

MULTI-STEP NEURAL NETWORK BASED SYSTEM IDENTIFICATION AND  
NMPC FOR AN UNDERACTUATED UNMANNED SURFACE VEHICLE

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES  
OF  
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA YILDIRIM

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR  
THE DEGREE OF MASTER OF SCIENCE  
IN  
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2026



Approval of the thesis:

**MULTI-STEP NEURAL NETWORK BASED SYSTEM IDENTIFICATION  
AND NMPC FOR AN UNDERACTUATED UNMANNED SURFACE  
VEHICLE**

submitted by **MUSTAFA YILDIRIM** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Naci Emre Altun  
Dean, Graduate School of **Natural and Applied Sciences** \_\_\_\_\_

Prof. Dr. Nevzat Güneri Gençer  
Head of Department, **Electrical and Electronics Engineering** \_\_\_\_\_

Assoc. Prof. Dr. Mustafa Mert Ankaralı  
Supervisor, **Electrical and Electronics Engineering, METU** \_\_\_\_\_

**Examining Committee Members:**

Assist. Prof. Dr. Kenan Ahıska  
Electrical and Electronics Engineering, METU \_\_\_\_\_

Assoc. Prof. Dr. Mustafa Mert Ankaralı  
Electrical and Electronics Engineering, METU \_\_\_\_\_

Assoc. Prof. Dr. İsmail Uyanık  
Electrical and Electronics Engineering, Hacettepe Üniversitesi \_\_\_\_\_

Date:22.01.2026

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Surname: Mustafa Yıldırım

Signature :

## ABSTRACT

### **MULTI-STEP NEURAL NETWORK BASED SYSTEM IDENTIFICATION AND NMPC FOR AN UNDERACTUATED UNMANNED SURFACE VEHICLE**

Yıldırım, Mustafa

M.S., Department of Electrical and Electronics Engineering

Supervisor: Assoc. Prof. Dr. Mustafa Mert Ankaralı

January 2026, 91 pages

This thesis focuses on learning a dynamics model for an underactuated 3-DOF unmanned surface vehicle (USV) using neural networks and deploying the learned model inside nonlinear model predictive control (NMPC). Physics-based maneuvering models provide useful structure, but accuracy can be affected by parameter uncertainty, noisy measurements, and actuator behavior.

In real operation, the USV dynamics depend on current velocity and the actual thrust, while the available signals are noisy velocity measurements and thruster commands that differ from delivered thrust because of nonlinearities and delays. To reduce this mismatch, the neural identifiers use a short history of body-frame velocities together with a short history of thruster commands, enabling the networks to implicitly estimate effective thrust and handle noise.

Three predictors are emphasized and compared: (i) a conventional 3-DOF Fossen maneuvering model, (ii) a multi-step MLP that outputs the full prediction horizon in a single forward pass, and (iii) a sequence-to-sequence (encoder-decoder) LSTM with the same horizon-length output. The neural models are trained on trajectory segments

whose length matches the NMPC horizon to limit drift in multi-step predictions.

Data are generated and the controllers are tested in a Gazebo simulation environment using a Clearpath Robotics Heron USV model. The NMPC formulation is designed to accept either the learned model or the physics-based model and is evaluated on body-frame velocity tracking and position/trajectory tracking. Results indicate that multi-step neural predictors can provide accurate horizon predictions and support real-time NMPC.

Keywords: System Identification, Model Predictive Control, Optimization, Neural Networks

## ÖZ

### **KISMİ TAHRİKLİ İNSANSIZ SU ÜSTÜ ARACI İÇİN ÇOK ADIMLI YAPAY SİNİR AĞI TABANLI SİSTEM TANIMLAMASI VE LİNEER OLMAYAN MODEL ÖNGÖRÜLÜ KONTROLCU TASARIMI**

Yıldırım, Mustafa

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Mustafa Mert Ankaralı

Ocak 2026 , 91 sayfa

Bu tez, eksik tahrikli 3-DOF insansız su üstü aracı (USV) için sinir ağları kullanarak pratik bir dinamik model öğrenmeye ve öğrenilen modeli doğrusal olmayan model öngörülü kontrol (NMPC) içinde kullanmaya odaklanmaktadır. Fizik tabanlı manevra modelleri yararlı bir kuramsal yapı sunsa da, doğruluk parametre belirsizliği, gürültü ölçümler ve eyleyici (aktüatör) davranışı nedeniyle pratikte etkilenebilir.

Gerçek işletimde dinamikler anlık hıza ve gerçek itkiye bağlıdır; ancak eldeki sinyaller çoğunlukla gürültülü hız ölçümleri ve doğrusal olmayanlıklar ile gecikmeler nedeniyle üretilen itkiye eşit olmayan itici komutlarıdır. Bu uyumsuzluğu azaltmak için sinir ağı tanımlayıcıları, gövde-çerçevesi hızlarının kısa bir geçmişini ve itici komutlarının kısa bir geçmişini birlikte girdi olarak kullanır; böylece ağlar etkin (effective) itkiyi örtük olarak tahmin edebilir ve gürültüye daha dayanıklı hale gelir.

Üç kestirici vurgulanmakta ve karşılaştırılmaktadır: (i) geleneksel 3-DOF Fossen manevra modeli, (ii) tüm öngörü ufku tek bir ileri geçişte çıktıl原因 çok-adımlı bir

MLP ve (iii) aynı ufuk uzunluğunda çıktı üreten sequence-to-sequence (encoder–decoder) bir LSTM. Sinir ağı modelleri, çok-adımlı öngörülerde sapmayı sınırlamak için NMPC ufku ile aynı uzunlukta yörünge kesitleri üzerinde eğitilmiştir.

Veriler Gazebo benzetim ortamında üretilmiş ve denetleyiciler Clearpath Robotics Heron USV modeli kullanılarak aynı ortamda test edilmiştir. NMPC formülasyonu, hem öğrenilen modeli hem de fizik tabanlı modeli kullanabilecek şekilde tasarlanmış ve gövde-çerçevesi hız izleme ile konum/yörünge izleme görevlerinde değerlendirilmiştir. Sonuçlar, çok-adımlı sinir ağlarının ufuk boyunca doğru kestirimler sağlayabildiğini ve gerçek zamanlı NMPC uygulamalarını destekleyebildiğini göstermektedir.

**Anahtar Kelimeler:** Sistem Tanımlama, Model Öngörülü Kontrol, Optimizasyon, Yapay Sinir Ağları

To my beloved family...

## ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Mustafa Mert Ankaralı, for his guidance, encouragement, and patience throughout my M.Sc. studies. His technical insight and trust in my work shaped both the direction and the quality of this thesis.

This thesis was partially supported by the TÜBİTAK 1001 program under project 122E249. I gratefully acknowledge this support, which provided the resources needed to pursue and complete this research.

I would like to thank my colleague Eminaalp Koyuncu for his contributions related to the Heron USV simulator and for his help in testing the methods proposed in this thesis. Our discussions and his practical support were valuable during the implementation and evaluation stages.

I am also thankful to my colleagues and friends for contributing to a positive working environment throughout this journey. In particular, I would like to thank Selin Ezgi Özcan for her helpful feedback and support, especially during the comparison of the methods used in this study.

I would like to thank my colleague Kemal Atacan Mucukgil for being with me during the writing process and for his steady encouragement throughout this period.

Above all, I owe my deepest thanks to my family. I would like to thank my sister, Merve Yıldırım, and my parents, Semahat Yıldırım and Ramazan Yıldırım, for their constant love, patience, and sacrifices throughout my education. Their belief in me has been my greatest motivation, and I would not have reached this point without their support and understanding.

## TABLE OF CONTENTS

ABSTRACT . . . . .	v
ÖZ . . . . .	vii
ACKNOWLEDGMENTS . . . . .	x
TABLE OF CONTENTS . . . . .	xi
LIST OF TABLES . . . . .	xiv
LIST OF FIGURES . . . . .	xvi
LIST OF ABBREVIATIONS . . . . .	xx
CHAPTERS	
1 INTRODUCTION . . . . .	1
1.1 Motivation of this Thesis . . . . .	1
1.2 Literature Review . . . . .	2
2 PRELIMINARIES AND BACKGROUND . . . . .	5
2.1 USV Dynamics . . . . .	5
2.2 Neural Network Architectures for System Identification . . . . .	7
2.2.1 Multi-Layer Perceptron (MLP) . . . . .	8
2.2.2 Long short term Memory (LSTM) . . . . .	9
2.2.3 Nonlinear Model Predictive Control (NMPC) . . . . .	10
2.2.4 General NMPC Formulation . . . . .	10

2.2.5	Real-Time NMPC using Acados SQP_RTI solver . . . . .	11
3	METHODOLOGY . . . . .	13
3.1	Methodology Overview . . . . .	13
3.2	System Identification . . . . .	14
3.2.1	Identifying Fossen Parameters for the USV . . . . .	14
3.2.2	Identifying USV Dynamics via MS-MLP . . . . .	16
3.2.2.1	Data Processing and Input Structure . . . . .	16
3.2.2.2	Network Architecture . . . . .	17
3.2.3	Training Implementation . . . . .	18
3.2.4	Identifying USV Dynamics via LSTM . . . . .	18
3.2.4.1	Encoder Stage (State Estimation) . . . . .	19
3.2.4.2	Decoder Stage and MLP Head (Trajectory Simulation) . . . . .	20
3.3	NMPC Formulation for Velocity Control . . . . .	20
3.3.1	Problem Statement and Notation . . . . .	20
3.3.2	Modifying the NN Models for Velocity Control . . . . .	21
3.3.3	Fossen NMPC Model for Velocity Control . . . . .	22
3.3.4	Seq2Seq NMPC Model for Velocity Control . . . . .	22
3.3.5	MS-MLP NMPC Model for Velocity Control . . . . .	23
3.4	NMPC Formulation for Position Control . . . . .	24
3.4.1	Pose propagation used in the horizon rollout . . . . .	26
3.4.2	NMPC objective with stage and terminal outputs . . . . .	26
3.4.3	Model-dependent NMPC state and transition . . . . .	27
4	IMPLEMENTATION AND RESULTS . . . . .	31

4.1	Simulation Setup and Data Acquisition . . . . .	31
4.2	Data Acquisition for System Identification . . . . .	34
4.3	Fossen System Identification . . . . .	36
4.4	Neural Network System Identification Results . . . . .	39
4.4.1	Data Synchronization and Normalization . . . . .	39
4.4.2	Training Setup . . . . .	40
4.4.3	Comparison and Architecture Selection . . . . .	40
4.4.4	Discussions . . . . .	47
4.5	Velocity Control Results . . . . .	48
4.5.1	Yaw-Rate Tracking . . . . .	48
4.5.2	Surge Tracking (Noiseless) . . . . .	53
4.5.3	Velocity Tracking with Noisy Measurements . . . . .	57
4.5.4	Discussions . . . . .	62
4.6	Position Control Results . . . . .	64
4.6.1	Star-Shaped Map . . . . .	64
4.6.2	S-Shaped Map . . . . .	69
4.7	Discussions . . . . .	76
5	CONCLUSIONS . . . . .	79
	REFERENCES . . . . .	81
	APPENDICES . . . . .	87
A	NMPC formulation and acados solution method (SQP_RTI + full condensing) . . . . .	87

## LIST OF TABLES

### TABLES

Table 4.1 Mapping between normalized propulsor input on <code>/cmd_drive</code> and the corresponding thrust output used by the simulator[11]. . . . .	33
Table 4.2 Identified Fossen discrete-time model parameters. . . . .	38
Table 4.3 Training data fit percentages for the nonlinear gray-box Fossen model. The overall mean squared error (MSE), computed jointly over surge, sway, and yaw dynamics, is 0.007278. . . . .	38
Table 4.4 Physical limits used for normalization of velocities and thruster forces.	39
Table 4.5 Physical validation MSE for the <b>selected</b> MS-MLP and Seq2Seq LSTM architectures, compared against the recursive Fossen baseline (same validation dataset and horizon-wise metric). . . . .	43
Table 4.6 NMPC computation time for noiseless yaw-rate tracking ( <b>ms</b> ): mean and maximum computation time per control update, including NMPC preparation (state/reference/bounds update and warm start; for Seq2Seq the encoder pass) and the SQP_RTI solve time. . . . .	53
Table 4.7 Surge-velocity tracking performance under noisy measurements ( $H = 20, 2.0$ s): fit percentage $\text{Fit}_u$ using Eq. (4.8), computed between the reference $u_{\text{ref}}$ and the actual (noise-free) surge velocity $u$ over $t \geq 2$ s (i.e., $\mathcal{K} = \{k \mid t_k \geq 2 \text{ s}\}$ ). . . . .	60

Table 4.8 Yaw-rate tracking performance under noisy measurements ( $H = 20, 2.0$ s): fit percentage $\text{Fit}_r$ using Eq. (4.8), computed between the yaw-rate reference $r_{\text{ref}}$ and the actual (noise-free) yaw rate $r$ over the full simulation interval. . . . .	62
Table 4.9 Star-shaped map: total mission completion time (time to finish all waypoints) for each prediction model. . . . .	69
Table 4.10 NMPC computation time for star-shaped map position control (ms): mean and maximum computation time per control update, including NMPC preparation and SQP_RTI solve time, for the Fossen, MS-MLP, and Seq2Seq prediction models. . . . .	69
Table 4.11 S-shaped map: tracking-error statistics computed from the distance between the closed-loop position $(x, y)$ and the closest point on the reference track. The fit metric is computed using (4.8). . . . .	76

## LIST OF FIGURES

### FIGURES

Figure 2.1	A simple representation of an Unmanned Surface Vessel . . . . .	5
Figure 2.2	Basic structure of an MLP. . . . .	8
Figure 2.3	LSTM Schematic . . . . .	9
Figure 3.1	Architecture of the MS-MLP based system identification model with residual learning. . . . .	16
Figure 3.2	Architecture of the LSTM-based Sequence-to-Sequence system identification model. The Encoder captures historical dynamics, while the Decoder recursively predicts future velocities based on planned thrusts. . . . .	19
Figure 4.1	Gazebo Simulator for the Heron USV . . . . .	32
Figure 4.2	Host–Docked communication diagram for ROS-based simula- tion control. . . . .	34
Figure 4.3	Training data snapshot over a time interval obtained using the event-based thruster command sequence. . . . .	35
Figure 4.4	Validation data snapshot over a time interval obtained using an independently randomized event-based command sequence. . . . .	36
Figure 4.5	Validation physical MSE for MS-MLP and Seq2Seq LSTM mod- els with a prediction horizon of $H = 5$ steps (0.5 s). . . . .	41
Figure 4.6	Validation physical MSE for MS-MLP and Seq2Seq LSTM mod- els with a prediction horizon of $H = 12$ steps (1.2 s). . . . .	42

Figure 4.7	Validation physical MSE for MS-MLP and Seq2Seq LSTM models with a prediction horizon of $H = 20$ steps (2.0 s).	42
Figure 4.8	Horizon-wise RMSE accumulation on the validation dataset for $H = 5$ .	44
Figure 4.9	Example validation segment: ground-truth vs. predicted velocity trajectories for $H = 5$ .	44
Figure 4.10	Horizon-wise RMSE accumulation on the validation dataset for $H = 12$ .	45
Figure 4.11	Example validation segment: ground-truth vs. predicted velocity trajectories for $H = 12$ .	45
Figure 4.12	Horizon-wise RMSE accumulation on the validation dataset for $H = 20$ .	46
Figure 4.13	Example validation segment: ground-truth vs. predicted velocity trajectories for $H = 20$ .	46
Figure 4.14	Yaw-rate tracking NMPC results without measurement noise for horizon $H = 5$ (0.5 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.	50
Figure 4.15	Yaw-rate tracking NMPC results without measurement noise for horizon $H = 12$ (1.2 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.	51
Figure 4.16	Yaw-rate tracking NMPC results without measurement noise for horizon $H = 20$ (2.0 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.	52
Figure 4.17	Surge-tracking NMPC results without measurement noise for horizon $H = 5$ (0.5 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.	54

Figure 4.18	Surge-tracking NMPC results without measurement noise for horizon $H = 12$ (1.2 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models. . . . .	55
Figure 4.19	Surge-tracking NMPC results without measurement noise for horizon $H = 20$ (2.0 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models. . . . .	56
Figure 4.20	Velocity NMPC with noisy measurements for horizon $H = 20$ (2.0 s): surge-focused test comparing Fossen-, MS-MLP-, and Seq2Seq-based prediction models. . . . .	59
Figure 4.21	Surge ( $u$ ) velocity tracking under noisy measurements for horizon $H = 20$ (2.0 s): actual (noise-free) surge response from the same experiment as Fig. 4.20, compared against the reference for Fossen-, MS-MLP-, and Seq2Seq-based NMPC. . . . .	60
Figure 4.22	Velocity NMPC with noisy measurements for horizon $H = 20$ (2.0 s): yaw-rate-focused test comparing Fossen-, MS-MLP-, and Seq2Seq-based prediction models. . . . .	61
Figure 4.23	Yaw-rate ( $r$ ) velocity tracking under noisy measurements for horizon $H = 20$ (2.0 s): actual (noise-free) yaw-rate response from the same experiment as Fig. 4.22, compared against the reference for Fossen-, MS-MLP-, and Seq2Seq-based NMPC. . . . .	62
Figure 4.24	Star-shaped map position-control snapshots at $t = 0$ and $t = 20$ s.	65
Figure 4.25	Star-shaped map position-control snapshots at $t = 35$ and $t = 45$ s.	66
Figure 4.26	Star-shaped map: closed-loop trajectories for the three prediction models (Fossen, MS-MLP, and Seq2Seq) under position NMPC. . .	67
Figure 4.27	Star-shaped map: Tracking results comparing the three prediction models. . . . .	68
Figure 4.28	S-shape map snapshots at $t = 0$ s and $t = 5$ s. . . . .	71

Figure 4.29	S-shape map snapshots at $t = 7.5$ s and $t = 12.5$ s. . . . .	72
Figure 4.30	S-shape map snapshots at $t = 15$ s and $t = 20$ s. . . . .	73
Figure 4.31	S-shaped map: closed-loop trajectories for the three prediction models (Fossen, MS-MLP, and Seq2Seq) under position NMPC. . . . .	74
Figure 4.32	S-shaped map: position NMPC results comparing the three prediction models. . . . .	75

## LIST OF ABBREVIATIONS

USV	Unmanned Surface Vehicle
DOF	Degree(s) of freedom
NARX	Nonlinear AutoRegressive with eXogenous inputs
NN	Neural Network
MLP	Multi-Layer Perceptron
MS-MLP	Multi-Step Multi-Layer Perceptron
LSTM	Long Short-Term Memory
Seq2Seq	Sequence-to-Sequence
RNN	Recurrent Neural Network
OCP	Optimal Control Problem
MPC	Model Predictive Control
NMPC	Nonlinear Model Predictive Control
AD	Automatic Differentiation
NLS	Nonlinear Least Squares
QP	Quadratic Program
SQP	Sequential Quadratic Programming
SQP_RTI	Sequential Quadratic Programming Real-Time Iteration Scheme
IPM	Interior Point Method
HPIPM	High-Performance Interior Point Method
KKT	Karush-Kuhn-Tucker
ROS	Robot Operating System
MEX	MATLAB Executable (MEX file)
MSE	Mean Squared Error

RMSE	Root Mean Squared Error
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LOS	Line-of-Sight
CPU	Central Processing Unit
GPU	Graphics Processing Unit



# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation of this Thesis

Unmanned Surface Vehicles (USVs) are becoming increasingly important for applications such as environmental monitoring, search and rescue, and surveillance [1]. These missions demand stable and reliable control performance. However, designing controllers for USVs is challenging because their dynamics are both nonlinear and underactuated, and they are affected by uncertain hydrodynamic parameters, external disturbances, and actuator non-idealities [2, 3].

Nonlinear Model Predictive Control (NMPC) is a strong candidate for USVs because it can explicitly handle nonlinear dynamics and input constraints while solving a finite-horizon tracking problem [4]. In practice, NMPC performance depends heavily on the prediction model used inside the optimizer. Physics-based maneuvering models provide a structured and interpretable starting point, but their accuracy can degrade when the model does not capture all relevant effects observed in data. This motivates system identification methods that use data to improve prediction quality, especially in operating regimes where parameter uncertainty and unmodeled dynamics are significant [5].

Neural networks have been widely used for nonlinear system identification in marine and robotic systems because they can approximate complex input–output relationships without requiring explicit hydrodynamic parameterization [6, 7]. When such learned models are used as predictors in NMPC, accurate multi-step prediction becomes critical because the controller optimizes over a horizon rather than a single step [8]. At the same time, the predictor must remain computationally efficient, since

NMPC is solved repeatedly in real time and relies on fast derivative information for efficient optimization [9, 10].

A key computational concern is how the prediction horizon is generated. In a single-step predictor setup, the model is applied recursively across the horizon, which can increase the complexity of the computation and differentiation needed during optimization. In contrast, multi-step predictors can produce the full trajectory more directly, potentially improving computational efficiency while still supporting accurate horizon-based control decisions [8].

In this thesis, we study how the choice of prediction model affects both prediction quality and real-time NMPC performance for an underactuated 3-DOF USV. We compare an identified discrete-time Fossen baseline [2] with two learned multi-step predictors (MS-MLP and Seq2Seq LSTM), and we evaluate their closed-loop performance in velocity and position tracking tasks in the Clearpath Heron simulation environment [11].

## 1.2 Literature Review

Traditional methods for modeling surface marine craft dynamics rely on physics-based maneuvering equations [2, 3]. These models give a useful structure and interpretable parameters, but their practical prediction accuracy can still drop when the identified parameters are uncertain, the measurements are noisy, the experiments do not excite all dynamics well, or the actuator behavior is not well represented by the simplified input model [3, 5, 12]. Because of this gap between an idealized model and real behavior, many works either use grey-box identification to tune physics parameters from data or move toward data-driven identification when matching measured trajectories is the main goal [3].

Neural networks have a long history in nonlinear system identification, often in nonlinear autoregressive-with-exogenous-input (NARX) settings where a finite history of states/outputs and inputs is used as the regressor [13]. Recurrent architectures extend this idea by keeping an internal memory, and LSTM networks [14] are a standard choice when longer temporal dependencies or delayed effects are important [15].

From a system identification viewpoint, deep learning is a flexible tool for nonlinear dynamics [16], but it also has well-known practical issues such as sensitivity to training data coverage, extrapolation outside the training distribution, and the effect of noise and regressor design on generalization [5, 17]. In marine and USV-related settings, neural identifiers have been used to capture nonlinear maneuvering behavior from data [6] and to learn USV dynamics using deep networks such as LSTMs [7]. In addition, physics-informed and hybrid strategies are common in the broader literature: some methods enforce physical structure during learning [18, 19, 20], while others combine a nominal physics model with learned components to improve accuracy without losing the core physical meaning of the model [21, 22].

On the control side, nonlinear model predictive control (NMPC) is a natural choice for constrained tracking for USV case because it can handle nonlinear dynamics and input/state constraints directly, but its performance depends strongly on the prediction model used inside the optimizer [4]. For USVs, the physics-based 3-DOF maneuvering model is attractive because it encodes known rigid-body and hydrodynamic relationships and remains interpretable [2]. However, when the physics model is not accurate enough under realistic disturbances, noise, and actuator effects, model mismatch can translate into weaker tracking and more aggressive control actions [3]. This motivates using identified data-driven models inside NMPC when they can provide better predictive behavior over the optimization horizon while still being implementable in real time [5].

Using neural prediction models in NMPC raises practical concerns beyond pure prediction accuracy [22, 23, 25, 24]. The learned model must be numerically well behaved during finite-horizon rollout and must not cause unstable or inconsistent predictions that can mislead the optimizer. Encoder-decoder (sequence-to-sequence) LSTM predictors have been used in MPC frameworks as a way to represent horizon behavior using recurrent structures [26, 27], and several NMPC studies using LSTM/GRU predictors emphasize solver runtime and sensitivity as major practical constraints as model complexity grows [28, 29, 30, 31]. To reduce online cost, related work proposes algorithmic modifications such as linearization-based QP subproblems or locally approximated dynamics while still benefiting from recurrent predictors [32, 33]. In parallel, stability and robustness properties of gated recurrent predictors have been

studied, reflecting the concern that learned dynamics should behave safely when used in closed-loop control [34, 35].

Efficient derivative computation for Optimal Control Problem is another key requirement for neural NMPC[22], because modern solvers rely on gradients and Jacobians for fast convergence. Automatic differentiation was already explored in early neural predictive control work to improve training and optimization for noisy nonlinear systems [36]. More recent NMPC-oriented identification studies highlight that deep networks can be integrated more effectively when gradients/Jacobians are available through AD, which is especially relevant for large multi-output predictors [37]. Toolchains such as CasADi provide a practical foundation for algorithmic differentiation and nonlinear optimization in NMPC [9], while embedded NMPC frameworks such as acados target fast runtimes and real-time feasibility [10]. Current work also focuses on making learned models easier to deploy in these solvers while controlling computational overhead [38, 22, 39].

Overall, the literature supports the view that physics-based USV models provide a strong and interpretable starting point but can be limited by uncertainty, noise, and actuator realism [2, 3]. Neural identifiers can capture nonlinear and history-dependent behavior when the regressor reflects what is measurable and the training data cover the operating range [5, 14, 13]. For NMPC, learned models are most useful when they can be differentiated efficiently and used reliably inside a real-time solver [37, 9, 10]. This thesis follows these lessons for an underactuated 3-DOF USV by comparing a physics-structured Fossen baseline [2] with neural identification models, and by evaluating NMPC performance when these predictors are embedded in a real-time optimization loop, using the Clearpath Heron simulation environment as the test platform [11].

## CHAPTER 2

### PRELIMINARIES AND BACKGROUND

#### 2.1 USV Dynamics

In this thesis, we consider an underactuated Unmanned Surface Vessel (USV) as our case study for system identification and control. The vessel we use is equipped with two thrusters that generate surge forces and yaw moments. A simple representation of the considered USV is shown in Fig. 2.1.

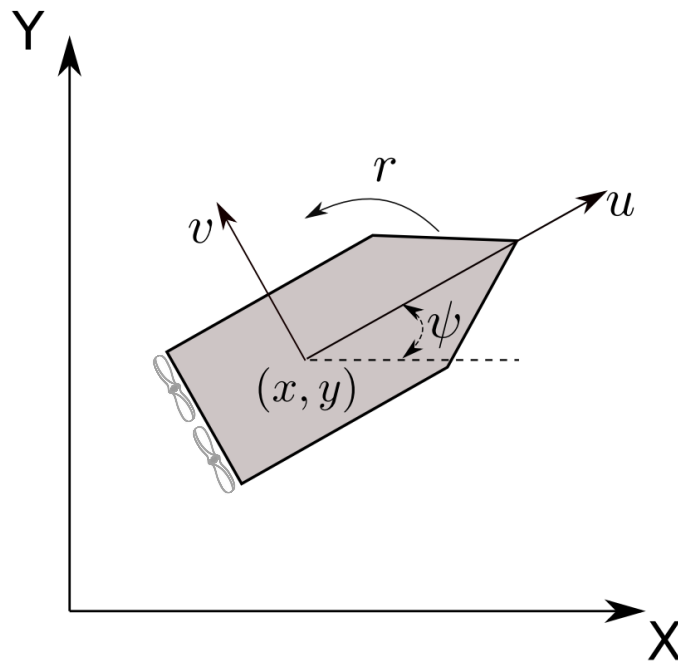


Figure 2.1: A simple representation of an Unmanned Surface Vessel

The USV can be modeled with the following state vectors:

- Position:  $\boldsymbol{\eta} = [x \ y \ \psi]^T$ , with  $x$  and  $y$  as positions on the planar environment and  $\psi$  as the yaw angle.
- Velocity:  $\boldsymbol{\nu} = [u \ v \ r]^T$ , where  $u$  is the surge,  $v$  is the sway, and  $r$  is the yaw rate.

The kinematic relationship between the inertial position  $\boldsymbol{p}$  and the body-frame velocity  $\boldsymbol{\nu}$  is expressed as

$$\dot{\boldsymbol{\eta}} = R(\psi) \boldsymbol{\nu}, \quad (2.1)$$

where  $R(\psi)$  is the planar rotation matrix that transforms body-frame velocities to the inertial frame can be described as

$$R(\psi) = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.2)$$

The dynamic model follows the standard 3-DOF marine craft formulation [2] and using (2.4) it can be expressed as

$$M\dot{\boldsymbol{\nu}} + C(\boldsymbol{\nu})\boldsymbol{\nu} + D(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau}_{thr} + \boldsymbol{\tau}_{wind} + \boldsymbol{\tau}_{wave}, \quad (2.3)$$

where  $M$  is the inertia matrix including added mass,  $C(\boldsymbol{\nu})$  is the Coriolis and centripetal matrix, and  $D(\boldsymbol{\nu})$  is hydrodynamic damping, composed of linear and nonlinear terms.

