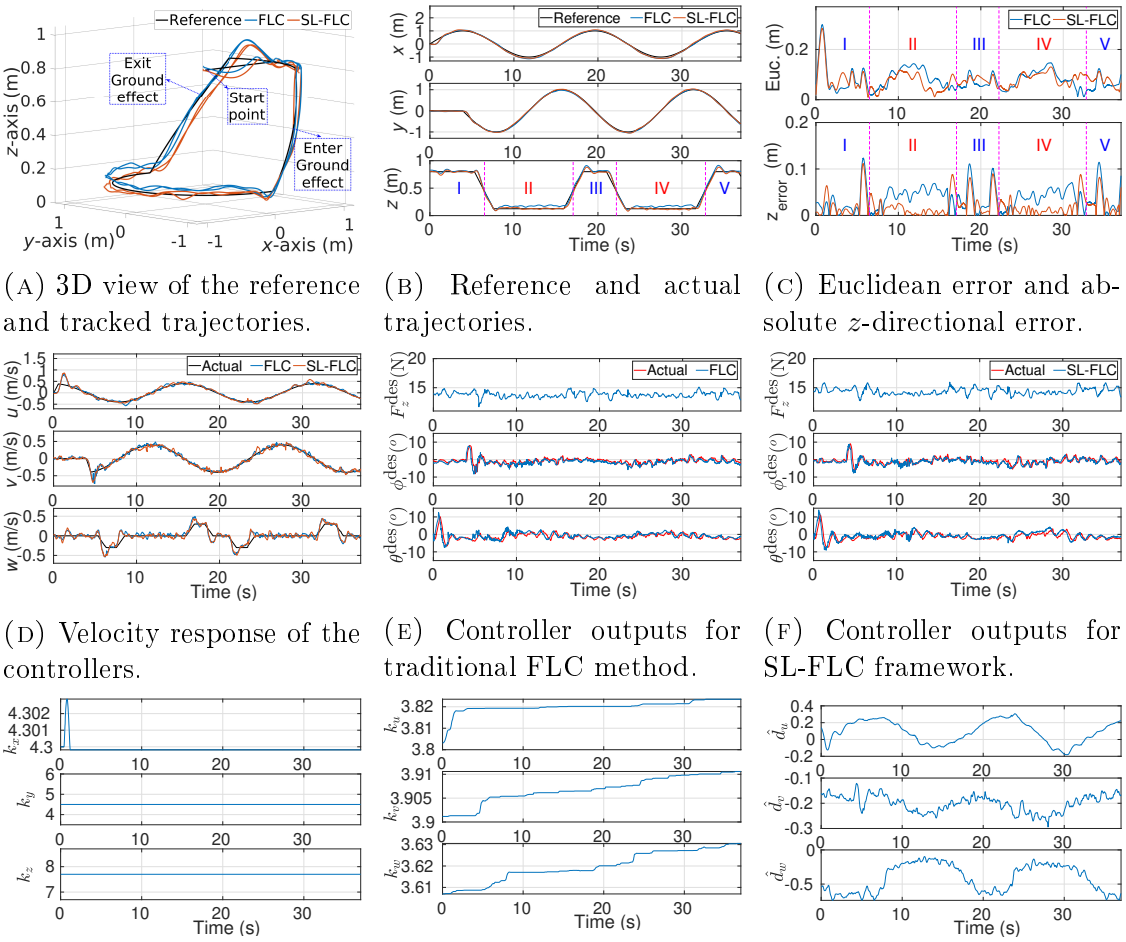
Both the controllers are tuned utilizing the control gains as specified in (4.40) and (4.41), and the SL-FLC framework is designed with the controller learning rates given in (4.42). Additionally, for this scenario, the initial value for disturbance vector and corresponding learning rates in the SL-FLC framework are selected to be:

$$\hat{d}_u(0) = 0.15, \quad \hat{d}_v(0) = -0.02, \quad \hat{d}_u(0) = -0.3,$$
$$\alpha_{\hat{d}_u} = 0.2, \quad \alpha_{\hat{d}_v} = 0.2, \quad \alpha_{\hat{d}_w} = 0.5.$$

Note that to increase the ground clearance for the tricopter, the foam attached to its landing skid is removed which further reduced its empty weight to $m = 1.410$ kg. This is the reason why a new disturbance vector is obtained.

Remark 25. In terms of the disturbance learning rates, a higher value for $\alpha_{\hat{d}_w}$ is utilized as the disturbance due to ground effect is expected to vary at a higher rate in comparison to the previous case scenario.

The position tracking performance for the traditional FLC method and the SL-FLC framework are presented in Figs. 4.7a and 4.7b. As anticipated, when the tricopter is not exposed to ground effect both result in similar tracking performance. However, the SL-FLC framework dominates the traditional FLC method under the ground effect influence, primarily in terms of z-directional tracking. The superiority is best visualized from the Euclidean error and z_{error} plots presented in Fig. 4.7c. Particularly in the z_{error} plot, it is seen that the error for the traditional FLC method is always more in comparison to the SL-FLC framework. Although there



(A) 3D view of the reference and tracked trajectories.

(B) Reference and actual trajectories.

(C) Euclidean error and absolute z-directional error.

(D) Velocity response of the controllers.

(E) Controller outputs for traditional FLC method.

(F) Controller outputs for SL-FLC framework.

(G) Controller gains for position.

(H) Controller gains for velocity.

(I) Disturbance estimates from SL algorithm.

FIGURE 4.7: Case Scenario II: trajectory tracking performance of the tilt-rotor tricopter under the influence of ground effect. It is to be noted that in Figs. (B) and (C) the vertical magenta lines represent the transition points, wherein the areas I, III, and V are the regions without ground effect influence while areas II and IV are with the influence of ground effect.

are instances (mainly during descending and climbing) when the errors for both the controllers get almost the same, these instances are mostly associated with the transition response of the SL-FLC framework. That is, since the learning is in progress for the SL-FLC framework during that phase, the tracking performance degrades a bit. Moreover, the mean values for Euclidean error and z_{error} in the traditional FLC's case are 0.0803m and 0.0369m, respectively, while for the SL-FLC framework, the respective mean error values are 0.0783m and 0.0213m. Hence, the SL algorithm resulted in an improvement of 0.2 cm in terms of Euclidean error and 1.56 cm in terms of z_{error}. At first glance, these numbers may appear less. However, if one may look closely at Fig. 4.7c, it is evident that within the ground

effect zone (represented by areas II and IV), the maximum z_{error} for the traditional FLC method reaches until 7 cm, whereas for the SL-FLC framework z_{error} always stays below 2.5 cm. This difference could be very crucial in certain situations, for instance, search and rescue, where a precise path tracking is required from the robot while staying very close to the ground.

Even in this case scenario, the velocity tracking results for both the controllers, presented in Fig. 4.7d, are similar. Their plots are mostly overlapping but with a bit more oscillatory response from the SL-FLC framework. In terms of the control outputs, again a similar performance is observed by both, as also visualized from Figs. 4.7e and 4.7f. Furthermore, the variation of the position and velocity control gains are presented in Figs. 4.7g and 4.7h. Although the position control gains do not change much during the trajectory (same reason as Remark 23), the velocity control gains get updated whenever a disturbance is induced. Note that even though there is no explicit induction of the disturbance along x and y directions, the velocity control gains are also updated to achieve a precise tracking performance. Finally, disturbance estimates from the SL algorithm are presented in Fig. 4.7i. They vary with the induced disturbance to result in an offset-free tracking. Moreover, from the plot of $\hat{d}_w$, it is deduced that whenever the tricopter enters the ground effect zone, the magnitude of $\hat{d}_w$ increases and decreases for vice-versa. This is also per intuition as the robot is expected to experience a positive force along z-direction under the ground effect influence.

Case Scenario III - Wind Gust

The third experiment examines the wind gust disturbance that is also frequently experienced during outdoor flights. In a similar way to the case 3 of Section 3.4.1.3, the wind gust disturbance are artificially induced with the help of two industrial fans that are placed perpendicular to each other at a distance of 2.2 m from the origin, as can be seen in Fig. 4.3. For this experiment, they produce wind with speeds in the range of 4.2-4.8 m/s (each fan) along $-x$- and $+y$-directions. To test the efficacy of SL-FLC framework in compensating for the wind gust disturbance, the trajectory tracking performance analyzed for a hover with varying z-position reference. At the start, the state of the tricopter is $\mathbf{x}(0) = [0, 0, 1.3, 0, 0, 0]^T$, and then, the z-position reference varies sinusoidally with magnitude ± 0.5m. Note

that the wind gust disturbance is induced from the very beginning for both the controllers.

Remark 26. It is worth to mention that due to a changing z-position reference, the overall magnitude of wind gust disturbance acting on the tricopter varies with time. However, it is emphasized that the average change in the wind gust disturbance is much slower than the states e_1 and e_2. Consequently, our assumption $\dot{d} = 0$ is still valid.

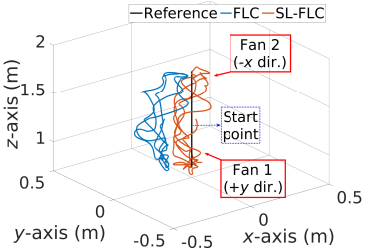
Again, both the controllers are tuned utilizing the control gains in (4.40) and (4.41), and the SL-FLC framework is designed with the controller learning rates given in (4.42). In addition, the initial value for the disturbance vector and the corresponding learning rates in the SL-FLC framework are selected to be:

$$\hat{d}_u(0) = 0.15, \quad \hat{d}_v(0) = -0.02, \quad \hat{d}_u(0) = -0.3,$$
$$\alpha_{\hat{d}_u} = 0.4, \quad \alpha_{\hat{d}_v} = 0.4, \quad \alpha_{\hat{d}_w} = 0.4,$$

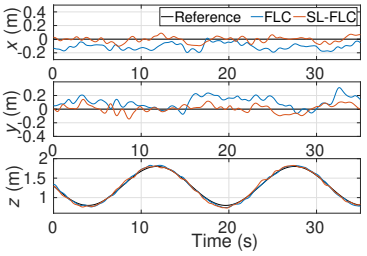
which are obtained by the trial-and-error procedure.

Remark 27. During the tuning phase for the SL-FLC framework, it is observed that for low values (less than 0.2) of disturbance learning rates, the disturbance estimation is unable to follow the variation within the induced wind gust disturbance due to which the overall control performance is poor. On the other hand, for the values higher than 0.5, the disturbance estimation follows the variation of wind gust disturbance but with high-frequency oscillations. Hence, the values in between these extreme limits are utilized in the experiment.

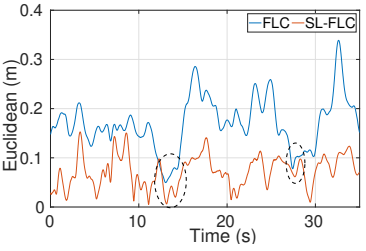
Position tracking results for both, traditional FLC method and SL-FLC framework, in the presence of wind gust disturbance are presented in Figs. 4.8a and 4.8b. As anticipated, the SL algorithm helps the FLC method to predict and compensate for the acting wind gust disturbance. Consequently, the tricopter is able to closely follow the commanded trajectory. On the other hand, there lies a constant offset (along $-x$- and $+y$-directions) in tracking performance by the traditional FLC method, which is mainly because of the induced plant-model uncertainties. Additionally, the superior tracking by the SL-FLC framework is even more explicit in Fig. 4.8c, where the Euclidean errors for both the controllers are shown. As can be seen, the Euclidean error for the SL-FLC framework stays substantially below



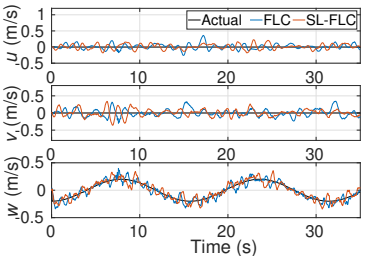
(A) 3D view of the reference and tracked trajectories.



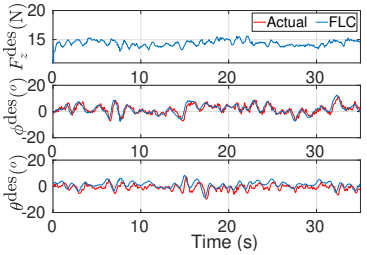
(B) Reference and actual trajectories.



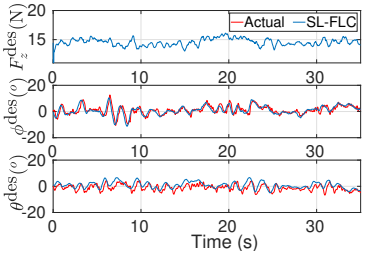
(C) Time propagation of Euclidean error.



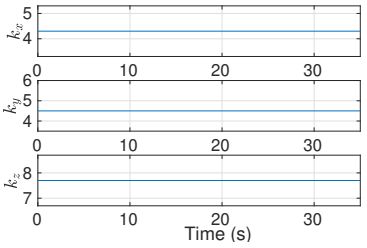
(D) Velocity response for the controllers.



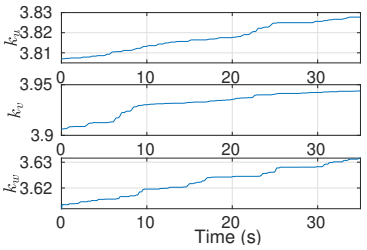
(E) Controller outputs for traditional FLC method.



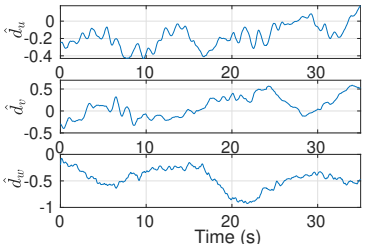
(F) Controller outputs for SL-FLC.



(G) Controller gains for position.



(H) Controller gains for velocity.



(I) Disturbance estimates from SL algorithm.

FIGURE 4.8: Case Scenario III: trajectory tracking performance of the tilt-rotor tricopter in presence of wind gust disturbance for a hover with varying z-position reference.

the Euclidean error for the traditional FLC method throughout the experiment. Moreover, the mean Euclidean error values for the traditional FLC method and the SL-FLC framework are 0.1713m and 0.0725m, respectively, which eventually illustrates an improvement of 9.88 cm due to the SL algorithm for this case scenario.

Remark 28. In Fig. 4.8c, one may notice two instances (marked by two black ellipses) when the error for both the controllers become almost the same. It is emphasized here that at these instants the tricopter is at the maximum height, which is outside the fans' influence, as can also be seen in Fig. 4.8a.

The velocity tracking performance for both the controllers is presented in Fig. 4.8d. Yet again, nothing much is deducted from the figure as both the plots are almost

overlapping throughout the trajectory. Next, the control outputs for the traditional FLC method and the SL-FLC framework are presented in Figs. 4.8e and 4.8f, respectively. In terms of controller gains for the SL-FLC framework, the position control gains are presented in Fig. 4.8g while the velocity control gains are shown in Fig. 4.8h. Although there is no change in the position control gains (same reason as Remark 23), the velocity control gains get updated throughout to realize an accurate tracking performance. In addition, the disturbance estimation performance by the SL algorithm, depicted in Fig. 4.8i, further facilitates a minimum-error trajectory tracking by the SL-FLC framework.

Remark 29. Although the wind gust disturbance is introduced along $-x$- and $+y$-directions, some of its influence is also experienced along z-direction (due to a non-stationary trajectory). As a result, a non-zero disturbance value is estimated along z-direction in Fig. 4.8i.

Finally, in order to validate the claim that SL algorithm helps to improve the tracking performance in presence of wind gust disturbance which is primarily stochastic in nature, some statistical results are also obtained. In essence, the tracking performance of each controller is tested for three different levels of fan speeds, whereby each individual experiment is repeated ten times. From every single test, the mean Euclidean error values for both the controllers are recorded and later accumulated to obtain a box plot which is shown in Fig. 4.9. As visualized, the mean Euclidean error values for both the controllers increase with the rising fan speed level. However, the error rise for the SL-FLC framework is negligible when compared to the traditional FLC method. This behavior is in accordance with the expectation, as the SL algorithm within the SL-FLC framework helps to compensate for the induced disturbance irrespective of its magnitude.

Remark 30. Note that the control gains in Fig. 4.8h get updated continuously even without much change in the disturbance value. The reason for this continuous update is a low-value selection for the dead-zone threshold (Remark 21), which is kept the same for all the experiments. Nevertheless, while the increase is less in comparison to the overall magnitude of the control gains, this continuous update can itself be avoided by selecting a higher value for the dead-zone threshold.

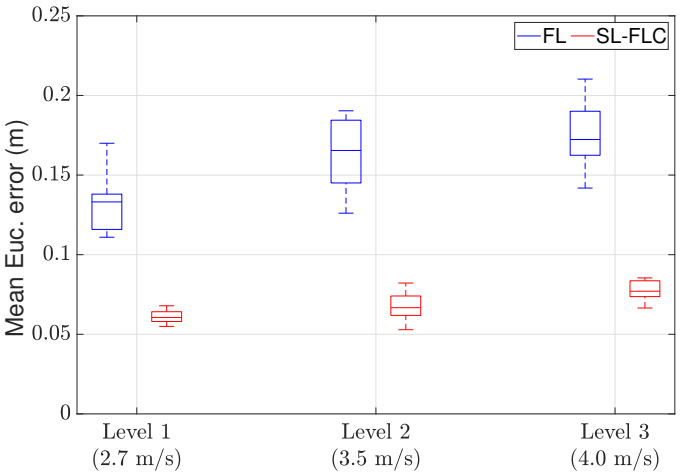


FIGURE 4.9: Mean Euclidean error for three wind speed levels, wherein ten tests for each speed level are performed.

4.6 Conclusions

A simple learning strategy for feedback linearization-based control of uncertain nonlinear systems has been proposed in this chapter. The efficacy of the proposed SL-FLC framework has been first validated via a simulation study where the control problem of a second-order dynamical system is considered. The obtained results have illustrated that the SL-FLC framework can ensure the desired closed-loop error dynamics in the presence of modeling uncertainties and external disturbance. Secondly, the performance of the SL-FLC framework has been experimentally validated for the position tracking of a 3D-printed tilt-rotor tricopter, wherein the disturbances in the form of mass variation, ground effect, and wind gust are explicitly induced. Thanks to the SL algorithm, the SL-FLC framework has resulted in z-directional tracking improvements of 80.05% and 42.27% for the case scenarios I and II, respectively, over the traditional FLC method. In addition, the tracking improvement of 57.68% in terms of the Euclidean error has also been achieved for the case scenario III. Furthermore, the additional statistical results presented for the case scenarios I and III have illustrated a consistent tracking improvement even with the increasing magnitude of disturbances.

Chapter 5

Iterative Learning-based Nonlinear Model Predictive Control for Precise Visualization of Geological Outcrops

This chapter presents the iterative learning control approach. As the name implies, ILC is based on the iterative learning paradigm that encompasses multiple system repetitions to identify/learn the unknown system model. Once the model is identified, it subsequently gets updated within the controller definition, thereby improving the tracking accuracy. Amongst numerous nonlinear regression approaches, for instance, neural networks, support vector regression (SVR), and Gaussian process, the GP-based regression is preferred in this chapter. The main reason for this selection is its ability to produce the distribution for the prediction in addition to the prediction value.

The proposed ILC approach is evaluated for the virtual outcrop modeling problem. In essence, a virtual outcrop model is a digital 3D representation of the outcrop surface, which plays a vital role in geoscience. While manual visualization is cheaper in comparison to airborne laser scanning (e.g. using helicopter-based lidars), there are numerous challenges. Firstly, the images are mostly taken in flights over a straight path due to which the camera focus has to be consistently adjusted, thereby resulting in distorted images. Secondly, a specific overlap value $(50 - 60\%)$ is required to be maintained in the offshore environment which is frequently windy, hence giving a hard-time even to a skilled pilot. Thirdly, the pilot

has to consistently steer (yaw) to always face the outcrop. What is more, since the outcrop generation process usually requires minutes to hours, depending on the size, hiring a skilled pilot itself gets costly.