The rigid-body inertia and added-mass matrices [2] are given by

$$M_{RB} = \begin{bmatrix} m & 0 & -m y_G \\ 0 & m & m x_G \\ -m y_G & m x_G & I_z + m(x_G^2 + y_G^2) \end{bmatrix}, M_A = \begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & Y_{\dot{r}} & N_{\dot{r}}, \end{bmatrix}$$

$$C_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & -m(v + x_G r) \\ 0 & 0 & m(u - y_G r) \\ m(v + x_G r) & -m(u - y_G r) & 0 \end{bmatrix},$$

$$C_A(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & Y_{\dot{v}} v + Y_{\dot{r}} r \\ 0 & 0 & -X_{\dot{u}} u \\ -Y_{\dot{v}} v - Y_{\dot{r}} r & X_{\dot{u}} u & 0 \end{bmatrix},$$

$$D_l = \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix}, D_n(\boldsymbol{\nu}) = \begin{bmatrix} X_{|u|u} |u| & 0 & 0 \\ 0 & Y_{|v|v} |v| & 0 \\ 0 & 0 & N_{|r|r} |r| \end{bmatrix}$$

$$M = M_A + M_{RB}, C = C_A + C_{RB}, D = D_l + D_{nl} \quad (2.4)$$

The vectors  $\boldsymbol{\tau}_{\text{wind}}$  and  $\boldsymbol{\tau}_{\text{wave}}$  represents disturbances caused by environments due to wind and waves, while  $\boldsymbol{\tau}_{\text{thr}}$  is the control input vector generated by the two thrusters behind the vessel. The control input vector is expressed as follows:

$$\boldsymbol{\tau}_{\text{thr}} = \begin{bmatrix} F_1 + F_2 \\ 0 \\ \frac{b}{2}(F_1 - F_2) \end{bmatrix} \quad (2.5)$$

Here,  $F_1$  and  $F_2$  are left and right thruster forces, and  $b$  represents the distance between the thrusters. As we can see, the sway component of the control input is always zero, which makes the system underactuated.

## 2.2 Neural Network Architectures for System Identification

We use different neural network architectures to identify the dynamic model of our Unmanned Surface Vessel example. The objective of the system identification task is to create a mapping from the current velocity and input trajectory considered using at that time, which predicts the next velocity trajectory by the length of the horizon it's trained on, by taking the current velocity and the control input trajectory as inputs. We use two types of neural network architectures; the first one MS-MLP, includes the multilayer perceptron (MLP) architecture [40], while the second one is Seq2Seq which uses the long short-term memory (LSTM) architecture.

## 2.2.1 Multi-Layer Perceptron (MLP)

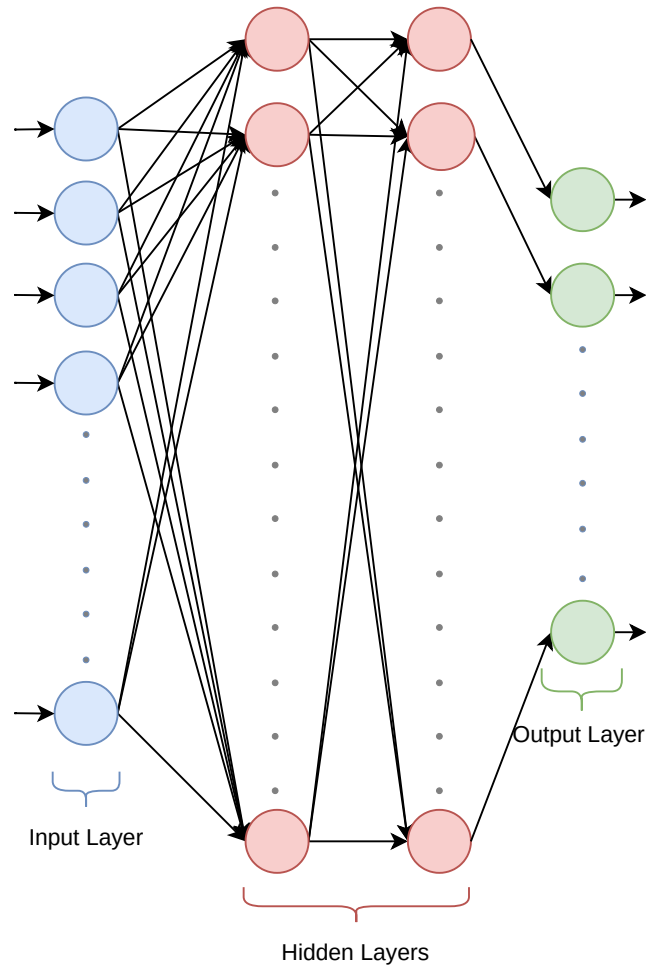


Figure 2.2: Basic structure of an MLP.

A compact  $N_L$ -layer MLP can be written as

$$\begin{aligned}
 h^{(0)} &= x, \\
 h^{(\ell)} &= \phi(W^{(\ell)}h^{(\ell-1)} + b^{(\ell)}), \quad \ell = 1, \dots, N_L - 1, \\
 \hat{y} &= W^{(N_L)}h^{(N_L-1)} + b^{(N_L)},
 \end{aligned} \tag{2.6}$$

where  $x$  is the input,  $\hat{y}$  is the output,  $W^{(\ell)}$  and  $b^{(\ell)}$  are learnable parameters, and  $\phi(\cdot)$  is an activation function (e.g.,  $\tanh(\cdot)$ , which is used for the MLP-based system identifier in this thesis).

## 2.2.2 Long short term Memory (LSTM)

LSTM networks are one of the classes of recurrent neural networks (RNN) designed to model temporal dependencies and solve the vanishing and exploding gradient problems that conventional RNNs have [41]. An LSTM cell uses gated mechanisms like input, forget, and output gates in order to maintain an internal memory state that is updated at each time step[14].

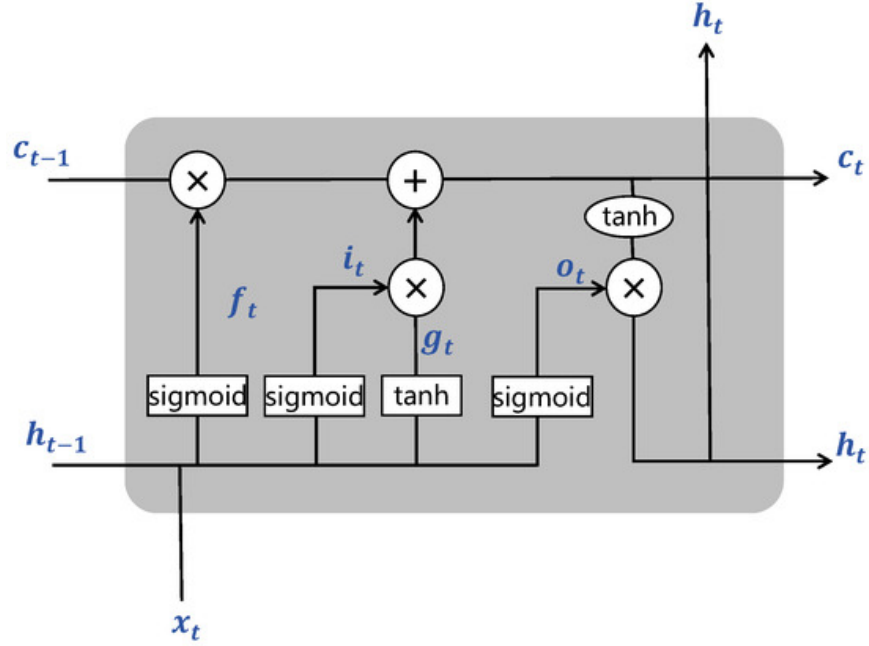


Figure 2.3: LSTM Schematic

The equations for the LSTM to get these gate parameters given in Fig. 2.3 are as follows:

$$\begin{aligned}
 f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f), \\
 i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i), \\
 o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o), \\
 \tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c), \\
 c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
 h_t &= o_t \odot \sigma_h(c_t).
 \end{aligned} \tag{2.7}$$

In these equations  $W_*$ ,  $U_*$  and  $b_*$  are learnable parameters for the LSTM,  $W_*$  are *input*

weight matrices,  $U_*$  are recurrent weight matrices,  $b_*$  are bias vectors to each gate. Each gate has its own set of weights, which allows independent control of information flow.

### 2.2.3 Nonlinear Model Predictive Control (NMPC)

Nonlinear Model Predictive Control (NMPC) is chosen to control the USV in this thesis. MPC computes an optimal sequence of control inputs that minimizes a cost function which is created by our control task over a receding prediction horizon. Because it is model-based, NMPC requires on a dynamic model to predict future system behavior according to a control trajectory. The USV dynamics, described by the 3-DOF Fossen model is non-linear, so we need to use non-linear solver to solve the optimization problem.

### 2.2.4 General NMPC Formulation

The NMPC problem is formulated for the discrete time USV dynamics as a finite-horizon optimal control problem. The discrete system dynamics can be represented as,

$$\boldsymbol{\nu}[k+1] = \mathbf{f}(\boldsymbol{\nu}[k], \boldsymbol{\tau}[k]), \quad (2.8)$$

where the control input at step  $k$  is

$$\boldsymbol{\tau}[k] = \begin{bmatrix} F_1[k] \\ F_2[k] \end{bmatrix}. \quad (2.9)$$

For NMPC, we optimize over a sequence of decided future control inputs,

$$\mathcal{T} = \begin{bmatrix} \boldsymbol{\tau}[k] \\ \boldsymbol{\tau}[k+1] \\ \vdots \\ \boldsymbol{\tau}[k+H-1] \end{bmatrix} \in \mathbb{R}^{2H}, \quad (2.10)$$

which represents all inputs over the prediction horizon  $H$ .

The cost function minimized by the NMPC controller is

$$\begin{aligned}
 J(\boldsymbol{\nu}[k], \mathcal{T}) = & \sum_{i=0}^{H-1} (\|\boldsymbol{\nu}[k+i] - \boldsymbol{\nu}_{\text{ref}}[k+i]\|_Q^2 + \|\boldsymbol{\tau}[k+i]\|_R^2) \\
 & + \|\boldsymbol{\nu}[k+H] - \boldsymbol{\nu}_{\text{ref}}[k+H]\|_{Q_f}^2.
 \end{aligned} \tag{2.11}$$

The complete NMPC problem is therefore

$$\begin{aligned}
 \min_{\mathcal{T}} \quad & J(\boldsymbol{\nu}[k], \mathcal{T}) \\
 \text{s.t.} \quad & \boldsymbol{\nu}[k+i+1] = \mathbf{f}(\boldsymbol{\nu}[k+i], \boldsymbol{\tau}[k+i]), \quad i = 0, \dots, H-1, \\
 & \boldsymbol{\tau}[k+i] \in [\boldsymbol{\tau}_{\min}, \boldsymbol{\tau}_{\max}], \quad i = 0, \dots, H-1,
 \end{aligned} \tag{2.12}$$

After solving the optimization, only the first control input of optimal control input trajectory  $\boldsymbol{\tau}^*[n]$  is applied to the USV. At the next sampling instant, the state is updated and the optimization is repeated over a shifted (receding) horizon.

### 2.2.5 Real-Time NMPC using Acados SQP\_RTI solver

NMPC is implemented with the open-source *acados* framework [10]. The model, cost, and constraints are written in *CasADi*, which provides automatic differentiation so that the solver can evaluate the required gradients and Jacobians efficiently [9]. At each sampling time, the OCP is solved with the SQP\_RTI algorithm, also known as the real-time iteration (RTI) scheme [43]. Instead of iterating until full convergence, RTI performs one SQP step per control update: it linearizes the dynamics and constraints around the previous solution, builds one QP approximation, solves it, and applies only the first control input. The remaining solution is shifted forward to warm-start the next update, which helps keep the runtime small and consistent.

To further reduce computation, the cost is written in a nonlinear least-squares form and `hessian_approx = "GAUSS_NEWTON"` is used. With this choice, the QP uses a Gauss–Newton Hessian approximation, so the solver avoids expensive second

derivatives and typically obtains a well-conditioned (positive semidefinite) curvature model. The QP is solved using full condensing with `FULL_CONDENSING_HPIPM`, which eliminates the state increments and produces a smaller dense QP in the input variables, solved efficiently by HPIPM [44]. The full RTI update relations and the detailed QP construction used here are given in Appendix A.

## CHAPTER 3

### METHODOLOGY

#### 3.1 Methodology Overview

This chapter has two main parts: system identification and motion control. The goal is to build discrete-time models of the USV surge, sway, and yaw dynamics from experimental data, and then use those models inside an NMPC controller. The models are required to be accurate over a finite prediction horizon, fast enough for real-time control, and paired with an OCP (Optimal Control Problem) formulation that preserves feasibility and solver reliability.

For system identification, a physics-based Fossen model with estimated parameters is compared against data-driven neural models. The neural models include a multi-step MLP [6, 5, 7] and a Seq2Seq LSTM [8, 14, 26], which are common choices for non-linear system identification and multi-step prediction. All networks are trained using PyTorch [42]. Together, these models provide a spectrum from physics-structured to purely data-driven representations.

For motion control, NMPC is formulated for both velocity and position tracking. The controller predicts future motion using the identified models and chooses thruster inputs to reduce tracking error while respecting thruster constraints [11], and its performance is evaluated in the presence of measurement noise. The Fossen and neural network models are redefined in CasADi[9], which provides automatic differentiation through an efficient computational graph for differentiation, and then used as prediction models inside an NMPC implementation built with acados [10]. For the neural network predictors, input and output normalization and denormalization layers are embedded explicitly in the CasADi model definition to ensure numerically well

scaled optimization variables during prediction and differentiation. Real-time solution follows an SQP-RTI style scheme [43], whose formulation and update equations are summarized in Appendix A.

## 3.2 System Identification

The experimental data include commanded thruster signals and measured velocities, but neither is ideal for identification. The commanded inputs do not perfectly match the thrust applied to the hull because of actuator dynamics, saturation, and operating-point effects. In addition, the velocity measurements contain sensor noise. These non-idealities directly affect parameter estimation and multi-step prediction quality.

As a structured baseline, the Fossen equations provide a standard hydrodynamic form that captures inertia, added-mass, damping, and Coriolis/centripetal coupling effects, but the associated parameters are difficult to identify and may vary with operating conditions [2, 3]. Data-driven neural models avoid an explicit parametric structure, but their reliability depends on the coverage of the training data [5]. For comparison, three distinct identification models are built: **(i)** a physics-structured Fossen model with identified parameters, **(ii)** a multi-step MLP that predicts the full horizon in a single inference, and **(iii)** Seq2Seq encoder-decoder LSTMs that predicts the trajectory one step at a time with shared recurrent weights.

### 3.2.1 Identifying Fossen Parameters for the USV

For the Heron catamaran, the center of mass is at the geometric midpoint, so  $x_G = y_G = 0$  in (2.4). With this simplification, the vector equation (2.3) expands into scalar surge ( $u$ ), sway ( $v$ ), and yaw rate ( $r$ ) dynamics.

For MPC, the continuous dynamics are approximated by a discrete-time difference equation model. This is computationally efficient, and with a sampling period of  $\Delta t = 0.1$  s it provides sufficient fidelity because the dominant USV motion evolves on time scales that are slow relative to the control update rate [12]. Let  $F_1[k]$  and  $F_2[k]$  be the left and right thruster forces at time step  $k$ . The discrete state evolution

is

$$\begin{aligned}u[k + 1] &= \alpha_1 u[k] + \alpha_2 |u[k]|u[k] + \alpha_3 v[k]r[k] + \alpha_4 r[k]^2 + \alpha_5 (F_1[k] + F_2[k]) \\v[k + 1] &= \beta_1 v[k] + \beta_2 |v[k]|v[k] + \beta_3 u[k]r[k] \\r[k + 1] &= \gamma_1 r[k] + \gamma_2 |r[k]|r[k] + \gamma_3 u[k]v[k] + \gamma_4 u[k]r[k] + \gamma_5 (F_1[k] - F_2[k])\end{aligned}\tag{3.1}$$

The coefficients  $(\alpha_i, \beta_i, \gamma_i)$  collect inertia, added mass, linear and quadratic damping, and Coriolis and centripetal coupling effects over the sampling period.

To estimate these coefficients, we use the Nonlinear Grey-Box System Identification framework in the MATLAB System Identification Toolbox [45]. This keeps the physical structure of the Fossen model while estimating parameters from data [2, 3]. The same training dataset used by the other identification approaches is applied here, with data discretized at a sampling period of 0.1 s (10 Hz).

### 3.2.2 Identifying USV Dynamics via MS-MLP

To capture nonlinear USV dynamics in a purely data-driven way, we use a Multi-Layer Perceptron (MLP) trained for multi-step prediction. The model predicts the future velocity trajectory over a finite horizon using past states and control inputs, which is effective for nonlinear system identification and USV modeling [5].

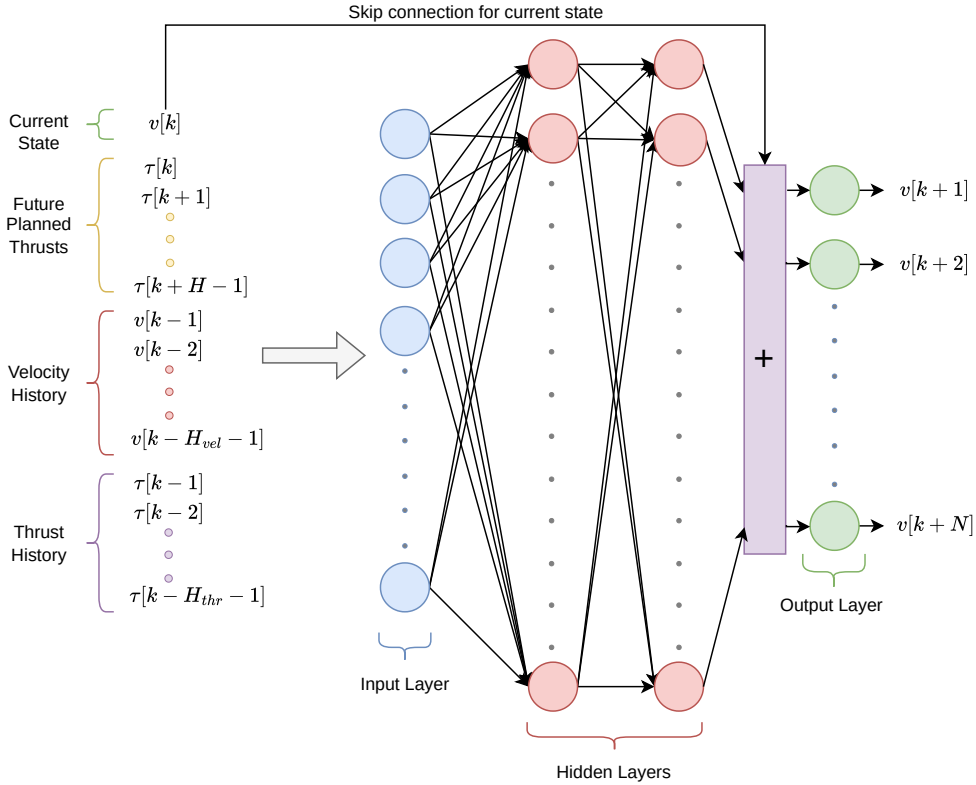


Figure 3.1: Architecture of the MS-MLP based system identification model with residual learning.

#### 3.2.2.1 Data Processing and Input Structure

The dataset consists of time-synchronized velocity vectors  $\boldsymbol{\nu} = [u, v, r]^T$  (surge, sway, and yaw rate) and thruster command vectors  $\boldsymbol{\tau} = [F_1, F_2]^T$ .

To build the supervised learning dataset, we use a sliding window on training data for

creating training sequences. For a given time step  $k$ , the input vector  $\mathbf{x}_k$  is formed by concatenating:

1. **Historical Velocities:** A sequence of past velocities of length  $H_{vel}$ :

$$\mathcal{V}_{hist} = [\boldsymbol{\nu}_{k-H_{vel}+1}, \dots, \boldsymbol{\nu}_k], \quad (3.2a)$$

$$\mathcal{T}_{hist} = [\boldsymbol{\tau}_{k-H_{thr}+1}, \dots, \boldsymbol{\tau}_{k-1}], \quad (3.2b)$$

$$\mathcal{T} = [\boldsymbol{\tau}_k, \dots, \boldsymbol{\tau}_{k+H-1}]. \quad (3.2c)$$

The flattened input vector  $\mathbf{x}_k$  is built by concatenating these sequences. Before training, all inputs and targets are normalized using min–max scaling to the interval  $[-1, 1]$ , based on the known physical limits of the vessel and thrusters, to match the effective operating range of the  $\tanh(\cdot)$  activation functions used in the hidden layers and to improve numerical conditioning during training.

### 3.2.2.2 Network Architecture

The neural network maps the input vector  $\mathbf{x}_k$  to the future velocity trajectory  $\hat{\mathcal{V}}_{future} \in \mathbb{R}^{3H}$ .

The network consists of an input layer,  $L$  fully connected hidden layers with  $n$  neurons per layer (chosen as tuning/design hyperparameters), and an output layer. Hyperbolic tangent ( $\tanh$ ) activation is used in the hidden layers.

A key part of the architecture is a residual (skip) connection. Instead of learning absolute dynamics from scratch, the network learns the change in velocity relative to the current state, similar to [18]. Let  $f_\theta(\cdot)$  denote the transformation learned by the MLP, where  $\theta$  are the network weights. The network predicts a deviation sequence  $\Delta\boldsymbol{\nu}$  over the horizon, and the final trajectory prediction is computed by adding this deviation to the current velocity  $\boldsymbol{\nu}_k$  (broadcasted across the horizon):

$$\hat{\boldsymbol{\nu}}_{k+j} = \boldsymbol{\nu}_k + [f_\theta(\mathbf{x}_k)]_j \quad \text{for } j = 1, \dots, H. \quad (3.3)$$

This formulation helps the network learn incremental dynamics and improves training stability.

### 3.2.3 Training Implementation

The model is trained using the PyTorch framework. The objective is to minimize the mean squared error (MSE) between the predicted future velocities and the ground truth trajectory over a dataset of  $N_{\text{seq}}$  training sequences (constructed via the sliding-window procedure) and over the prediction horizon  $H$ . For the  $n$ -th training sequence, let  $\mathbf{Y}_i^{(n)} \in \mathbb{R}^3$  denote the ground-truth velocity vector at the  $i$ -th step of the prediction horizon, and let  $\hat{\mathbf{Y}}_i^{(n)} \in \mathbb{R}^3$  be the corresponding network prediction.

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{seq}} H} \sum_{n=1}^{N_{\text{seq}}} \sum_{i=1}^H \left\| \mathbf{Y}_i^{(n)} - \hat{\mathbf{Y}}_i^{(n)} \right\|^2 \quad (3.4)$$

We use the Adam optimizer with an initial learning rate of  $10^{-2}$ . A scheduler decays the learning rate by a factor of 0.5 every 100 epochs. The model is trained for 1000 epochs with a batch size of 512, using a validation set to monitor generalization and save the best-performing weights.

### 3.2.4 Identifying USV Dynamics via LSTM

The MS-MLP model is a useful baseline, but in our tests its validation error plateaued, and larger models tended to overfit. In addition, the MS-MLP predicts the entire future trajectory in a single forward pass, which may introduce practical limitations for long-horizon prediction: because the full sequence of future inputs is provided at once, the model does not explicitly enforce temporal causality within the prediction horizon, and errors or inconsistencies can propagate across time steps in an unstructured manner, which can also degrade the optimality of the resulting NMPC solution. To address these issues, we use a Long Short-Term Memory (LSTM) network [14] in a Sequence-to-Sequence (Seq2Seq) encoder-decoder architecture. This recurrent structure shares parameters across the horizon, reduces model size, and preserves temporal consistency for long predictions. Similar encoder-decoder LSTM structures have been used for multi-step predictive control [26].

The proposed network architecture is illustrated in Figure 3.2. The figure shows an unrolled view, but in implementation the encoder and decoder are two distinct LSTM

blocks (with separate parameters), each applied recursively over its respective time indices: the encoder processes the history window with shared weights across encoder steps, and the decoder reuses its own LSTM block across the prediction horizon.

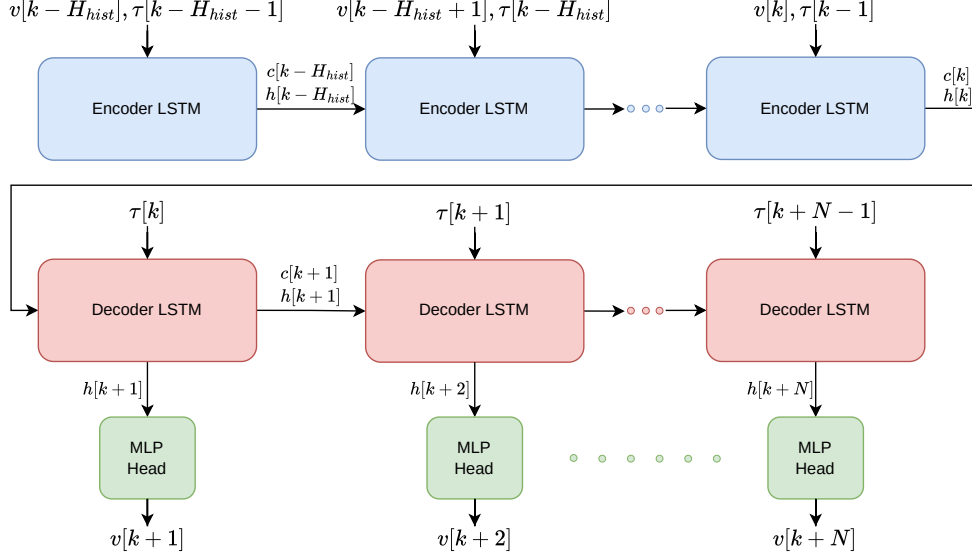


Figure 3.2: Architecture of the LSTM-based Sequence-to-Sequence system identification model. The Encoder captures historical dynamics, while the Decoder recursively predicts future velocities based on planned thrusts.

The model consists of three distinct stages:

### 3.2.4.1 Encoder Stage (State Estimation)

The encoder acts as a context extractor. It processes a historical window of state-action pairs  $\{(\boldsymbol{\nu}_t, \boldsymbol{\tau}_{t-1})\}_{t=k-H_{hist}}^k$ , where  $\boldsymbol{\nu}$  is the velocity vector and  $\boldsymbol{\tau}$  is the thrust vector. Unlike the MS-MLP, the recurrent structure requires synchronized input vectors, so the history length for velocity and thrust must be the same.

By iterating through the sequence with the shared LSTM cell, the network maintains a cell state  $c_k$  and hidden state  $h_k$ . These final states at time step  $k$  compress the vessel's recent motion into a fixed-size latent vector.

### 3.2.4.2 Decoder Stage and MLP Head (Trajectory Simulation)

The final hidden state  $h_k$  and cell state  $c_k$  from the encoder are passed to the decoder. This starts the future prediction with the correct context from the recent past. The decoder acts as a learned forward simulator. For a prediction horizon  $H$ , the decoder takes the planned future control inputs  $\{\tau_t\}_{t=k}^{k+H-1}$ . At each time step  $t$ , the decoder's LSTM cell updates its hidden state  $h_t$  based on the control input  $\tau_t$  and the previous state. This hidden state represents the latent physics of the vessel at that moment.

To map this latent representation back to the physical domain, a Multilayer Perceptron (MLP) head is attached to the output of each decoder step. The MLP head consists of a single hidden layer with 32 neurons and projects the decoder hidden features to the predicted state space ( $\mathbb{R}^3$ ), yielding the velocity sequence  $\{\hat{\nu}_t\}_{t=k+1}^{k+H}$ .

## 3.3 NMPC Formulation for Velocity Control

### 3.3.1 Problem Statement and Notation

Let the body-frame velocity state be

$$\nu[k] = [u[k], v[k], r[k]]^\top \in \mathbb{R}^3$$

and the control input (twin thrusters) be

$$\tau[k] = [F_1[k], F_2[k]]^\top \in \mathbb{R}^2.$$

All models are discretized at a fixed sampling time of  $\Delta t = 0.1$  s. Given a reference velocity trajectory  $\nu^{\text{ref}}[k]$ , the NMPC minimizes tracking error and control effort over a finite horizon. Predicted velocities along the horizon are denoted by  $\hat{\nu}[k]$ .