Motivated by the aforementioned challenges, a fully autonomous visualization solution is proposed in this chapter. The first aspect of the solution is the navigation algorithm that generates a suitable mapping path while maintaining a specific distance to the outcrop. This circumvents the continuous focusing of the camera, thereby resulting in consistent-quality images. Additionally, utilizing the data from the two inclined distance sensors, the navigation algorithm commands the yaw angle such that the robot always faces the outcrop. The second aspect involves the precise tracking of the generated path in the presence of wind disturbances for which the NMPC is incorporated in conjunction with a GP-based regression technique. Although the performance of NMPC is expected to degrade due to the offshore working environment, thanks to the accurate learning of disturbance forces by the GP, the optimal performance of NMPC is restored. Moreover, this accurate learning ability is mainly credited to the long-short term memory feature of the designed GP model.

The outline of this chapter is as follows: Section 5.1 overviews other works based on the iterative learning concept. The customized quadrotor robot and the underlying model is described in Section 5.2. Then, Section 5.3 discusses the fundamental formulations for GP-based regression followed by the illustration of the incorporated ILC framework in Section 5.4. Thereafter, the simulation and real-world test results are elaborated in Section 5.5. Lastly, some conclusions are drawn in Section 5.6.

5.1 Literature Overview

Neural networks have been widely explored for modeling the nonlinear dynamics within numerous learning-based control applications. For instance, an adaptive NN-based controller design is proposed to achieve a reconfigurable flight control system in [129]. A deep NN-based quadrotor model has been demonstrated to synthesize control for trajectories not used while training in [130]. In [131], a hybrid artificial NN-assisted proportional-derivative controller has been demonstrated for

the learning control during different flight phases, namely, fast and agile flight, motor failure, and safe landing. Although the NN-based model can learn the system uncertainty, it does not provide any distribution for the prediction rather than just the prediction value in contrast to the GP-based regression.

Kernels-based SVR has also been demonstrated along with MPC for the lateral control of a fixed-wing aerial robot in [132]. While SVR is less affected by the unstructured data noise, obtaining the suitable kernel hyperparameters is challenging. Whereas, the kernel hyperparameters can be learned from the data through evidence maximization for the GP-based regression.

A recent GP-based iterative learning control implementation includes [133], wherein, a GP is used to model the uncertainties in a ground robot that is operating on uncertain terrain. However, unlike the disturbances considered in this thesis, the level of stochasticity involved is significantly low (if not zero), as the terrain topography remains the same over each successive trial. Furthermore, a GP is utilized to learn the full quadrotor model using a huge dataset in [134]. While learning the full model from the beginning would work in the case without uncertainty, the GP model will run into computational issues for the cases with uncertain dynamics. This is because the inversion of the kernel matrix has to be performed every time a new dataset is observed.

5.2 Problem Statement

To conduct the offshore visualization, an off-the-shelf quadrotor platform 'DJI Matrice 100' having an 'x-configuration' is utilized, as shown in Fig. 5.1. For outdoor localization, an RTK GPS system with 'SIRIUS RTK GNSS Base (M8P)' from Drotek as the base module and 'NEO-M8P RTK receiver' from CsgShop as the rover module, are incorporated. Additionally, a 'LIDAR-Lite v3' and a 'PX4flow' optical sensor are added to achieve precise position feedback. For visualization of the outcrop, three 'TeraRanger Evo 60m' distance sensors from Terabee are utilized, wherein one is mounted along the body x-axis while the other two are inclined by $\pm 10\,\text{deg}$ yaw angle on each side of the first. In essence, they perform two tasks: firstly, to get the feedback of wall distance and secondly, to compute the

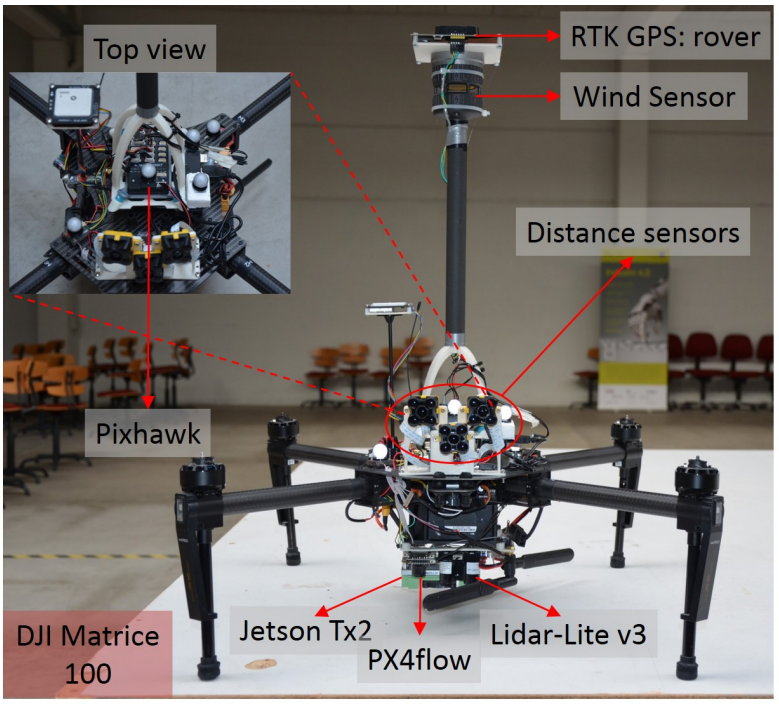


FIGURE 5.1: DJI Matrice 100 aerial robot with onboard electronics. To be able to command it in ROS environment, its flight controller is replaced with a Pixhawk flight controller that performs the low-level stabilization tasks.

desired yaw angle based on the difference in the value of two inclined sensors. Furthermore, a wind sensor 'FT205' from FT Technologies is mounted on the robot to record the realtime wind speed and directional data.[1] For executing the controller codes onboard, an Nvidia 'Jetson TX2' with 'Orbitty' carrier board is utilized in the system. Overall, the total takeoff mass (m) of the whole robotic platform is around 3.8 kg.

The incorporated discrete-time nonlinear model of the quadrotor at high-level, utilizing the previous definition in (2.33), is of the form:

$$\mathbf{x}_{k+1} = \underbrace{\mathbf{f}_{\mathrm{d}}(\mathbf{x}_k, \mathbf{u}_k)}_{\textit{a priori}\text{ model}} + \underbrace{\mathbf{g}_{\mathrm{d}}(\mathbf{x}_k, \mathbf{u}_k)}_{\textit{unknown}\text{ model}}, \quad \mathbf{z}_k = \mathbf{x}_k + \boldsymbol{\nu}, \tag{5.1}$$

where $\mathbf{x} \in \mathbb{R}^6 = [x, y, z, u, v, w]^T$, $\mathbf{u} \in \mathbb{R}^4 = [\phi, \theta, \psi, F_z]^T$, and $\mathbf{z} \in \mathbb{R}^6$ are the system states, inputs, and measured outputs, respectively. Due to the involved time-varying uncertainties, the overall model is again considered to be comprised

[1] It is to be noted that the wind sensor provides the ground truth data but it is not used in the controller design.

of *a priori*-known nominal function $\mathbf{f}_{\mathtt{d}}(\,\cdot\,,\,\cdot\,) : \mathbb{R}^6 \times \mathbb{R}^4 \mapsto \mathbb{R}^6$ and an *uncertain/unknown* function $\mathbf{g}_{\mathtt{d}}(\,\cdot\,,\,\cdot\,) : \mathbb{R}^6 \times \mathbb{R}^4 \mapsto \mathbb{R}^6$. While the nominal function $\mathbf{f}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k)$ is generally obtained using the first principle approach (as illustrated in Section 2.3), the function $\mathbf{g}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k)$ is essentially a disturbance model that represents the discrepancy between the known nominal function and actual/observed system behavior. Moreover, for the considered robotic application it is expressed as:

$$\mathbf{g}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k) = [0, 0, 0, F_k^{x_{dist}}, F_k^{y_{dist}}, F_k^{z_{dist}}]^T, \tag{5.2}$$

where $F_k^{x_{dist}}$, $F_k^{y_{dist}}$, and $F_k^{z_{dist}}$ are the respective disturbance forces along the three directions.

5.3 Gaussian Process

A Gaussian process is usually regarded as a distribution over functions. In other words, a given multivariate Gaussian distribution defined over a vector $\mathbf{g} \in \mathbb{R}^n$, for $\mathbf{g} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, transforms into a GP for an infinitely long and dense vector $\mathbf{g}$. Consequently, the vector $\mathbf{g}$ becomes a function $g(\,\cdot\,) : \mathbb{R} \mapsto \mathbb{R}$, and the corresponding GP describes a distribution over such function. More formally, a Gaussian process is defined as a collection of (possibly infinite) random variables, any finite number of which are jointly Gaussian distributed [135]. In general, GP has an associated continuous index variable $\mathbf{a}$, which is time for classical GPs. However, in this chapter, the index set is considered to be a $D-$dimensional vector space defined over $\mathbb{R}^D$.

In a nonparametric (scalar) regression procedure, the goal is to obtain an approximation for a (non)linear map, $g(\mathbf{a}) : \mathbb{R}^D \mapsto \mathbb{R}$, from an input (or index) vector $\mathbf{a}$ to the function value $g(\mathbf{a})$. One of the artifices is by assuming Gaussian distributions for the function values $g(\mathbf{a})$ and the corresponding index vector $\mathbf{a}$. Since this is also the basis for a Gaussian process, GP-based regression is often utilized for nonparametric regression.

While a multivariate Gaussian distribution $\mathbf{g} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is fully specified by the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$, a GP $g \sim \mathcal{GP}(m(\,\cdot\,), k(\,\cdot\,,\,\cdot\,))$ is also fully defined by a mean function $m(\,\cdot\,)$ and covariance function $k(\,\cdot\,,\,\cdot\,)$ that are

expressed as follows:

$$m(\mathbf{a}) \triangleq \mathbb{E}_g[g(\mathbf{a})], \tag{5.3}$$

$$k(\mathbf{a}, \mathbf{a}') \triangleq \mathrm{cov}_g[g(\mathbf{a}), g(\mathbf{a}')] = \mathbb{E}_g[(g(\mathbf{a}) - m(\mathbf{a}))(g(\mathbf{a}') - m(\mathbf{a}'))], \tag{5.4}$$

where $\mathbf{a}$ and $\mathbf{a}'$ are two arbitrary index points. Recall that within the GP definition, the mean function refers to the (average) function shape, whereas the covariance function signifies the covariance between the function values which is computed based on the correlation of the corresponding inputs in the index set.

5.3.1 Gaussian Process Prior

The regression process is about making an inference of the underlying function $g(\mathbf{a})$, which is observed with a zero-mean Gaussian noise ($\nu \sim \mathcal{N}(0, \sigma_\nu^2)$) as:

$$\widetilde{g}(\mathbf{a}) = g(\mathbf{a}) + \nu, \tag{5.5}$$

given a set of input vectors $\mathbf{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_N] \in \mathbb{R}^{D \times N}$, measured outputs $\widetilde{\mathbf{g}} = [\widetilde{g}(\mathbf{a}_1), \cdots, \widetilde{g}(\mathbf{a}_N)]^T \in \mathbb{R}^N$, and a GP model $g(\mathbf{a}) \sim \mathcal{GP}\big(m(\,\cdot\,), k(\,\cdot\,,\,\cdot\,)\big)$. Also, $\mathcal{D} = \{\mathbf{a}_i, \widetilde{g}(\mathbf{a}_i)\}_{i=1}^N$ is the combined dataset that is obtained for notational convenience. Utilizing the Bayesian setting, inference of $g(\,\cdot\,)$ can be formulated as:

$$p(g|\mathcal{D}, \boldsymbol{\theta}) = \frac{p(\widetilde{\mathbf{g}}|g, \mathbf{A}, \boldsymbol{\theta}) p(g|\boldsymbol{\theta})}{p(\widetilde{\mathbf{g}}|\mathbf{A}, \boldsymbol{\theta})} \tag{5.6}$$

where $p(g|\boldsymbol{\theta})$ is the GP prior and $p(g|\mathcal{D}, \boldsymbol{\theta})$ represents the GP posterior, whereas the term $\boldsymbol{\theta}$ is the hyperparameter vector for the underlying GP. Since no prior knowledge is generally available for the wind disturbance, a suitable GP prior is a zero-mean function: $m(\mathbf{a}) = 0$, along with a squared-exponential (SE) covariance function:

$$k(\mathbf{a}, \mathbf{a}') = \sigma_g^2 \exp\left[-\frac{1}{2}(\mathbf{a} - \mathbf{a}')^T \boldsymbol{\Lambda}^{-1} (\mathbf{a} - \mathbf{a}')\right], \tag{5.7}$$

where the term $\mathbf{\Lambda} = \text{diag}([\lambda_1, \cdots, \lambda_D])$, specifies the rate of change of $g(\mathbf{a})$ with respect to $\mathbf{a}$. Additionally, by incorporating noisy measurements, the above covariance definition changes to:

$$\widetilde{k}_{\mathbf{aa}'} = \text{cov}(\widetilde{g}(\mathbf{a}), \widetilde{g}(\mathbf{a}')) = k(\mathbf{a}, \mathbf{a}') + \delta_{\mathbf{aa}'}\sigma_\nu^2, \tag{5.8}$$

where $\delta_{\mathbf{aa}'}$ denotes the Kronecker delta symbol defined as:

$$\delta_{\mathbf{aa}'} = \begin{cases} 1, & \text{for } \mathbf{a} \text{ and } \mathbf{a}' \text{ to be the same points,} \\ 0, & \text{otherwise.} \end{cases}$$

As seen from the definitions in (5.7) and (5.8), the covariance function is parameterized by: length scales $(\lambda_1, \cdots, \lambda_D)$, process variance (σ_g^2), and measurement noise variance (σ_ν^2). They are collected together in a hyperparameter vector $\boldsymbol{\theta} = [\lambda_1, \cdots, \lambda_D, \sigma_g^2, \sigma_\nu^2]$ that is usually learned from the data by solving the following evidence maximization (or maximum log-likelihood) problem utilizing the gradient ascent methods [135]:

$$\max_{\theta} -\frac{1}{2}\widetilde{\mathbf{g}}^T\widetilde{\mathbf{K}}_{\mathbf{AA}}^{-1}(\boldsymbol{\theta})\widetilde{\mathbf{g}} - \frac{1}{2}\log|\widetilde{\mathbf{K}}_{\mathbf{AA}}(\boldsymbol{\theta})| - \frac{1}{2}D\log(2\pi), \tag{5.9}$$

where the covariance matrix $\widetilde{\mathbf{K}}_{\mathbf{AA}} = \mathbf{K}_{\mathbf{AA}} + \sigma_\nu^2\mathbf{I}_N \in \mathbb{R}^{N\times N}$, and its element are given as: $k(\mathbf{a}_i, \mathbf{a}_j) + \sigma_\nu^2 \; \forall \; i, j \in [1, N]$. It is to be noted that in the above equation, the matrix $\widetilde{\mathbf{K}}_{\mathbf{AA}}$ is explicitly shown to be a function of $\boldsymbol{\theta}$. This dependence is avoided hereon for notational simplicity.

5.3.2 Prediction: Deterministic Inputs

Once the GP prior is defined for the set of past observations $\mathcal{D}$, a distribution for $p(g(\mathbf{a}_*))$ (or the function value $g(\mathbf{a}_*)$) at a given test input $\mathbf{a}_* \in \mathbb{R}^D$, can be obtained utilizing the GP framework. To do that, first the joint probability distribution of the past measurements and the function value at the test point is obtained as:

$$\begin{bmatrix} \widetilde{\mathbf{g}} \\ g(\mathbf{a}*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}_{N+1}, \begin{matrix} \widetilde{\mathbf{K}}_{\mathbf{AA}} & \mathbf{k}_{\mathbf{a}_*\mathbf{A}}^T \\ \mathbf{k}_{\mathbf{a}_*\mathbf{A}} & k(\mathbf{a}_*, \mathbf{a}_*) \end{matrix}\right), \tag{5.10}$$

where $\mathbf{0}_{N+1} \in \mathbb{R}^{N+1}$ represents a zero vector and the vector $\mathbf{k}_{\mathbf{a}_*\mathbf{A}}$ that contains the covariances between the test input $\mathbf{a}_*$ and all the other observed data points in $\mathcal{D}$ is expressed as: $\mathbf{k}_{\mathbf{a}_*\mathbf{A}} = [k(\mathbf{a}_*, \mathbf{a}_1), \cdots, k(\mathbf{a}_*, \mathbf{a}_N)]$. Subsequently, by utilizing the conditioning property of the joint Gaussian distributions, the distribution for GP posterior at the given test input is: $p(g(\mathbf{a}_*)|\mathcal{D}, \boldsymbol{\theta}) \sim \mathcal{N}(m^+(\mathbf{a}_*), k^+(\mathbf{a}_*, \mathbf{a}_*))$, where the mean and covariance functions are expressed as follows (cf. [135]):

$$m^+(\mathbf{a}_*) = \mathbf{k}_{\mathbf{a}_*\mathbf{A}} \widetilde{\mathbf{K}}_{\mathbf{AA}}^{-1} \widetilde{\mathbf{g}}, \tag{5.11}$$

$$k^+(\mathbf{a}_*, \mathbf{a}_*) = k(\mathbf{a}_*, \mathbf{a}_*) - \mathbf{k}_{\mathbf{a}_*\mathbf{A}} \widetilde{\mathbf{K}}_{\mathbf{AA}}^{-1} \mathbf{k}_{\mathbf{a}_*\mathbf{A}}^T. \tag{5.12}$$

Remark 31. The covariance matrix $\widetilde{\mathbf{K}}_{\mathbf{AA}}$ in the above equation is of size $(N \times N)$. As a result, computing its inverse requires $\mathcal{O}(N^3)$ operations, which may lead to computational issues for systems with a large dataset. However, note that the inverse has to be computed only once for a given dataset to make predictions.

5.3.3 Prediction: Uncertain Inputs

The above procedure for obtaining the GP posterior is valid when the test input is deterministic; in other words, while predicting one-step ahead. However, when the multi-step predictions are required, the conventional way of recursively performing predictions using (5.11) and (5.12) results in a subgrade performance. This is mainly because the recursive multi-step ahead predictions neglect the additional uncertainty induced with each successive prediction, which is vital in time-series regression [136]. Nevertheless, this uncertainty propagation problem is dealt with by considering $\mathbf{a}_*$ to be a Gaussian random variable $\mathbf{a}_* \sim \mathcal{N}(\widetilde{\mathbf{a}}_*, \boldsymbol{\Sigma}_{\mathbf{a}_*})$, with a measured value $\widetilde{\mathbf{a}}_*$ and $\boldsymbol{\Sigma}_{\mathbf{a}_*}$ being the corresponding measurement noise variance. Then, to obtain the exact predictive distribution $p(g(\mathbf{a}_*)|\widetilde{\mathbf{a}}_*, \boldsymbol{\Sigma}_{\mathbf{a}_*})$, marginalization over $\mathbf{a}_*$ is performed as:

$$p(g(\mathbf{a}_*)|\widetilde{\mathbf{a}}_*, \boldsymbol{\Sigma}_{\mathbf{a}_*}) = \int p(g(\mathbf{a}_*)|\mathbf{a}_*) p(\mathbf{a}_*) d\mathbf{a}_*. \tag{5.13}$$

The above integral is analytically intractable while the resulting distribution is normally non-Gaussian. The easiest way out is to consider the distribution to be

Gaussian anyway and subsequently, utilize the exact moment matching technique[2] to obtain the underlying distribution. Finally, the mean $\widetilde{m}_*^+$ and variance $\widetilde{k}_{**}^+$ at an uncertain input for a zero-mean function and a squared-exponential covariance function can be obtained as (cf. [137]):

$$\widetilde{m}_*^+ = \overline{\mathbf{K}}_{*\mathbf{A}} \widetilde{\mathbf{K}}_{\mathbf{AA}}^{-1} \widetilde{\mathbf{g}}, \tag{5.14}$$

$$\widetilde{k}_{**}^+ = \sigma_g^2 - \sigma_g^4 \sqrt{\frac{|\mathbf{\Lambda}|}{|\mathbf{\Lambda} + 2\mathbf{\Sigma}_{\mathbf{a}_*}|}} \sum_{i=1}^{N} \sum_{j=1}^{N} \left\{ \mathbf{Q}_{\mathbf{a}_i \mathbf{a}_j} \exp\left(-\frac{1}{2} \big(\mathbf{a}_i - \mathbf{a}_j\big)^T (2\mathbf{\Lambda})^{-1} \big(\mathbf{a}_i - \mathbf{a}_j\big) \right) \right.$$
$$\left. \exp\left(-\frac{1}{2} \left(\frac{\mathbf{a}_i + \mathbf{a}_j}{2} - \widetilde{\mathbf{a}}_* \right)^T \left(\frac{1}{2}\mathbf{\Lambda} + \mathbf{\Sigma}_{\mathbf{a}_*} \right)^{-1} \left(\frac{\mathbf{a}_i + \mathbf{a}_j}{2} - \widetilde{\mathbf{a}}_* \right) \right) \right\} - (\widetilde{m}_*^+)^2, \tag{5.15}$$

where the respective elements of the vector $\overline{\mathbf{K}}_{*\mathbf{A}}$ and matrix $\mathbf{Q}$ are expressed as:

$$\overline{\mathbf{K}}_{*\mathbf{a}_i} = \sigma_g^2 \sqrt{\frac{|\mathbf{\Lambda}|}{|\mathbf{\Lambda} + \mathbf{\Sigma}_{\mathbf{a}_*}|}} \exp\left(-\frac{1}{2} (\widetilde{\mathbf{a}}_* - \mathbf{a}_i)^T (\mathbf{\Lambda} + \mathbf{\Sigma}_{\mathbf{a}_*})^{-1} (\widetilde{\mathbf{a}}_* - \mathbf{a}_i) \right), \tag{5.16}$$

$$\mathbf{Q}_{\mathbf{a}_i \mathbf{a}_j} = \widetilde{\mathbf{K}}_{\mathbf{a}_i \mathbf{a}_j}^{-1} - \left(\widetilde{\mathbf{K}}_{\mathbf{a}_i \mathbf{a}_j}^{-1} \widetilde{g}(\mathbf{a}_i) \right) \left(\widetilde{\mathbf{K}}_{\mathbf{a}_i \mathbf{a}_j}^{-1} \widetilde{g}(\mathbf{a}_i) \right)^T. \tag{5.17}$$

Remark 32. Note that the presented GP framework only considers the case of scalar function $g(\mathbf{a})$, although a vector function is required to approximate the unknown function $\mathbf{g}_\mathrm{d}(\mathbf{x}, \mathbf{u})$. However, in that case, the scalar covariance function $k(\mathbf{a}, \mathbf{a}') \in \mathbb{R}$ has to be replaced with a matrix covariance function $\mathbf{K}(\mathbf{a}, \mathbf{a}') \in \mathbb{R}^{6\times 6}$, which would result in a covariance matrix $\widetilde{\mathbf{K}}_{\mathbf{AA}}$ of size $(6N \times 6N)$. Since this matrix is too big for an efficient inversion, independent GPs are designed along each dimension of $\mathbf{g}_\mathrm{d}(\mathbf{x}, \mathbf{u})$ for computational tractability.

5.4 Iterative Learning-based NMPC

In this work, the aim is to find a GP-based approximation for the uncertain function $\mathbf{g}_\mathrm{d}(\mathbf{x}_k, \mathbf{u}_k)$ and the associated confidence intervals, utilizing the available measurement data. This information subsequently updates the model within the NMPC which is employed for trajectory tracking. As the estimate for $\mathbf{g}_\mathrm{d}(\mathbf{x}_k, \mathbf{u}_k)$ becomes

[2] The main idea behind moment matching is to approximate any complicated distribution with an easier (Gaussian in this case) distribution having the exact same moments. Mean – first moment, variance – second moment, skewness – third moment, and so on.

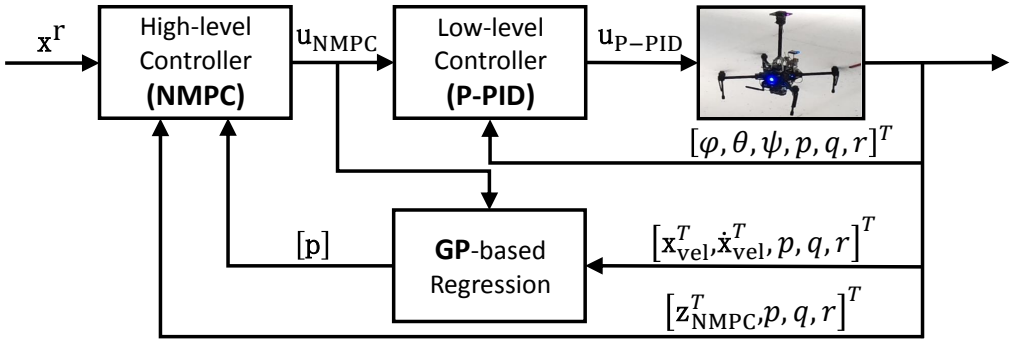


FIGURE 5.2: The proposed learning-based control framework, wherein the reference vector from the navigation algorithm is represented by $\mathbf{x}^{\text{r}} = [x^{\text{r}}, y^{\text{r}}, z^{\text{r}}, u^{\text{r}}, v^{\text{r}}, w^{\text{r}}, \psi^{\text{r}}]^T$, $\mathbf{u}_{\text{P-PID}} = [\Omega_1, \Omega_2, \Omega_3, \Omega_4]^T$ is the control output of Pixhawk, and $\mathbf{x}_{\text{vel}} = [u, v, w]^T$ is the linear velocity and p, q, r are the rotational rates of the aerial robot.

precise, the tracking performance of the controller improves significantly. Moreover, the overall control scheme is summarized in Fig. 5.2.

5.4.1 GP-based Disturbance Regression

In this chapter, three individual GPs ($\text{GP}^{x_{dist}}$, $\text{GP}^{y_{dist}}$, and $\text{GP}^{z_{dist}}$) are employed to learn the respective wind disturbance forces, namely, $F^{x_{dist}}$, $F^{y_{dist}}$, and $F^{z_{dist}}$. Note that all the three GPs are identically designed except for the features that are stated otherwise. Since the disturbance forces are primarily stochastic, a mixture model is adopted to achieve a *long-short term memory* feature. In essence, each GP model consists of two sub-GP models. The first sub-GP model (GP_1^{sub}) that represents the long-term memory is designed with the data collected for the first 15s. Whereas, the second sub-GP (GP_2^{sub}) is designed based on the data collected over a past window with 200 samples. Additionally, this window moves forward in time, granting the short-term memory feature to the GP_2^{sub}. Finally, the overall GP ($\text{GP}^{\text{merged}}$) is constructed by concatenating the two sub-GPs, as also shown in Fig. 5.3.

Recall that for training a GP, firstly, a suitable selection of input and output/target data is required, and subsequently, the optimized hyperparameters are obtained. In that vein, to approximate the function $\mathbf{g}_{\text{d}}(\mathbf{x}, \mathbf{u})$ that represents the mismatch between the nominal model and the measurements, the difference $\boldsymbol{\xi}_k = \mathbf{z}_k - \mathbf{f}_{\text{d}}(\mathbf{x}, \mathbf{u})$, is regarded as the appropriate input. Hence, the input to each GP is expressed as: $\mathbf{a}_k = [\boldsymbol{\xi}_{k-1}, \Delta\boldsymbol{\xi}_{k-1}, \cdots, \boldsymbol{\xi}_{k-5}, \Delta\boldsymbol{\xi}_{k-5}]^T$, where the term $\Delta\boldsymbol{\xi}$ represents the rate

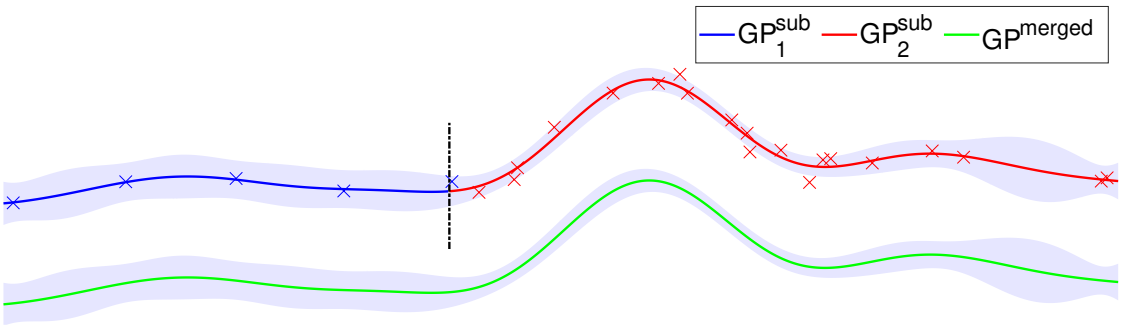


FIGURE 5.3: The proposed GP model with the LSTM feature to estimate the disturbance forces. It comprises two sub-GPs (GP_1^{sub} and GP_2^{sub}) that are concatenated to result in the final GP model (GP^{merged}).

of change of the difference $\boldsymbol{\xi}$. Moreover, it is to be noted that $\boldsymbol{\xi}$ signifies the difference along the respective direction for the individual GP. On the other hand, the difference at the current time instant $\boldsymbol{\xi}_k$ is selected to be the target value for the underlying GP. Furthermore, utilizing the conjugate gradient method, the hyper-parameters are optimized over the data obtained in GP_1^{sub}, wherein 200 gradient updates are performed.

In terms of prediction, the mean and variance of each GP are obtained over a window $[k, k+N_c+1]$, where k is the current time instant and N_c is the prediction horizon for NMPC. While the formulations in (5.11) and (5.14) are used to compute the mean at instant k and for the rest of the window, respectively, the variance for the entire window is computed utilizing the expression (5.12). This is primarily due to the computational issues with the other expression in (5.15). Furthermore, the external disturbance forces at the instant k are selected to be the respective mean values at that instant from the corresponding GP.

5.4.2 High-level NMPC Design

The high-level NMPC is designed to track the trajectory generated by the navigation algorithm. In contract to the definition in (3.1), the OCP is modified to cater to the disturbance estimates such that the optimal control outputs from NMPC

minimizes their overall magnitudes. Hence, the modified OCP is expressed as:

$$\min_{\mathbf{x}_k,\mathbf{u}_k} \frac{1}{2}\left\{\sum_{k=j}^{j+N_c-1}\left(\|\mathbf{x}_k-\mathbf{x}_k^{\mathrm{r}}\|^2_{\mathbf{W_x}}+\|\mathbf{u}_k-\mathbf{u}_k^{\mathrm{r}}\|^2_{\mathbf{W_u}}+\|\mathbf{p}_k\|^2_{\mathbf{W_p}}\right)+\left\|\mathbf{x}_{N_c}-\mathbf{x}_{N_c}^{\mathrm{r}}\right\|^2_{\mathbf{W}_{N_c}}\right\} \tag{5.18a}$$

$$\text{s.t. } \mathbf{x}_j=\hat{\mathbf{x}}_j, \tag{5.18b}$$

$$\mathbf{x}_{k+1}=\mathbf{f_d}(\mathbf{x}_k,\mathbf{u}_k)+\mathbf{g_d}(\mathbf{x}_k,\mathbf{u}_k), \qquad k=j,\cdots,j+N_c-1, \tag{5.18c}$$

$$\mathbf{x}_{k,\min}\le\mathbf{x}_k\le\mathbf{x}_{k,\max}, \qquad k=j,\cdots,j+N_c, \tag{5.18d}$$

$$\mathbf{u}_{k,\min}\le\mathbf{u}_k\le\mathbf{u}_{k,\max}, \qquad k=j,\cdots,j+N_c-1, \tag{5.18e}$$

where the additional term $\mathbf{p}_k\in\mathbb{R}^3$ incorporates the disturbance estimates and $\mathbf{W_p}\in\mathbb{R}^{3\times3}$ is the corresponding weight matrix.

The overall NMPC design is similar to the one utilized in Section 3.4.1. In summary, the state, control and measurement vectors are comprised of:

$$\mathbf{x}_{\mathrm{NMPC}}=[x,y,z,u,v,w]^T, \quad \mathbf{u}_{\mathrm{NMPC}}=[\phi,\theta,\psi,F_z]^T,$$
$$\mathbf{z}_{\mathrm{NMPC}}=\mathbf{x}_{\mathrm{NMPC}},$$

while the parameter vector $\mathbf{p}$ is expressed as:

$$\mathbf{p}_k=[F_k^{x_{dist}},F_k^{y_{dist}},F_k^{z_{dist}}]^T.$$

The following state and control trajectories, obtained by the trial-and-error method, are given as references for the optimization problem:

$$\mathbf{x}^{\mathrm{r}}=\mathbf{x}_{N_c}^{\mathrm{r}}=[x^{\mathrm{r}},y^{\mathrm{r}},z^{\mathrm{r}},u^{\mathrm{r}},v^{\mathrm{r}},w^{\mathrm{r}}]^T, \tag{5.19}$$

$$\mathbf{u}^{\mathrm{r}}=\left[sin^{-1}\left(\frac{1.7F_0^{y_{dist}}}{F_z}\right),sin^{-1}\left(\frac{-1.7F_0^{x_{dist}}}{F_z}\right),\psi^{\mathrm{r}},1.2mg-3.65F_0^{z_{dist}}\right]^T, \tag{5.20}$$

where ψ^{r} is the reference yaw commanded by the navigation algorithm. Note that the subscript 0 within the disturbance variables definition signify their values at the beginning of the prediction horizon. Additionally, the following constraints are defined in the optimization problem:

$$0.3mg\ (\mathrm{N})\le F_z\le 2mg\ (\mathrm{N}), \quad -40\ (^\circ)\le\phi,\theta\le 40\ (^\circ). \tag{5.21}$$

Moreover, the selected weight matrices for the optimization are:

$$\mathbf{W_x} = \text{diag}(25, 25, 30, 1.5, 1.5, 1.3), \qquad \mathbf{W_u} = \text{diag}(35, 35, 80, 0.015),$$
$$\mathbf{W_p} = \text{diag}(1/k_0^{+,x_{dist}}, 1/k_0^{+,y_{dist}}, 1/k_0^{+,z_{dist}}), \quad \mathbf{W}_{N_c} = 1.5\mathbf{W_x},$$

wherein the matrices $\mathbf{W_x}$ and $\mathbf{W_u}$ are obtained utilizing the algorithm proposed in Authors' previous work [138]. Whereas, the matrix $\mathbf{W_p}$ is selected to be the inverse of the obtained variance values at the beginning of the prediction horizon window. Furthermore, the prediction horizon $N_c = 30$ is selected for the computational tractability of the control framework.

Remark 33. Note that there is a version of MPC, known as the stochastic MPC, that can cater to uncertainties within the problem formulation. In essence, it considers a probabilistic description of the uncertainties and also brings in the probability of constraints violation via the chance constraints. Although the stochastic MPC facilitates limited constraint handling for uncertainties, there lies a major problem of ensuring recursive feasibility. This is mainly because of a finite constraint violation probability which contradicts the assumption that the optimal control sequence will remain feasible in the next sampling time. On the contrary, the approach presented in this chapter is a sample-based method, wherein the uncertain distribution is approximated before-hand such that the resulting optimization problem becomes deterministic.

5.5 Case Study: Visualization of Chalk Quarry

In this chapter, the outcrop to be visualized is the Rødal chalk quarry in Aalborg, Denmark. For scanning the chalk wall, the navigation algorithm generates a trajectory with a flight speed of 0.5 m/s while maintaining 4.5m distance from it. For thoroughly investigating the performance of the GP-based regression in learning the wind disturbances, the testing is first performed in a Gazebo simulation environment followed by real-world testing. Moreover, the tracking performance of NMPC for both, with and without learning cases are presented.

5.5.1 Simulation Testing

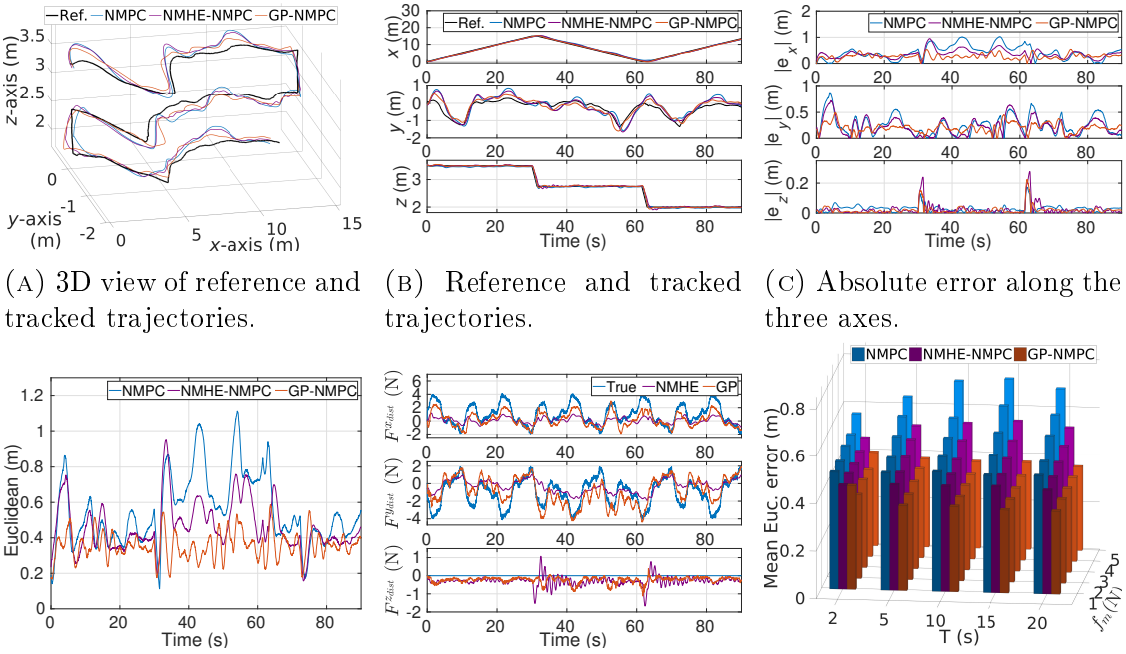
The motivation to test in a Gazebo environment is to create various wind scenarios that cannot be realized on the same day in the real-world. In that vein, the following function is used to generate the wind disturbance that satisfactorily replicates the real wind ($s : sin$, $c : cos$):

$$\mathbf{F}(t) = \boldsymbol{\delta}\left\{ f_m\left(0.5s^4\left(\frac{4\pi t}{\mathtt{T}}\right) + c^3\left(\frac{\pi t}{\mathtt{T}}\right) + s^2\left(\frac{2\pi t}{\mathtt{T}}\right) + s\left(\frac{2\pi t}{\mathtt{T}}\right)\right) + \sigma_\omega^2\right\}, \quad (5.22)$$

where $\mathbf{F}(t) \in \mathbb{R}^3$ represents the generated forces along x, y, and z directions, $\boldsymbol{\delta} \in \mathbb{R}^3$ is a directional vector that specifies the respective force components along the three directions, f_m is the mean force value, $\mathtt{T}$ is the time-period, and σ_ω represents the standard deviation of the added white Gaussian noise ($\omega \sim \mathcal{N}(0, \sigma_\omega^2)$).

The tracking results for the generated wind disturbance are presented in Fig. 5.4, wherein the utilized values are: $\boldsymbol{\delta} = [0.5, 0.5, 0]^T$, $f_m = 3\text{N}$, $\mathtt{T} = 10\text{s}$, $\sigma_\omega = 0.3$. These values are selected based on the wind that is frequently encountered in Aalborg. Below, the tracking performance of NMPC incorporating the GP-based regression (GP-NMPC framework) is compared with the conventional NMPC (without learning). Besides, to evaluate the learning performance of GP-based regression over an estimator, the tracking results are also obtained with the NMHE that is employed in a similar way with NMPC. While most of the NMHE design is the same as utilized in Section 3.4.1, the fourth term of the control reference trajectory vector from (5.19) is changed to $\mathbf{u}^{\mathrm{r}}(4) = 1.2mg - 2F_0^{z_{dist}}$, for NMHE-NMPC framework. Moreover, since the designed NMHE can only predict the disturbance value at the next time instant, the same value is replicated over the whole prediction horizon, as required by the NMPC's optimization problem. Lastly, the term $\mathbf{p}_k$ is not considered in the problem formulation (5.18a) within the NMHE-NMPC framework.

The overall commanded trajectory along with the tracking performance of all the controllers can be seen in Figs. 5.4a and 5.4b. For a quantitative comparison, the absolute errors and the Euclidean errors from each controller are presented in Figs. 5.4c and 5.4d, respectively. In terms of the respective absolute errors along x, y, and z directions, the convectional NMPC results in 42.62 cm, 25.63 cm, 2.93 cm, the NMHE-NMPC framework results in 36.29 cm, 22.36 cm, 2.39



(A) 3D view of reference and tracked trajectories.

(B) Reference and tracked trajectories.