For a generic horizon length  $H$ , define the input trajectory

$$\mathcal{T} = \{\tau[0], \dots, \tau[N-1]\}.$$

The NMPC problem is

$$\min_{\mathcal{T}} \sum_{k=0}^{H-1} \|\hat{\boldsymbol{\nu}}[k] - \boldsymbol{\nu}^{\text{ref}}[k]\|_{\mathbf{Q}}^2 + \sum_{k=0}^{H-1} \|\boldsymbol{\tau}[k]\|_{\mathbf{R}}^2 + \|\hat{\boldsymbol{\nu}}[H] - \boldsymbol{\nu}^{\text{ref}}[H]\|_{\mathbf{Q}_f}^2 \quad (3.5)$$

$$\text{s.t. } \mathbf{x}[k+1] = f(\mathbf{x}[k], \boldsymbol{\tau}[k]), \quad k = 0, \dots, H-1, \quad (3.6)$$

$$\mathbf{x}[0] = \hat{\mathbf{x}}[0], \quad (3.7)$$

$$\boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau}[k] \leq \boldsymbol{\tau}_{\max}. \quad (3.8)$$

Here  $\mathbf{Q}$ ,  $\mathbf{Q}_f$ , and  $\mathbf{R}$  are positive (semi-)definite weighting matrices chosen during tuning. The state trajectory is obtained by forward simulation of  $f(\cdot)$  given  $\mathcal{T}$ , and the state definition  $\mathbf{x}[k]$  depends on the selected model, described next. Indexing starts at  $k = 0$  for convenience; since  $\mathbf{x}[0]$  is fixed, the stage cost at  $k = 0$  is constant and can be omitted without changing the optimizer. The first control input  $\boldsymbol{\tau}[k]$  is applied to the plant. The remaining optimal input sequence is then shifted forward by one step and used to warm-start the optimization problem at the next control period.

### 3.3.2 Modifying the NN Models for Velocity Control

The learned models are trained in PyTorch with normalized inputs and outputs. In the NMPC implementation, these normalization/denormalization steps are embedded inside the CasADi model graphs. For any physical input vector  $\mathbf{z}$  and output vector  $\mathbf{y}$ , we use

$$\tilde{\mathbf{z}} = (\mathbf{z} - \boldsymbol{\mu}_z) \oslash \boldsymbol{\sigma}_z, \quad (3.9)$$

$$\mathbf{y} = \tilde{\mathbf{y}} \odot \boldsymbol{\sigma}_y + \boldsymbol{\mu}_y, \quad (3.10)$$

where  $\oslash$  and  $\odot$  denote elementwise division and multiplication. We therefore use composed functions that map physical quantities directly to physical predictions, without explicit normalization terms in the remaining equations. Concretely, we use two model types: the Seq2Seq LSTM with encoder, decoder, and output-head mappings  $f_{\text{enc,vel}}(\cdot)$ ,  $f_{\text{dec}}(\cdot)$ , and  $f_{\text{head,vel}}(\cdot)$ , and the MS-MLP mapping  $f_{\text{mlp,vel}}(\cdot)$  to indicate the embedded normalization layers. In particular, normalization is embedded at the encoder input and at the output head (which maps latent features to physical velocities), while the decoder recurrence evolves only the latent states.

For example, the composed mappings are

$$f_{\text{enc,vel}}(\mathbf{z}) = f_{\text{enc}}((\mathbf{z} - \boldsymbol{\mu}_z) \otimes \boldsymbol{\sigma}_z), \quad (3.11)$$

$$f_{\text{dec}}(\tilde{\mathbf{z}}, \mathbf{h}, \mathbf{c}) = f_{\text{dec}}(\tilde{\mathbf{z}}, \mathbf{h}, \mathbf{c}), \quad (3.12)$$

$$f_{\text{head,vel}}(\mathbf{h}) = (f_{\text{head}}(\mathbf{h})) \odot \boldsymbol{\sigma}_y + \boldsymbol{\mu}_y, \quad (3.13)$$

$$f_{\text{mlp,vel}}(\mathbf{z}) = (f_{\text{mlp}}((\mathbf{z} - \boldsymbol{\mu}_z) \otimes \boldsymbol{\sigma}_z)) \odot \boldsymbol{\sigma}_y + \boldsymbol{\mu}_y. \quad (3.14)$$

### 3.3.3 Fossen NMPC Model for Velocity Control

For the physics-based baseline, the discrete-time dynamics are written as

$$\hat{\boldsymbol{\nu}}[k+1] = f_{\text{fossen}}(\boldsymbol{\nu}[k], \boldsymbol{\tau}[k]), \quad (3.15)$$

where  $f_{\text{fossen}}(\cdot)$  denotes the identified discrete model. The NMPC state is simply

$$\mathbf{x}[k] = \boldsymbol{\nu}[k],$$

so the optimization follows (3.5) with the Fossen dynamics (3.15) and input bounds.

### 3.3.4 Seq2Seq NMPC Model for Velocity Control

The Seq2Seq model uses an encoder–decoder structure. The encoder processes a history of measured velocities and applied thrusts to compute a latent context. Let the past window lengths be  $H_{\text{hist}}$  for velocities and thrusts. The encoder is applied outside the optimization to initialize the NMPC state:

$$[\mathbf{h}[k], \mathbf{c}[k]] = f_{\text{enc,vel}}(\{\boldsymbol{\nu}[k - H_{\text{hist}} + 1], \dots, \boldsymbol{\nu}[k]\}, \{\boldsymbol{\tau}[k - H_{\text{hist}}], \dots, \boldsymbol{\tau}[k - 1]\}), \quad (3.16)$$

where the inputs are the recent velocity and thrust histories. The NMPC state is augmented with the LSTM hidden and cell states:

$$\mathbf{x}[k] = \begin{bmatrix} \hat{\boldsymbol{\nu}}[k] \\ \mathbf{h}[k] \\ \mathbf{c}[k] \end{bmatrix}.$$

The current velocity is included in the state for tracking and consistency; at time  $k$  it is fixed by measurement, so it does not add decision variables.

The decoder advances the LSTM using the current thrust input, and the discrete dynamics are

$$[\mathbf{h}[k+1], \mathbf{c}[k+1]] = f_{\text{dec}}(\boldsymbol{\tau}[k], \mathbf{h}[k], \mathbf{c}[k]), \quad (3.17)$$

$$\hat{\boldsymbol{\nu}}[k+1] = f_{\text{head,vel}}(\mathbf{h}[k+1]). \quad (3.18)$$

The NMPC objective remains (3.5), with  $\hat{\mathbf{x}}[0]$  formed from the measured  $\boldsymbol{\nu}[0]$  and the encoder output (3.16).

### 3.3.5 MS-MLP NMPC Model for Velocity Control

The MS-MLP model predicts the full velocity horizon in a single forward pass. It uses a history of velocities and past thrusts, and it takes the future thrust sequence as input. Let the prediction horizon be  $H$ . The NMPC decision variable is the entire future thrust sequence

$$\boldsymbol{\mathcal{T}} = [\boldsymbol{\tau}[k]^\top, \boldsymbol{\tau}[k+1]^\top, \dots, \boldsymbol{\tau}[k+H-1]^\top]^\top.$$

Define the stacked histories and horizon prediction as

$$\mathcal{V}_{\text{hist}} = \boldsymbol{\nu}[k - H_v + 1 : k], \quad \boldsymbol{\mathcal{T}}_{\text{hist}} = \boldsymbol{\tau}[k - H_u : k - 1], \quad \hat{\mathcal{V}} = \hat{\boldsymbol{\nu}}[k : k + H]$$

Here  $\mathcal{V}_{\text{hist}}$  includes the current measured velocity at time  $k$ . The NMPC state contains the velocity and thrust histories:

$$\mathbf{x}[k] = \begin{bmatrix} \mathcal{V}_{\text{hist}} \\ \boldsymbol{\mathcal{T}}_{\text{hist}} \end{bmatrix},$$

The network outputs the horizon prediction

$$\hat{\mathcal{V}} = f_{\text{mlp,vel}}(\mathcal{V}_{\text{hist}}, \boldsymbol{\mathcal{T}}_{\text{hist}}, \boldsymbol{\mathcal{T}}), \quad (3.19)$$

where the residual/shortcut connection with the current velocity is handled inside the model.

Define the stacked reference

$$\boldsymbol{\nu}^{\text{ref}} = [\boldsymbol{\nu}^{\text{ref}}[k], \dots, \boldsymbol{\nu}^{\text{ref}}[k+H]]^\top,$$

and block-diagonal weights

$$\bar{\mathbf{Q}} = \mathbf{I}_H \otimes \mathbf{Q}, \quad \bar{\mathbf{R}} = \mathbf{I}_H \otimes \mathbf{R}.$$

Because the MLP predicts the whole horizon, the NMPC can be written as a single optimization step with decision variable  $\mathcal{T}$ :

$$\min_{\mathcal{T}} \left\| \hat{\mathcal{V}} - \mathcal{V}^{\text{ref}} \right\|_{\bar{\mathbf{Q}}}^2 + \|\mathcal{T}\|_{\bar{\mathbf{R}}}^2, \quad (3.20)$$

$$\text{s.t. } \tau_{\min} \leq \tau[k+i] \leq \tau_{\max}, \quad i = 0, \dots, H-1.$$

The NMPC problems using the Fossen, MS-MLP and Seq2Seq LSTM prediction models are solved using a Gauss-Newton approximation of the Hessian and an SQP\_RTI scheme for the resulting nonlinear program, consistent with the real-time requirements of velocity control.

### 3.4 NMPC Formulation for Position Control

Let the inertial-frame pose and body-frame velocity be

$$\boldsymbol{\eta}[k] = [x[k], y[k], \psi[k]]^\top, \quad \boldsymbol{\nu}[k] = [u[k], v[k], r[k]]^\top,$$

and the twin-thruster input be

$$\boldsymbol{\tau}[k] = [F_1[k], F_2[k]]^\top.$$

In position control, tracking only  $(x, y)$  can leave the heading unconstrained: the USV may reach the correct position but with an arbitrary yaw angle. We therefore also penalize yaw error so the vehicle aligns with the reference direction along the path. This reduces unnecessary turning and thrust effort, and it often leads the NMPC optimizer to more efficient (more optimal) motion plans.

Instead of tracking the yaw angle directly, we track its cosine and sine to avoid wrap-

around and to keep an angle-aware cost. Define the state-to-output map

$$\mathbf{y}_x[k] = h_x(\mathbf{x}[k]), \quad h_x(\mathbf{x}[k]) = \begin{bmatrix} x[k] \\ y[k] \\ \cos(\psi[k]) \\ \sin(\psi[k]) \\ u[k] \\ v[k] \\ r[k] \end{bmatrix}. \quad (3.21)$$

For the *stage* cost we append the control input and define

$$\mathbf{y}[k] = h(\mathbf{x}[k], \boldsymbol{\tau}[k]) = \begin{bmatrix} h_x(\mathbf{x}[k]) \\ \boldsymbol{\tau}[k] \end{bmatrix} = \begin{bmatrix} x[k] \\ y[k] \\ \cos \psi[k] \\ \sin \psi[k] \\ u[k] \\ v[k] \\ r[k] \\ F_1[k] \\ F_2[k] \end{bmatrix}. \quad (3.22)$$

The reference signals are assumed *given* as sequences

$$\mathbf{y}_x^{\text{ref}}[k] \in \mathbb{R}^7, \quad \mathbf{y}^{\text{ref}}[k] \in \mathbb{R}^9.$$

A common choice (used when there is no input reference) is,

$$\mathbf{y}^{\text{ref}}[k] = \begin{bmatrix} \mathbf{y}_x^{\text{ref}}[k] \\ \mathbf{0}_{2 \times 1} \end{bmatrix},$$

but in general  $\mathbf{y}^{\text{ref}}[k]$  can include a nonzero input reference.

This yaw representation is consistent with the familiar  $1 - \cos(\Delta\psi)$  error. Indeed, with  $\Delta\psi = \psi - \psi_{\text{ref}}$ ,

$$(\cos \psi - \cos \psi_{\text{ref}})^2 + (\sin \psi - \sin \psi_{\text{ref}})^2 = 2 - 2 \cos(\Delta\psi) = 2(1 - \cos(\Delta\psi)),$$

so a quadratic penalty on  $(\cos \psi, \sin \psi)$  is equivalent (up to a constant factor) to penalizing  $1 - \cos(\Delta\psi)$ . In addition, this yaw-error term is smooth and periodic, avoiding wrap-around discontinuities and yielding a well-behaved objective for gradient based optimization which our NMPC formulation.

### 3.4.1 Pose propagation used in the horizon rollout

All position NMPC variants require a pose rollout over the horizon. We use a midpoint (average-velocity) update to reduce discretization bias. Using the planar rotation (2.2) which is

$$\mathbf{R}(\psi) = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and with sampling period  $\Delta t = 0.1$  s, define

$$\begin{aligned} \hat{\boldsymbol{\nu}}_{\text{avg}}[k] &= \frac{1}{2}(\hat{\boldsymbol{\nu}}[k] + \hat{\boldsymbol{\nu}}[k+1]), \\ \psi_{\text{avg}}[k] &= \hat{\psi}[k] + \frac{1}{2} \Delta t \hat{r}_{\text{avg}}[k], \end{aligned} \quad (3.23)$$

where  $\hat{r}_{\text{avg}}[k]$  is the third component of  $\hat{\boldsymbol{\nu}}_{\text{avg}}[k]$ . The midpoint pose update is

$$\hat{\boldsymbol{\eta}}[k+1] = \hat{\boldsymbol{\eta}}[k] + \Delta t \mathbf{R}(\psi_{\text{avg}}[k]) \hat{\boldsymbol{\nu}}_{\text{avg}}[k]. \quad (3.24)$$

### 3.4.2 NMPC objective with stage and terminal outputs

We use a nonlinear least-squares objective with different stage and terminal outputs. The stage output includes the input (so stage weights are effectively  $\text{diag}(Q, R)$ ), while the terminal output excludes the input (so terminal weight is  $Q_f$  only). Define

$$\mathbf{W} = \text{diag}(\mathbf{Q}, \mathbf{R}), \quad \mathbf{W}_H = \mathbf{Q}_f.$$

The position NMPC problem is

$$\min_{\{\boldsymbol{\tau}[k]\}_{k=0}^{H-1}} \sum_{k=0}^{H-1} \|\mathbf{y}[k] - \mathbf{y}^{\text{ref}}[k]\|_{\mathbf{W}}^2 + \|\mathbf{y}_x[H] - \mathbf{y}_x^{\text{ref}}[H]\|_{\mathbf{Q}_f}^2 \quad (3.25)$$

$$\text{s.t. } \hat{\mathbf{x}}[k+1] = f(\hat{\mathbf{x}}[k], \boldsymbol{\tau}[k]), \quad k = 0, \dots, N-1, \quad (3.26)$$

$$\hat{\mathbf{x}}[0] = \mathbf{x}_0, \quad (3.27)$$

$$\boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau}[k] \leq \boldsymbol{\tau}_{\max}, \quad k = 0, \dots, N-1, \quad (3.28)$$

where  $\mathbf{y}[k] = h(\hat{\mathbf{x}}[k], \boldsymbol{\tau}[k])$  and  $\mathbf{y}_x[k] = h_x(\hat{\mathbf{x}}[k])$  as defined above. The only remaining model-dependent element is the transition map  $f(\cdot)$ , which differs for the Fossen, LSTM, and MS-MLP predictors.

### 3.4.3 Model-dependent NMPC state and transition

#### Fossen-based position NMPC

For the physics-based baseline, the NMPC state is

$$\mathbf{x}[k] = [x[k], y[k], \psi[k], u[k], v[k], r[k]]^\top,$$

and the transition

$$\hat{\mathbf{x}}[k+1] = f_{\text{fossen}}(\hat{\mathbf{x}}[k], \boldsymbol{\tau}[k])$$

updates both velocity and pose. Conceptually, the model first updates  $\hat{\boldsymbol{v}}[k+1]$  using the identified discrete dynamics, then updates  $\hat{\boldsymbol{\eta}}[k+1]$  using the kinematics (implemented consistently with the chosen discretization represented in (3.24)). The stage and terminal costs are evaluated using  $h(\cdot)$  and  $h_x(\cdot)$  in (3.25).

#### Seq2Seq LSTM-based position NMPC

For the Seq2Seq predictor, the NMPC state augments the physical state with the LSTM hidden and cell states:

$$\mathbf{x}[k] = \begin{bmatrix} x[k] \\ y[k] \\ \psi[k] \\ u[k] \\ v[k] \\ r[k] \\ h[k] \\ c[k] \end{bmatrix}$$

At each prediction step, the LSTM decoder advances  $(\mathbf{h}, \mathbf{c})$  using the current input, and an output head produces the next velocity:

$$[\mathbf{h}[k+1], \mathbf{c}[k+1]] = f_{\text{dec}}(\mathbf{h}[k], \mathbf{c}[k], \boldsymbol{\tau}[k]), \quad \hat{\boldsymbol{v}}[k+1] = f_{\text{head,vel}}(\mathbf{h}[k+1]).$$

The pose is then propagated by the midpoint update (3.24) using  $\hat{\boldsymbol{v}}[k]$  and  $\hat{\boldsymbol{v}}[k+1]$ . This composite update defines  $f(\cdot)$  in (3.26). The output maps  $h(\cdot)$  and  $h_x(\cdot)$  only use the physical components  $(x, y, \psi, u, v, r)$ , and ignore  $(\mathbf{h}, \mathbf{c})$ .

### MS-MLP-based position NMPC (condensed horizon form)

The MS-MLP predictor is used in a condensed form where the decision variable is the entire future input sequence

$$\mathcal{T} = [\boldsymbol{\tau}[k]^\top, \dots, \boldsymbol{\tau}[k+H-1]^\top]^\top \in \mathbb{R}^{2H}.$$

The state contains the current pose and the required histories:

$$\mathbf{x}[k] = \begin{bmatrix} x[k] \\ y[k] \\ \psi[k] \\ \mathcal{V}_{\text{hist}} \\ \mathcal{T}_{\text{hist}} \end{bmatrix},$$

where  $\mathcal{V}_{\text{hist}}$  is the stacked velocity history (including the current velocity) and  $\mathcal{T}_{\text{hist}}$  is the stacked input history used by the network, as defined in (3.2).

Given  $\mathbf{x}[k]$  and  $\mathcal{T}$ , the MS-MLP predicts a velocity horizon  $\{\hat{\boldsymbol{v}}[k+i]\}_{i=1}^H$ , and the corresponding pose horizon is obtained by rolling out using (3.24). Define the stacked stage outputs

$$\hat{\mathcal{Y}} = \begin{bmatrix} \hat{\mathbf{y}}[k] \\ \hat{\mathbf{y}}[k+1] \\ \vdots \\ \hat{\mathbf{y}}[k+H-1] \end{bmatrix} \in \mathbb{R}^{9H}, \quad \hat{\mathbf{y}}[k+i] = \begin{bmatrix} \hat{x}[k+i] \\ \hat{y}[k+i] \\ \cos \hat{\psi}[k+i] \\ \sin \hat{\psi}[k+i] \\ \hat{u}[k+i] \\ \hat{v}[k+i] \\ \hat{r}[k+i] \\ F_1[k+i] \\ F_2[k+i] \end{bmatrix}. \quad (3.29)$$

With the stacked reference  $\mathcal{Y}^{\text{ref}}[k] \in \mathbb{R}^{9H}$  and block-diagonal stage weights

$$\bar{\mathbf{W}} = \mathbf{I}_H \otimes \text{diag}(\mathbf{Q}, \mathbf{R}), \quad (3.30)$$

the condensed MS-MLP position NMPC objective is

$$\min_{\mathcal{T}} \left\| \hat{\mathcal{Y}} - \mathcal{Y}^{\text{ref}} \right\|_{\bar{\mathbf{W}}}^2 + \left\| \hat{\mathbf{y}}_x[k+H] - \mathbf{y}_x^{\text{ref}}[k+H] \right\|_{\mathbf{Q}_f}^2. \quad (3.31)$$

subject to the input bounds at each step of the horizon. This matches the same structure as (3.25): stage penalties use  $\text{diag}(Q, R)$  (because inputs are in  $\hat{\mathcal{Y}}$ ) and the terminal penalty uses only  $Q_f$  (because it uses  $\hat{\mathbf{y}}_x$ ).



## CHAPTER 4

### IMPLEMENTATION AND RESULTS

#### 4.1 Simulation Setup and Data Acquisition

The simulation environment in this thesis is based on ROS and Gazebo, using the official Clearpath Robotics packages for the *Heron* USV [11]. These packages give a reasonable approximation of the real Heron. For example, in [48] the authors successfully created a control policy using only this simulation setup for real-life application, which suggests that it is accurate enough for control development. These packages are designed for *ROS Melodic* on *Ubuntu 18.04*. Since this software stack is relatively old and can conflict with newer libraries needed in our implementation, the entire simulator (ROS, Gazebo, and the Clearpath Heron packages) was built and executed inside a Docker container based on Ubuntu 18.04. This keeps the simulator compatible with ROS Melodic while allowing the host system to use a more flexible and modern Python environment without breaking the simulator dependencies, and it also makes the overall setup easier to reproduce on different machines. The host machine used for development runs Ubuntu 24.04 on a 12th Gen Intel(R) Core(TM) i5-12400F CPU and an NVIDIA GeForce RTX 4070 SUPER GPU. In the simulator, the Heron model provides the main sensor and actuator interfaces used in this thesis. The primary sensors are GPS and IMU, and the vessel is actuated by two thrusters (left and right)[11].

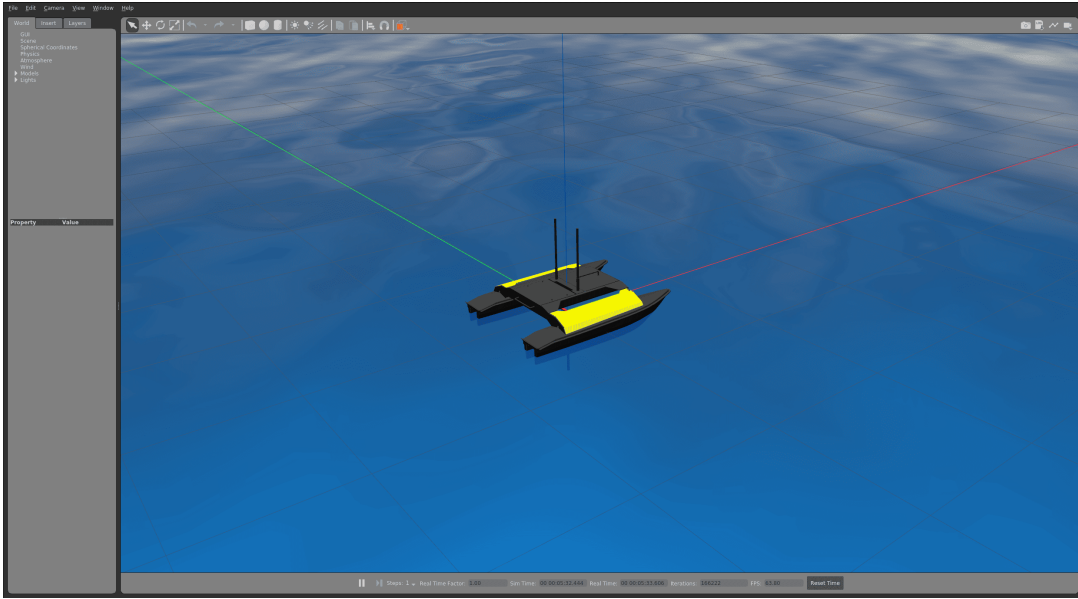


Figure 4.1: Gazebo Simulator for the Heron USV

Control commands are applied by publishing to `/cmd_drive`, where the left and right command values are normalized to the range  $[-1, +1]$ . Inside the Clearpath simulation stack, these normalized inputs are converted into physical thrust using a fixed mapping. The command-to-thrust relationship used in our setup is shown in Table 4.1.

A practical limitation of `/cmd_drive` in the built-in packages is that the message does not include a standard ROS header, so it does not carry a timestamp. In the default Heron simulation, `/cmd_drive` is processed by an intermediate node before reaching the Gazebo thruster plugin, and this node publishes the per-thruster commands on `/heron/thrusters/` at 20 Hz. For logging and synchronization, we use this downstream thruster command stream to associate each command with a timestamp based on ROS time at publish/receive.

While checking the Gazebo implementation of the thrusters, we also found that the thruster response is not instantaneous. Instead, each thruster is modeled as a first-order continuous-time system with unit steady-state gain and time constant

$$\tau_{\text{constant}} = 0.05 \text{ s.} \quad (4.1)$$

Table 4.1: Mapping between normalized propulsor input on `/cmd_drive` and the corresponding thrust output used by the simulator[11].

<b>Input to propulsor</b>	<b>Output thrust (N)</b>
-1.0	-19.88
-0.8	-16.52
-0.6	-12.60
-0.4	-5.60
-0.2	-1.40
0.0	0.00
0.2	2.24
0.4	9.52
0.6	21.28
0.8	28.00
1.0	33.60

This behavior can be expressed as:

$$\tau_{\text{constant}} \dot{T}(t) + T(t) = T_{\text{cmd}}(t), \quad (4.2)$$

where  $T_{\text{cmd}}(t)$  is the commanded thrust and  $T(t)$  is the actual thrust applied in simulation. As a result, the commanded thrust and the applied thrust can differ, especially when the input changes quickly. We intentionally did *not* provide this thruster model (including  $\tau_{\text{constant}}$ ) to our algorithms, so that the methods do not rely on simulator-only internal information.

For feedback and data collection, the simulation provides odometry topics. The `/odometry/filtered` topic provides filtered estimates of global position and global-frame velocities, but the built-in packages do not provide a topic that directly outputs filtered body-frame velocities. Since our controller requires body-frame velocity feedback, we implied the `/odometry/base` topic provided in [47], which provides twist information in the base frame (body frame) and runs at 20 Hz in our setup.

All real-time control and data acquisition code was written in Python and executed on

the host computer, while the simulator runs inside the Docker container. To communicate between the host-side Python process and the ROS graph inside Docker, we used `rosbridge_server` [49] inside the container, exposing ROS topics over a WebSocket connection. The container network was configured so that the `rosbridge` port is forwarded to a local port on the host machine, and the host-side code connects using the `roslibpy` library [50]. This allows the controller code to subscribe to odometry topics for feedback and logging and to publish drive commands to `/cmd_drive` in real time.



Figure 4.2: Host–Docker communication diagram for ROS-based simulation control.

## 4.2 Data Acquisition for System Identification

The system identification stage requires an input–output dataset that is informative enough to excite the main vessel dynamics over a wide operating range. For this reason, the data were collected in simulation by applying a long sequence of thruster commands designed to include different temporal patterns (slow/fast variations, steady inputs, and intervals with zero thrust). This approach improves the richness of the training data and reduces the risk of fitting a model that only performs well under a narrow set of maneuvers.

The thruster commands were generated as a sequence of *events*. Each event represents a specific command type with its own parameters and duration, and the overall sequence is executed with a fixed sampling period of  $T_s = 0.1$  s, which is used both as the discretization step for the discrete-time model and as the controller command-update period. Three main event families were used. The first family uses *sinusoidal variations*, where the thruster commands are defined by sine signals with different frequencies, bias values, amplitudes, and phase differences between the left and right thrusters. The second family uses *chirp-type* inputs to sweep frequency content over

time; these events are defined by different start and end frequencies, durations, amplitudes, bias values, and left-right phase differences. The third family consists of *constant-thrust* events, where fixed left and right thrust values are applied for different durations to create steady-state segments. In addition to these, many *break* events were inserted (zero thrust on both thrusters) to include coasting intervals; in the no-disturbance case, the USV velocities converge to zero under zero net thrust.

After defining a large library of events, the full input sequence was created by randomly shuffling the event order and then applying the resulting command timeline to the Gazebo simulator. During execution, we logged the commanded thruster signals together with their timestamps and the corresponding measured velocities from the simulator. The resulting training dataset has a total duration of 764.1 s, which corresponds to 7641 samples at  $T_s = 0.1$  s. An overview of the training sequence structure is shown in Fig. 4.3.