(C) Absolute error along the three axes.

(D) Euclidean error incurred by all the controllers.

(E) True and estimated wind disturbance forces in body frame.

(F) Bar plot of mean Euclidean error for varying mean and time-period.

FIGURE 5.4: Simulation tracking results in the presence of wind disturbance generated using (5.22). The GP-NMPC framework results in the minimum tracking error in comparison to all the other controllers. This is primarily due to the superior learning ability of the designed GP model that incorporates the LSTM feature.

cm, whereas the GP-NMPC framework results in the lowest error values of 27.48 cm, 20.0 cm, 1.97 cm. On the other hand, the mean values of the Euclidean errors incurred by conventional NMPC, NMHE-NMPC, and GP-NMPC are 55.29 cm, 46.31 cm, and 35.63 cm, respectively. This implies an improvement of 35.6% over the conventional NMPC and 23% over the NMHE-NMPC framework. While the tracking improvement over conventional NMPC is intuitive (since there is no learning involve), the significant improvement over the NMHE-NMPC framework is primarily due to the LSTM feature that gives substantial information to GP for predicting future disturbance values. Recall that, although NMHE uses past data over a window for estimating the disturbance value, the information captured is still not enough to predict well the disturbances that involve a substantial level of stochasticity. Furthermore, one may notice a sudden rise in the Euclidean error between $30-65$s in Fig. 5.4d. It is to be noted that during that duration, the aerial robot is flying against the wind which makes it harder for conventional NMPC and NMHE-NMPC framework to track the commanded trajectory. Whereas for

the GP-NMPC framework, the Euclidean error increases at first but it decreases quickly thereafter, once the GP can learn the rise in the disturbance values.

The learned disturbances from NMHE and GP-based regression together with the true values are shown in Fig. 5.4e. As can be seen, the learned disturbance values from GP are significantly more accurate compared to NMHE. This improved prediction can again be credited to the LSTM feature of the designed GP. Another point to take note in Fig. 5.4e is that the estimation from NMHE shows more noise reduction capability in comparison to GP. However, this issue can be resolved by conservatively selecting the hyperparameter related to the measurement noise (σ_ν^2).

Now, in order to evaluate the robustness of the GP-NMPC framework to various levels of wind disturbances, some statistical analysis is performed, as shown in Fig. 5.4f; wherein, the mean and time-period are varied between $1-5$N and $2-20$s, respectively, while $\sigma_\omega = 0.3$ is maintained. It is to be noted that these values are selected based on the wind data over the recent past in Aalborg [139]. The first point to take note in Fig. 5.4f is that for conventional NMPC and NMHE-NMPC framework, the Euclidean error values increase with the disturbance mean, while in the case of GP-NMPC framework, the error rise is negligible (less than 2 cm except for $\mathtt{T} = 2$s); besides, the overall error value itself being substantially less. This signifies the robustness of GP-based regression to the increasing disturbance level to which NMHE significantly gets affected (error is around $7-8$ cm on average). Secondly, it can be seen that the error for conventional NMPC and NMHE-NMPC framework rises with the time-period with their maximum values at $T = 15$s, whereas, in the case of the GP-NMPC framework, maximum error is incurred at $T = 2$s. While the former is occurring due to the disturbance getting more time to act on the robot, in the latter case, GP does not get enough time to react to the abruptly varying disturbance thus results in a higher error value. Lastly, it can be visualized that the magnitude of average Euclidean error decreases for all the controllers at $T = 20$s. The plausible reason for this behavior could be that from this time-period on, the disturbance change is slow enough to which the controllers have sufficient time to respond.

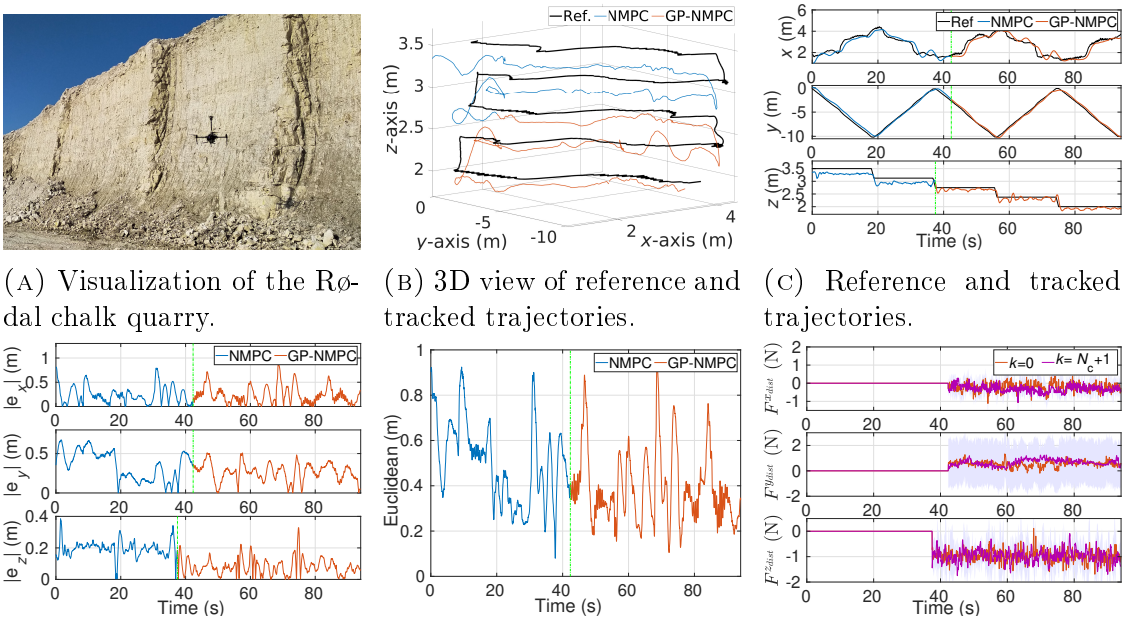
5.5.2 Real-world Testing

This section presents the tracking results from the real-world test[3] that is conducted in the Rødal chalk quarry, shown in Fig. 5.5a. During the experiment, the controller codes are executed onboard TX2 with a sampling frequency of 50-Hz, while the GP codes are running on a laptop having an Intel i7 processor and 16 GHz RAM, with approximately 17-Hz sampling frequency. Moreover, all the codes are communicating with each other over ROS topics. It is to be noted that in the real-world experiment, the aerial robot flies with conventional NMPC for the first 42s, wherein it collects the data for GP-based regression, and subsequently switches to the GP-NMPC framework. This is done to achieve a similar wind condition that would be difficult to ensure in case of sequential tests.

The overall commanded trajectory, as well as the tracking performance of conventional NMPC and GP-NMPC framework, can be visualized in Figs. 5.5b and 5.5c, wherein, 'blue' and 'red' colors represent the trajectory tracked by conventional NMPC and GP-NMPC framework, respectively. For the quantitative comparison, the absolute errors and Euclidean errors are presented in Figs. 5.5d and 5.5e, respectively. While the respective absolute errors along x, y, and z directions by conventional NMPC are 24.07 cm, 35.45 cm, 19.53 cm, in case of the GP-NMPC framework, the error values are reduced to 23.75 cm, 26.38 cm, 8.09 cm. While a substantial tracking improvement is achieved along y and z directions, the tracking improvement along x-direction may seem negligible. However, this is mainly due to the rough wall surface that resulted in sudden movements and partially due to the wind direction. On the other hand, the mean Euclidean error values incurred by conventional NMPC and GP-NMPC framework are 50.24 cm and 39.10 cm, respectively. Consequently, the GP-based regression of the wind disturbances resulted in a tracking improvement of 22.2% over the conventional NMPC.

Finally, the learned disturbance values by GP-based regression at $k = 0$ (beginning of the window) and $k = N_c + 1$ (end of the window) are presented together in Fig. 5.5f. Besides, the 95% confidence interval (at $k = 0$) is also presented to evaluate the predictions. As can be seen, the predicted values are significantly noisy, resulting in the wider confidence intervals (especially for $F^{y_{dist}}$), even though some pattern is visible for $F^{y_{dist}}$ and $F^{z_{dist}}$. The main reason for obtaining such

[3]The day of the experiment was less windy, with an average speed of around $2-3$ m/s, in comparison to the usual days in that chalk quarry.



(A) Visualization of the Rødal chalk quarry.

(B) 3D view of reference and tracked trajectories.

(C) Reference and tracked trajectories.

(D) Absolute error along the three axes.

(E) Euclidean error incurred by all the controllers.

(F) Estimated disturbance forces in body frame with 95% confidence intervals.

FIGURE 5.5: Real-world tracking results in the Rødal chalk quarry. It is illustrated that the overall tracking performance gets satisfactorily improved once the GP-NMPC framework takes over the control.

noisy estimates is essentially the noisy feedback data that is provided to the GPs for regression. A possible way to correct it by collecting more data for $\mathrm{GP}^{\mathrm{sub}_1}$ and subsequently, performing higher gradient updates for hyperparameter optimization. Another reason for not obtaining a wind pattern within the predicted values could be the low magnitude of wind disturbance in the first place. As already seen in Fig. 5.4f, the regression performance from GP gets degraded for low disturbance magnitudes (case with $f_m = 1\mathrm{N}$ and $\mathtt{T} = 2\mathrm{s}$). Since the GP is trained with the data that is based on the difference of measurements and the nominal model, for the case with a lower difference but noisy measurements, the learning may get dominated by the noise itself. Nevertheless, better GP training along with the selection of a more windy day would result in a significantly improved tracking performance by the GP-NMPC framework over the conventional NMPC, as also validated by the simulation results.

5.6 Conclusions

In this chapter, the iterative learning control framework has been utilized to tackle the challenges associated with manual visualization via aerial robots for generating virtual outcrops in offshore environments. In essence, the proposed GP-NMPC framework employs a GP-based regression algorithm to learn the wind disturbance forces which are later utilized to update the model of a position tracking NMPC. Both the simulation and the real-world testings have validated a precise tracking performance by the GP-NMPC framework over its conventional counterpart. Moreover, the simulation tests have also manifested the superior learning ability of the GP-based regression over the NMHE.

Chapter 6

Automated Tuning of Nonlinear Model Predictive Controller

The model predictive controller has shown remarkable success for the control and path planning of numerous robotic systems. However, the non-trivial weight tuning process is a major concern for its safe realtime implementation, especially over aerial robots. In essence, an MPC design process involves formulating a cost function for the application-related optimization problem, followed by a suitable selection of some weighting parameters. These weighting parameters, commonly known as MPC weights, reflect the relative importance of each element in the underlying optimization problem. In most cases, users prefer the trial-and-error method to obtain these weighting parameters either on a real robot or in simulation. Whereas the former might be dangerous, the latter requires an accurate system model.

To tackle the aforementioned challenges, this chapter presents an active exploration-based tuning methodology. The proposed auto-tuning mechanism significantly reduces the time and effort for MPC implementation and hence, can be fairly useful for unskilled MPC users. Previously, the authors' have developed and tested another automated (N)MPC tuning framework in [138]. However, that method utilizes a Gazebo model (system model from the first principle) of the robotic platform of interest for training, which can be difficult to create for novice users. Hence, the proposed deep neural network (DNN) model in this chapter replaces the former Gazebo model. Accordingly, the weight sets that are tuned over a high fidelity DNN model circumvents dangerous trials over real robots; while being real

flight-worthy at the same time, which facilitates their direct deployment over the real robots. Furthermore, to cater to several operational uncertainties – decreasing battery voltage, communication delays – that may not be captured within the DNN model, fine-tuning of the weight sets is also performed over the real robot. This essentially demonstrates the real flight tuning feasibility of the proposed algorithm. Lastly, a comparison with the manual tuning procedure through a user-based study is performed. During the tuning process, users implicitly apply various strategies. Naively, they start by recognizing the dominating parameters and their effect on the performance, followed by the appropriate weight set selection. In essence, they optimize performance by exploring the selection space in a Bayesian way. It is emphasized that unlike the existing tuning guidelines (discussed next) in the literature, this chapter provides a complete tuning framework that begins with the robot modeling and finally, results in the real flight-worthy weight sets.

The outline of this chapter is as follows: Section 6.1 overviews other MPC tuning methods from the literature. The utilized robotic platform is introduced in Section 6.2 followed by its NMPC problem formulation in Section 6.3. Then, Section 6.4 illustrates the proposed tuning approach in detail. The implementation results for the proposed tuning methodology are discussed in Section 6.5, while the tracking results of the explored MPC weight sets in real flights are exhibited in Section 6.6. Lastly, some conclusions are drawn in Section 6.7.

6.1 Literature Overview

Several systematic auto-tuning guidelines are proposed for the MPC weight tuning in the literature. In [140], some static tuning rules are obtained by first identifying the dominating tunable parameters and later analyzing their influence on the closed-loop MPC behavior. In [141], a simplified tuning expression is obtained for unconstrained linear MPC. In [142], a controller matching technique is utilized for selecting MPC weights. However, these approaches are restricted to linear systems. In [143], an objective function is proposed to tune a continuously linearized MPC for wind turbines. While linearized models are sufficient to describe the dynamics within wind turbines, they may result in suboptimal performance for robotic systems, especially aerial robots, due to their cross-coupled, nonlinear dynamics. In [144], nonlinear process models are utilized for an offline, optimization-based

tuning procedure for MPC. Nevertheless, this method requires a precise nonlinear model of the system. Besides, an online, adaptive tuning strategy is proposed in [145], wherein analytical expressions for closed-loop response sensitivity to the tunable MPC parameters are obtained. Yet, this methodology may not be feasible for many robotic systems because of their complex, nonlinear dynamics.

Machine learning techniques are also incorporated for MPC tuning. As an extension to the work in [145], a fuzzy logic-based online tuning method for NMPC is proposed in [146]. The restricted gradient computation, which is required to determine the time propagation of the NMPC tuning parameters, is substituted by fuzzy logic. Although fuzzy logic facilitates its implementation for nonlinear systems, it requires notable experience with the underlying system. Another machine learning technique for MPC tuning is reinforcement learning (RL), whose two recent implementations are presented in [147] and [148]. The RL technique is utilized to tune NMPC for position tracking in the former and collision-free trajectory planning in the latter. Although both approaches show satisfactory performance in simulation, their real-world applicability is not yet demonstrated. Besides, there also exist works in the literature that demonstrate RL-based (online) tuning of PID controllers [149–152]. However, in most of these implementations, the tuning starts from some precomputed gains via the Ziegler-Nichols (Z-N) strategy. Since the Z-N–like strategy is not available for MPC, the proposed active-exploration approach obtains the flight-worthy MPC weight sets from scratch.

6.2 Quadrotor Aerial Robot

In this chapter, a microscale quadrotor platform with 'x-configuration' is utilized, having arm lengths $l_1 = 0.073$m and $l_2 = 0.098$m, as displayed in Fig. 2.3. The overall frame comprises of off-the-shelf carbon fiber arms that are assembled in-house. A Pixhawk flight controller is utilized for the low-level stabilization control, whereas an Odroid XU4 computer executes all the control codes onboard. The total takeoff mass (m) including all the onboard electronics is equal to 0.89 kg.

The incorporated discrete-time nonlinear model of the quadrotor at high-level can be written as:

$$\mathbf{x}_{k+1} = \mathbf{f}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k), \quad \mathbf{z}_k = \mathbf{x}_k + \boldsymbol{\nu}_k, \tag{6.1}$$

where $\mathbf{x} \in \mathbb{R}^6$, $\mathbf{u} \in \mathbb{R}^4$, and $\mathbf{z} \in \mathbb{R}^6$ are the corresponding state, control, and measurement vectors, as defined before in Section 5.2. Note that the above equation is the modified version of (2.33), wherein the model uncertainties and dependence on model parameters are neglected. Furthermore, the nominal function is $\mathbf{f}_{\mathtt{d}}(\cdot\,,\cdot) : \mathbb{R}^6 \times \mathbb{R}^4 \mapsto \mathbb{R}^6$.

6.3 Position Tracking Nonlinear Model Predictive Controller

For the considered quadrotor model, the OCP formulation of high-level NMPC is the same as utilized in (3.1) except that the following model equation replaces (3.1b).

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{f}_{\mathtt{d}}(\mathbf{x}_k, \mathbf{u}_k), \qquad k = j, \cdots, j + N_c - 1. \tag{6.2}$$

Moreover, it is to be noted that the high-level model in the above equation is obtained using the first principle approach.

In summary, the state, control, and measurement vectors are composed of:

$$\mathbf{x}_{\text{NMPC}} = [x, y, z, u, v, w]^T, \quad \mathbf{u}_{\text{NMPC}} = [\phi, \theta, \psi, F_z]^T,$$
$$\mathbf{z}_{\text{NMPC}} = \mathbf{x}_{\text{NMPC}},$$

while the following state and control trajectories, obtained by the trial-and-error method, are given as the corresponding references:

$$\mathbf{x}^{\text{r}} = \mathbf{x}^{\text{r}}_{N_c} = [x^{\text{r}}, y^{\text{r}}, z^{\text{r}}, u^{\text{r}}, v^{\text{r}}, w^{\text{r}}]^T, \quad \mathbf{u}^{\text{r}} = [0, 0, 0, 1.2mg]^T.$$

The control outputs from NMPC are passed to the Pixhawk with the control vector $\mathbf{u}_{\text{P-PID}} = [\Omega_1, \Omega_2, \Omega_3, \Omega_4]^T$. The overall control scheme is summarized in a block diagram shown in Fig. 6.2. Additionally, the following constraints are defined in

the optimization problem:

$$0.5mg \text{ (N)} \leq F_z \leq 1.8mg \text{ (N)}, \quad -35 \text{ (°)} \leq \phi, \theta \leq 35 \text{ (°)}. \tag{6.3}$$

Moreover, within the optimization problem of NMPC, the diagonal elements of state and control weighting matrices represented as $\mathbf{W_x}$ and $\mathbf{W_u}$, respectively, are tuned utilizing the proposed framework. In essence, these weighting matrices penalize the deviations of predicted state and control trajectories from their specified references. Also, the terminal weight matrix is selected as: $\mathbf{W}_{N_c} = 1.3 \times \mathbf{W_x}$ along with the prediction window length of $N_c = 30$ for stability reasons. Note that while the proposed tuning algorithm can be utilized to obtain the terminal weight matrix, it is preselected for simplicity.

6.4 Proposed Auto-Tuning Approach

The proposed approach brings together concepts from reinforcement learning, conventional trial-and-error–based MPC tuning, and DNN-based system modeling. The presented methodology can be viewed as an advanced version of a traditional trial-and-error–based tuning process. It enhances the baseline trial-and-error method in two key aspects: eliminating the need for numerous trials on a real robot by utilizing a DNN model of it, and expediting the tuning process by benefiting from the active exploration paradigm which is inspired from RL. One may recall the reason to adopt DNN-based robot modeling over the model-based approaches. Being a data-based approach, the former does not require the non-trivial first principle model which is difficult to obtain for novice users.

In summary, RL is a nature-inspired, experience-based, learning method having three main elements: state, action, and reward. Each trial begins with the agent making a selection among possible actions while observing the state of the system. Thereafter, a reward is given as a consequence of that action, which is utilized in deciding the next trial. Eventually, the goal of the agent is to find those actions that maximize the overall reward [153].

6.4.1 DNN-based System Modeling

Creating the overall tuning framework starts with obtaining a DNN model of the robot of interest. In that vein, the flight data is collected from the real aerial robot at 50Hz. Recall that the aerial robot needs a stable position controller to follow a reference trajectory. Since a test NMPC (pre-trained via the trial-and-error method) is already available for the quadrotor, it has been utilized for collecting the flight data. However, one can use any position controller including standard PIDs that are already implemented in Pixhawk flight controller. Within the collected data, the twelve states of the quadrotor – position, linear velocity, attitude, angular velocity over three axes – and 4 control inputs of the high-level controller – roll angle, pitch angle, yaw angle, thrust – for a finite duration in the past (0.8s) are regarded as inputs, whereas the resultant states of the quadrotor at the current time instant are regarded as outputs. This data representing input-output relations of the quadrotor is fed into a DNN whose architecture is depicted in Fig. 6.1.

The utilized DNN comprises of five main lanes for five inputs. 'Position', 'linear velocity', 'attitude', and 'angular velocity' are taken in the first four lanes separately and fed through two consecutive, fully connected layers of size 16 on each lane. Then, the outputs of these lanes are combined and fed through two consecutive layers of size 32. Meanwhile, 'controls' is fed through two consecutive layers of size 32 in the fifth lane. Then, the output of the combination of the first four lanes and the output of the fifth lane are combined in a layer of size 32. This layer is fully connected to the last layer of size 12 which yields the twelve states of the quadrotor. All layers except the last layer apply rectified linear unit. Finally, this network is trained using the Adam optimizer [154] in PyTorch with default settings through $510,000$ data samples over 6500 epochs. Since the data used for training is obtained from the real robot over a variety of trajectories resulting in persistent excitation, the DNN represents the real robot's dynamics fairly well. In this way, it is assured that all the MPC weight sets obtained over the DNN-based model can be directly tested on the real robot, hence are real flight-worthy.

Remark 34. It is to be noted that other networks could also be utilized for robot modeling as the network architecture does not play a vital role within the proposed algorithm. Besides, one can also adopt other data-based modeling approaches such as GP regression. Essentially, the choice depends on the ease of modeling and

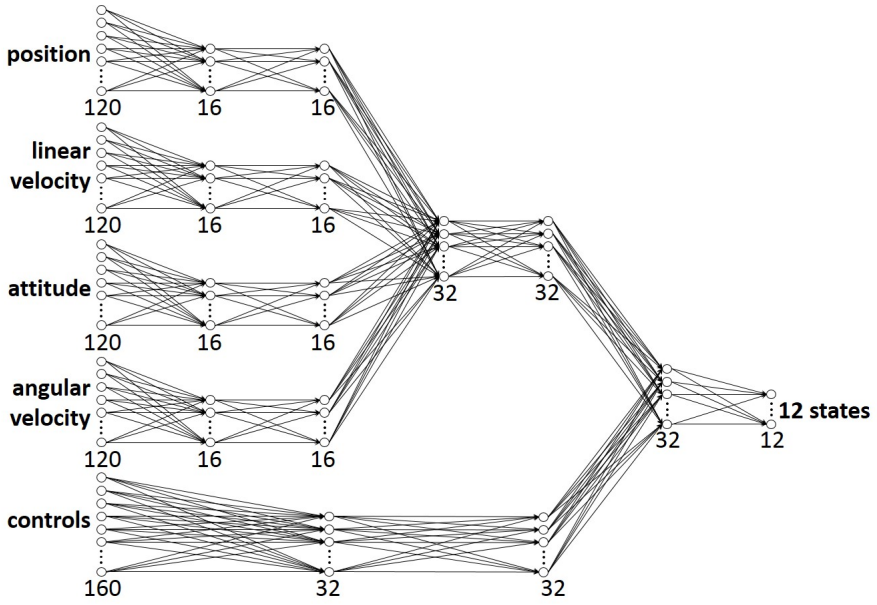


FIGURE 6.1: Utilized DNN for modeling the quadrotor aerial robot.

the available training resources, including the amount of data and computational power.

6.4.2 Active Exploration of Weight Sets

The obtained DNN model can also be used for conventional trial-and-error–based tuning. However, it might take a considerable amount of time and effort from the users to try different weight sets, observe their performance on the model, and tweak them till finding decent weight sets. Thus, this process is automated by benefiting from the active exploration paradigm which is inspired by the conventional RL technique [153].

In the proposed auto-tuning method, different weight sets are deployed by adopting the exploration-exploitation concept from RL. Exploration refers to deploying a random MPC weight set while exploitation represents deploying a similar[1] one to the best MPC weight set obtained until the current trial. The balance between exploration and exploitation is governed by a parameter ϵ. By employing exploitation besides exploration, the grades (analogous to rewards in RL) of the previously deployed weight sets are utilized to interpret their neighborhood in the

[1]Similarity is governed by a parameter λ which represents the neighborhood radius of a weighting parameter as a percentage of its magnitude.

search space. This strategy caters to a more efficient exploration of the search space and yields better MPC weight sets in a shorter duration, as validated in the subsequent section.

The overall tuning procedure undertakes a few steps, as shown in Fig. 6.2. Throughout the process, each trial run starts with the 'Trial Configuration' module, wherein the 'Weight set selection' and 'Stable flight mode' submodules give necessary instructions to the NMPC. Taking these as inputs, NMPC controls the quadrotor model over the commanded trajectory while its performance is being observed simultaneously. Once the flight is completed, the performance of the NMPC with the utilized weight set is graded based on the five criteria (elaborated later). Subsequently, the trial is finalized by updating the weight set cluster. Thereafter, a new trial begins with the selection of a new weight set within the 'Trial Configuration' module, and thus, closes the loop. The tuning procedure is also summarized in Algorithm 1. In the next section, each of these steps is illustrated in detail.

6.4.3 Overall Framework with Implementation Details

A critical issue for the successful implementation of the proposed approach is to define reasonable bounds to the weight set cluster. For this application, these bounds are specified as the following order of magnitude for each weighting parameter:

$$\begin{aligned} \mathbf{W_x} \in \mathbb{W}_\mathbf{x} &= \mathrm{diag}\big([\mathcal{O}(10^2)]^{1\times 3}, [\mathcal{O}(10^1)]^{1\times 3}\big), \\ \mathbf{W_u} \in \mathbb{W}_\mathbf{u} &= \mathrm{diag}\big([\mathcal{O}(10^2)]^{1\times 3}, \mathcal{O}(10^{-2})\big), \end{aligned} \tag{6.4}$$

where $\mathcal{O}$ represents the order of magnitude, and $\mathbb{W}_\mathbf{x}$ and $\mathbb{W}_\mathbf{u}$ are the corresponding search spaces in $\mathbb{R}^{++}$. Note that to expedite the tuning process, the integer values for the weighting parameters are explored first, and are subsequently scaled by the corresponding factors such that they lie within the specified bounds.

During auto-tuning, three essential flight configurations are considered that show the characteristics of common flight envelopes for quadrotors: 'hover', 'move-to-a-setpoint', and 'follow-sequential-setpoints'. In hover mode, the robot tries to maintain its original position in the air. In move-to-a-setpoint mode, it tries to navigate to a predefined setpoint as fast as possible. In aptly named follow-sequential-setpoints mode, the robot tracks a sequence of setpoints.

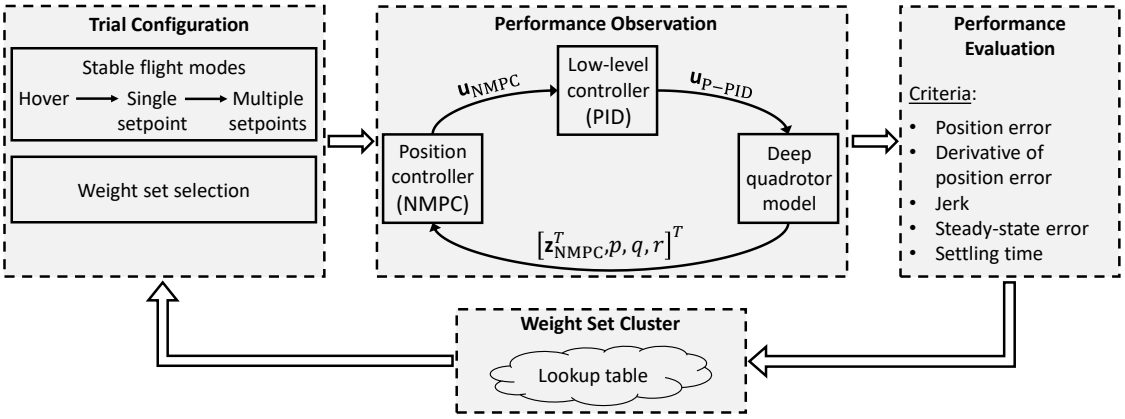


FIGURE 6.2: Proposed (N)MPC weight tuning framework.

Remark 35. Considering the common waypoint based operations of quadrotors, the first two modes may seem sufficient. However, as observed during the trials, successfully achieving these two flight modes may not guarantee a stable flight with certain MPC weights. That is, sequential waypoint performance plays an important role in assessing the weight set quality. Therefore, the follow-sequential-setpoints mode is included as the third flight configuration.

The criteria to assess different weight sets are position error (e), derivative of position error ($\dot{e}$), jerk (j), steady-state error (e_{ss}), and settling time (t_s). After a flight trial, a weight set gets a sub-grade for each of these criteria by:

$$G_c = \begin{cases} G_{c,\max}, & \text{for } G_{c,\max} \leq G_c \\ \dfrac{c_{\text{tol,init}}}{c}, & \text{for } c \leq c_{\text{tol}} \\ -G_{c,\max}, & \text{for } c_{\text{tol}} < c, \end{cases} \tag{6.5}$$

where c is the corresponding criteria, $c_{\text{tol,init}}$ and c_{tol} are the respective initial and current tolerance values for it. These sub-grades together form the grade G as follows:

$$G = \begin{cases} \dfrac{1}{n}\text{sum}(G_e, G_{\dot{e}}, G_j, G_{e_{ss}}, G_{t_s}), & \text{for } G_e, \cdots, G_{t_s} \in \mathbb{R}^+ \\ -G_{c,\max}, & \text{otherwise,} \end{cases} \tag{6.6}$$

wherein, $n = 13$, represents the number of criteria, the terms G_e, $G_{\dot{e}}$, G_j, $G_{e_{ss}}$, G_{t_s} represent the sub-grades and $G_{c,\max}$ is the maximum sub-grade value which is set to 100 in this chapter. Moreover, the following expressions are defined for the

Algorithm 1: Auto-tuning approach

Result: Real flight-worthy weight sets
Specify *maximum episodes*, ϵ, and λ
Specify the search space bounds as in (6.4)
Initialize the tolerance values as in Table 6.1
while *episode* $<$ *maximum episodes* **do**
  $\epsilon^* \sim \text{rand}([0,1])$ ▷ Randomly select within $[0,1]$
  if $\epsilon* < \epsilon$ **then**
    $W_{\mathtt{x},\mathtt{u}} \sim \text{rand}(W^*_{\mathtt{x},\mathtt{u}}, \lambda^2)$ ▷ Sample $W_{\mathtt{x},\mathtt{u}}$ within λ radius of $W^*_{\mathtt{x},\mathtt{u}}$
  else
    $W_{\mathtt{x},\mathtt{u}} \sim \text{rand}([\mathbb{W}_{\mathtt{x}}, \mathbb{W}_{\mathtt{u}}])$ ▷ Randomly sample $W_{\mathtt{x},\mathtt{u}}$ within (7)
  end
  foreach *flight mode* **do**
    Observe flight performance over the DNN model
    Compute G using (6.5) and (6.6)
    if $G < 0$ **then**
      episode $\leftarrow$ *episode* $+ 1$ ▷ Continue with next episodic trial
    end
  end
  Append $W_{\mathtt{x},\mathtt{u}}$ to the lookup table
  $N_w \leftarrow N_w + 1$
  Update c_{tol} based on (6.8)
end

weight set with maximum grade value in the lookup table:

$$\mathbf{W}^*_{\mathbf{x},\mathbf{u}} = [\mathbf{W}^*_{\mathbf{x}}, \mathbf{W}^*_{\mathbf{u}}] = \max_{\mathbf{W}_{\mathbf{x}}, \mathbf{W}_{\mathbf{u}}} G. \tag{6.7}$$

Besides, the relation between $c_{\text{tol,init}}$ and c_{tol} is defined as:

$$c_{\text{tol}} = (e^{-N_w/50}) c_{\text{tol,init}}, \tag{6.8}$$

where N_w is the number of available weight sets with a positive grade in the lookup table. The motive behind this tolerance decaying approach is to always look for better weight sets that can satisfy stricter criteria. In this chapter, the initial parameters $c_{\text{tol,init}}$ for the three different flight modes are selected as in Table 6.1, considering the usual values observed during common operations of quadrotors. However, since an exponential decay is incorporated on $c_{\text{tol,init}}$ as described in (6.8), the initial value selection is flexible as long as the values comply with safe flight conditions.

TABLE 6.1: Initial tolerance values for a stable flight.

Characteristics		**Tolerance values for each configuration**		
		Hover	Move-to-a-setpoint	Follow-sequential-setpoints
Position error (m)	e_x	0.15	0.3	0.35
	e_y	0.15	0.3	0.35
	e_z	0.15	0.3	0.35
Derivative of position error (m/s)	$\dot{e_x}$	0.025	0.4	0.5
	$\dot{e_y}$	0.025	0.4	0.5
	$\dot{e_z}$	0.03	0.45	0.55
Jerk (m/s^3)	j_x	0.5	2	2
	j_y	0.5	2	2
	j_z	1	3	3
Steady-state error (m)	e_{ss_x}	0.15	0.15	0.15
	e_{ss_y}	0.15	0.15	0.15
	e_{ss_z}	0.15	0.15	0.15
Settling time (s)	t_s	1.5	4.5	4.5

6.5 Tuning Approach in Action

In this section, the implementation results for the proposed active exploration-based tuning methodology are presented. Firstly, each design setting is discussed over batched tuning sessions in simulation and then the best setting which caters to the requirements is selected. Subsequently, the weight sets from simulation-based tuning are fine-tuned in real flights. Note that, while the simulation tuning sessions are performed on a workstation computer with 2.6GHz Intel Core i9-7980XE (octadeca-core) processor with 128GB RAM, the real flight tuning is performed on a laptop having Intel Core i7-8750H processor and 16GB RAM.

6.5.1 Benchmark Study for Simulation-based Tuning

A benchmark study is conducted to justify the design choices for the proposed auto-tuning method. For each design choice, a batch of ten tuning sessions is performed to have generalized results. Firstly, a heuristic to expedite the tuning process is examined: the weighting parameters along x and y body axes are close to each other since quadrotors are symmetrical with respect to these axes. Hence, the respective weights along these axes are generated within each other's 10% of magnitude neighborhood. As can be seen in Table 6.2, without the heuristic, no

positively graded weight set is explored in more than half of the sessions even in 1000 episodes setting. By exploiting this heuristic, on the other hand, desirable weight sets are obtained in most of the sessions for 500 episodes and some of the sessions for even 100 episodes. Therefore, this heuristic is employed for the rest of the results presented in this chapter.

Table 6.3 presents the number of sessions in which no positively graded weight set is explored. For both active ($\epsilon = 0.5$, $\lambda = 20\%$ – the exploration neighborhood –) and random ($\epsilon = 1$) explorations, 100 episodes seem to be less to explore desirable weight sets. There are seven failed sessions out of ten sessions for active exploration in this case. For the 1000 episodes, on the other hand, there is only one failed session (90% success) both for active and random exploration. There are only two failed sessions when the episode number is 500, which implies 80% success. This setting is selected for the tuning purposes since 1000 episodes take around twice the duration of 500 episodes while the success rate improvement is only 10%.

Next, Table 6.4 presents the average and the maximum number of positively graded weight sets. The proposed active exploration outperforms the random exploration by yielding a higher number of positively graded weight sets. It caters for obtaining substantially more weight sets in a similar amount of time. A remark here is that the number of weight sets obtained does not change significantly when the episode number is increased from 500 to 1000 while the overall tuning duration increases approximately two times. This result further supports the previous episode number selection of 500 due to the trade-off between the tuning duration and success rate.

Furthermore, Table 6.5 presents the average and the maximum grade values for the positively graded weight sets obtained over ten sessions. In all the three episode settings, average and maximum grade values are higher for active exploration. In

TABLE 6.2: Number of sessions without any positively graded weight set by using or omitting the heuristic.

Episodes	**Without heuristic**	**With heuristic**
100	10	7
500	7	2
1000	7	1

TABLE 6.3: Number of sessions without any positively graded weight set with active exploration ($\epsilon = 0.5$) and random exploration ($\epsilon = 1$).

Episodes	$\epsilon = \mathbf{0.5}$	$\epsilon = \mathbf{1}$
100	7	8
500	2	4
1000	1	1

TABLE 6.4: Number of weight sets obtained with active exploration ($\epsilon = 0.5$) and random exploration ($\epsilon = 1$) averaged over ten batched sessions.

	$\epsilon = \mathbf{0.5}$		$\epsilon = \mathbf{1}$	
Episodes	Average	Maximum	Average	Maximum
100	1.4	6	0.2	1
500	12.7	24	1	2
1000	15.5	23	1.3	3
Mean	9.87	17.67	0.83	2

TABLE 6.5: Grade values averaged over ten batched sessions for the positively graded weight sets obtained with active exploration ($\epsilon = 0.5$) and random exploration ($\epsilon = 1$). One may notice the same average and maximum grades for $\epsilon = 1$ with 100 episodes. This is due to the single positively graded weight set as resulted by the corresponding setting (Table 6.4).

	$\epsilon = \mathbf{0.5}$		$\epsilon = \mathbf{1}$	
Episodes	Average	Maximum	Average	Maximum
100	10.5226	10.7634	7.1828	7.1828
500	9.0791	10.2100	8.6367	9.2077
1000	9.0089	10.2899	8.6427	9.1658
Mean	9.5371	10.4211	8.1541	8.5188

other words, the weight sets obtained via active exploration satisfy stricter criteria (e, $\dot{e}$, j, e_{ss}, t_s), and hence, they are better in quality. All these results prove the superiority of the proposed tuning method over a random trial-and-error method.

Additionally, Table 6.5 presents the average and maximum grade values for the positively graded weight sets obtained over ten sessions. In all the three episode settings, average and maximum grade values are higher for active exploration. In other words, the weight sets obtained via active exploration yield less values for the aforementioned five characteristics (e, $\dot{e}$, j, e_{ss}, t_s), and hence, they are better in quality. All these results prove the superiority of the proposed tuning method over a random trial-and-error method.

6.5.2 Fine-tuning in Real Flight

Although the DNN model represents the robot's dynamics with high fidelity, certain operational uncertainties are challenging to capture. The underlying reason is their highly stochastic nature, which would require an indefinite amount of data for training. In this vein, one could decide to collect hours-and-hours of operational

data for creating a highly accurate DNN model and just perform DNN-based MPC tuning. On the other hand, another approach is to create a fairly accurate DNN model with a moderate amount of data and perform MPC tuning over it; followed by repeating the tuning procedure over the real robot to further improve the quality of the obtained weight sets. In this chapter, the latter is utilized, even though it is the user's preference.

For fine-tuning in real flights, Firstly, new grade values are obtained for all the positively graded weight sets from simulation-based tuning. This step is conducted to account for possible real-world operational uncertainties. As expected, only 17 out of 19 weight sets yield new grade values as positive. In other words, there are some weight sets, which can fulfill the design criteria (Table 6.1) over the DNN model but are unable to do so over the real robot. Then, the real flight tuning is performed over 30 episodes with $\epsilon = 0$ and $\lambda = 10\%$ using the new grades. These hyperparameters are conservative versions of the ones selected for simulation-based tuning. Note that selecting $\epsilon = 0$ makes sure that there will not be any trial with a random weight set, whereas, $\lambda = 10\%$ focuses the search closer to the desirable weight sets obtained from simulation-based tuning. Throughout the real flight tuning, the average and maximum grade values increase from 9.34 and 12.36 to 9.68 and 13.33. This result demonstrates the successful fine-tuning in real flights.

6.6 Trajectory Tracking Results

This section presents evaluative results for the weight sets obtained using the proposed auto-tuning method. The weight set with the highest grade from simulation-based tuning are deployed in the position tracking NMPC for tracking two types of trajectories: hover ($x = 0$, $y = 0$, $z = 1.2$m), and sequential-setpoints (setpoints being 0.6m-1.2m apart). Weight set having the highest grade from the real flight tuning is also utilized for tracking the same trajectories. Then, a performance comparison is conducted to further justify the fine-tuning in real flights.

6.6.1 Simulation-based Tuning

The best weight set from simulation-based tuning is:

$$\mathbf{W_x} = \text{diag}(16, 11, 13, 2.0, 2.1, 2.3), \quad \mathbf{W_u} = \text{diag}(17, 13, 76, 0.058).$$

For hovering, this weight set results in a precise tracking with mean Euclidean error values of $3 - 4$ cm over both the DNN model and real robot. As for the sequential-setpoints tracking, it again yields precise tracking with mean Euclidean error values of 21 cm (Fig. 6.3). Similar performance obtained over the DNN model and real robot for both the trajectories further validates the high fidelity of the DNN model.

Remark 36. The error values for sequential-setpoints are significantly higher as compared to the ones obtained for hovering. The main reason behind this is the underlying jumps of 1.2m that the robot has to execute in order to reach the commanded setpoint as fast as possible.