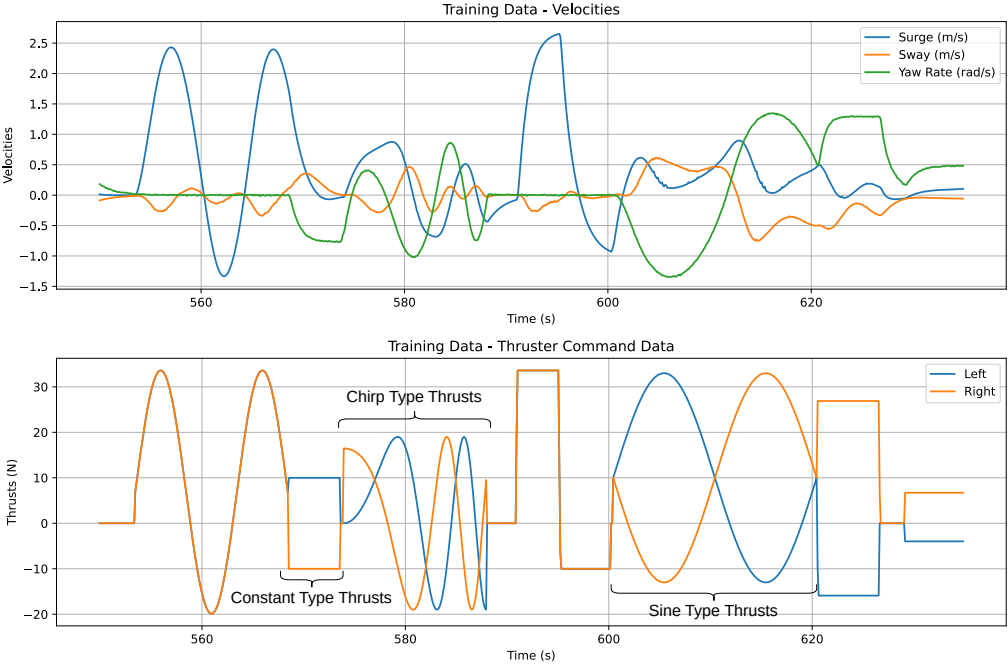


Figure 4.3: Training data snapshot over a time interval obtained using the event-based thruster command sequence.

In order to evaluate generalization, a separate validation dataset was generated using the same event families (sinusoidal, chirp, constant thrust, and break segments),

but with randomized event parameters and durations chosen independently from the training set. In other words, the validation trajectory keeps the same *structure* of excitations while changing the specific numerical characteristics so that model performance is measured on unseen input realizations rather than a replay of the training sequence. The total validation duration is 180.0 s, corresponding to 1800 samples at  $T_s = 0.1$  s. The validation sequence and the resulting velocity response are illustrated in Fig. 4.4.

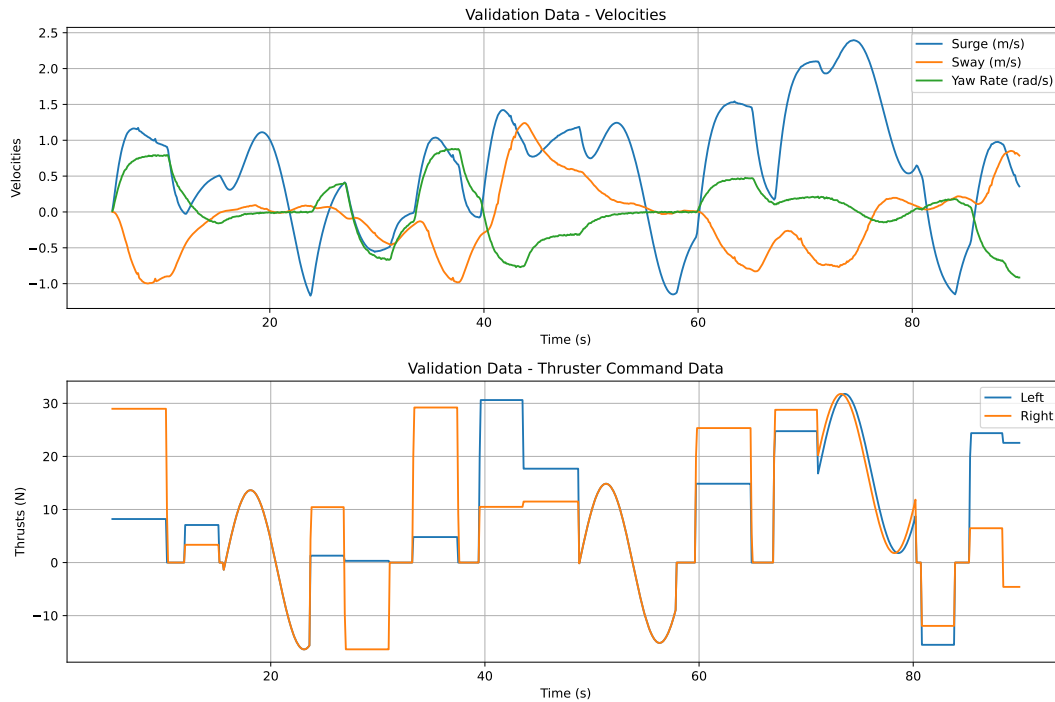


Figure 4.4: Validation data snapshot over a time interval obtained using an independently randomized event-based command sequence.

### 4.3 Fossen System Identification

The nonlinear system identification procedure based on the discrete time Fossen model given in (3.1). The training data used for parameter estimation were obtained as described in Section 4.1.

The dataset consists of thruster command inputs and vessel odometry measurements sampled at a nominal frequency of 10 Hz. Minor time-stamp mismatches were ob-

served between the input and output signals. Since the thruster commands exhibit an impulsive, piecewise-constant behavior, the velocity measurements were linearly interpolated onto the thruster command time stamps, allowing the inputs and outputs to be treated as time-aligned. The synchronized signals were then assembled into a MATLAB `iddata` object for training.

The model structure follows the discrete Fossen equations in (3.1) and employs a gray-box formulation. The physical structure of the dynamics is fixed, while 13 parameters, grouped in the vectors  $\alpha$ ,  $\beta$ , and  $\gamma$ , are estimated from data. The nonlinear gray-box model was implemented using MATLAB's `idnlgrey` framework, referenced as [45]. Parameter estimation was performed using the `fmincon` optimizer [46]. Due to the use of finite-difference gradients and evaluation of the cost function, the optimization required a large number of model evaluations; therefore, the model equations were implemented in C and compiled as a MATLAB executable (MEX) function to improve computational efficiency. No additional noise was added to the training data.

The final identified parameter values of the discrete-time Fossen model are listed in Table 4.2.

Table 4.2: Identified Fossen discrete-time model parameters.

Parameter	Value
$\alpha_1$	0.947464
$\alpha_2$	-0.0103382
$\alpha_3$	0.0928303
$\alpha_4$	0.0000317
$\alpha_5$	0.00324865
$\beta_1$	0.943917
$\beta_2$	-0.0182524
$\beta_3$	-0.103244
$\gamma_1$	0.947303
$\gamma_2$	-0.0441524
$\gamma_3$	0.000520024
$\gamma_4$	-0.00250532
$\gamma_5$	-0.00339975

The identified model was evaluated on the training dataset. The fit percentages for each motion component are summarized in Table 4.3. The overall mean squared error (MSE), computed jointly over surge, sway, and yaw dynamics, was 0.007278. These results indicate that the model captures the dominant vessel dynamics with high accuracy, with slightly reduced performance in sway due to stronger nonlinear coupling effects.

Table 4.3: Training data fit percentages for the nonlinear gray-box Fossen model. The overall mean squared error (MSE), computed jointly over surge, sway, and yaw dynamics, is 0.007278.

State	Fit (%)
Surge	92.67
Sway	85.67
Yaw Rate	94.09

## 4.4 Neural Network System Identification Results

This section reports the identification performance of the two neural network models introduced in Chapter 2: the multi-step MLP (MS-MLP) (Fig. 3.1) and the Seq2Seq LSTM (Fig. 3.2). Both models are trained to predict the future body-frame velocity trajectory from measured velocity histories and thruster inputs.

### 4.4.1 Data Synchronization and Normalization

As in the Fossen identification workflow (Section 4.3), the velocity measurements and the thruster commands are first time-aligned. Neural networks are sensitive to input scaling, and the states and inputs in this problem have different physical units and ranges. Moreover, both network architectures use  $\tanh(\cdot)$  nonlinearities; therefore, scaling inputs and targets to a bounded and approximately symmetric interval (which is  $[-1, +1]$ ) improves numerical conditioning and training stability.

We apply an affine scaling based on the known physical limits of the simulated vessel and thrusters. Table 4.4 summarizes the limits used for normalization.

Table 4.4: Physical limits used for normalization of velocities and thruster forces.

Signal	Min	Max
$u$ (surge, m/s)	-1.781828	2.679578
$v$ (sway, m/s)	-1.484772	1.492883
$r$ (yaw rate, rad/s)	-1.503650	1.504609
$F_1, F_2$ (thrust, N)	-19.88	33.6

For each scalar physical signal  $z$  with limits  $(z_{\min}, z_{\max})$ , we define the center and half-range as

$$\mu_z = \frac{z_{\max} + z_{\min}}{2}, \quad \sigma_z = \frac{z_{\max} - z_{\min}}{2}. \quad (4.3)$$

Normalization to approximately  $[-1, 1]$  is then performed by

$$\tilde{z} = \frac{z - \mu_z}{\sigma_z}. \quad (4.4)$$

In vector form, the normalization and denormalization mappings used throughout training and evaluation are

$$\tilde{\mathbf{z}} = (\mathbf{z} - \boldsymbol{\mu}_z) \oslash \boldsymbol{\sigma}_z, \quad (4.5)$$

$$\mathbf{y} = \tilde{\mathbf{y}} \odot \boldsymbol{\sigma}_y + \boldsymbol{\mu}_y, \quad (4.6)$$

where  $\oslash$  and  $\odot$  denote elementwise division and multiplication, respectively.

After normalization, the supervised input–output sequences are generated using the sliding-window procedure described in Section 3.2.2.1 for both the training and the validation datasets.

#### 4.4.2 Training Setup

Both networks use the same history lengths in this thesis:  $H_v = 5$  velocity samples and  $H_u = 5$  thrust samples. The Seq2Seq decoder includes an MLP output head with a single hidden layer of size 32.

The MS-MLP models are trained for 1000 epochs with batch size 1024 using the Adam optimizer and an initial learning rate of  $10^{-2}$ . The Seq2Seq LSTM models are trained for 2000 epochs with batch size 512, again using Adam with an initial learning rate of  $5 \times 10^{-2}$ . In both cases, a learning-rate scheduler reduces the step size during training to improve convergence. To promote generalization and avoid overfitting, we monitor the validation loss during training and save model checkpoints; all results reported below use the checkpoint that achieved the lowest validation loss.

#### 4.4.3 Comparison and Architecture Selection

Model performance is evaluated on the independent validation dataset described in Section 4.2. Errors are computed in *physical* units by denormalizing the network outputs using (4.6) and comparing predicted trajectories against the ground-truth validation velocities.

The training objective is the mean-squared error (MSE), consistent with (3.4). For a prediction horizon  $H$ , let  $\boldsymbol{\nu}_{k+j} \in \mathbb{R}^3$  be the ground-truth velocity at step  $j$  ahead of

time  $k$ , and let  $\hat{\mathbf{v}}_{k+j}$  be the corresponding model prediction. We compute the physical validation MSE as

$$\text{MSE}(H) = \frac{1}{N_{\text{seq}} H} \sum_{s=1}^{N_{\text{seq}}} \sum_{j=1}^H \|\mathbf{v}_{s,j} - \hat{\mathbf{v}}_{s,j}\|_2^2, \quad (4.7)$$

where  $N_{\text{seq}}$  is the number of evaluated validation sequences. For the neural models,  $\hat{\mathbf{v}}_{s,j}$  denotes the denormalized prediction in physical units.

To compare against the physics-based baseline, the discrete Fossen model is evaluated on the same validation sequences and the same metric in (4.7) is used. Since the Fossen model is a one-step predictor, multi-step predictions are generated recursively by feeding each predicted state back into the model over the horizon.

We compare prediction accuracy for three horizons:  $H = 5$  (0.5 s),  $H = 12$  (1.2 s), and  $H = 20$  (2.0 s). For each horizon, we sweep several network sizes (hidden width) and depths (number of layers) and report the validation MSE in physical units (i.e., using denormalized values). The resulting comparisons are shown in Figs. 4.5–4.7.

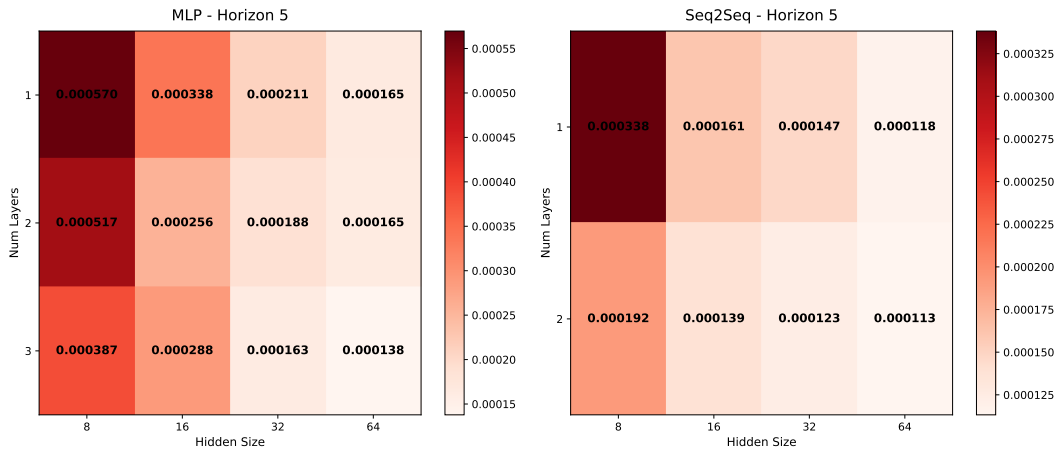


Figure 4.5: Validation physical MSE for MS-MLP and Seq2Seq LSTM models with a prediction horizon of  $H = 5$  steps (0.5 s).

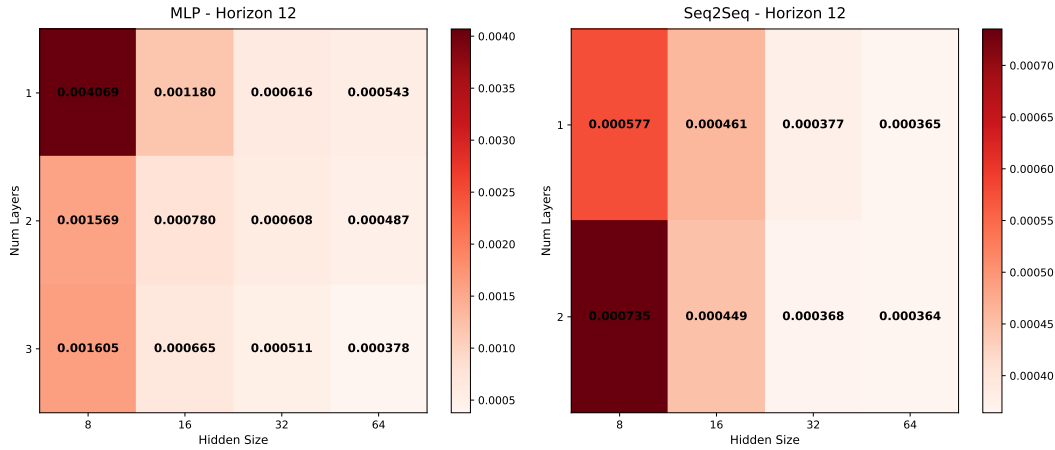


Figure 4.6: Validation physical MSE for MS-MLP and Seq2Seq LSTM models with a prediction horizon of  $H = 12$  steps (1.2 s).

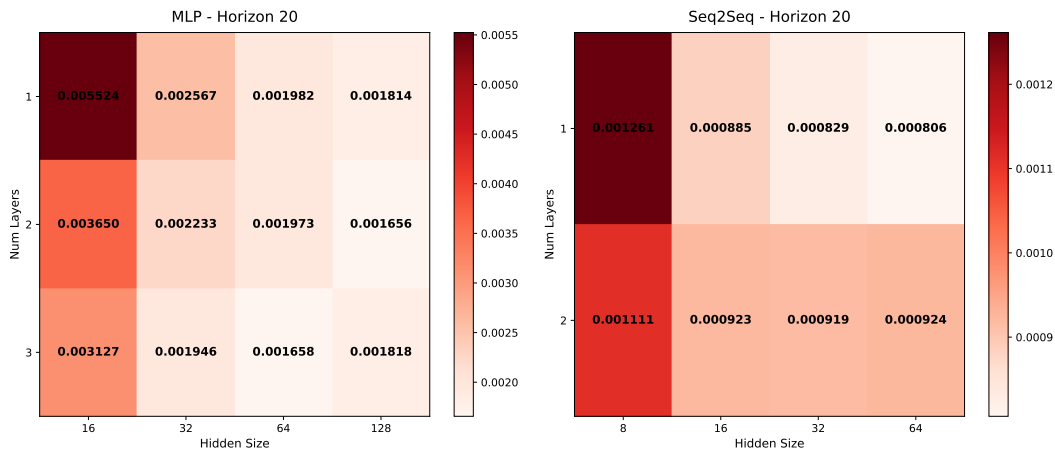


Figure 4.7: Validation physical MSE for MS-MLP and Seq2Seq LSTM models with a prediction horizon of  $H = 20$  steps (2.0 s).

Across all horizons, increasing the number of trainable parameters (larger hidden sizes and/or more layers) generally reduces validation error. However, for NMPC deployment, model complexity also affects real-time optimization cost: larger networks increase evaluation and differentiation time inside the NMPC solver and may reduce convergence robustness. Therefore, we **select** architectures that provide a favorable accuracy–complexity trade-off. Based on the heatmap sweeps, we **select** the MS-MLP with **2** hidden layers and hidden size **64**, and the Seq2Seq LSTM with **1**

encoder LSTM layer and **1** decoder LSTM layer with hidden size **32**, for the control experiments in the remainder of the thesis.

After selecting these final architectures, Table 4.5 summarizes the physical validation MSE values for the MS-MLP, Seq2Seq LSTM, and the recursive Fossen baseline under the same horizons and the same evaluation procedure in (4.7).

Table 4.5: Physical validation MSE for the **selected** MS-MLP and Seq2Seq LSTM architectures, compared against the recursive Fossen baseline (same validation dataset and horizon-wise metric).

<b>Horizon</b>	<b>Seq2Seq LSTM</b>	<b>MS-MLP</b>	<b>Fossen</b>
$H = 5$	0.000123	0.000165	0.001331
$H = 12$	0.000377	0.000487	0.003084
$H = 20$	0.000829	0.001973	0.004467

To further illustrate prediction quality beyond a single scalar error, we also report horizon-wise RMSE accumulation on the validation dataset for  $H = 5$ ,  $H = 12$ , and  $H = 20$ , and visualize representative predicted trajectories by selecting a random segment from the validation sequence. The RMSE growth curves and example trajectory comparisons are shown in Figs. 4.8–4.12 and Figs. 4.9–4.13, respectively.

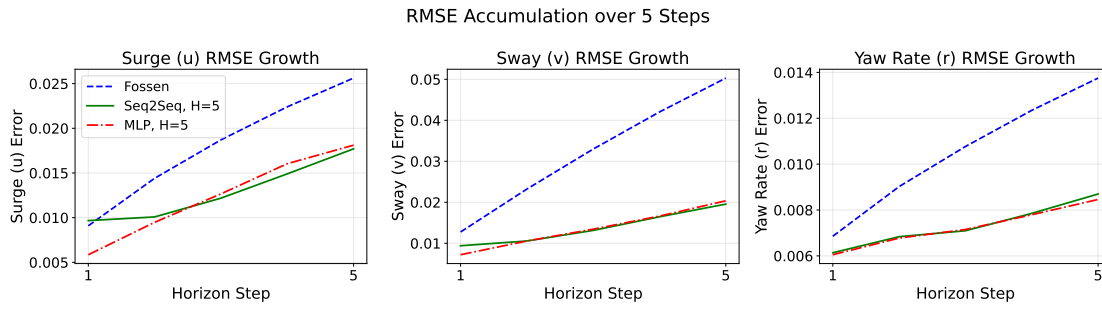


Figure 4.8: Horizon-wise RMSE accumulation on the validation dataset for  $H = 5$ .

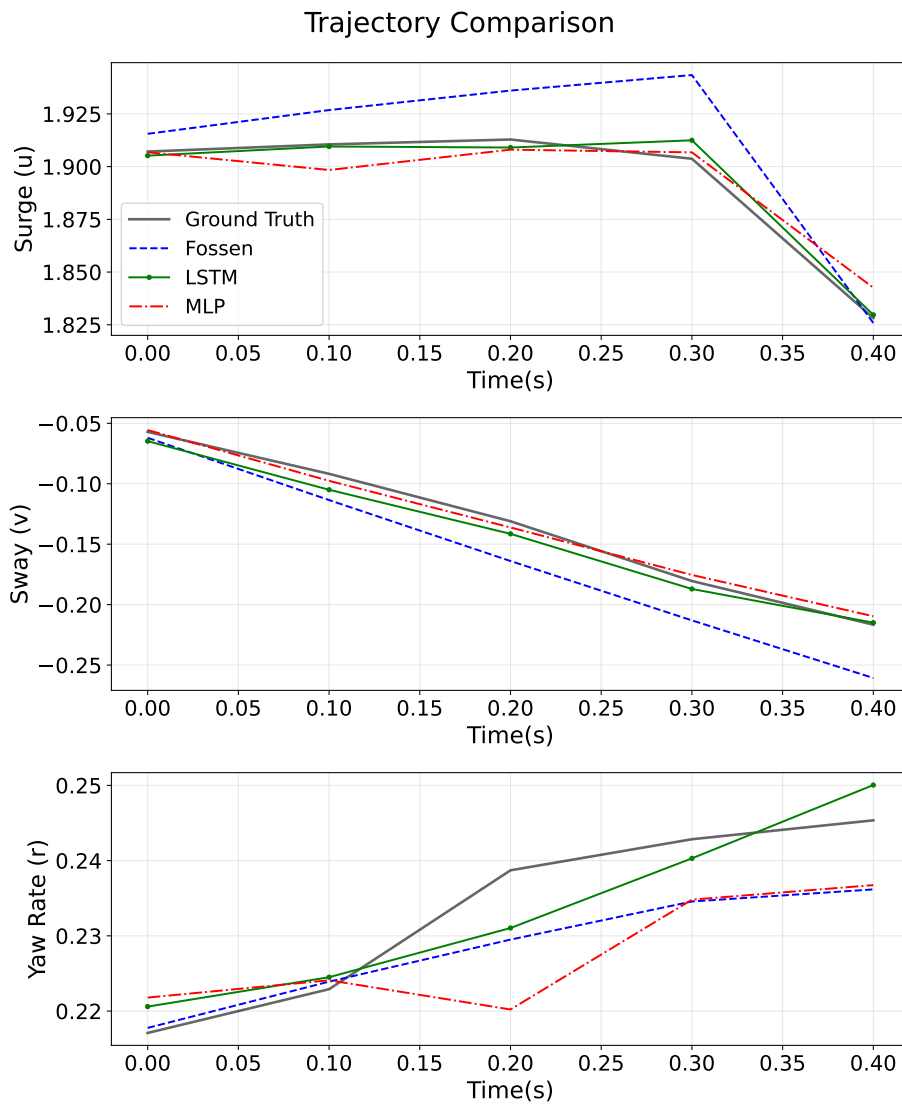


Figure 4.9: Example validation segment: ground-truth vs. predicted velocity trajectories for  $H = 5$ .

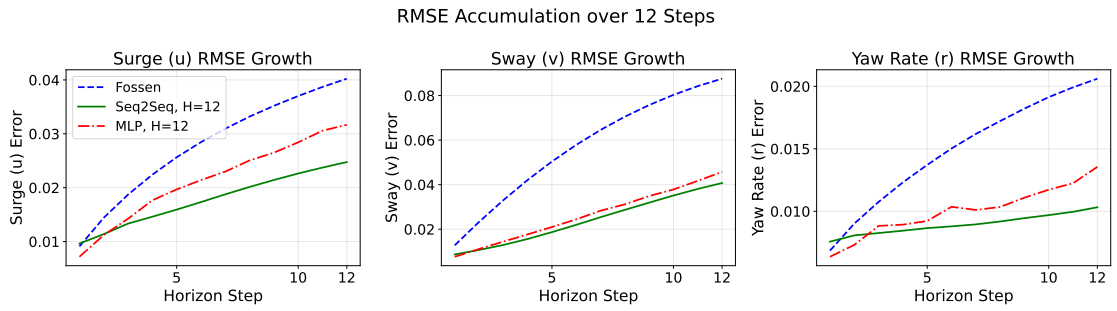


Figure 4.10: Horizon-wise RMSE accumulation on the validation dataset for  $H = 12$ .

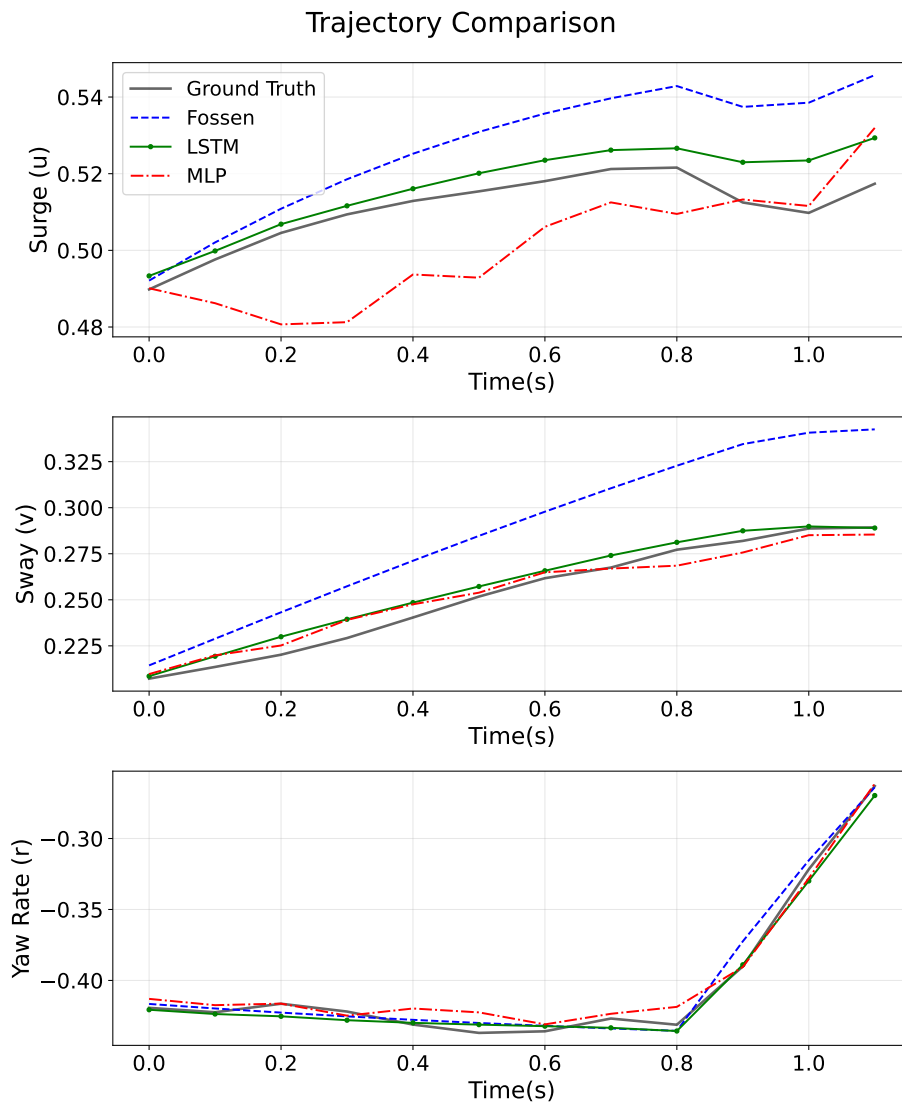


Figure 4.11: Example validation segment: ground-truth vs. predicted velocity trajectories for  $H = 12$ .

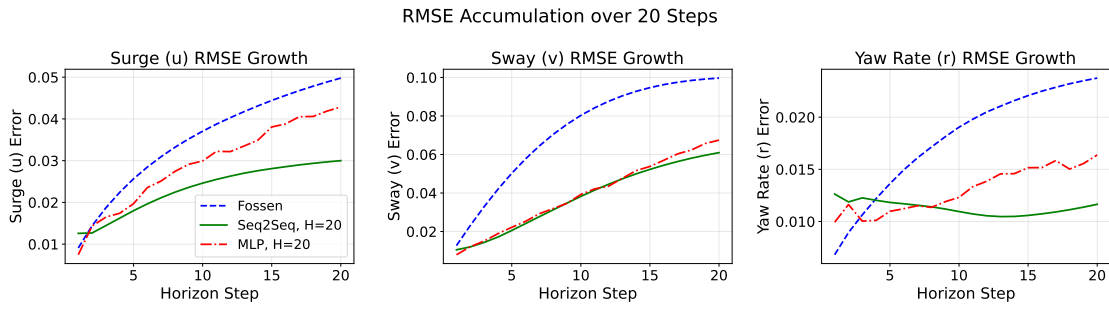


Figure 4.12: Horizon-wise RMSE accumulation on the validation dataset for  $H = 20$ .