The above experiments are repeated with the ten best weight sets from simulation-based tuning to assess the overall quality. Their corresponding mean Euclidean error values are presented on the left side of Fig. 6.5. The average and maximum error values over the DNN model are 2.98 cm and 3.49 cm for hovering. The same values over the real robot are 5.11 cm and 7.94 cm. A similar result is obtained for sequential-setpoints. The respective average and maximum error values rise from 22.12 cm and 26.61 cm over the DNN model to 22.21 cm and 28.27 cm over the real robot. The error values rise slightly in the case of the real robot because of the operational uncertainties that are possibly not captured by the DNN model.

6.6.2 Real Flight Tuning

The best weight set from real flight tuning is:

$$\mathbf{W_x} = \text{diag}(17, 11, 14, 1.5, 1.8, 1.9), \quad \mathbf{W_u} = \text{diag}(20, 15, 68, 0.064).$$

For hovering, this weight set results in a precise tracking with mean Euclidean error values of $3 - 4$ cm over both the DNN model and real robot. As for the

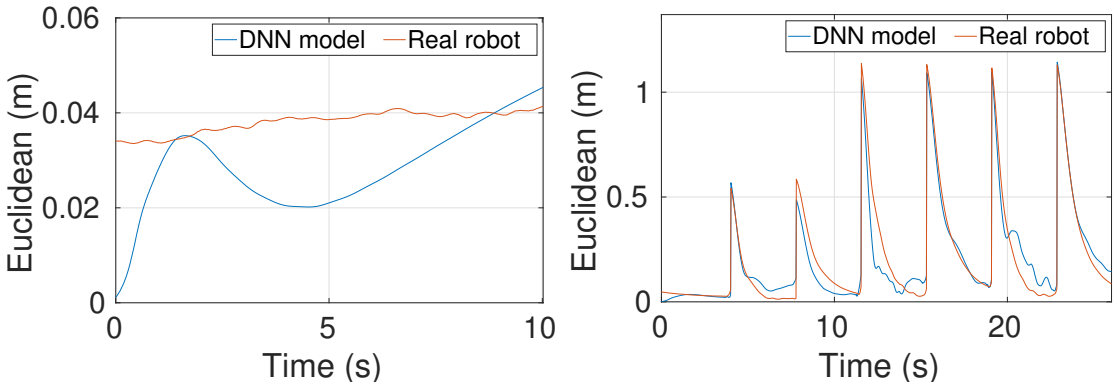


FIGURE 6.3: Euclidean errors for hover (left) and sequential-setpoints (right) by the weight set from simulation tuning.

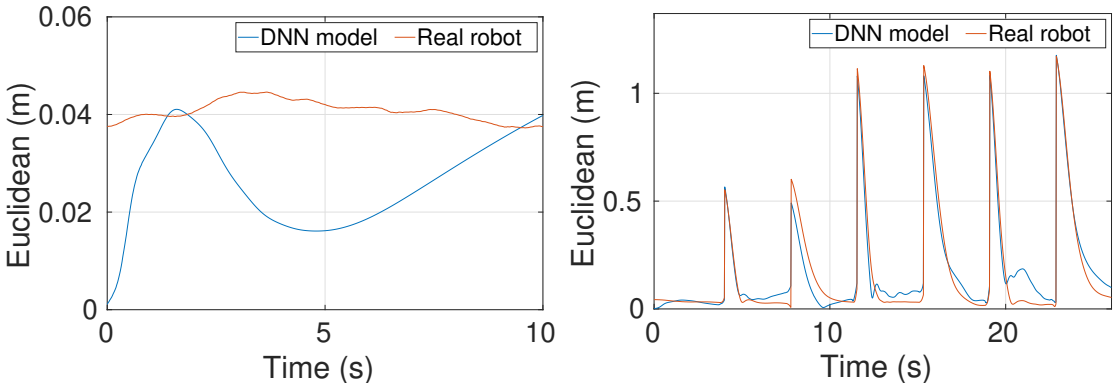


FIGURE 6.4: Euclidean errors for hover (left) and sequential-setpoints (right) by the weight set from real flight tuning.

sequential-setpoints tracking, it again yields precise tracking performance with the mean Euclidean error values of around 18 cm (Fig. 6.4) which are slightly less as compared to the ones in the former subsection. This implies the improvement by fine-tuning in real flights.

Again, the above experiments are repeated with the ten best weight sets from real flight tuning and complete the right side of Fig. 6.5. While the average and maximum error values for hovering with the DNN model are 2.63 cm and 2.85 cm, they increase slightly over the real robot to 4.54 cm and 6.12 cm. As for sequential-setpoints the average and maximum error values changes from 19.43 cm and 21.16 cm over the DNN model to 18.49 cm and 21.7 cm over the real robot.

An important remark in Fig. 6.5 is that the respective average and maximum mean Euclidean error values obtained over the real robot reduce from 5.11 cm and 7.94 cm to 4.54 cm and 6.12 cm by real flight tuning, which implies 11.15% and 22.92% improvement in these values. For the sequential-setpoints tracking case, these numbers reduce from 22.21 cm and 28.27 cm to 18.49 cm and 21.7 cm, which

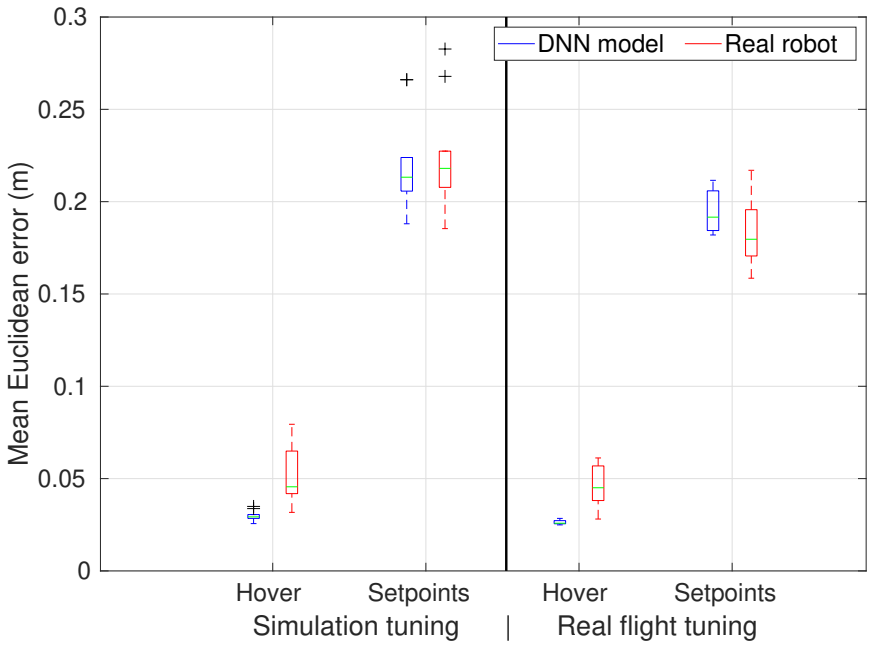


FIGURE 6.5: Boxplot of mean Euclidean error obtained by the weight sets having top ten grade values in the lookup tables from simulation (left) and real flight (right) tuning.

suggests a performance improvement by 16.75% and 23.24% in terms of average and maximum mean Euclidean errors. These small improvements exhibit both the high fidelity of the DNN model and reveal the possible benefits of fine-tuning in real flights.

6.6.3 User-based Tuning Study

To further analyze the proposed auto-tuning method's efficacy, a user-based tuning study is conducted for comparison. Ten users with different quadrotor experience are given the task of tuning NMPC over a Gazebo model[2] of the quadrotor for two hours. Two weight sets are recorded from each user: one after the first hour and one after the second hour of tuning. Then, trajectory tracking is performed with over the DNN model[3] to evaluate the performance of these weight sets. The resulting mean Euclidean errors are listed in Table 6.6 along with the observed number of oscillations, depicting the oscillatory behavior of a particular weight set.

[2]The Gazebo simulation platform is utilized as it has a graphical user interface. In other words, the vehicle's response can be observed visually as being similar to the manual tuning in real flights.

[3]These weight sets are not tested on the real robot for safety.

It is to be noted that oscillation is characterized as an abrupt change of more than 3 cm in the Euclidean error response. As can be seen, most users resulted in high Euclidean errors with moderate to high oscillations, implying an eventual crash of the robot. Whereas, three users obtained the real flight-worthy weight sets (marked with bold-font) which accounts for only 15% success. It is emphasized that most users could obtain some meaningful weight sets in a limited time as they could try abrupt values over the simulation model which otherwise might not be possible over the real robot. Essentially, the proper tuning procedure in a general case takes around 3-4 hours (or even more) as noticed during informal observations. Another point to take note in Table 6.6 is that for almost half of the users, the tracking performance decreased from the first to the second hour of tuning. This is counterintuitive as one expects to perform better after gaining some experience with the system. Nevertheless, utilizing the relation in (6.8), the proposed algorithm always looks for the weight sets which could perform better by satisfying stricter criteria.

For a quantitative comparison, the best weight sets from user-based tuning (User 8 after the first hour) and simulation-based tuning are deployed for hover and

TABLE 6.6: Mean Euclidean error and the corresponding oscillatory response for the weight sets obtained by user-based tuning, wherein $\mathsf{e}^{\text{Euc}}_{\text{mean}}$ and ω represent the mean Euclidean error and number of oscillations, respectively. Accordingly, the weight sets whose results are marked with bold-font are regarded as real flight-worthy.

User	**First hour**				**Second hour**			
	Hover		Sequential-setpoints		Hover		Sequential-setpoints	
	$\mathsf{e}^{\text{Euc}}_{\text{mean}}$ (cm)	ω	$\mathsf{e}^{\text{Euc}}_{\text{mean}}$ (cm)	ω	$\mathsf{e}^{\text{Euc}}_{\text{mean}}$ (cm)	ω	$\mathsf{e}^{\text{Euc}}_{\text{mean}}$ (cm)	ω
1	12.70	31	1.21×10^{5}	176	4.83	1	1.10×10^{4}	204
2	30.64	20	3.09×10^{4}	220	34.51	20	89.47	34
3	15.10	34	7.21×10^{4}	138	14.42	29	1.03×10^{5}	232
4	10.74	20	25.74	50	10.09	18	38.13	47
5	13.29	36	7.82×10^{4}	146	**6.24**	**0**	**23.41**	**17**
6	3.96	2	24.87	37	12.46	19	33.78	45
7	**5.28**	**0**	**35.11**	**5**	9.70	6	31.88	31
8	**3.41**	**3**	**22.01**	**21**	2.69	0	23.62	44
9	8.50	16	4.36×10^{6}	250	4.20	1	36.55	29
10	3.02	0	38.26	32	8.23	10	26.78	35

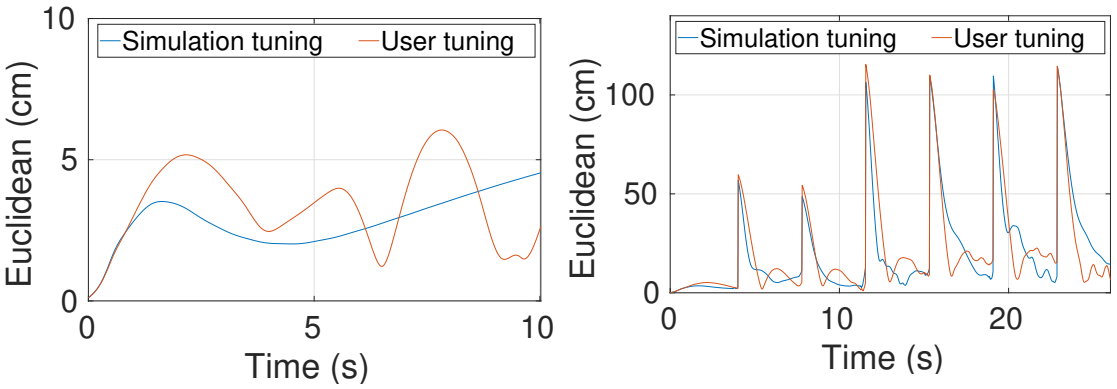


FIGURE 6.6: Euclidean errors for hover (left) and sequential-setpoints (right) by the weight sets from simulation and user-based tuning.

sequential-setpoints tracking of the DNN model. The comparison plots are given in Fig. 6.6. While the corresponding mean Euclidean error values for user-based weight set are 3.41 cm and 22.01 cm, the proposed algorithm outperforms it by resulting in the error values of 2.85 cm and 21.16 cm, respectively. Besides, in terms of the number of oscillations, the user-based weight set exhibits 3 and 21 for hover and sequential-setpoints, respectively, whereas the simulation-based weight set exhibits 0 and 9, respectively. These values are significantly less when compared to the user-based weight set, thereby implying a safer weight set. All these results signify that the proposed method can realize better weight sets in a limited time (less than an hour) that an average user cannot obtain even after two hours.

6.7 Conclusions

This chapter caters to one of the arduous yet unavoidable tasks associated with the realtime implementation of MPC over robots. A novel, active exploration-based tuning framework has been presented for obtaining MPC weight sets which otherwise are obtained by the trial-and-error methods. To avoid the weight set trials on the real robot, which could jeopardize the safety of the system, a DNN model has been incorporated. Thanks to its high fidelity representation of the real robot's dynamics, it has facilitated the direct utilization of the obtained weight sets over the real robot. Besides, the fine-tuning possibility over the real robot has also been demonstrated in this chapter. Extensive statistical analysis, real flight trajectory tracking results, and a comparative user study validate that this chapter

could help researchers with the realtime implementation of their MPCs by saving a considerable amount of tuning time without compromising the safety of the robot.

Chapter 7

Conclusion and Future Scope

7.1 Conclusions

In this thesis, the challenges associated with the precise tracking control of multirotor aerial robots in uncertain environmental conditions have been addressed. These robotic platforms are strongly coupled, inherently nonlinear, and open-loop unstable which imply a challenging control problem. In addition, their operation in unknown environments results in an uncertain model which negatively affects the tracking accuracy for model-based controllers. Besides, most of the control algorithms require a tuning phase that is tedious for novice users and dangerous to be carried out directly on real robots. In essence, the main motivation in this thesis has been to realize an aerial robot that can learn from its past interactions with the environment, and thus, can perform sufficiently well in unknown working conditions for which it has not been trained. Accordingly, the main tasks have been distributed along four objectives: (i) design control algorithms to explicitly cater to the nonlinear dynamics of the aerial robots without linearization, (ii) accommodate fast solution methodologies to facilitate the implementation of NMPC over low-cost embedded processors, (iii) develop reliable online learning algorithms to account for the uncertain dynamics due to operation in unknown environment, and (iv) develop a safe automated tuning framework for the underlying control algorithm.

In order to achieve the first objective, two control approaches have been adopted. First is the NMPC that directly accommodates the nonlinear dynamics along with

system constraints in the problem formulation. Its efficacy has been illustrated for the tracking control of two types of multirotor aerial robots, namely, quadrotor and tilt-rotor tricopter. The second control approach is the FLC method that reduces the nonlinear system into its linear equivalent. Its tracking performance has been demonstrated for the trajectory tracking application of the tilt-rotor tricopter.

The adopted fast solution methods and efficient C++ scripts have resulted in successful onboard implementations of the control algorithms, further realizing the second objective. In particular, the adopted direct multiple shooting method along with the GGN-based Hessian approximation and the special RTI scheme has facilitated the implementation of NMPC and NMHE algorithms on a low-cost embedded processor, i.e., Raspberry Pi 3.

To realize the third objective, two model learning approaches, namely, instantaneous learning and iterative learning, have been demonstrated. In terms of the InLC scheme, two control frameworks have been developed. The first one employs an NMHE to estimate the time-varying model parameters that makes the NMPC adaptive to the changing working conditions. Thanks to the constrained learning ability of NMHE, superior tracking performance of the NMPC-NMHE framework over the conventional NMPC has been illustrated from the two case studies. The second control framework incorporates an SL strategy to circumvent the limitations of the traditional FLC method by updating the controller gains and the disturbance estimate according to the changing environment. The tracking results have exhibited that unlike the traditional FLC method, the SL-FLC framework ensures the nominal control performance in the presence of modeling uncertainties and external disturbance.

In terms of the second model learning approach, i.e., the ILC scheme, a GP-based regression has been adopted to learn the disturbance forces that are encountered in an offshore visualization operation of the aerial robot. Essentially, updating these disturbance forces in NMPC's model definition facilitates its adaptive nature. This results in improved overall tracking accuracy over the conventional NMPC, which has been depicted by the simulated as well as the real-world test results. Furthermore, thanks to the LSTM feature of the designed GP model, a superior learning performance over the NMHE has been realized.

Lastly, an active exploration approach has been proposed for achieving the fourth objective. The proposed auto-tuning approach extends the basic trial-and-error method and tunes the weight sets more intelligently by benefiting from the retrospective knowledge gained in previous trials. Moreover, a DNN-based robot model has been incorporated to circumvent the tuning over the real robot, hence ensures a safe tuning process. Additionally, owing to the high-fidelity representation of the DNN model, an impeccable sim-to-real transition has been exhibited by directly deploying the weight sets from simulation tuning over the real robot.

Overall, the main goal of this work has been to facilitate a safe transition of aerial robots from a controlled lab environment to the uncertain real-world. In that vein, both the proposed control frameworks (InLC and ILC) render them to learn from their interactions with the environment, thereby gracefully accommodating the underlying uncertainties. Besides, the auto-tuning methodology circumvents the unsafe manual-tuning process while resulting in the flight-worthy controller parameters. All this work has been a small step towards a greater vision, which is "designed with more adaptive control structures – also with the help of artificial intelligence and machine learning methods –, it is not so far-fetched that the robots of the future could learn from their own experience, interact with the environment, and eventually, adapt to different working conditions".

7.2 Future Scope

The RTI scheme that has been discussed in Chapter 2, utilizes a GGN approximation to obtain a solution to the reduced SQP of type (2.23). In essence, for a least square problem, the GGN method approximates the Hessian of the Lagrangian utilizing a residual function, shown in (2.26). However, this approximation works well for the applications with a low magnitude of the residual function, i.e., moderately nonlinear problems. Since the aerial robots are significantly nonlinear, the underlying approximation may fail, especially during their agile operations. Therefore, there is a need to obtain better approximations for the Hessian in the case of highly nonlinear problems like tracking control of aerial robots.

Chapter 3 has illustrated the implementations of NMPC and NMHE over a low-cost embedded processor, i.e., Raspberry Pi 3. However, the sampling frequency

that could be achieved, especially for NMHE, is barely sufficient for sustaining the closed-loop. Therefore, to support fast robotic applications, additional research efforts can be spent to develop efficient solution techniques that could solve the OCPs of NMPC and NMHE even faster. Additionally, the feasibility of field-programmable-gate-arrays (FPGAs), which have already been utilized with linear MPCs in [155, 156], could also be explored for aerial applications of NMPC. Essentially, an FPGA represents reconfigurable hardware that contains a far more fixed/floating-point arithmetic-logic units than are present in any common central processing unit. Hence, this flexibility and a higher number of processing units in a single chip make them a strong candidate for fast NMPC applications. Although a simulated aircraft trajectory planning implementation of NMPC utilizing a 3D kinematic model is reported in [157], the real implementation incorporating the dynamic model is still missing. Furthermore, the implementation of NMPC over ultra low power processors, for instance, a parallel ultra-low-power-platform (PULP), which are essential for micro-aerial robotic applications can also be exploited in the future.

Chapter 4 has presented an SL strategy to circumvent the limitation of a traditional FLC method for an uncertain nonlinear system. The updated feedback control law incorporating the SL strategy in (4.10), assumes that the disturbances consist of a constant and vanishing perturbation terms, whereby its average rate of change is much lower than the error states. Since the wind disturbance may continuously vary during the offshore operation of the aerial robots, the SL algorithm can further be investigated for non-vanishing perturbation terms. Besides, the SL-FLC framework can also be experimentally compared with its other InLC counterpart, i.e., the NMPC-NMHE framework.

Chapter 5 has demonstrated a GP-based regression technique to estimate the wind disturbance forces that are encountered during the offshore operation. In this thesis, a zero-mean along with a commonly used SE covariance function has been adopted as a suitable GP prior. Although the wind disturbance is primarily stochastic, it does exhibit some pseudo-periodic behavior over short time-scales. Hence, there is a possibility to investigate the usage of pseudo-periodic covariance functions within the GP-based wind disturbance modeling.