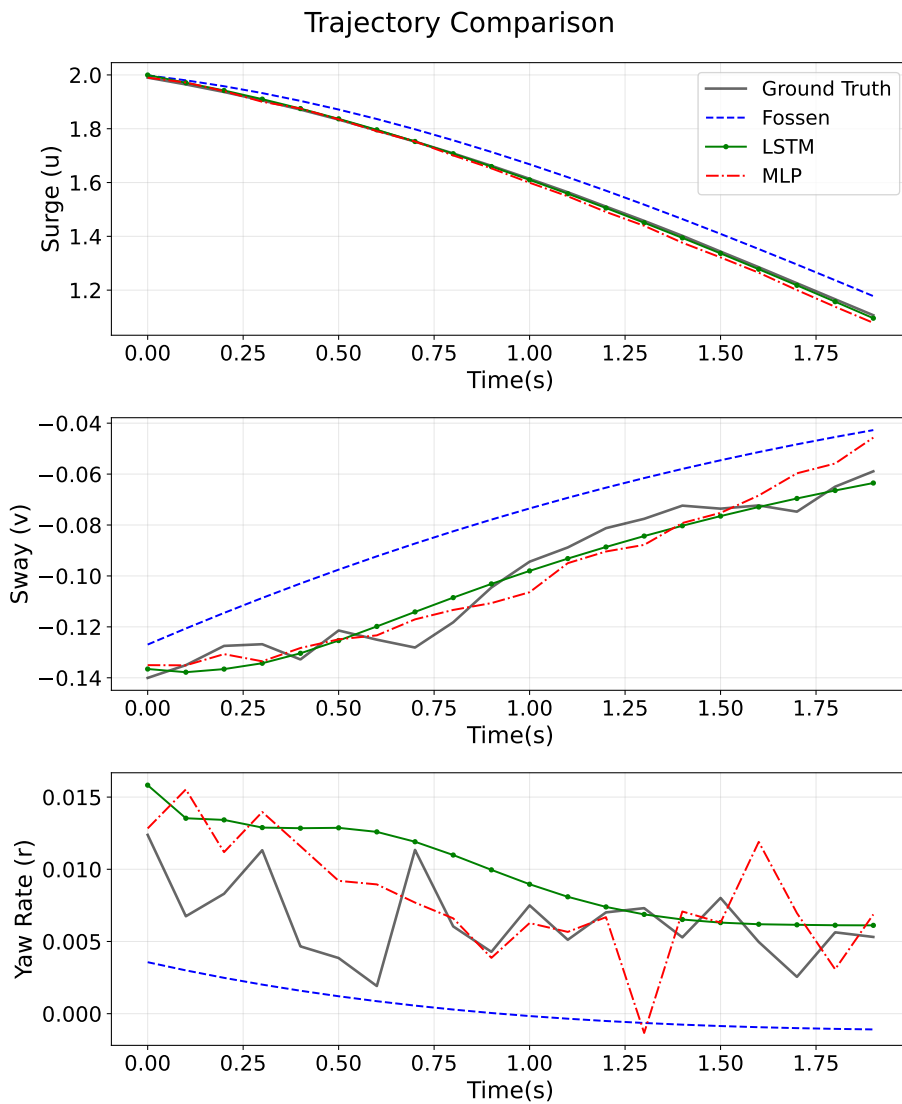


Figure 4.13: Example validation segment: ground-truth vs. predicted velocity trajectories for  $H = 20$ .

#### 4.4.4 Discussions

The results show that both neural models predict multi-step velocities more accurately than the recursive Fossen baseline on the validation set. Table 4.5 shows that the Seq2Seq LSTM achieves the lowest physical MSE for all horizons. The MS-MLP is close for short horizons, but its error increases faster as the horizon grows. This is also visible in the RMSE growth curves in Figs. 4.8–4.12, where the Seq2Seq model accumulates error more slowly while the Fossen model deviates substantially.

A likely reason is the prediction structure. The MS-MLP outputs the whole trajectory in one pass, so small inconsistencies can spread across future steps. The Seq2Seq decoder predicts step-by-step with shared recurrent weights, which better matches the causal rollout used in NMPC and tends to produce more consistent long-horizon predictions. The example trajectories in Figs. 4.9–4.13 support this trend.

Finally, the heatmaps in Figs. 4.5–4.7 show that larger networks usually reduce validation error, but they also increase computation. Since NMPC requires repeated model evaluation and differentiation, we select compact architectures that balance accuracy and real-time cost. These results motivate using the selected neural models for NMPC in the next chapter, while keeping the Fossen model as a physics-based reference.

## 4.5 Velocity Control Results

Table 4.5 indicates that the neural identification models (Seq2Seq LSTM and MS-MLP) achieve lower multi-step prediction errors than the identified discrete Fossen model. For NMPC, this accuracy is important, but it is not the only requirement. The identified model must also be *feasible as an NMPC prediction model*: it must allow a well-posed optimal control problem (OCP) that can be constructed and solved reliably in real time. In practice, this means the model should produce stable multi-step rollouts and should be time efficient.

In this section, we compare velocity NMPC performance using three prediction models: (i) the discrete Fossen model, (ii) the MS-MLP, and (iii) the Seq2Seq LSTM. All controllers use the same NMPC formulation in Section 3.3, sampling time  $\Delta t = 0.1$  s, and the same thruster bounds. We test three horizons,  $H \in \{5, 12, 20\}$ , for noiseless tracking, and we use  $H = 20$  for the noisy tracking experiments.

All experiments in this section use the same stage-cost weighting matrices in the NMPC objective (Section 4.5):

$$Q = \text{diag}(10000, 10000, 10000), \quad R = \text{diag}(0.5, 0.5),$$

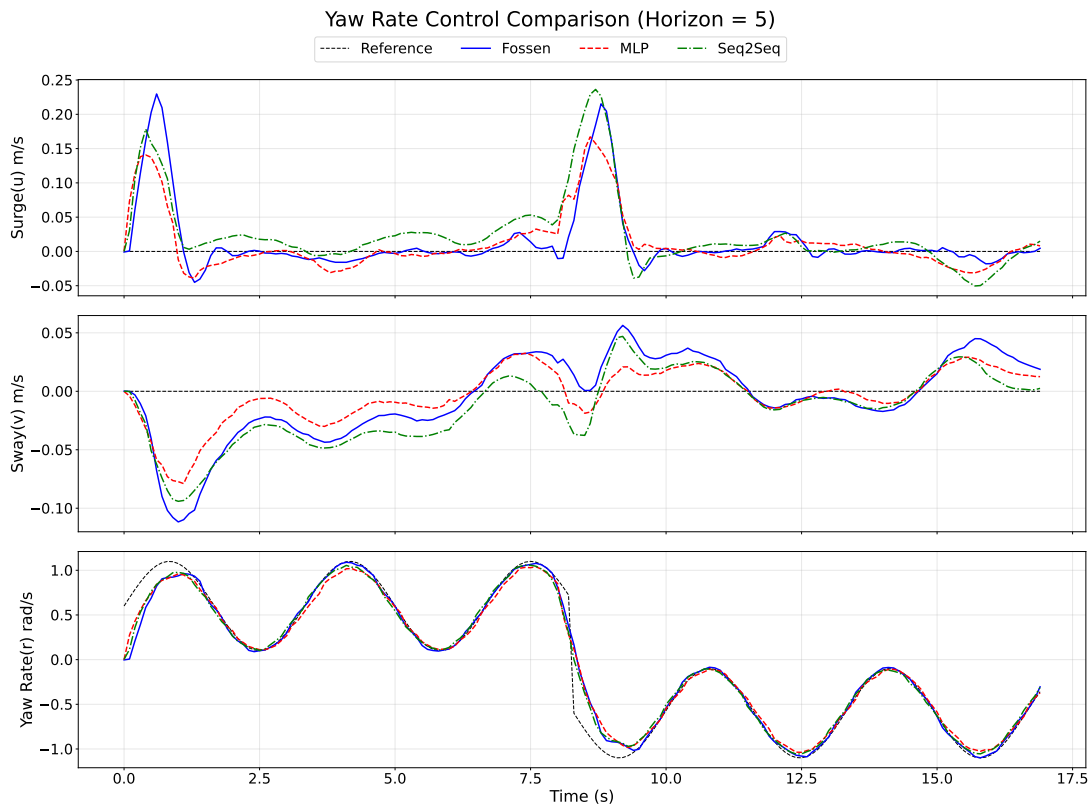
where  $Q$  penalizes the body-frame velocity tracking error  $(u, v, r) - (u_{\text{ref}}, v_{\text{ref}}, r_{\text{ref}})$  and  $R$  penalizes the control inputs (thrusters). Consequently, in experiments where only one velocity component is commanded (e.g., surge tracking), the remaining reference components are set to zero, so the NMPC tracks the commanded velocity while regulating the non-commanded velocities toward 0.

### 4.5.1 Yaw-Rate Tracking

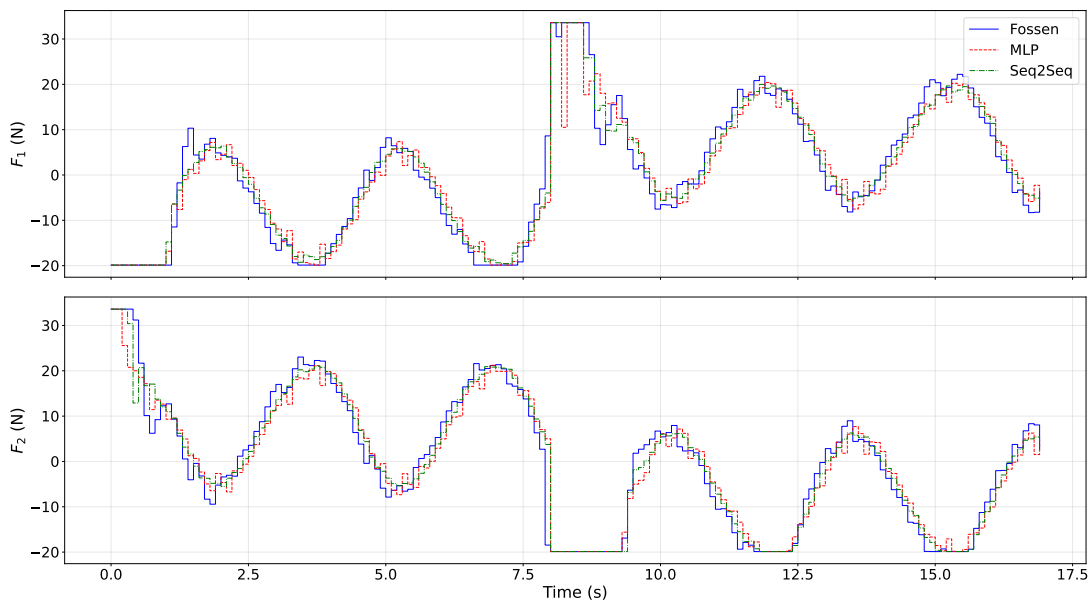
Figures 4.14–4.16 show yaw-rate ( $r$ ) tracking without measurement noise. The total simulation is split into two halves with different sine-wave references to test both positive and negative bias conditions. In the first half, the yaw-rate reference is a sinusoid with bias 0.6, amplitude 0.5, and frequency 0.3 Hz. In the second half, the same sinusoid is applied with bias  $-0.6$  (amplitude 0.5, frequency 0.3 Hz).

Across horizons, the neural NMPC controllers generally track the reference more

closely than the Fossen-based NMPC. For the short horizon ( $H = 5$ ), the difference is limited because the controller mainly reacts to near-term error. For longer horizons ( $H = 12$  and  $H = 20$ ), the advantage of improved multi-step prediction becomes clearer in the closed loop.

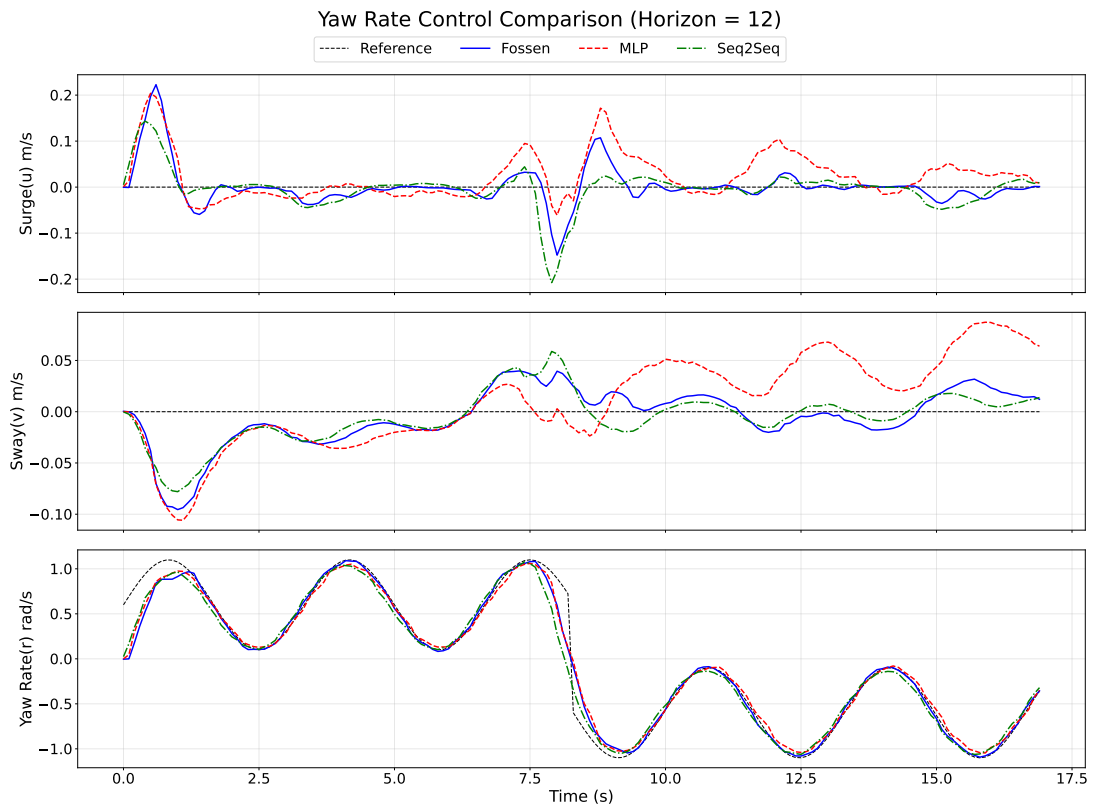


(a) Body-frame velocity responses ( $u, v, r$ ).

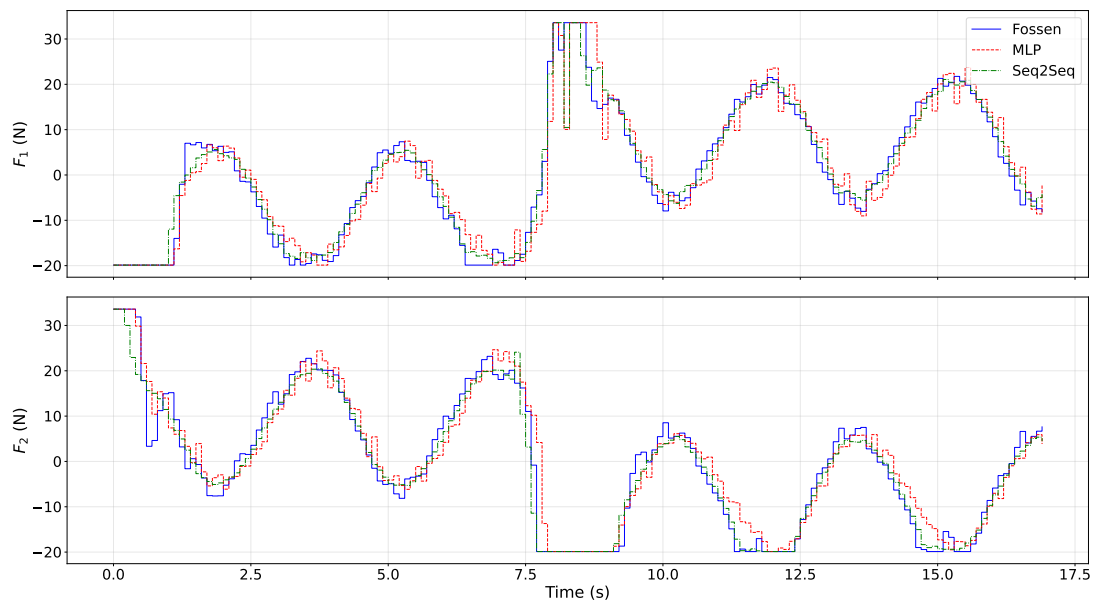


(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.14: Yaw-rate tracking NMPC results without measurement noise for horizon  $H = 5$  (0.5 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.

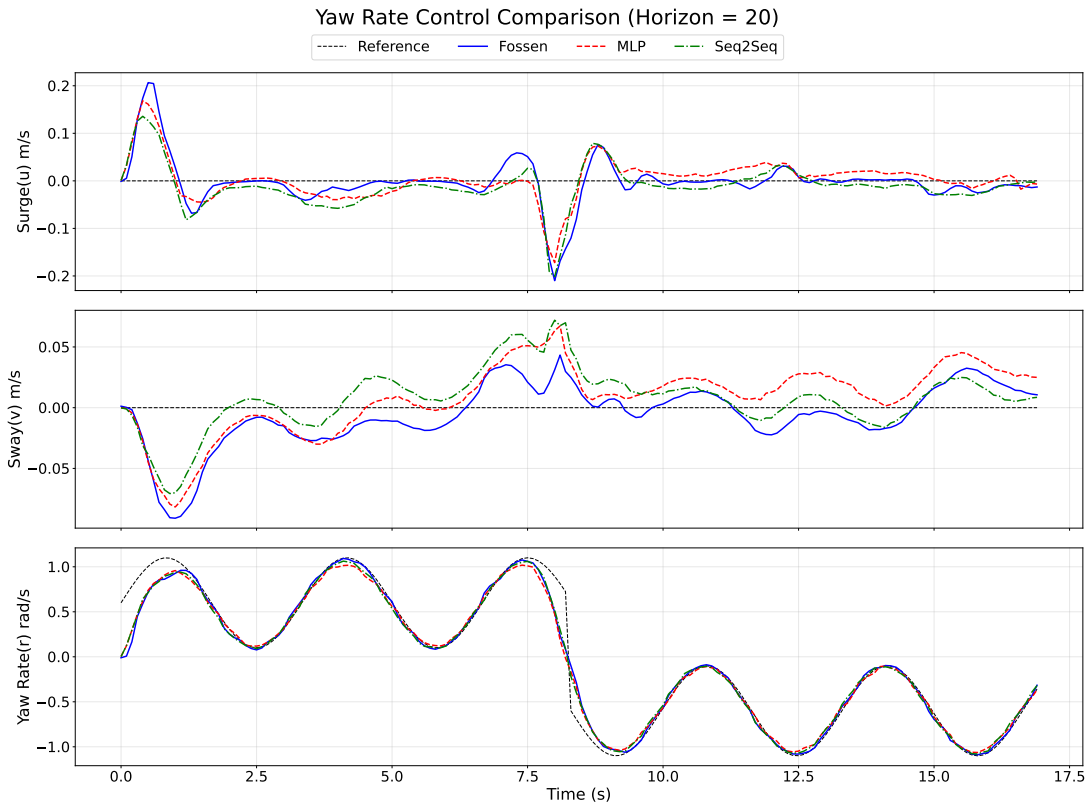


(a) Body-frame velocity responses ( $u, v, r$ ).

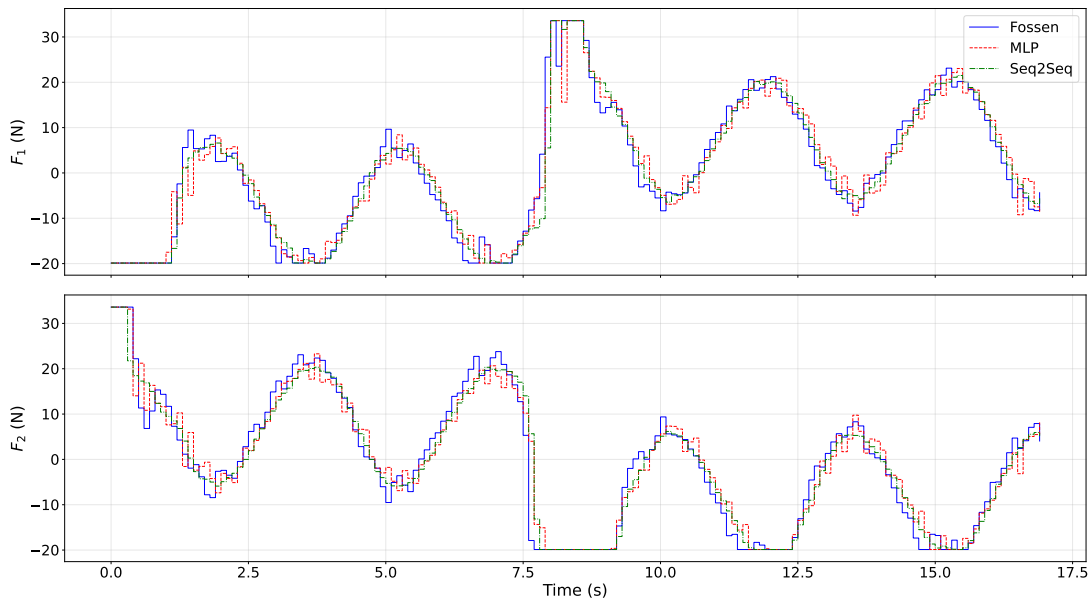


(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.15: Yaw-rate tracking NMPC results without measurement noise for horizon  $H = 12$  (1.2 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.



(a) Body-frame velocity responses ( $u, v, r$ ).



(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.16: Yaw-rate tracking NMPC results without measurement noise for horizon  $H = 20$  (2.0 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.

Table 4.6 reports the mean and worst-case computation time per NMPC update (in ms), including NMPC preparation (state/reference/bounds update and warm-start/initial guess; for Seq2Seq the encoder pass) and the SQP\_RTI solver time.

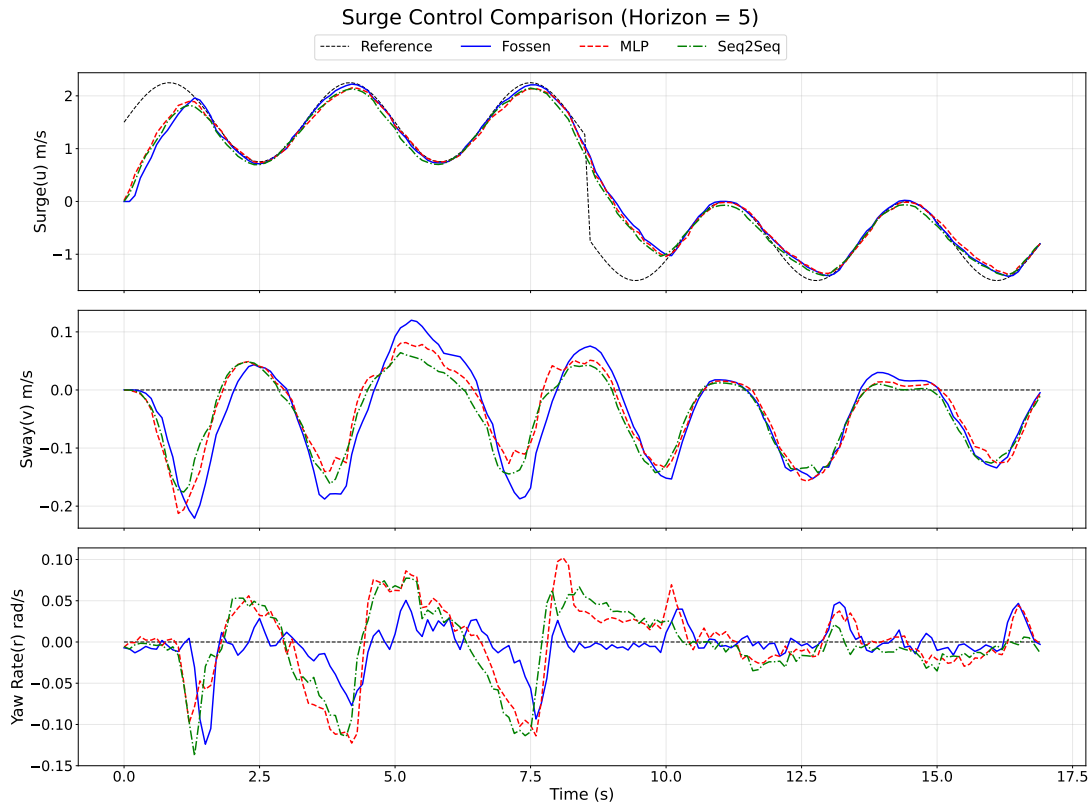
Table 4.6: NMPC computation time for noiseless yaw-rate tracking (**ms**): mean and maximum computation time per control update, including NMPC preparation (state/reference/bounds update and warm start; for Seq2Seq the encoder pass) and the SQP\_RTI solve time.

Model	$H = 5$		$H = 12$		$H = 20$	
	Mean	Max	Mean	Max	Mean	Max
Fossen	1.029	2.528	1.825	3.554	2.778	4.552
MS-MLP	1.191	2.613	1.702	2.758	2.261	3.602
Seq2Seq	2.571	4.145	5.110	7.631	8.417	10.402

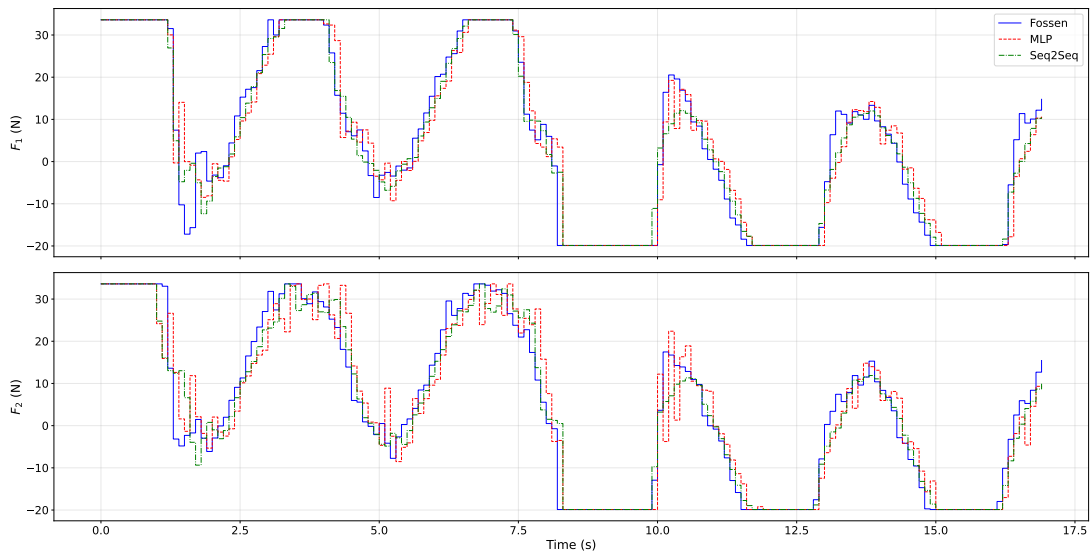
#### 4.5.2 Surge Tracking (Noiseless)

Figures 4.17–4.19 show surge velocity ( $u$ ) tracking without measurement noise. As in yaw-rate tracking, the simulation is split into two halves with different biases to excite both positive and negative operating regions. In the first half, the surge reference is a sinusoid with bias 1.5, amplitude 0.75, and frequency 0.3 Hz. In the second half, the reference is switched to bias  $-0.75$  with amplitude 0.75 and frequency 0.3 Hz.

The results follow the same trend: neural prediction models inside NMPC typically reduce tracking error compared to the Fossen-based NMPC, and the difference is more visible for longer horizons.

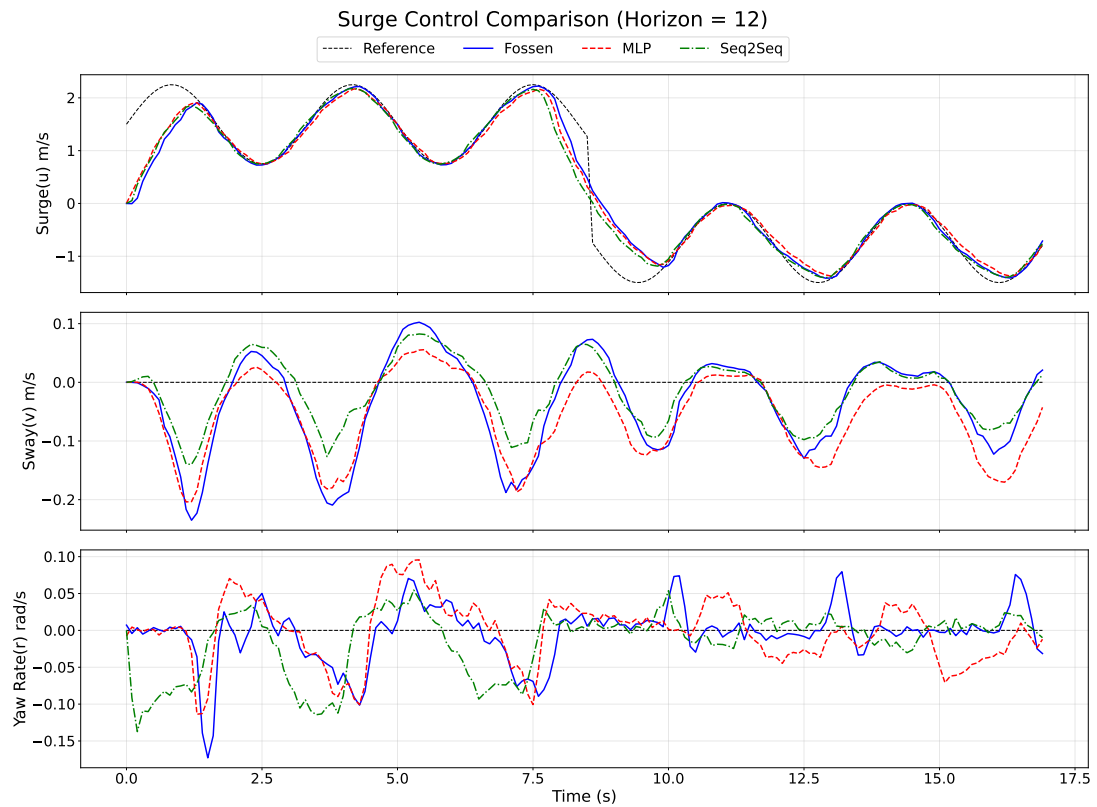


(a) Body-frame velocity responses ( $u, v, r$ ).

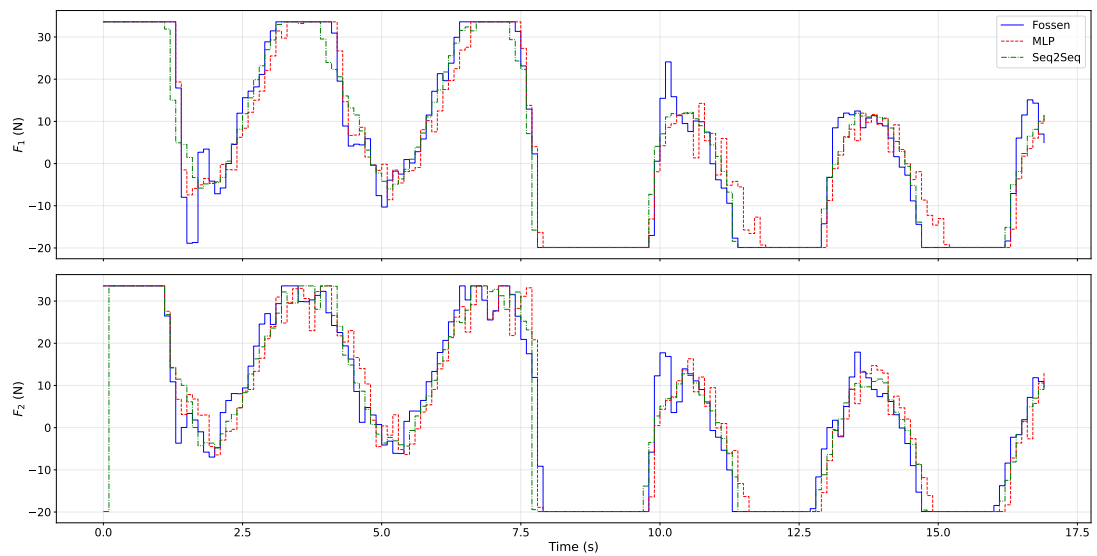


(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.17: Surge-tracking NMPC results without measurement noise for horizon  $H = 5$  (0.5 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.

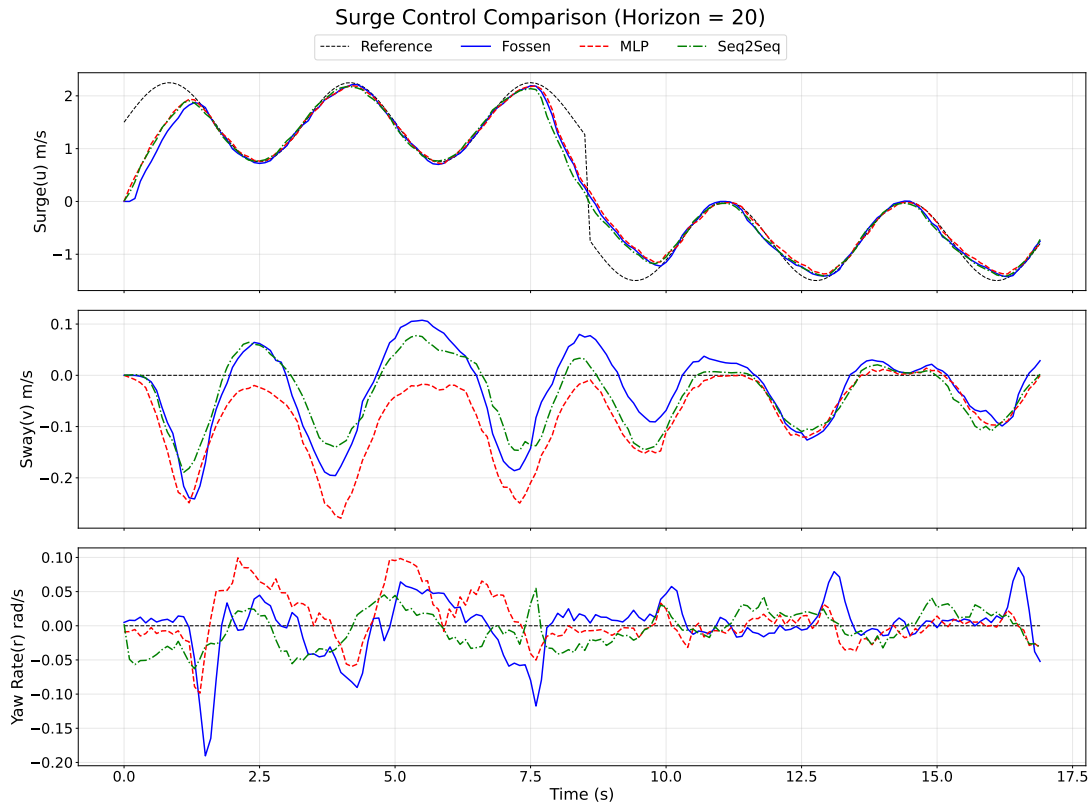


(a) Body-frame velocity responses ( $u, v, r$ ).

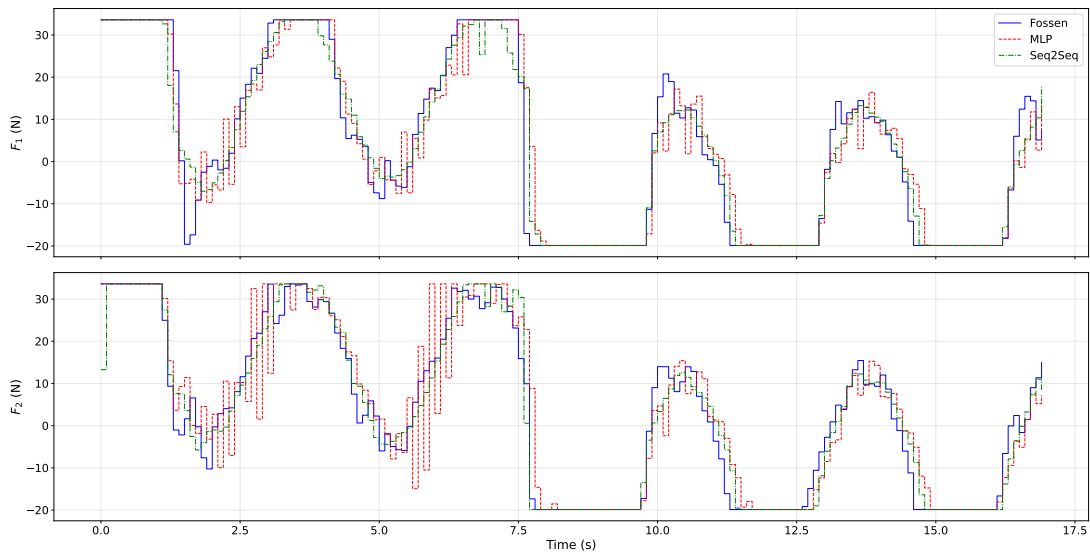


(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.18: Surge-tracking NMPC results without measurement noise for horizon  $H = 12$  (1.2 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.



(a) Body-frame velocity responses ( $u, v, r$ ).



(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.19: Surge-tracking NMPC results without measurement noise for horizon  $H = 20$  (2.0 s): comparison of Fossen-, MS-MLP-, and Seq2Seq-based prediction models.

Since the NMPC formulation and real-time implementation (preparation steps and SQP\_RTI solver) are identical, the noiseless surge-tracking experiments exhibit nearly the same computation times as yaw-rate tracking; therefore, Table 4.6 is representative for the surge case as well.

### 4.5.3 Velocity Tracking with Noisy Measurements

To evaluate robustness, we add measurement noise to all velocity measurements (surge, sway, and yaw). The noise is zero-mean Gaussian with standard deviation  $\sigma = 0.1$ . In these noisy experiments, we only report the  $H = 20$  (2.0 s) horizon case.

Before comparing NMPCs performance under noise, we retrain the neural models using training data augmented with the same type of noise (zero-mean Gaussian,  $\sigma = 0.1$ ). This is done to reduce the mismatch between training and deployment conditions. The Fossen model is not retrained under noisy conditions. Because it is a fixed-structure, memoryless one-step model that does not rely on an input history window, and augmenting its training data with noise would not improve robustness and would only degrade its fit to the true system dynamics.

For the noisy surge-velocity test, the surge reference is a sinusoid with bias 0.5, amplitude 2.0, and frequency 0.2 Hz. For the noisy yaw-rate test, the yaw-rate reference is a sinusoid with bias 0, amplitude 1.5, and frequency 0.2 Hz.

Figures 4.20 and 4.22 show the body-frame velocity responses ( $u, v, r$ ) and the corresponding control actions for the noisy surge-focused and yaw-focused experiments, respectively. Figures 4.21 and 4.23 use the same experiments, but they plot only the *actual* surge or yaw-rate signals (not the noisy measured signals) against the reference for easier comparison.

To quantify tracking performance, we use the fit percentage between a reference signal and the corresponding *actual* (noise-free) closed-loop response. For a generic reference signal vector  $y_{\text{ref}}$  and actual response vector  $y$ , the fit is computed as,

$$\text{Fit}_y(\%) = 100 \left( 1 - \frac{\|y_{\text{ref}} - y\|_2}{\|y_{\text{ref}} - \bar{y}_{\text{ref}}\|_2} \right), \quad (4.8)$$

where  $\bar{y}_{\text{ref}}$  denotes the mean of the reference over the evaluation window  $\mathcal{K}$ ,

$$\bar{y}_{\text{ref}} = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} y_{\text{ref}}[k]. \quad (4.9)$$

For yaw-rate tracking, the reference and actual signals are defined as

$$y_{\text{ref}} = r_{\text{ref}}, \quad y = r,$$

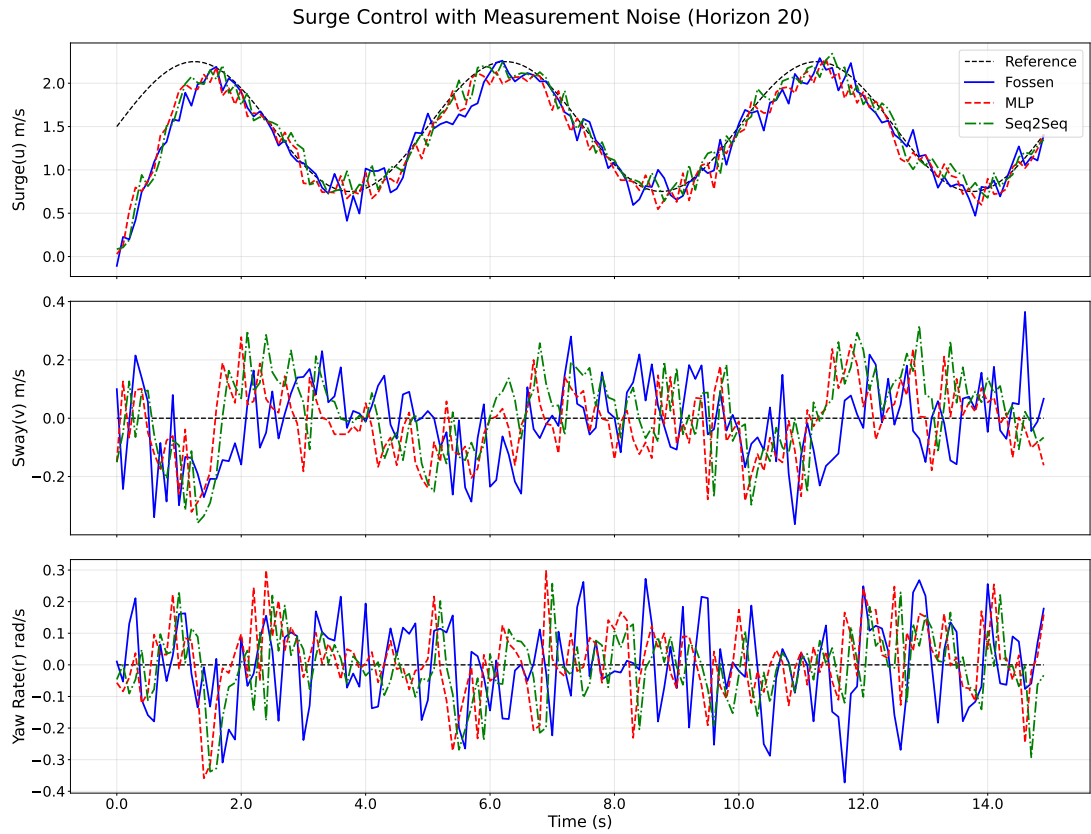
whereas for surge-velocity tracking,

$$y_{\text{ref}} = u_{\text{ref}}, \quad y = u.$$

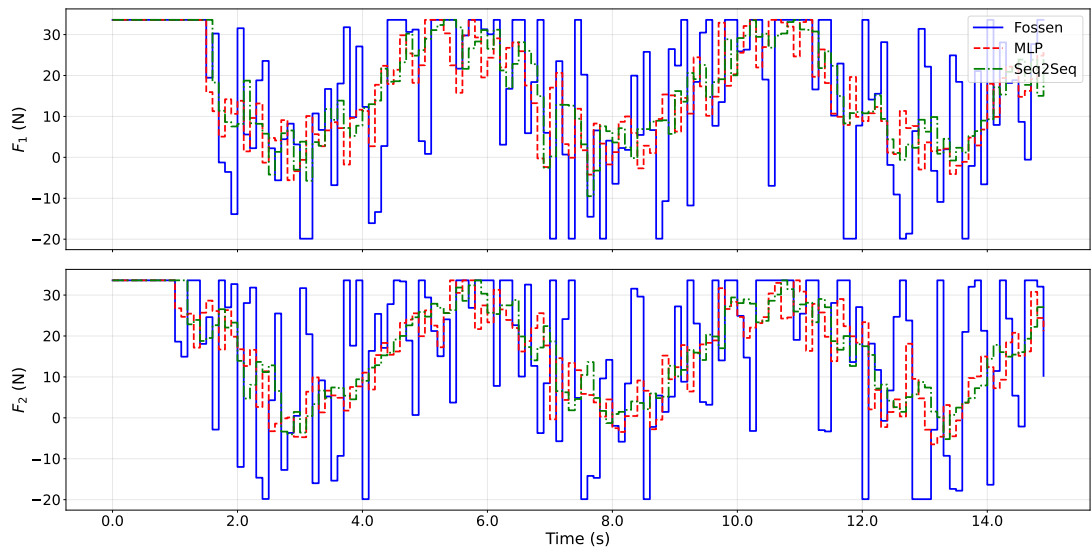
For the surge-focused noisy experiment, the evaluation window is chosen as

$$\mathcal{K} = \{k \mid t_k \geq 2 \text{ s}\},$$

in order to exclude initial transient effects. For the yaw-rate-focused noisy experiment, the evaluation window  $\mathcal{K}$  spans the full simulation interval.



(a) Body-frame velocity tracking ( $u, v, r$ ) with noisy measurements.



(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.20: Velocity NMPC with noisy measurements for horizon  $H = 20$  (2.0 s): surge-focused test comparing Fossen-, MS-MLP-, and Seq2Seq-based prediction models.

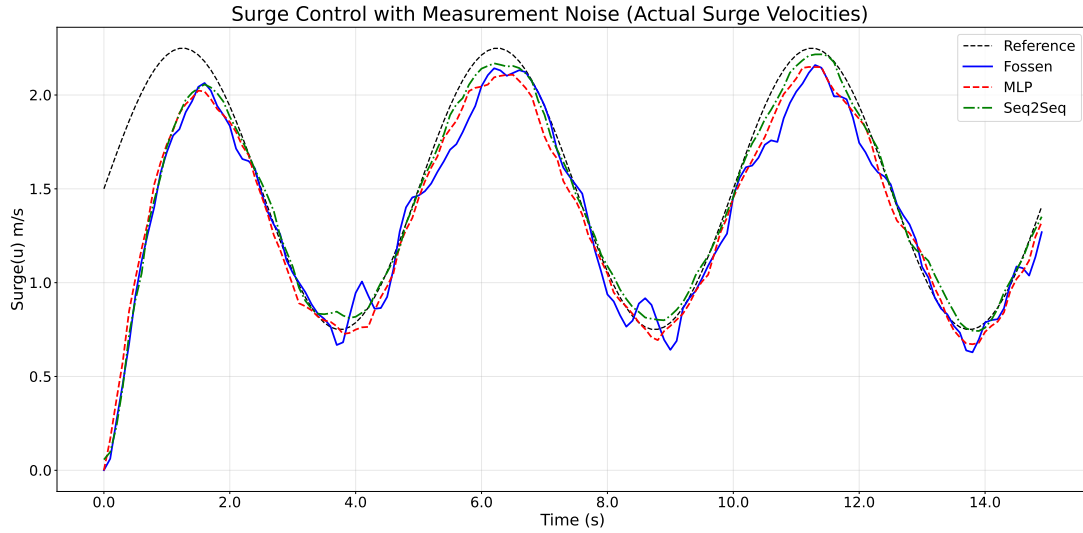
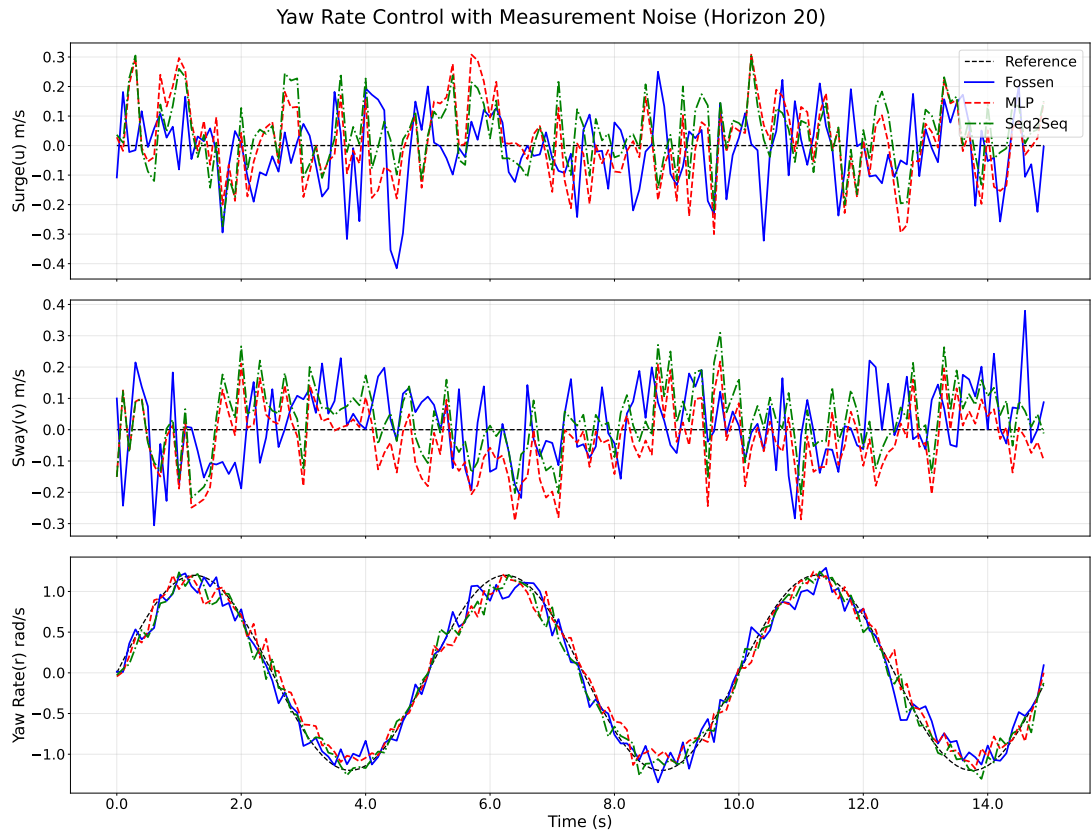


Figure 4.21: Surge ( $u$ ) velocity tracking under noisy measurements for horizon  $H = 20$  (2.0 s): actual (noise-free) surge response from the same experiment as Fig. 4.20, compared against the reference for Fossen-, MS-MLP-, and Seq2Seq-based NMPC.

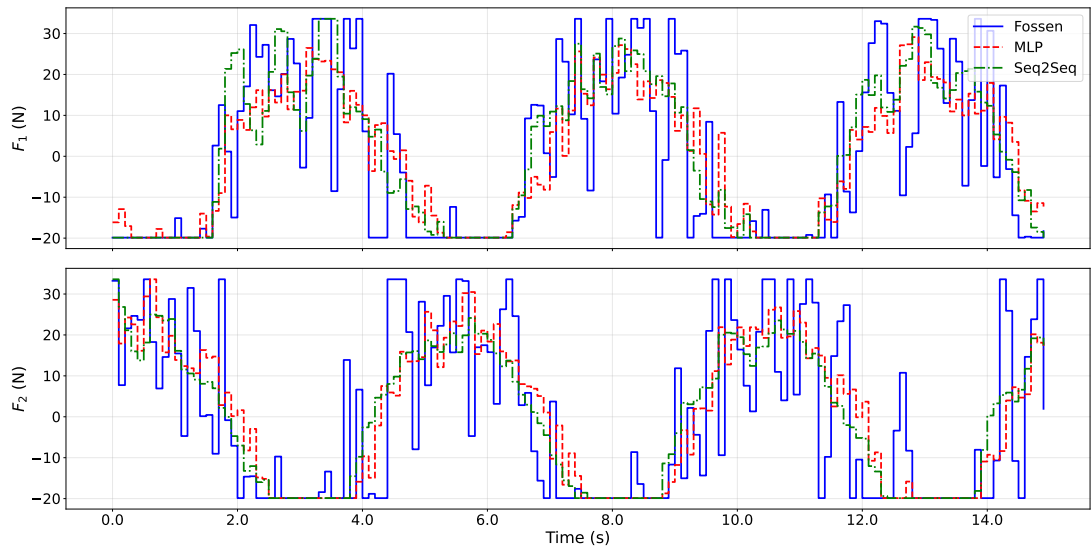
The fit values for surge tracking with different prediction models are shown in Table 4.7.

Table 4.7: Surge-velocity tracking performance under noisy measurements ( $H = 20$ , 2.0 s): fit percentage  $\text{Fit}_u$  using Eq. (4.8), computed between the reference  $u_{\text{ref}}$  and the actual (noise-free) surge velocity  $u$  over  $t \geq 2$  s (i.e.,  $\mathcal{K} = \{k \mid t_k \geq 2 \text{ s}\}$ ).

Prediction model	Fit (%)
Fossen	77.32
MS-MLP	83.33
Seq2Seq	90.56



(a) Body-frame velocity tracking ( $u, v, r$ ) with noisy measurements.



(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.22: Velocity NMPC with noisy measurements for horizon  $H = 20$  (2.0 s): yaw-rate-focused test comparing Fossen-, MS-MLP-, and Seq2Seq-based prediction models.

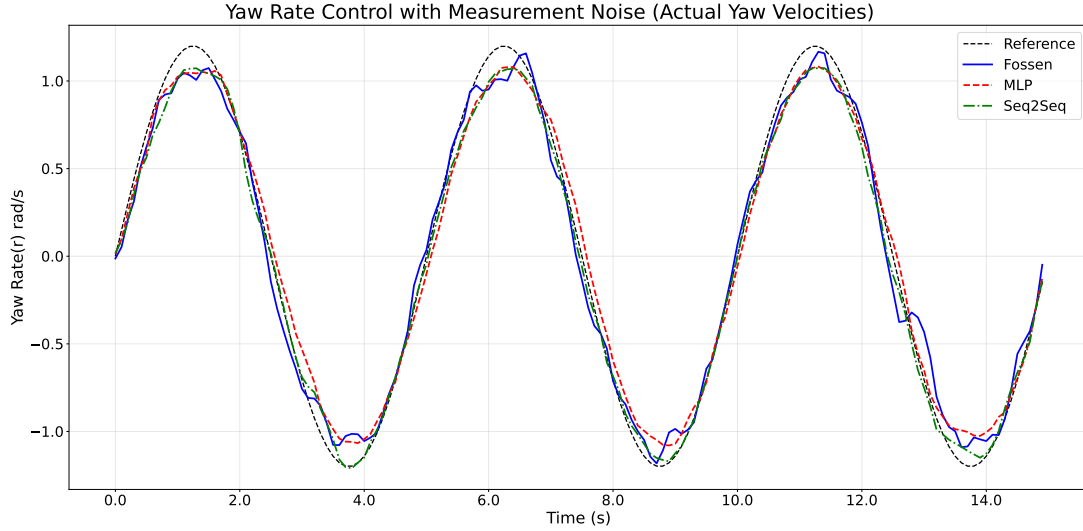


Figure 4.23: Yaw-rate ( $r$ ) velocity tracking under noisy measurements for horizon  $H = 20$  (2.0 s): actual (noise-free) yaw-rate response from the same experiment as Fig. 4.22, compared against the reference for Fossen-, MS-MLP-, and Seq2Seq-based NMPC.

Table 4.8: Yaw-rate tracking performance under noisy measurements ( $H = 20$ , 2.0 s): fit percentage  $\text{Fit}_r$  using Eq. (4.8), computed between the yaw-rate reference  $r_{\text{ref}}$  and the actual (noise-free) yaw rate  $r$  over the full simulation interval.

Prediction model	Fit (%)
Fossen	87.88
MS-MLP	87.89
Seq2Seq	90.57

The noisy tracking experiments use the same horizon  $H = 20$ , so their per-step computation times match the  $H = 20$  results reported in Table 4.6.

#### 4.5.4 Discussions

Using the selected prediction models, we evaluated surge and yaw-rate tracking for horizons  $H \in \{5, 12, 20\}$  under noiseless measurements. The reference signals include bias changes and relatively high amplitudes, so the tracking tasks are not triv-

ial. However, in all noiseless experiments and for all horizons, the closed-loop responses of the three NMPC variants are very close to each other (Figs. 4.17–4.19 and Figs. 4.14–4.16). This suggests that, for these scenarios, all three prediction models provide sufficiently accurate short-horizon rollouts, so the NMPC optimizer converges to very similar input sequences even when the internal models differ.

Since noiseless tracking performance is similar, computation time becomes an important differentiator. Table 4.6 shows that MS-MLP and Fossen achieve comparable runtimes, while Seq2Seq is consistently slower and its computation time increases strongly with the horizon. This is expected: MS-MLP predicts the full horizon in a single forward pass, so increasing  $H$  has a limited impact on per-step NMPC time. Seq2Seq, in contrast, requires a recursive decoder rollout across the horizon and also includes an encoder pass during the preparation step for the solver, which increases both evaluation and differentiation cost inside the SQP\_RTI loop.