A DNN-based robot model has been adopted in Chapter 6 to ensure the safety of the robot during the tuning phase. While the incorporated DNN model has

represented the robot's dynamics fairly well, it may be tedious to obtain in the first place. Therefore, there is a possibility to explore the usage of different modeling techniques, for instance, GP-based modeling. Furthermore, at the end of the tuning process, the final weight has been selected based on the highest grade value in the lookup table. Although this selection results in stable tracking performance, the desired flight behavior, i.e., agile or smooth, remains unaccounted. Therefore, the overall framework can be extended to accommodate the user's choice of the required flight characteristics.

Video Links

- **Chapter 3:**
  https://youtu.be/suHGUnEetj4
- **Chapter 4:**
  https://youtu.be/inMhrK9tf8A
- **Chapter 5:**
  https://youtu.be/6zVDKXxXuWs
- **Chapter 6:**
  https://youtu.be/GLxRPCyNogc

List of Author's Awards, Patents, and Publications[1]

Book Chapters

- **Mohit Mehndiratta**, Erkan Kayacan, Siddharth Patel, Erdal Kayacan and Girish Chowdhary, "Learning-based fast nonlinear model predictive control for custom-made 3D printed ground and aerial robots." In Handbook of Model Predictive Control, pp. 581-605. Birkhäuser, Cham, 2019. **Chapter 3**

Journal Articles

- **Mohit Mehndiratta**, Efe Camci and Erdal Kayacan. "Can Deep Models Help a Robot to Tune Its Controller? A Step Closer to Self-tuning Model Predictive Controller." In IEEE/ASME Transactions on Mechatronics. (Submitted) **Chapter 6**

- **Mohit Mehndiratta**, Erkan Kayacan, Mahmut Reyhanoglu and Erdal Kayacan. "Robust Tracking Control of Aerial Robots Via a Simple Learning Strategy-Based Feedback Linearization," in IEEE Access, vol. 8, pp. 1653-1669, 2020, doi: 10.1109/ACCESS.2019.2962512. **Chapter 4**

- **Mohit Mehndiratta** and Erdal Kayacan. "A constrained instantaneous learning approach for aerial package delivery robots: onboard implementation and experimental results." In Autonomous Robots 43, no. 8 (2019): 2209-2228. **Chapter 3**

[1]The superscript * indicates joint first authors

- **Mohit Mehndiratta** and Erdal Kayacan. "Receding horizon control of a 3 DOF helicopter using online estimation of aerodynamic parameters." In Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering (2017): 0954410017703414.

Conference Proceedings

- **Mohit Mehndiratta** and Erdal Kayacan. "Gaussian Process-based Learning Control of Aerial Robots for Precise Visualization of Geological Outcrops," 2020 European Control Conference (ECC), Saint Petersburg, Russia, 2020, pp. 10-16. **Chapter 5**

- **Mohit Mehndiratta**, Efe Camci and Erdal Kayacan. "Automated Tuning of Nonlinear Model Predictive Controller by Reinforcement Learning." In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3016-3021. IEEE, 2018. **Chapter 6**

- **Mohit Mehndiratta** and Erdal Kayacan. "Reconfigurable fault-tolerant NMPC for Y6 coaxial tricopter with complete loss of one rotor." In 2018 IEEE conference on control technology and applications (CCTA), pp. 774-780. IEEE, 2018.

- **Mohit Mehndiratta**, Erkan Kayacan and Erdal Kayacan. "A Simple Learning Strategy for Feedback Linearization Control of Aerial Package Delivery Robot." In 2018 IEEE Conference on Control Technology and Applications (CCTA), pp. 361-367. IEEE, 2018. **Chapter 4**

- **Mohit Mehndiratta** and Erdal Kayacan. "Online Learning-based Receding Horizon Control of Tilt-rotor Tricopter: A Cascade Implementation." In 2018 Annual American Control Conference (ACC), pp. 6378-6383. IEEE, 2018. **Chapter 3**

- Wilson Ying Jun Lee, **Mohit Mehndiratta** and Erdal Kayacan. "Fly without borders with additive manufacturing: a microscale tilt-rotor tricopter design." In Proceedings of the 3rd International Conference on Progress in Additive Manufacturing (Pro-AM 2018), pp 256-261, 2018.

- Ruddhi Gokhale, Yi Wan, **Mohit Mehndiratta**, and Erdal Kayacan. "Fixed-wing vertical-takeoff-and-landing UAV with additive manufacturing: a dual-rotor version." In Proceedings of the 3rd International Conference on Progress in Additive Manufacturing (Pro-AM 2018), pp 250-255, 2018.

- **Mohit Mehndiratta**, Anna Prach and Erdal Kayacan. "Numerical Investigation of Gaussian filters with a combined type Bayesian filter for nonlinear state estimation." IFAC-PapersOnLine 49, no. 18 (2016): 446-453.

- **Mohit Mehndiratta**, Erdal Kayacan and Tufan Kumbasar. "Design and experimental validation of single input type-2 fuzzy PID controllers as applied to 3 DOF helicopter testbed." In 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1584-1591. IEEE, 2016.

Bibliography

[1] USA TODAY. Amazon wants to drop packages by parachute, 2019. URL https://www.usatoday.com/story/tech/news/2017/02/15/how-amazon-might-deliver-packages-drone/97937664/?hootPostID=df519a616bc3f318bebaa63ceb8226c4. [accessed 9-November-2019]. xxi, 3

[2] sUAS News. Drone technology improves inspection of u.s. offshore wind platform, 2019. URL https://www.suasnews.com/2018/10/drone-technology-improves-inspection-of-u-s-offshore-wind-platforms/. [accessed 9-November-2019]. xxi, 3

[3] dronelife. Transurban: Using drones to maintain roads & bridges down under, 2019. URL https://dronelife.com/2019/02/20/transurban-using-drones-to-maintain-roads-bridges-down-under/. [accessed 9-November-2019]. xxi, 3

[4] Rien Quirynen. *Numerical Simulation Methods for Embedded Optimization*. PhD thesis, KU Leuven, 2017. xxi, 24, 37

[5] U. Eren, A. Prach, Başaran B. Koçer, Saša V. Raković, E. Kayacan, and B. Açıkmeşe. Model predictive control in aerospace systems: Current state and opportunities. *Journal of Guidance, Control, and Dynamics*, 40(7):1541–1566, 2017. ISSN 0731-5090. doi: 10.2514/1.G002507. URL https://doi.org/10.2514/1.G002507. xxv, 29, 35

[6] E. T. Alotaibi, S. S. Alqefari, and A. Koubaa. Lsar: Multi-uav collaboration for search and rescue missions. *IEEE Access*, 7:55817–55832, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2912306. 1

[7] Juntong Qi, Dalei Song, Hong Shang, Nianfa Wang, Chunsheng Hua, Chong Wu, Xin Qi, and Jianda Han. Search and Rescue Rotary-Wing UAV and Its Application to the Lushan Ms 7.0 Earthquake. *Journal of Field Robotics*, 33 (3):290–321, 2016. 1

[8] Amazon. Amazon prime air, 2017. URL https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011. [accessed 4-September-2017]. 1

[9] Jinhong Chen, Haoting Liu, Jingchen Zheng, Ming Lv, Beibei Yan, Xin Hu, and Yun Gao. Damage degree evaluation of earthquake area using UAV aerial image. *International Journal of Aerospace Engineering*, 2016, 2016. 1

[10] Miao Li, Lu Zhen, Shuaian Wang, Wenya Lv, and Xiaobo Qu. Unmanned aerial vehicle scheduling problem for traffic monitoring. *Computers & Industrial Engineering*, 122:15 – 23, 2018. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2018.05.039. URL `http://www.sciencedirect.com/science/article/pii/S0360835218302481`. 1

[11] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. Receding horizon "next-best-view" planner for 3d exploration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1462–1468, May 2016. doi: 10.1109/ICRA.2016.7487281. 1

[12] Sujit Rajappa, Heinrich Bülthoff, and Paolo Stegagno. Design and implementation of a novel architecture for physical human-UAV interaction. *The International Journal of Robotics Research*, 36(5-7):800–819, 2017. doi: 10.1177/0278364917708038. URL `https://doi.org/10.1177/0278364917708038`. 1

[13] Daniel Szafir, Bilge Mutlu, and Terrence Fong. Designing planning and control interfaces to support user collaboration with flying robots. *The International Journal of Robotics Research*, 36(5-7):514–542, 2017. doi: 10.1177/0278364916688256. URL `https://doi.org/10.1177/0278364916688256`. 1

[14] S. Bouabdallah, A. Noth, and R. Siegwart. PID vs LQ control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2451–2456 vol.3, Sept 2004. 1

[15] Erkan Kayacan, Erdal Kayacan, Herman Ramon, and Wouter Saeys. Distributed nonlinear model predictive control of an autonomous tractor-trailer system. *Mechatronics*, 24(8):926 – 933, 2014. ISSN 0957-4158. 2

[16] E. Kayacan, E. Kayacan, H. Ramon, and W. Saeys. Learning in centralized nonlinear model predictive control: Application to an autonomous tractor-trailer system. *IEEE Transactions on Control Systems Technology*, 23(1):197–205, Jan 2015. 35

[17] E. Kayacan, J. M. Peschel, and E. Kayacan. Centralized, decentralized and distributed nonlinear model predictive control of a tractor-trailer system: A comparative study. In *2016 American Control Conference (ACC)*, pages 4403–4408, July 2016. doi: 10.1109/ACC.2016.7525615. 2

[18] Wilson Ying Jun Lee, Mohit Mehndiratta, and Erdal Kayacan. Fly without borders with additive manufacturing: a microscale tilt-rotor tricopter design. In *Proceedings of the 3rd International Conference on Progress in Additive Manufacturing (Pro-AM 2018)*, pages 256–261, May 2018. doi: 10.25341/D43K5G. 2

[19] Anna Prach and Erdal Kayacan. An MPC-based position controller for a tilt-rotor tricopter VTOL UAV. *Optimal Control Applications and Methods*, 2017. ISSN 1099-1514. doi: 10.1002/oca.2350. URL `http://dx.doi.org/10.1002/oca.2350`.

[20] H. Seo, S. Kim, and H. J. Kim. Aerial grasping of cylindrical object using visual servoing based on stochastic model predictive control. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6362–6368, May 2017. doi: 10.1109/ICRA.2017.7989751.

[21] Kostas Alexis, Christos Papachristos, Roland Siegwart, and Anthony Tzes. Robust Model Predictive Flight Control of Unmanned Rotorcrafts. *Journal of Intelligent & Robotic Systems*, 81(3):443–469, 2016.

[22] G. Garimella and M. Kobilarov. Towards model-predictive control for aerial pick-and-place. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4692–4697, May 2015. doi: 10.1109/ICRA.2015.7139850. 2

[23] M. Mehndiratta and E. Kayacan. Reconfigurable fault-tolerant nmpc for y6 coaxial tricopter with complete loss of one rotor. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 774–780, Aug 2018. doi: 10.1109/CCTA.2018.8511444. 2

[24] Jan Dentler, Somasundar Kannan, Souad Bezzaoucha, Miguel Angel Olivares-Mendez, and Holger Voos. Model predictive cooperative localization control of multiple uavs using potential function sensor constraints. *Autonomous Robots*, Mar 2018. ISSN 1573-7527. doi: 10.1007/s10514-018-9711-z. URL `https://doi.org/10.1007/s10514-018-9711-z`. 2

[25] Erkan Kayacan, Erdal Kayacan, Herman Ramon, and Wouter Saeys. Nonlinear modeling and identification of an autonomous tractor-trailer system. *Computers and Electronics in Agriculture*, 106:1 – 10, 2014. ISSN 0168-1699. doi: http://dx.doi.org/10.1016/j.compag.2014.05.002. 2

[26] D. Mellinger, Q. Lindsey, M. Shomin, and V. Kumar. Design, modeling, estimation and control for aerial grasping and manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2668–2673, Sept 2011. doi: 10.1109/IROS.2011.6094871. 2

[27] J. A. Meda-Campana. Estimation of complex systems with parametric uncertainties using a JSSF heuristically adjusted. *IEEE Latin America Transactions*, 16(2):350–357, Feb 2018. ISSN 1548-0992. doi: 10.1109/TLA.2018.8327386. 2

[28] J. A. Meda-Campaña. On the estimation and control of nonlinear systems with parametric uncertainties and noisy outputs. *IEEE Access*, 6:31968–31973, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2846483. 2

[29] Junyan Hu and Alexander Lanzon. An innovative tri-rotor drone and associated distributed aerial drone swarm control. *Robotics and Autonomous Systems*, 103:162 – 174, 2018. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2018.02.019. URL `http://www.sciencedirect.com/science/article/pii/S0921889017308163`. 4

[30] Carlos Izaguirre-Espinosa, Aldo-Jonathan Muñoz-Vázquez, Anand Sanchez-Orta, Vicente Parra-Vega, and Pedro Castillo. Contact force tracking of quadrotors based on robust attitude control. *Control Engineering Practice*, 78:89 – 96, 2018. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2018.06.013. URL http://www.sciencedirect.com/science/article/pii/S0967066118301898.

[31] Pradeep R. Ambati and Radhakant Padhi. Robust auto-landing of fixed-wing uavs using neuro-adaptive design. *Control Engineering Practice*, 60:218 – 232, 2017. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2016.03.017. URL http://www.sciencedirect.com/science/article/pii/S0967066116300582.

[32] Devaprakash Muniraj, Mark C. Palframan, Kyle T. Guthrie, and Mazen Farhood. Path-following control of small fixed-wing unmanned aircraft systems with h_∞ type performance. *Control Engineering Practice*, 67:76 – 91, 2017. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2017.07.006. URL http://www.sciencedirect.com/science/article/pii/S0967066117301570.

[33] E. Kayacan. Multiobjective h_∞ control for string stability of cooperative adaptive cruise control systems. *IEEE Transactions on Intelligent Vehicles*, 2(1):52–61, March 2017. ISSN 2379-8858. doi: 10.1109/TIV.2017.2708607.

[34] B. Zhao, B. Xian, Y. Zhang, and X. Zhang. Nonlinear robust adaptive tracking control of a quadrotor UAV via immersion and invariance methodology. *IEEE Transactions on Industrial Electronics*, 62(5):2891–2902, May 2015. ISSN 0278-0046. doi: 10.1109/TIE.2014.2364982.

[35] Arkadiusz Mystkowski. Implementation and investigation of a robust control algorithm for an unmanned micro-aerial vehicle. *Robotics and Autonomous Systems*, 62(8):1187 – 1196, 2014. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2014.04.002. URL http://www.sciencedirect.com/science/article/pii/S0921889014000682.

[36] Hao Liu, Yongqiang Bai, Geng Lu, and Yisheng Zhong. Robust attitude control of uncertain quadrotors. *IET Control Theory & Applications*, 7(11): 1583–1589, 2013. 4

[37] Heriberto Ramirez-Rodriguez, Vicente Parra-Vega, Anand Sanchez-Orta, and Octavio Garcia-Salazar. Robust backstepping control based on integral sliding modes for tracking of quadrotors. *Journal of Intelligent & Robotic Systems*, 73(1):51–66, Jan 2014. ISSN 1573-0409. doi: 10.1007/s10846-013-9909-4. 4

[38] Ruifeng Zhang, Quan Quan, and K-Y Cai. Attitude control of a quadrotor aircraft subject to a class of time-varying disturbances. *IET control theory & applications*, 5(9):1140–1146, 2011. 4

[39] Erkan Kayacan. Sliding mode control for systems with mismatched time-varying uncertainties via a self-learning disturbance observer. *Transactions of the Institute of Measurement and Control*, 41(7):2039–2052, 2019. doi: 10.1177/0142331218794266. 4

[40] Shouzhao Sheng, Chenwu Sun, and Hong Zhao. Indirect adaptive attitude control for a ducted fan vertical takeoff and landing microaerial vehicle. *Mathematical Problems in Engineering*, 2015, 2015. 4, 65

[41] Z. T. Dydek, A. M. Annaswamy, and E. Lavretsky. Adaptive control of quadrotor uavs: A design trade study with flight evaluations. *IEEE Transactions on Control Systems Technology*, 21(4):1400–1406, July 2013. doi: 10.1109/TCST.2012.2200104. 4, 65

[42] Girish Chowdhary, Maximilian Mühlegg, Jonathan P. How, and Florian Holzapfel. *Concurrent learning adaptive model predictive control*, pages 29–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38253-6. doi: 10.1007/978-3-642-38253-6_3. URL `https://doi.org/10.1007/978-3-642-38253-6_3`. 4

[43] G. Chowdhary, M. Mühlegg, J. P. How, and F. Holzapfel. A concurrent learning adaptive-optimal control architecture for nonlinear systems. In *52nd IEEE Conference on Decision and Control*, pages 868–873, Dec 2013. doi: 10.1109/CDC.2013.6759991. 4

[44] A. Ataka, H. Tnunay, R. Inovan, M. Abdurrohman, H. Preastianto, A. I. Cahyadi, and Y. Yamamoto. Controllability and observability analysis of the gain scheduling based linearization for UAV quadrotor. In *2013 International Conference on Robotics, Biomimetics, Intelligent Computational Systems*, pages 212–218, Nov 2013. doi: 10.1109/ROBIONETICS.2013.6743606. 4

[45] Kostas Alexis, George Nikolakopoulos, and Anthony Tzes. Switching model predictive attitude control for a quadrotor helicopter subject to atmospheric disturbances. *Control Engineering Practice*, 19(10):1195 – 1207, 2011. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2011.06.010. URL `http://www.sciencedirect.com/science/article/pii/S0967066111001262`. 4

[46] G. Lai, Z. Liu, Y. Zhang, and C. L. P. Chen. Adaptive position/attitude tracking control of aerial robot with unknown inertial matrix based on a new robust neural identifier. *IEEE Transactions on Neural Networks and Learning Systems*, 27(1):18–31, Jan 2016. doi: 10.1109/TNNLS.2015.2406812. 5

[47] T. Dierks and S. Jagannathan. Output feedback control of a quadrotor UAV using neural networks. *IEEE Transactions on Neural Networks*, 21(1):50–66, Jan 2010. ISSN 1045-9227. doi: 10.1109/TNN.2009.2034145. 5

[48] C. Nicol, C. J. B. Macnab, and A. Ramirez-Serrano. Robust neural network control of a quadrotor helicopter. In *2008 Canadian Conference on Electrical*

and Computer Engineering, pages 001233–001238, May 2008. doi: 10.1109/CCECE.2008.4564736. 5

[49] Erdal Kayacan, Mojtaba Ahmadieh Khanesar, Jaime Rubio-Hervas, and Mahmut Reyhanoglu. Learning control of fixed-wing unmanned aerial vehicles using fuzzy neural networks. *International Journal of Aerospace Engineering*, 2017, 2017. 5

[50] Efe Camci, Devesh Raju Kripalani, Linlu Ma, Erdal Kayacan, and Mojtaba Ahmadieh Khanesar. An aerial robot for rice farm quality inspection with type-2 fuzzy neural networks tuned by particle swarm optimization-sliding mode control hybrid algorithm. *Swarm and Evolutionary Computation*, 2017. ISSN 2210-6502. doi: https://doi.org/10.1016/j.swevo.2017.10.003.