Because the noiseless results are close, we added a non-negligible level of measurement noise ( $\sigma = 0.1$ ) to stress robustness. In both noisy surge and noisy yaw-rate tracking, the neural NMPC controllers outperform the Fossen-based NMPC in terms of fit (Tables 4.7 and 4.8). The difference is most visible in the noisy surge test, where Seq2Seq achieves the highest fit and MS-MLP also improves over Fossen. In addition, the thrust commands produced by the Fossen-based NMPC become noticeably more irregular under noise (Figs. 4.20 and 4.22), while the neural-model NMPC solutions remain smoother. Overall, these results indicate that the learned predictors provide more consistent multi-step predictions under noisy measurements, which improves closed-loop tracking and yields more stable control actions, at the cost of increased computation for the Seq2Seq model.

## 4.6 Position Control Results

In this chapter, position NMPC is evaluated on two waypoint maps: a star-shaped path and an S-shaped path. All position-control experiments use the same prediction horizon ( $H = 20$ ), the same neural-network structures as in the velocity-control experiments, and noisy velocity measurements with  $\sigma = 0.1$ .

### 4.6.1 Star-Shaped Map

This position-control scenario uses a star-shaped waypoint map. The mission contains 10 waypoints tracked sequentially. At each NMPC update, the controller tracks the active waypoint. When the USV enters the 4 m acceptance radius (green circle), the target switches to the next waypoint.

For this map, the stage weights are

$$\mathbf{Q} = \text{diag}(100, 100, 1600, 1600, 5000, 0.1, 0.1), \quad \mathbf{R} = \text{diag}(0.5, 0.5)$$

The first two entries penalize  $(x, y)$  tracking error. The next two penalize yaw error via  $(\cos \psi, \sin \psi)$ . For velocity terms,  $u$  is weighted more strongly to support forward speed, while  $v$  and  $r$  are given small weights.

The reference outputs  $\mathbf{y}_x^{\text{ref}}$  and  $\mathbf{y}^{\text{ref}}$  are formed as in Section 3.4 by stacking the waypoint-based position reference, yaw reference, and velocity references.

For the star map, the yaw reference is a line-of-sight heading toward the active waypoint:

$$\psi_{\text{ref}}[k] = \text{atan2}(y_{\text{ref}}[k] - y[k], x_{\text{ref}}[k] - x[k]).$$

To maintain progress, a constant forward-speed reference is used:  $u_{\text{ref}} = 1.5$  m/s, with  $v_{\text{ref}} = 0$  and  $r_{\text{ref}} = 0$ . At each NMPC update, the waypoint-based  $(x_{\text{ref}}, y_{\text{ref}}, \psi_{\text{ref}})$  values are held constant over the prediction horizon  $H = 20$ .

Figures 4.24 and 4.25 show snapshots of the motion. Figure 4.26 compares the trajectories for the three prediction models (Fossen, MS-MLP, and Seq2Seq). The body-frame velocities and thruster inputs are shown in Fig. 4.27.

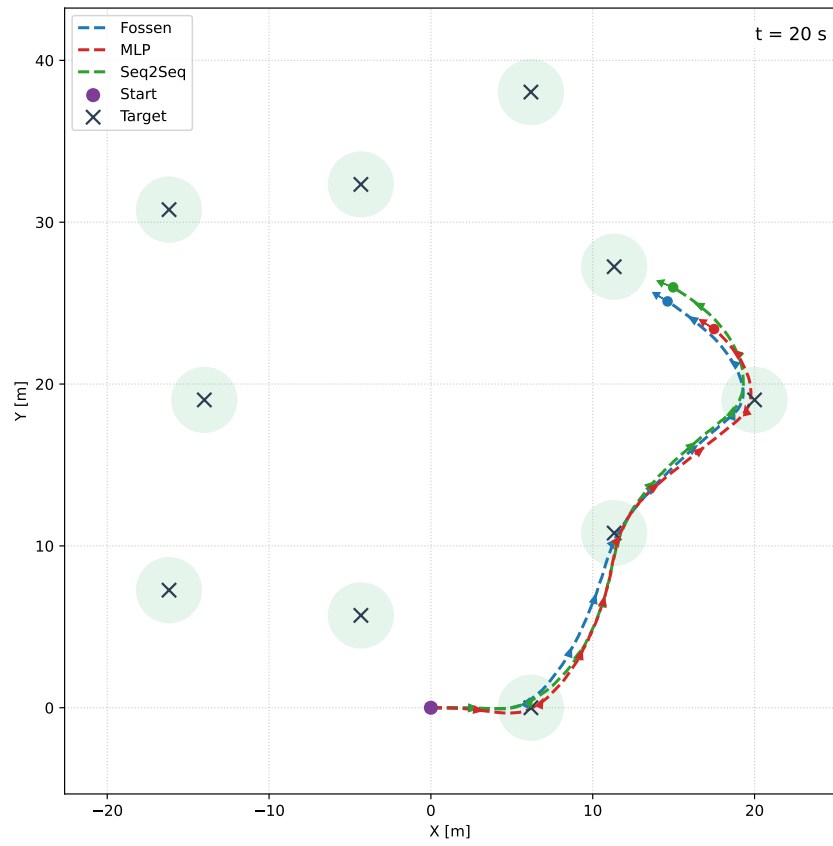
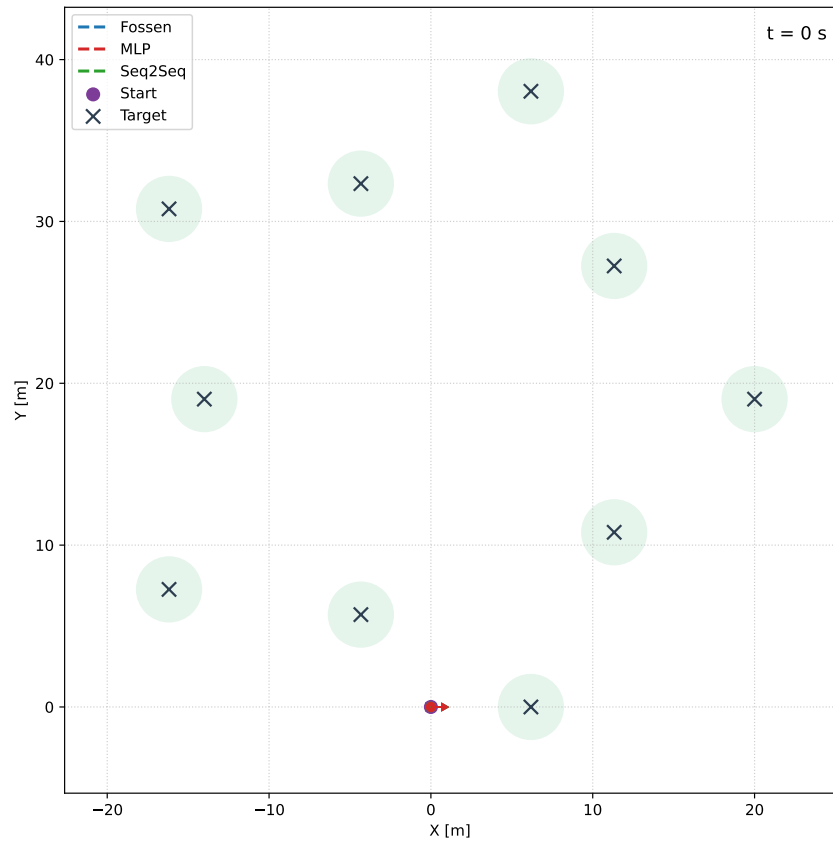


Figure 4.24: Star-shaped map position-control snapshots at  $t = 0$  and  $t = 20$  s.

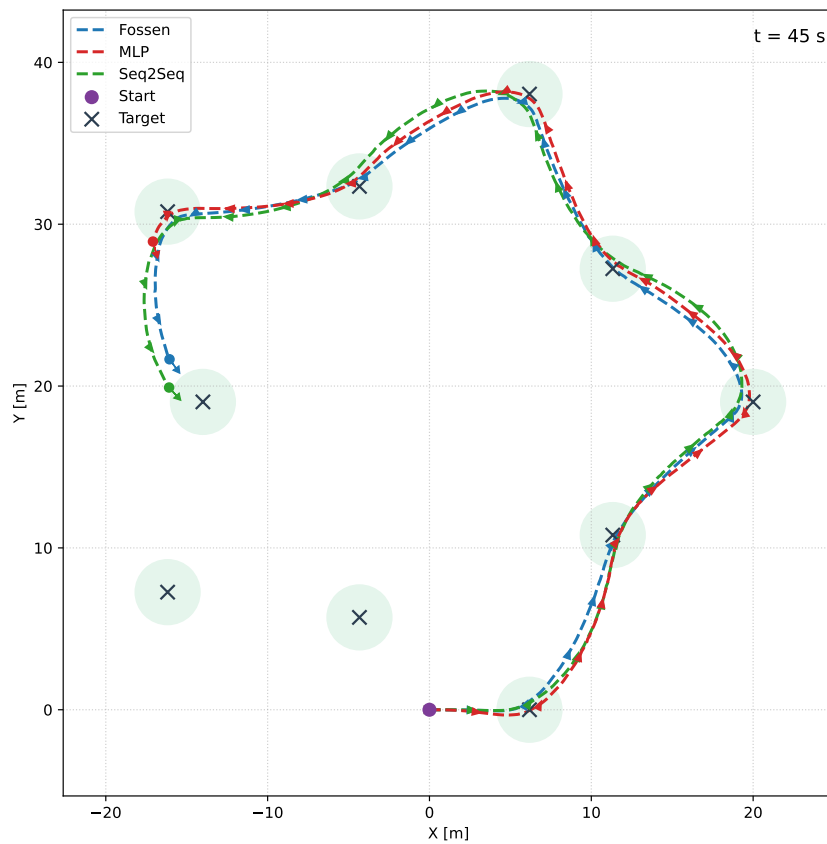
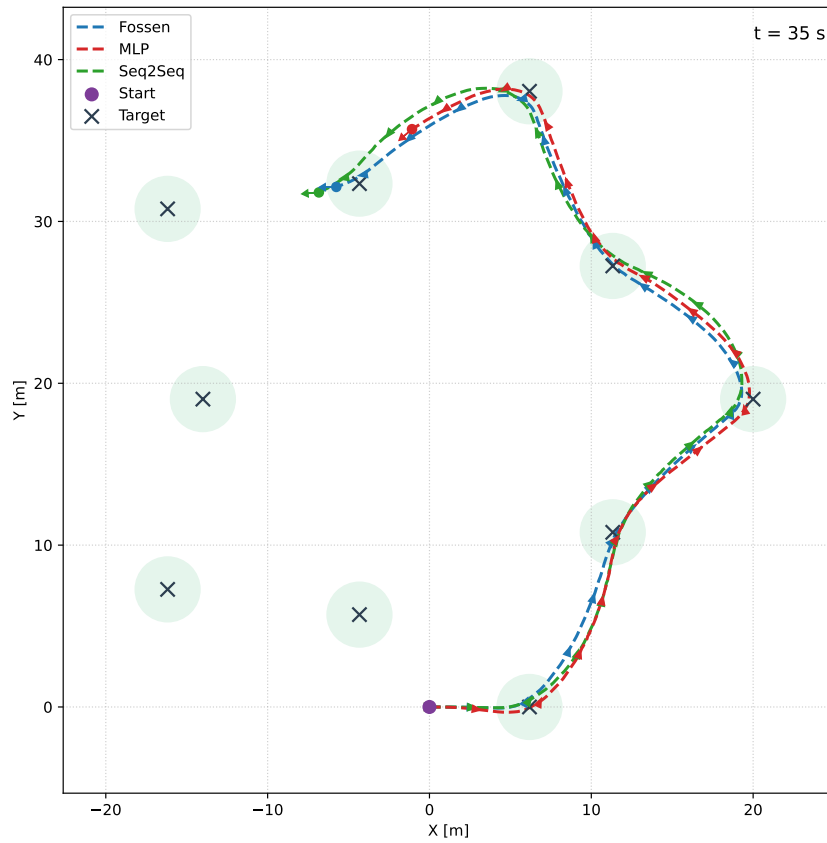


Figure 4.25: Star-shaped map position-control snapshots at  $t = 35$  and  $t = 45$  s.

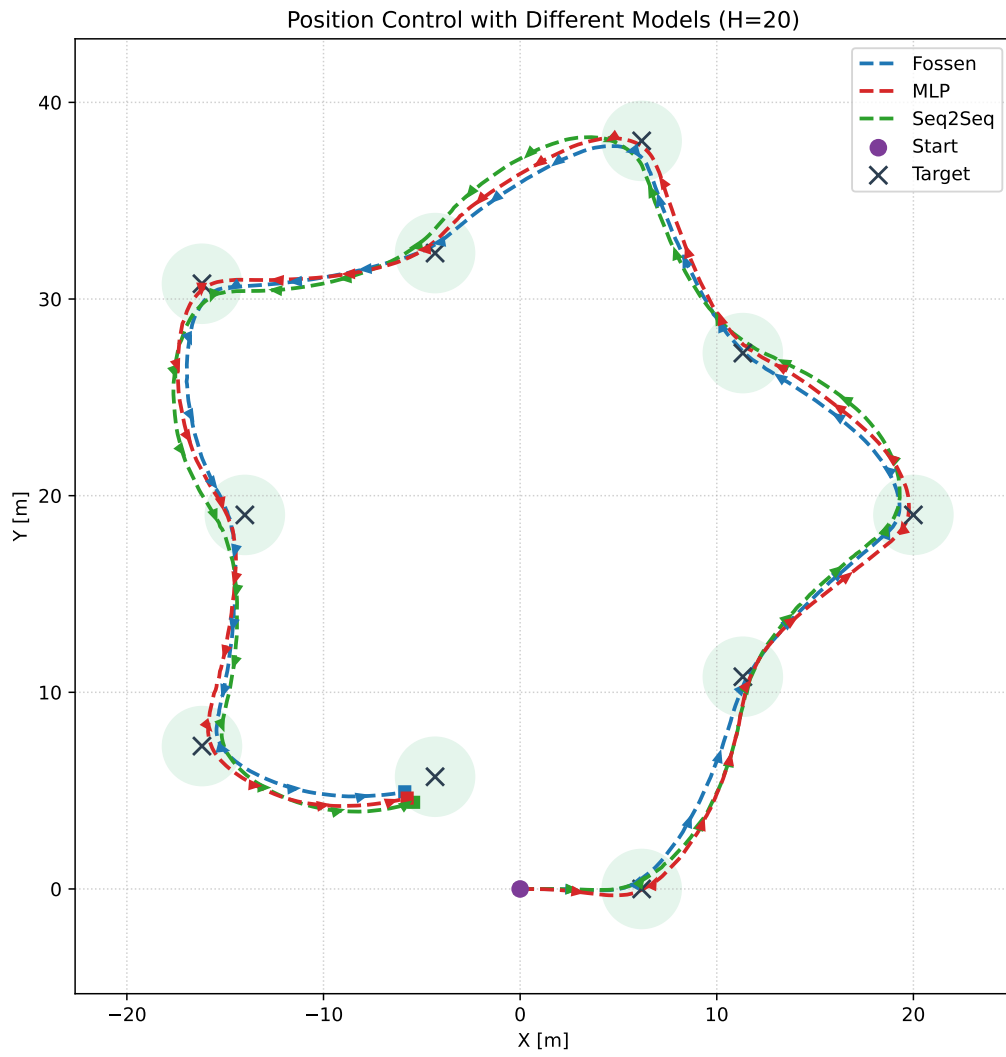
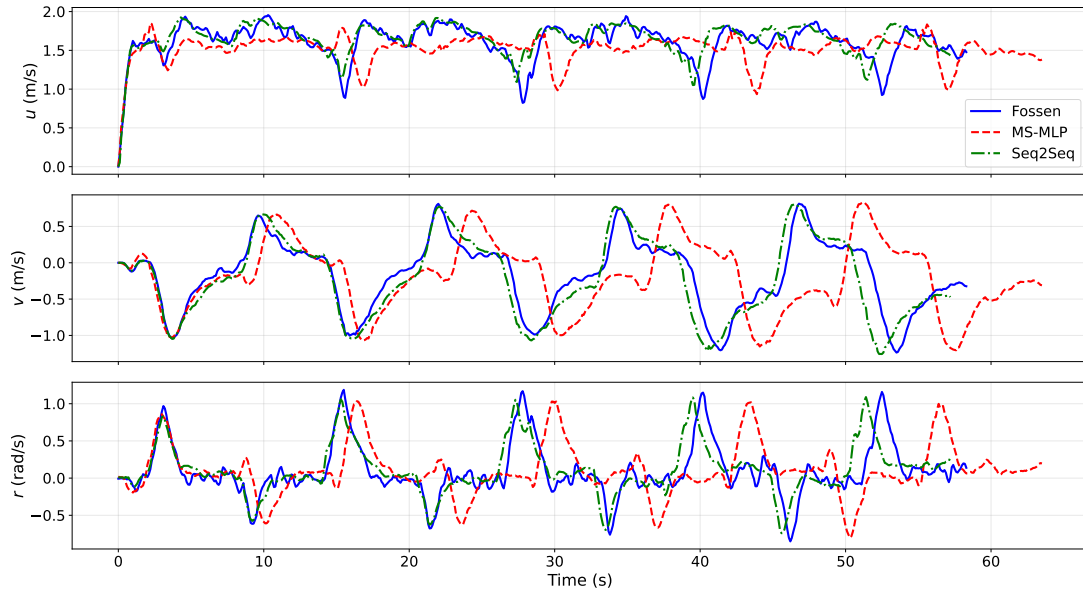
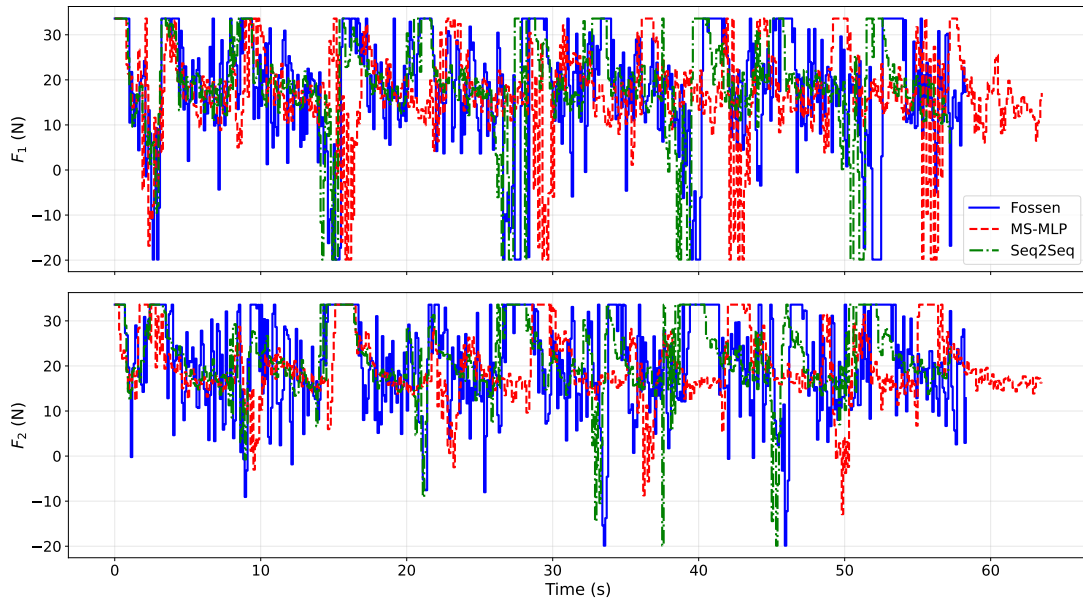


Figure 4.26: Star-shaped map: closed-loop trajectories for the three prediction models (Fossen, MS-MLP, and Seq2Seq) under position NMPC.



(a) Body-frame velocity responses ( $u, v, r$ ).



(b) Thruster inputs ( $F_1, F_2$ ).

Figure 4.27: Star-shaped map: Tracking results comparing the three prediction models.

Table 4.9: Star-shaped map: total mission completion time (time to finish all waypoints) for each prediction model.

<b>Model</b>	<b>Completion time (s)</b>
Fossen	58.4
MS-MLP	63.5
Seq2Seq	57.2

Table 4.10: NMPC computation time for star-shaped map position control (ms): mean and maximum computation time per control update, including NMPC preparation and SQP\_RTI solve time, for the Fossen, MS-MLP, and Seq2Seq prediction models.

<b>Model</b>	<b>Mean</b>	<b>Max</b>
Fossen	3.638	5.665
MS-MLP	2.274	5.285
Seq2Seq	8.512	14.423

Table 4.10 reports the computation time per NMPC update for the star-shaped map, including preparation and SQP\_RTI solve time. MS-MLP has the lowest mean run-time, while Seq2Seq is slower due to its recurrent structure (including the encoder pass in the preparation stage).

#### 4.6.2 S-Shaped Map

The star-shaped waypoint mission produced similar results for the three prediction models. To better highlight performance differences, a second position-control scenario is evaluated on a continuous S-shaped reference track. The S-shaped map contains two circular arcs (two half-circle curve segments) with radius 4 m and requires the vehicle to follow a smooth path instead of switching between waypoints. The same three prediction models are tested again (Fossen, MS-MLP, and Seq2Seq). The measured body-frame velocities are corrupted by zero-mean Gaussian noise with  $\sigma = 0.1$ .

For this map, the stage weights are

$$\mathbf{Q} = \text{diag}(400, 400, 250, 250, 400, 0.1, 0.1), \quad \mathbf{R} = \text{diag}(0.2, 0.2).$$

The  $(x, y)$  weights dominate the cost and drive the path tracking behavior. The yaw-error terms (through  $(\cos \psi, \sin \psi)$ ) are included mainly to guide the optimizer toward more efficient motion plans and to keep the vehicle aligned with the local path direction, but they do not dominate the objective. For the velocity terms,  $u$  is weighted more strongly to support the surge-speed objective, while  $v$  and  $r$  have very small weights and therefore have little effect on the cost.

A constant surge-speed reference is used:  $u_{\text{ref}} = 2.5$  m/s, which is close to the physical limit of the vessel. This makes the task more challenging, especially in the curved parts of the track. The remaining velocity references are set to zero:  $v_{\text{ref}} = 0$  and  $r_{\text{ref}} = 0$ .

At each NMPC update, the reference is generated directly from the S-shaped curve. First, the closest point on the track to the current vehicle position is found. From that point, the first reference point is selected by moving 1 m forward along the track (look-ahead point). Then, the remaining reference points are generated by moving forward along the track with the same surge-speed reference. With sampling time  $\Delta t = 0.1$  s, the distance between successive reference points is

$$\Delta s = u_{\text{ref}} \Delta t = 2.5 \times 0.1 = 0.25 \text{ m}.$$

This produces  $H = 20$  reference steps  $\{(x_{\text{ref}}[k+i], y_{\text{ref}}[k+i])\}_{i=0}^H$  on the path. The yaw reference at each step is chosen as the tangent direction of the track at that reference point, and it is applied through  $\cos \psi_{\text{ref}}$  and  $\sin \psi_{\text{ref}}$  in the output references.

Figures 4.28–4.31 show snapshots and the resulting trajectories on the S-shaped map. The body-frame velocities and the thruster inputs are shown in Figures 4.32a and 4.32b. The NMPC computation times are similar to those reported for the star-shaped position-control case in Table 4.10.

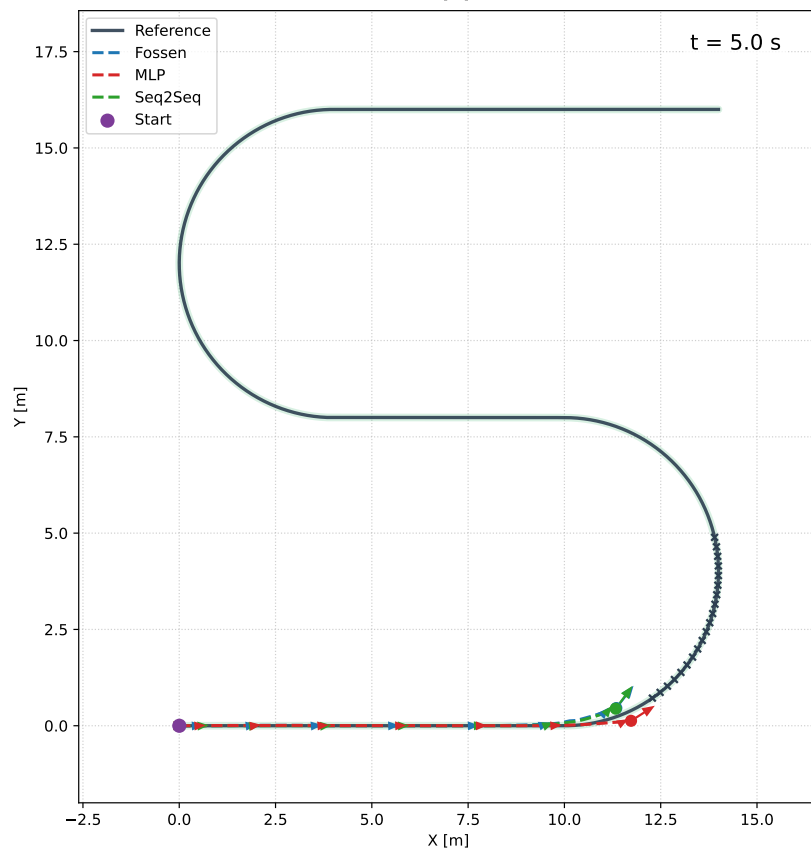
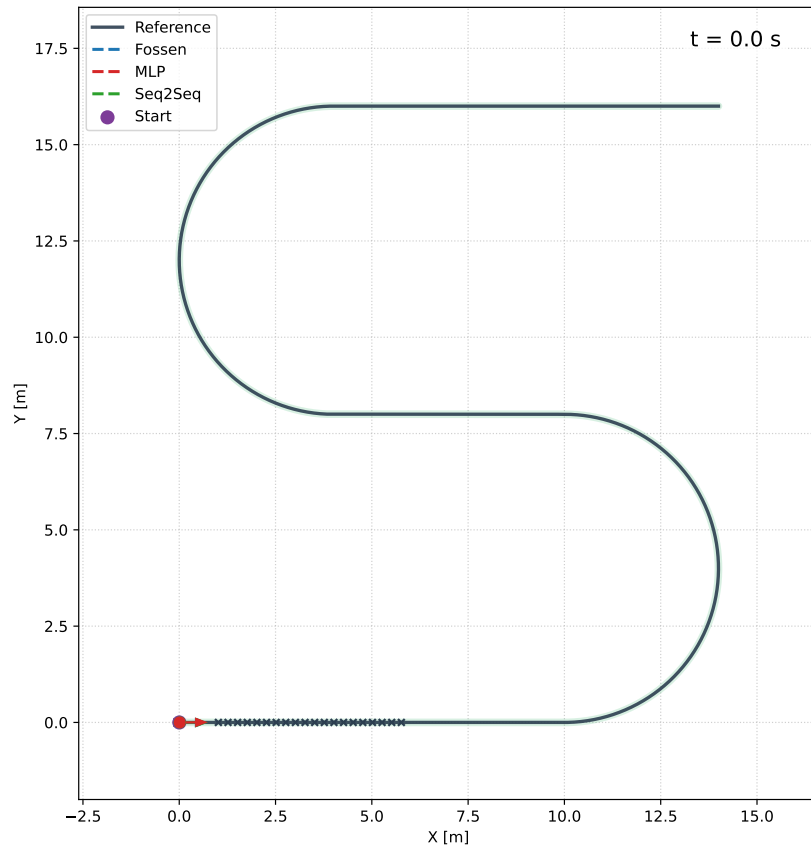


Figure 4.28: S-shape map snapshots at  $t = 0$  s and  $t = 5$  s.