[51] Andriy Sarabakha, Nursultan Imanberdiyev, Erdal Kayacan, Mojtaba Ahmadieh Khanesar, and Hani Hagras. Novel Levenberg-Marquardt based learning algorithm for unmanned aerial vehicles. *Information Sciences*, 417: 361 – 380, 2017. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2017.07.020. 5

[52] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 11

[53] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[54] Philip E Gill, Walter Murray, and Margaret H Wright. *Practical optimization*. Academic press, 1981. 11

[55] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017. doi: 10.1017/9781139061759. 12, 13, 14, 17

[56] Melanie Nicole Zeilinger. *Real-time Model Predictive Control*. PhD dissertation, Eidgenössische Technische Hochschule (ETH) Zürich, 2011. 15, 16

[57] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 2006. 15, 18, 28

[58] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994. 16

[59] Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Hervé Lebret. Applications of second-order cone programming. *Linear algebra and its applications*, 284(1-3):193–228, 1998. 16

[60] B Bank, J Guddat, D Klatte, B Kummer, and K Tammer. Non-linear parametric optimization. *Akademie-Verlag, Berlin*, 1982. 17

[61] Tomas Gal. *Postoptimal analyses, parametric programming and related topics*. Walter de Gruyter, 2 edition, 1995. 17

[62] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. Geometric algorithm for multiparametric linear programming. *Journal of optimization theory and applications*, 118(3):515–540, 2003. 17

[63] M. Diehl, H.G. Bock, H. Diedam, and P.-B. Wieber. *Fast Direct Multiple Shooting Algorithms for Optimal Robot Control*, pages 65–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. 19, 20, 21

[64] R Bellman. Dynamic programming: Princeton univ. press, 1957. 20

[65] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010. 21

[66] M.R Osborne. On shooting methods for boundary value problems. *Journal of Mathematical Analysis and Applications*, 27(2):417 – 433, 1969. 21

[67] Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the IFAC World Congress*, 1984. 21

[68] R Sargent and G Sullivan. The development of an efficient optimal control package. *Proceedings of the 8th IFIP Conference on Optimization Techniques (1977), Part 2 (Heidelberg, 1978)*, page 158 – 168, 1978. 23

[69] J. Albersmeyer and M. Diehl. The lifted newton method and its application in optimization. *SIAM Journal on Optimization*, 20(3):1655–1684, 2010. doi: 10.1137/080724885. URL `https://doi.org/10.1137/080724885`. 23

[70] R. P. Hettich and H.Th. Jongen. *Semi-infinite programming: Conditions of optimality and applications*, pages 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978. 23

[71] Moritz Diehl, H.Georg Bock, Johannes P. Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577 – 585, 2002. ISSN 0959-1524. doi: https://doi.org/10.1016/S0959-1524(01)00023-3. 25

[72] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation*, pages 391–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 26

[73] Michal Kvasnica, Pascal Grieder, Mato Baotić, and Manfred Morari. Multiparametric toolbox (mpt). In *International Workshop on Hybrid Systems: Computation and Control*, pages 448–462. Springer, 2004. 27

[74] Efstratios N Pistikopoulos, Nikolaos A Diangelakis, Richard Oberdieck, Maria M Papathanasiou, Ioana Nascu, and Muxin Sun. PAROC—an integrated framework and software platform for the optimisation and advanced model-based control of process systems. *Chemical Engineering Science*, 136: 115–138, 2015. 27

[75] Alberto Bemporad, Manfred Morari, and N Lawrence Ricker. Model predictive control toolbox user's guide. *The mathworks*, 2010. 27

[76] Pablo Zometa, Markus Kögel, and Rolf Findeisen. μAO-MPC: a free code generation tool for embedded real-time linear model predictive control. In *American Control Conference (ACC), 2013*, pages 5320–5325. IEEE, 2013. 27

[77] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014. 27

[78] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013. 27

[79] Alexander Domahidi and Juan Jerez. FORCES professional. embotech gmbh (http://embotech. com/forces-pro). *July*, 2014. 27

[80] Jonathan Currie and David I Wilson. Lightweight model predictive control intended for embedded applications. *IFAC Proceedings Volumes*, 43(5):278–283, 2010. 27

[81] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010. 27

[82] Stefano Riverso, Alberto Battocchio, and Giancarlo Ferrari-Trecate. *PnPMPC Toolbox v. 1.0-User manual*, 2014. 27

[83] Matthew J Tenny, Stephen J Wright, and James B Rawlings. Nonlinear model predictive control via feasibility-perturbed sequential quadratic programming. *Computational Optimization and Applications*, 28(1):87–121, 2004. 27

[84] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011. 27

[85] M Diehl, DB Leineweber, and Schäfer AAS. Muscod-ii users' manual'. preprint 2001-25. *Interdisciplinary Center for Scientific Computing, University of Heidelberg*, 2001. 28

[86] Zoltan K Nagy. Optcon-an efficient tool for rapid prototyping of nonlinear model predictive control applications. In *Proceedings of the AIChE Annual Meeting*, pages 16–21, 2008. 28

[87] Anthony Kelman, Sergey Vichik, and Francesco Borrelli. BLOM: The berkeley library for optimization modeling and nonlinear model predictive control. *Berkeley, CA, http://www. mpclab. net/Trac*, 2012. 28

[88] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006. 28

[89] Per E Rutquist and Marcus M Edvall. Propt-matlab optimal control software. *Tomlab Optimization Inc*, 260(1), 2010. 28

[90] Paola Falugi, Eric Kerrigan, and Eugene Van Wyk. Imperial college london optimal control software user guide (iclocs). *Department of Electrical and Electronic Engineering, Imperial College London, London, England, UK*, 2010. 28

[91] Jacob Mattingley and Stephen Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012. 28

[92] Alexander Domahidi, Eric Chu, and Stephen Boyd. ECOS: An socp solver for embedded systems. In *2013 European Control Conference (ECC)*, pages 3071–3076. IEEE, 2013. 28

[93] D. Ariens, B. Houska, H. Ferreau, and F. Logist. *ACADO: Toolkit for Automatic Control and Dynamic Optimization*. Optimization in Engineering Center (OPTEC), 1.0beta edition, 2010. `http://www.acadotoolkit.org/`. 28

[94] D. Ariens, B. Houska, H. Ferreau, and F. Logist. *ACADO for Matlab User's Manual*. Optimization in Engineering Center (OPTEC), 1.0beta edition, May 2010. `http://www.acadotoolkit.org/`. 28

[95] Samir Bouabdallah. *Design and control of quadrotors with application to autonomous flying*. PhD dissertation, EPFL, 2007. 30

[96] S. Bezzaoucha, H. Voos, and M. Darouach. A polytopic approach for the nonlinear unknown input functional observers design: Application to a quadrotor aerial robots landing. In *2016 European Control Conference (ECC)*, pages 1146–1152, June 2016. doi: 10.1109/ECC.2016.7810444. 34

[97] M. Tognon and A. Franchi. Nonlinear observer for the control of bi-tethered multi aerial robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1852–1857, Sep. 2015. doi: 10.1109/IROS.2015.7353619. 34

[98] Fuyang Chen, Wen Lei, Kangkang Zhang, Gang Tao, and Bin Jiang. A novel nonlinear resilient control for a quadrotor UAV via backstepping control and nonlinear disturbance observer. *Nonlinear Dynamics*, 85(2):1281–1295, 2016. 34

[99] M. Rida Mokhtari, Brahim Cherki, and Amal Choukchou Braham. Disturbance observer based hierarchical control of coaxial-rotor UAV. *ISA Transactions*, 67:466 – 475, 2017. ISSN 0019-0578. doi: https://doi.org/10.1016/j.isatra.2017.01.020. URL `http://www.sciencedirect.com/science/article/pii/S0019057817301283`. 34

[100] Jean Smith, Jinya Su, Cunjia Liu, and Wen-Hua Chen. Disturbance observer based control with anti-windup applied to a small fixed wing UAV for disturbance rejection. *Journal of Intelligent & Robotic Systems*, 88(2-4):329–346, 2017. 34

[101] Xiaobo Lin, Yao Yu, and Chang-yin Sun. A decoupling control for quadrotor uav using dynamic surface control and sliding mode disturbance observer. *Nonlinear Dynamics*, 97(1):781–795, 2019. 34

[102] P. Bouffard, A. Aswani, and C. Tomlin. Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 279–284, May 2012. doi: 10.1109/ICRA.2012.6225035. 34

[103] Basaran Bahadir Kocer, Tegoeh Tjahjowidodo, and Gerald Gim Lee Seet. Centralized predictive ceiling interaction control of quadrotor vtol uav. *Aerospace Science and Technology*, 76:455 – 465, 2018. ISSN 1270-9638. doi: https://doi.org/10.1016/j.ast.2018.02.020. URL `http://www.sciencedirect.com/science/article/pii/S1270963817315572`. 35

[104] Mohit Mehndiratta and Erdal Kayacan. Receding horizon control of a 3 DOF helicopter using online estimation of aerodynamic parameters. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 2017. doi: 10.1177/0954410017703414. 35

[105] M. Mehndiratta, E. Kayacan, M. Reyhanoglu, and E. Kayacan. Robust tracking control of aerial robots via a simple learning strategy-based feedback linearization. *IEEE Access*, 8:1653–1669, 2020. 35

[106] M. Mehndiratta, E. Kayacan, and E. Kayacan. A simple learning strategy for feedback linearization control of aerial package delivery robot. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 361–367, Aug 2018. doi: 10.1109/CCTA.2018.8511485. 35

[107] T. Kraus, H.J. Ferreau, E. Kayacan, H. Ramon, J. De Baerdemaeker, M. Diehl, and W. Saeys. Moving horizon estimation and nonlinear model predictive control for autonomous agricultural vehicles. *Computers and Electronics in Agriculture*, 98:25 – 33, 2013. 36

[108] H. CHEN and F. Allgöwer. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205 – 1217, 1998. ISSN 0005-1098. doi: https://doi.org/10.1016/S0005-1098(98)00073-9. URL `http://www.sciencedirect.com/science/article/pii/S0005109898000739`. 37

[109] Lars Grúne. NMPC without terminal constraints. *IFAC Proceedings Volumes*, 45(17):1–13, 2012. 37

[110] G. De Nicolao, L. Magni, and R. Scattolini. Stabilizing receding-horizon control of nonlinear time-varying systems. *IEEE Transactions on Automatic Control*, 43(7):1030–1036, Jul 1998. 37

[111] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789 – 814, 2000. 37

[112] Moritz Diehl, Ilknur Uslu, Rolf Findeisen, Stefan Schwarzkopf, Frank Allgöwer, H. Georg Bock, Tobias Bürner, Ernst Dieter Gilles, Achim Kienle, Johannes P. Schlöder, and Erik Stein. *Real-Time optimization for large scale processes: Nonlinear model predictive control of a high purity distillation column*, pages 363–383. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-662-04331-8. doi: 10.1007/978-3-662-04331-8_20. URL `https://doi.org/10.1007/978-3-662-04331-8_20`. 37

[113] Matthias A. Müller. Nonlinear moving horizon estimation in the presence of bounded disturbances. *Automatica*, 79:306 – 314, 2017. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2017.01.033. URL `https://www.sciencedirect.com/science/article/pii/S0005109817300432`. 38

[114] Boyang Li, Weifeng Zhou, Jingxuan Sun, Chihyung Wen, and Chihkeng Chen. Model predictive control for path tracking of a VTOL tailsitter UAV in an HIL simulation environment. In *2018 AIAA Modeling and Simulation Technologies Conference*, page 1919. American Institute of Aeronautics and Astronautics, 8-12 January 2018. 40

[115] Erkan Kayacan and Thor I. Fossen. Feedback linearization control for systems with mismatched uncertainties via disturbance observers. *Asian Journal of Control*, 21(4):1–13, 2019. doi: 10.1002/asjc.1802. 64, 68

[116] José de Jesús Rubio. Robust feedback linearization for nonlinear processes control. *ISA Transactions*, 74:155 – 164, 2018. ISSN 0019-0578. doi: https://doi.org/10.1016/j.isatra.2018.01.017. URL `http://www.sciencedirect.com/science/article/pii/S001905781830017X`. 64

[117] Savaş Şahin. Learning feedback linearization using artificial neural networks. *Neural Processing Letters*, 44(3):625–637, Dec 2016. ISSN 1573-773X. doi: 10.1007/s11063-015-9484-8. URL `https://doi.org/10.1007/s11063-015-9484-8`. 64

[118] J. Umlauft, T. Beckers, M. Kimmel, and S. Hirche. Feedback linearization using gaussian processes. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5249–5255, Dec 2017. doi: 10.1109/CDC.2017.8264435. 64

[119] Zhongsheng Hou and Shangtai Jin. *Model free adaptive control: theory and applications*. CRC press, 2013. 64

[120] Ronghu Chi, Zhongsheng Hou, Biao Huang, and Shangtai Jin. A unified data-driven design framework of optimality-based generalized iterative learning control. *Computers & Chemical Engineering*, 77:10 – 23, 2015. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2015.03.003.

[121] Y. Zhu and Z. Hou. Data-driven mfac for a class of discrete-time nonlinear systems with rbfnn. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):1013–1020, May 2014. ISSN 2162-237X. doi: 10.1109/TNNLS.2013.2291792.

[122] Z. Hou and Y. Zhu. Controller-dynamic-linearization-based model free adaptive control for discrete-time nonlinear systems. *IEEE Transactions on Industrial Informatics*, 9(4):2301–2309, Nov 2013. ISSN 1551-3203. doi: 10.1109/TII.2013.2257806.

[123] Z. Hou, R. Chi, and H. Gao. An overview of dynamic-linearization-based data-driven control and applications. *IEEE Transactions on Industrial Electronics*, 64(5):4076–4090, May 2017. ISSN 0278-0046. doi: 10.1109/TIE.2016.2636126. 64

[124] Jimoh O. Pedro, Muhammed Dangor, Olurotimi A. Dahunsi, and M. Montaz Ali. Intelligent feedback linearization control of nonlinear electrohydraulic suspension systems using particle swarm optimization. *Applied Soft Computing*, 24:50 – 62, 2014. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2014.05.013. URL `http://www.sciencedirect.com/science/article/pii/S1568494614002403`. 64

[125] Daewon Lee, H Jin Kim, and Shankar Sastry. Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. *International Journal of control, Automation and systems*, 7(3):419–428, 2009. 65

[126] M. Huang, B. Xian, C. Diao, K. Yang, and Y. Feng. Adaptive tracking control of underactuated quadrotor unmanned aerial vehicles via backstepping. In *Proceedings of the 2010 American Control Conference*, pages 2076–2081, 2010. 65

[127] Wen-Hua Chen. Disturbance observer based control for nonlinear systems. *IEEE/ASME transactions on mechatronics*, 9(4):706–710, 2004. 70

[128] W. Langson and A. Alleyne. A stability result with application to nonlinear regulation: theory and experiments. In *Proceedings of the 1999 American*

Control Conference (Cat. No. 99CH36251), volume 5, pages 3051–3056, June 1999. doi: 10.1109/ACC.1999.782322. 71

[129] Dong-Ho Shin and Youdan Kim. Reconfigurable flight control system design using adaptive neural networks. *IEEE Transactions on Control Systems Technology*, 12(1):87–100, Jan 2004. doi: 10.1109/TCST.2003.821957. 92

[130] 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)*, pages 4653–4660, Dec 2016. 92

[131] Siddharth Patel, Andriy Sarabakha, Dogan Kircali, and Erdal Kayacan. An intelligent hybrid artificial neural network-based approach for control of aerial robots. *Journal of Intelligent & Robotic Systems*, May 2019. ISSN 1573-0409. doi: 10.1007/s10846-019-01031-z. 92

[132] Jongho Shin, H. Jin Kim, Sewook Park, and Youdan Kim. Model predictive flight control using adaptive support vector regression. *Neurocomputing*, 73(4):1031 – 1037, 2010. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2009.10.002. Bayesian Networks / Design and Application of Neural Networks and Intelligent Learning Systems (KES 2008 / Bio-inspired Computing: Theories and Applications (BIC-TA 2007). 93

[133] Chris J. Ostafew, Angela P. Schoellig, Timothy D. Barfoot, and Jack Collier. Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking. *Journal of Field Robotics*, 33(1):133–152, 2016. ISSN 1556-4967. doi: 10.1002/rob.21587. 93

[134] G. Cao, E. M. K. Lai, and F. Alam. Gaussian process model predictive control of unmanned quadrotors. In *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, pages 200–206, April 2016. doi: 10.1109/ICCAR.2016.7486726. 93

[135] CE Rasmussen and C Williams. Gaussian processes for machine learning the mit press, 2006. 95, 97, 98

[136] Agathe Girard, Carl Edward Rasmussen, Joaquin Quinonero Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. In *Advances in neural information processing systems*, pages 545–552, 2003. 98

[137] Marc Peter Deisenroth. *Efficient Reinforcement Learning Using Gaussian Processes*. PhD thesis, KIT Scientific Publishing, Karlsruhe, 2010. 99

[138] M. Mehndiratta, E. Camci, and E. Kayacan. Automated tuning of nonlinear model predictive controller by reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018. 103, 111

[139] Windfinder. wind data in aalborg, 2019. URL `https://www.windfinder.com/windstatistics/aalborg`. [accessed 1-July-2019]. 106

[140] J.H. Lee and Z.H. Yu. Tuning of model predictive controllers for robust performance. *Computers & Chemical Engineering*, 18(1):15 – 37, 1994. ISSN 0098-1354. doi: https://doi.org/10.1016/0098-1354(94)85020-8. An International Journal of Computer Applications in Chemical Engineering. 112

[141] Rahul Shridhar and Douglas J. Cooper. A tuning strategy for unconstrained multivariable model predictive control. *Industrial & Engineering Chemistry Research*, 37(10):4003–4016, 1998. doi: 10.1021/ie980202s. 112

[142] S. Di Cairano and A. Bemporad. Model predictive control tuning by controller matching. *IEEE Transactions on Automatic Control*, 55(1):185–190, Jan 2010. ISSN 0018-9286. doi: 10.1109/TAC.2009.2033838. 112

[143] Achin Jain, Georg Schildbach, Lorenzo Fagiano, and Manfred Morari. On the design and tuning of linear model predictive control for wind turbines. *Renewable Energy*, 80:664 – 673, 2015. ISSN 0960-1481. doi: https://doi.org/10.1016/j.renene.2015.02.057. 112

[144] Emad Ali and Evanghelos Zafiriou. Optimization-based tuning of nonlinear model predictive control with state estimation. *Journal of Process Control*, 3(2):97 – 107, 1993. ISSN 0959-1524. doi: https://doi.org/10.1016/0959-1524(93)80005-V. 112

[145] Ashraf Al-Ghazzawi, Emad Ali, Adnan Nouh, and Evanghelos Zafiriou. On-line tuning strategy for model predictive controllers. *Journal of Process Control*, 11(3):265 – 284, 2001. ISSN 0959-1524. doi: https://doi.org/10.1016/S0959-1524(00)00033-0. 113

[146] Emad Ali. Heuristic on-line tuning for nonlinear model predictive controllers using fuzzy logic. *Journal of Process Control*, 13(5):383 – 396, 2003. ISSN 0959-1524. doi: https://doi.org/10.1016/S0959-1524(02)00064-1. 113

[147] K. M. Cabral, S. R. B. dos Santos, S. N. Givigi, and C. L. Nascimento. Design of model predictive control via learning automata for a single UAV load transportation. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–7, April 2017. doi: 10.1109/SYSCON.2017.7934800. 113

[148] P. T. Jardine, S. N. Givigi, and S. Yousefi. Experimental results for autonomous model-predictive trajectory planning tuned with machine learning. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–7, April 2017. doi: 10.1109/SYSCON.2017.7934801. 113

[149] William J. Shipman and Loutjie C. Coetzee. Reinforcement learning and deep neural networks for PI controller tuning. *IFAC-PapersOnLine*, 52(14):111 – 116, 2019. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2019.09.173. 18th IFAC Symposium on Control, Optimization and Automation in Mining, Mineral and Metal Processing, MMM 2019. 113

[150] Panagiotis Kofinas and Anastasios I. Dounis. Fuzzy q-learning agent for online tuning of PID controller for dc motor speed control. *Algorithms*, 11 (10):148, Sep 2018. ISSN 1999-4893. doi: 10.3390/a11100148.

[151] Jaime Junell, Tommaso Mannucci, Ye Zhou, and Erik-Jan Van Kampen. Self-tuning gains of a quadrotor using a simple model for policy gradient reinforcement learning. In *AIAA Guidance, Navigation, and Control Conference*. doi: 10.2514/6.2016-1387.

[152] M.N Howell and M.C Best. On-line PID tuning for engine idle-speed control using continuous action reinforcement learning automata. *Control Engineering Practice*, 8(2):147 – 154, 2000. ISSN 0967-0661. doi: https://doi.org/10.1016/S0967-0661(99)00141-0. 113

[153] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. 115, 117

[154] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 116

[155] Helfried Peyrl, Alessandro Zanarini, Thomas Besselmann, Junyi Liu, and Marc-Alexandre Boéchat. Parallel implementations of the fast gradient method for high-speed mpc. *Control Engineering Practice*, 33:22 – 34, 2014. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2014.08.010. URL `http://www.sciencedirect.com/science/article/pii/S0967066114002093`. 134

[156] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded predictive control on an fpga using the fast gradient method. In *2013 European Control Conference (ECC)*, pages 3614–3620, July 2013. doi: 10.23919/ECC.2013.6669598. 134

[157] Alexander Joos and Walter Fichter. Parallel implementation of constrained nonlinear model predictive controller for an fpga-based onboard flight computer. In Florian Holzapfel and Stephan Theil, editors, *Advances in Aerospace Guidance, Navigation and Control*, pages 273–286, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19817-5. 134