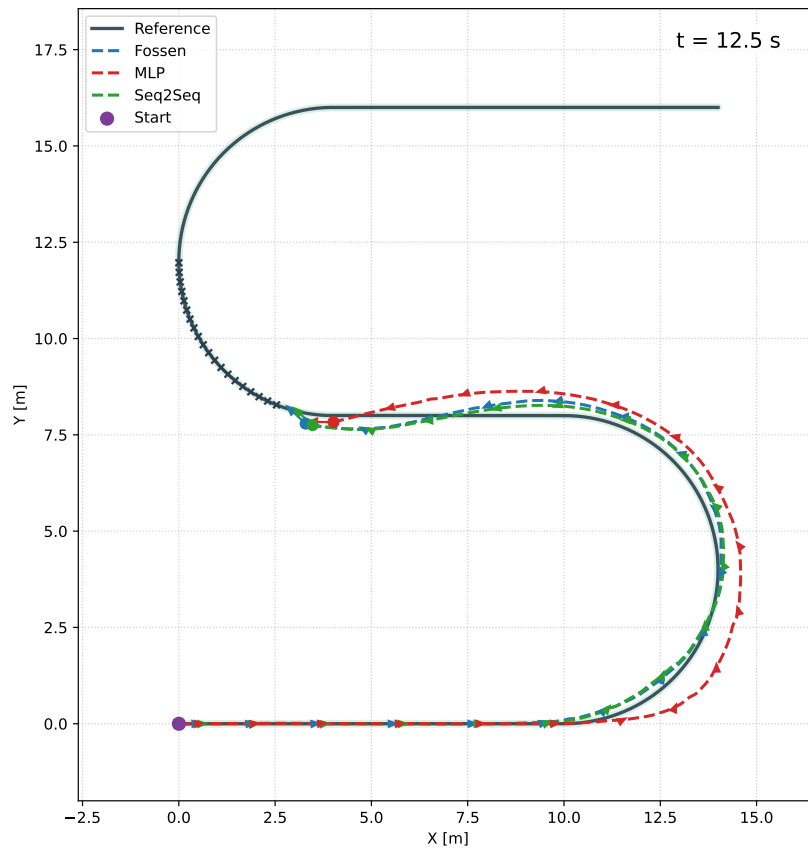
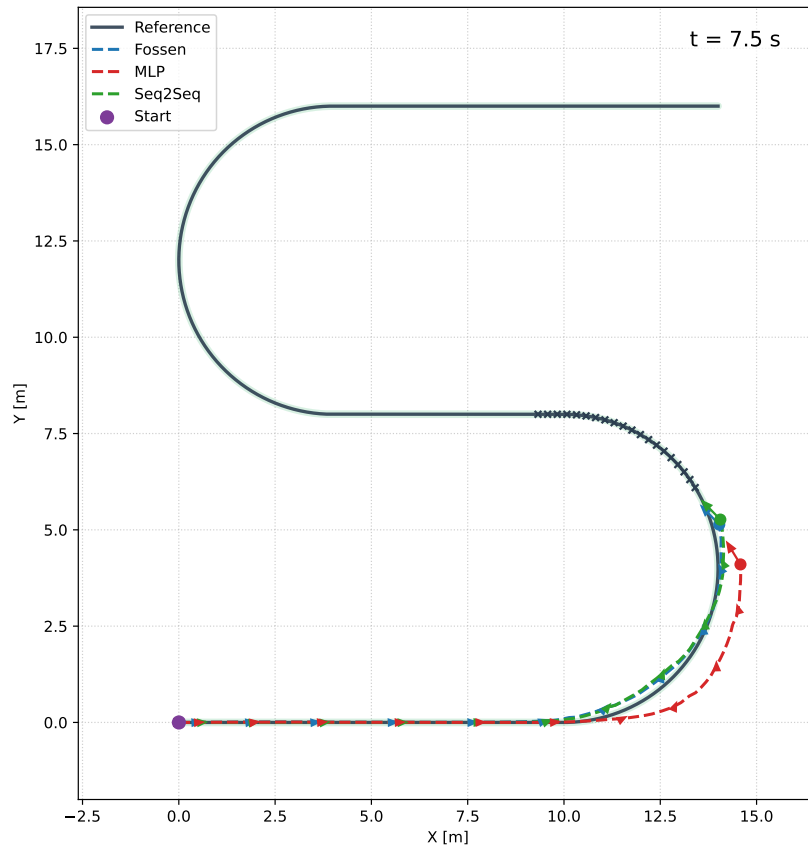


Figure 4.29: S-shape map snapshots at  $t = 7.5 s$  and  $t = 12.5 s$ .

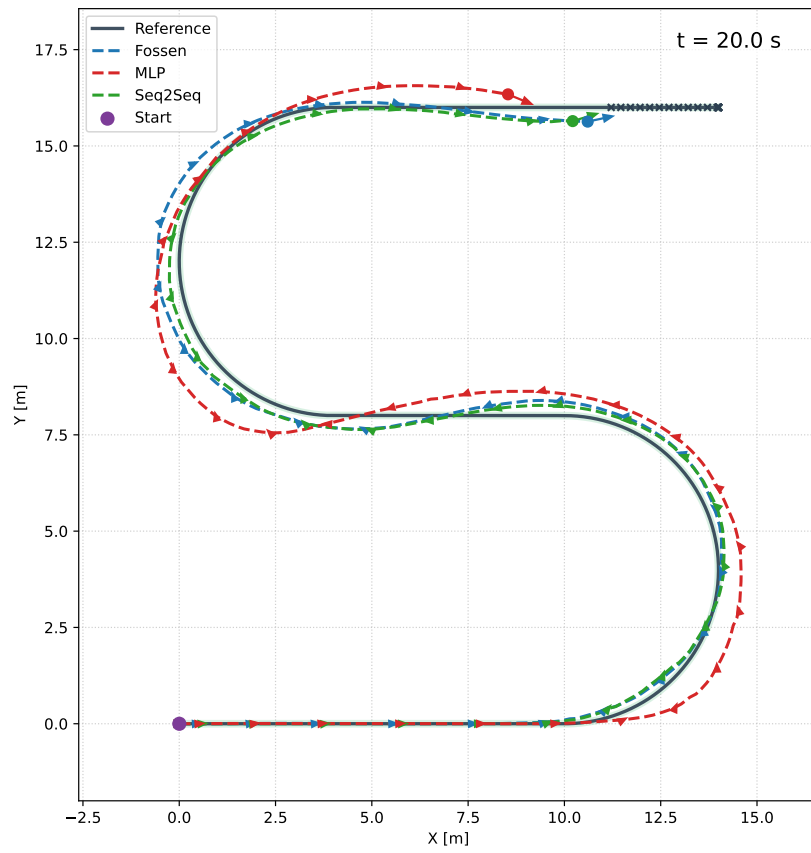
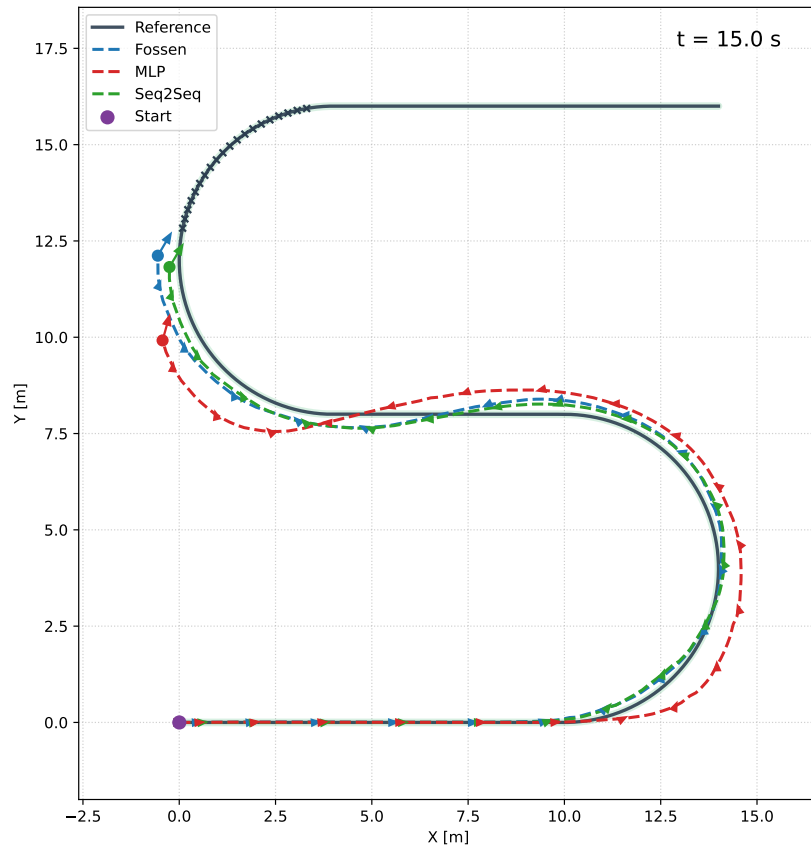


Figure 4.30: S-shape map snapshots at  $t = 15 \text{ s}$  and  $t = 20 \text{ s}$ .

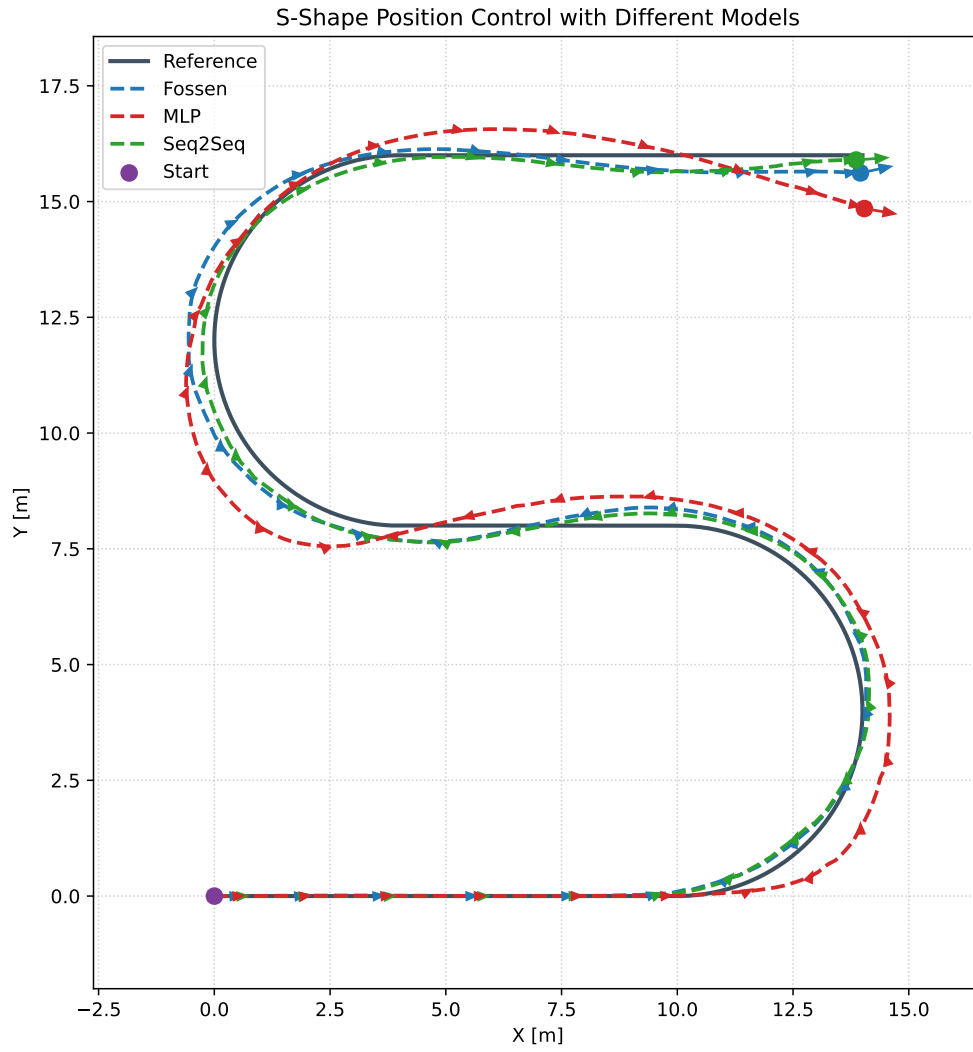
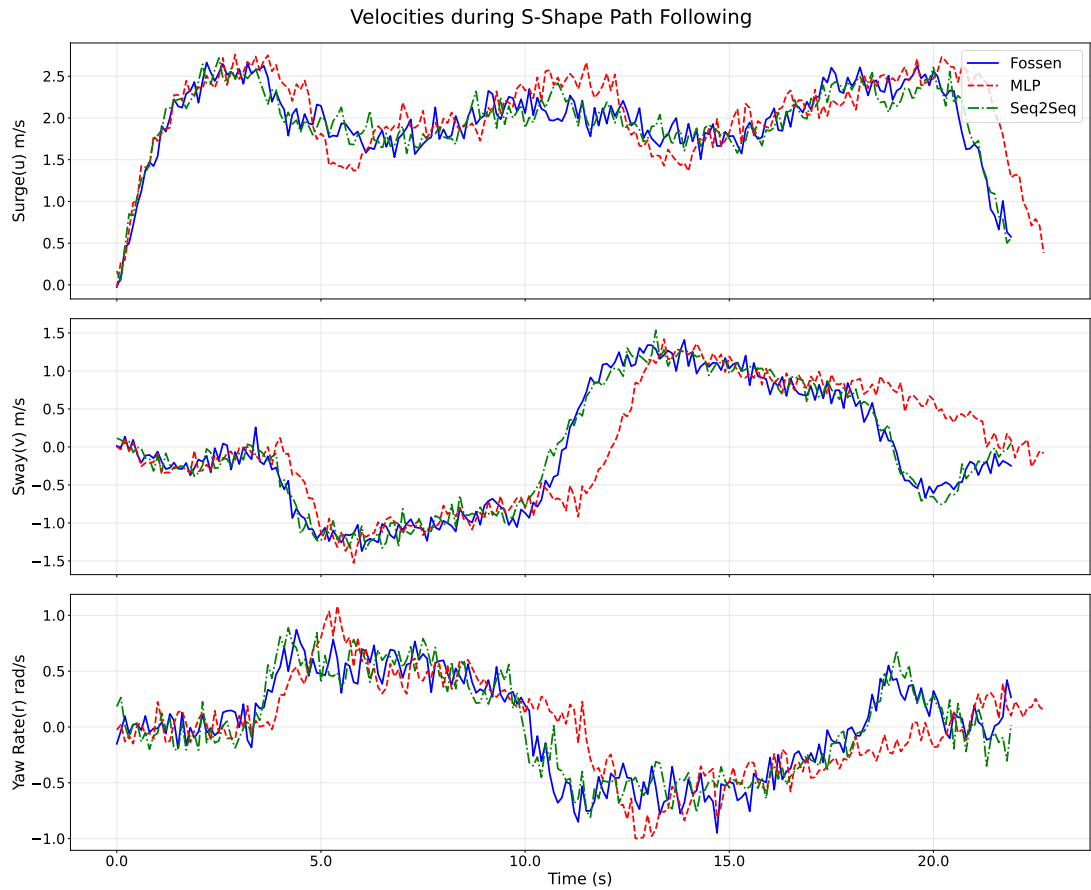
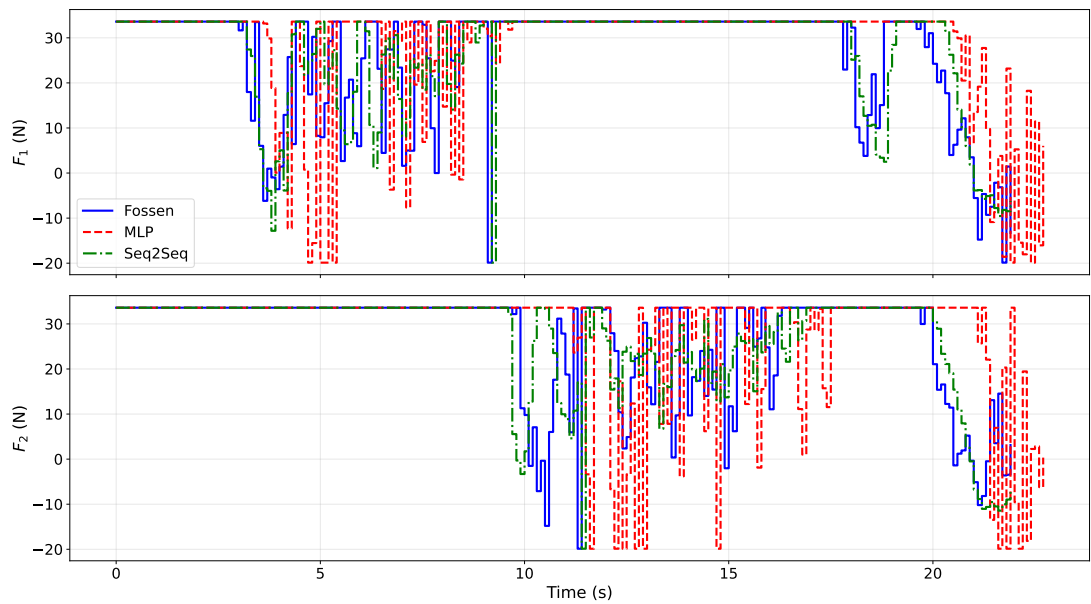


Figure 4.31: S-shaped map: closed-loop trajectories for the three prediction models (Fossen, MS-MLP, and Seq2Seq) under position NMPC.



(a) Body-frame velocity responses  $(u, v, r)$ .



(b) Thruster inputs  $(F_1, F_2)$ .

Figure 4.32: S-shaped map: position NMPC results comparing the three prediction models.

To measure tracking accuracy on that continuous curve, the vehicle position samples  $(x[k], y[k])$  are compared to the closest point on the reference track. We report the mean and maximum distance over the full run. We also report the fit metric defined in (4.8), which summarizes the overall match between the tracked trajectory and the reference.

Table 4.11: S-shaped map: tracking-error statistics computed from the distance between the closed-loop position  $(x, y)$  and the closest point on the reference track. The fit metric is computed using (4.8).

<b>Model</b>	<b>Mean distance (m)</b>	<b>Max distance (m)</b>	<b>Fit (%)</b>
Fossen	0.2157	0.5786	96.4832
Seq2Seq	0.1607	0.3772	97.4438
MS-MLP	0.4129	1.1519	93.2210

Table 4.11 shows that Seq2Seq gives the best tracking on this fast and curvy map. It has the lowest mean and maximum distance to the track and the highest fit value. The Fossen model also tracks the path well, but with slightly larger errors in the curved parts. MS-MLP has the largest errors, including the highest peak distance, which suggests that it is less robust when the task requires high-speed turning.

## 4.7 Discussions

In position control, the star-shaped map is mainly a waypoint task, so all three models give similar trajectories and reach the waypoints successfully. Seq2Seq finishes slightly faster than the others (Table 4.9), and its thruster commands are smoother and more consistent (Fig. 4.27b). This comes with a higher computation time per NMPC update compared to Fossen and MS-MLP (Table 4.10).

Even though MS-MLP performs well in system identification (Fig. 4.12), it gives weaker closed-loop position-control results than the Fossen model in both maps. One likely reason is its single-inference, full-horizon prediction structure, which can make the NMPC optimization more sensitive and more prone to suboptimal (local-

minimum) solutions. In the star-shaped case, the MS-MLP inputs are not as smooth as Seq2Seq and the overall performance does not surpass Fossen. In contrast, the Fossen controller sometimes produces more oscillatory thruster commands than the learned models (Fig. 4.27b), but it still tracks the waypoint mission reliably.

The S-shaped map is more challenging because it requires continuous tracking at high speed and through curved sections. Here, model differences are clearer. Seq2Seq gives the best tracking accuracy (Table 4.11) with stable control inputs. The Fossen model also tracks well, but shows slightly larger errors in the curved parts. MS-MLP has the largest tracking errors, and its thruster inputs show more oscillatory (high-frequency) variations (Fig. 4.32b), which indicates less robust behavior in fast turning.



## CHAPTER 5

### CONCLUSIONS

This thesis investigated learned prediction models for an underactuated Unmanned Surface Vehicle (USV) and their use as internal dynamics predictors in a real-time Nonlinear Model Predictive Control (NMPC) framework. We compared an identified discrete-time Fossen model with two data-driven predictors: a multi-step MLP (MS-MLP) that outputs the full horizon in one forward pass, and a Seq2Seq LSTM that generates the horizon step-by-step using an encoder–decoder structure.

In system identification, both learned models achieved lower multi-step prediction error than the recursive Fossen baseline for horizons  $H \in \{5, 12, 20\}$ . The Seq2Seq LSTM provided the best validation accuracy and slower error accumulation as the horizon increased, while the MS-MLP remained competitive for short horizons but degraded more for longer horizons. This shows a clear trade-off between compact single-pass prediction and temporally consistent recurrent rollout.

For control, each predictor was embedded inside the same NMPC formulation implemented with CasADi-based automatic differentiation and acados using an SQP-RTI scheme. In noiseless velocity tracking, the closed-loop performance of all three NMPC variants was similar, so computation time became the key difference: MS-MLP and Fossen achieved comparable runtimes, while Seq2Seq was slower due to the recurrent decoder and encoder computation. Under noisy measurements, the learned predictors improved robustness and produced smoother control actions; Seq2Seq achieved the best tracking fit, especially in the surge tracking experiment.

In position control, the waypoint-based star mission produced similar trajectories for all models, while the high-speed S-shaped path highlighted stronger differences:

Seq2Seq achieved the lowest tracking error, the Fossen model performed close behind, and MS-MLP showed larger deviations and more oscillatory inputs. Overall, MS-MLP is a practical choice when runtime is the dominant constraint, whereas Seq2Seq is preferable when long-horizon consistency and robustness are critical, at the cost of higher computation.

Future work includes validation on real hardware, hybrid physics–learning predictors to reduce model mismatch, and reducing the online cost of recurrent predictors through compression or more efficient implementations.

## REFERENCES

- [1] Z. Peng, J. Wang, L. Liu, D. Wang, and Q.-L. Han, “An Overview of Recent Advances in Coordinated Control of Multiple Autonomous Surface Vehicles,” *IEEE Transactions on Industrial Informatics*, 2020, doi:10.1109/tii.2020.3004343.
- [2] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011, doi:10.1002/9781119994138.
- [3] C. Sonnenburg and C. A. Woolsey, “Modeling, Identification, and Control of an Unmanned Surface Vehicle,” *Journal of Field Robotics*, 2013, doi:10.1002/rob.21452.
- [4] J. M. Maciejowski, *Predictive Control With Constraints*. Prentice-Hall, 2002.
- [5] L. Ljung, C. Andersson, K. Tiels, and T. B. Schön, “Deep learning and system identification,” *IFAC-PapersOnLine*, 2020, doi:10.1016/j.ifacol.2020.12.1329.
- [6] G. Rajesh and S. K. Bhattacharyya, “System identification for nonlinear maneuvering of large tankers using artificial neural network,” *Applied Ocean Research*, vol. 30, no. 4, pp. 256–263, 2008. doi: 10.1016/j.apor.2008.10.003.
- [7] J. Woo, J.-Y. Park, C.-W. Yu, and N. Kim, “Dynamic model identification of unmanned surface vehicles using deep learning network,” *Applied Ocean Research*, 2018, doi:10.1016/j.apor.2018.06.011.
- [8] N. Mohajerin and S. L. Waslander, “Multistep Prediction of Dynamic Systems With Recurrent Neural Networks,” *IEEE Transactions on Neural Networks*, 2019, doi:10.1109/tnnls.2019.2891257.
- [9] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “CasADi — A software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019,

doi: 10.1007/s12532-018-0139-4. <https://web.casadi.org/> (project website), <https://github.com/casadi/casadi> (source code).

- [10] R. Verschueren *et al.*, “acados — a modular open-source framework for fast embedded optimal control,” *Mathematical Programming Computation*, vol. 14, pp. 147–183, 2022, doi: 10.1007/s12532-021-00208-8. [Online]. Available: <https://docs.acados.org/> (documentation), <https://github.com/acados/acados> (source code).
- [11] Clearpath Robotics, “Heron simulator,” GitHub repository, [https://github.com/heron/heron\\_simulator](https://github.com/heron/heron_simulator).
- [12] H. Xu, P. Pires da Silva, and C. Guedes Soares, “Effect of sampling rate in sea trial tests on the estimation of hydrodynamic parameters for a nonlinear ship manoeuvring model,” *Journal of Marine Science and Engineering*, vol. 12, no. 3, art. 407, 2024. doi: 10.3390/jmse12030407.
- [13] T.-N. Lin, B. G. Horne, P. Tino, and C. L. Giles, “Learning long-term dependencies in NARX recurrent neural networks,” *IEEE Transactions on Neural Networks*, 1996, doi:10.1109/72.548162.
- [14] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, 1997, doi:10.1162/neco.1997.9.8.1735.
- [15] M. Jung, P. R. da Costa Mendes, M. Önnheim, and E. Gustavsson, “Model Predictive Control when utilizing LSTM as dynamic models,” *Engineering Applications of Artificial Intelligence*, 2023, doi:10.1016/j.engappai.2023.106226.
- [16] M. Forgione and D. Piga, “Continuous-time system identification with neural networks: Model structures and fitting criteria,” *European Journal of Control*, vol. 59, pp. 69–81, 2021, doi: 10.1016/j.ejcon.2021.01.008.
- [17] O. Ogunmolu, X. Gu, S. B. Jiang, and N. Gans, “Nonlinear Systems Identification Using Deep Dynamic Neural Networks,” *arXiv: Neural and Evolutionary Computing*, 2016.
- [18] X. Pengfei, P. Xu, C.-B. Han, C. Hongxia, H. Cheng, C. Cheng, and T. Ge, “A Physics-Informed Neural Network for the Prediction of Unmanned Sur-

- face Vehicle Dynamics,” *Journal of Marine Science and Engineering*, 2022, doi:10.3390/jmse10020148.
- [19] A. Saviolo, G. Li, and G. Loianno, “Physics-Inspired Temporal Learning of Quadrotor Dynamics for Accurate Model Predictive Trajectory Tracking,” *IEEE Robotics and Automation Letters*, 2022, doi:10.1109/lra.2022.3192609.
- [20] E. A. Antonelo, E. Camponogara, L. O. Seman, E. Rehbein De Souza, J. Jordanou, and J. F. Hubner, “Physics-informed neural nets for control of dynamical systems,” *Neurocomputing*, 2021, doi:10.1016/j.neucom.2024.127419.
- [21] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh, “KNODE-MPC: A Knowledge-Based Data-Driven Predictive Control Framework for Aerial Robots,” *IEEE Robotics and Automation Letters*, 2021, doi:10.1109/lra.2022.3144787.
- [22] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, “Real-Time Neural MPC: Deep Learning Model Predictive Control for Quadrotors and Agile Robotic Platforms,” *IEEE Robotics and Automation Letters*, 2023, doi:10.1109/lra.2023.3246839.
- [23] M. Lazar, M.-S. Popescu, and M. Schoukens, “Nonlinear Data-Driven Predictive Control Using Deep Subspace Prediction Networks,” in *IEEE Conference on Decision and Control*, 2023, doi:10.1109/cdc49753.2023.10384143.
- [24] B. Karg and S. Lucia, “Deep learning-based embedded mixed-integer model predictive control,” in *Proceedings of the 2018 European Control Conference (ECC)*, pp. 2075–2080, 2018. doi: 10.23919/ECC.2018.8550234.
- [25] S. S. Pon Kumar, A. Tulsyan, B. Gopaluni, and P. Loewen, “A deep learning architecture for predictive control,” *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 512–517, 2018, Proceedings of the 10th IFAC Symposium on Advanced Control of Chemical Processes (ADCHEM 2018). doi: 10.1016/j.ifacol.2018.09.373.
- [26] M. J. Ellis and V. Chinde, “An encoder–decoder LSTM-based EMPC framework applied to a building HVAC system,” *Chemical Engineering Research and Design*, vol. 160, pp. 508–520, 2020. doi: 10.1016/j.cherd.2020.06.008.

- [27] D. Masti, F. Smarra, A. D’Innocenzo, and A. Bemporad, “Learning affine predictors for MPC of nonlinear systems via artificial neural networks,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5233–5238, 2020, Proceedings of the 21st IFAC World Congress. doi: 10.1016/j.ifacol.2020.12.1199.
- [28] K. Zarzycki and M. Ławryńczuk, “LSTM and GRU neural networks as models of dynamical processes used in predictive control: A comparison of models developed for two chemical reactors,” *Sensors*, vol. 21, no. 16, p. 5625, 2021.
- [29] B.-K. Jeon and E.-J. Kim, “LSTM-based model predictive control for optimal temperature set-point planning,” *Sustainability*, vol. 13, no. 2, p. 894, 2021. doi: 10.3390/su13020894.
- [30] I. Hammoud, S. Hentzelt, T. Oehlschlaegel, and R. Kennel, “Learning-based model predictive current control for synchronous machines: An LSTM approach,” *European Journal of Control*, vol. 68, p. 100663, 2022. doi: 10.1016/j.ejcon.2022.100663.
- [31] T. V. Baby, S. M. Sotoudeh, and B. HomChaudhuri, “Data-driven prediction and predictive control methods for eco-driving in production vehicles,” *IFAC-PapersOnLine*, vol. 55, no. 37, pp. 633–638, 2022, Proceedings of the 2nd Modeling, Estimation and Control Conference (MECC 2022). doi: 10.1016/j.ifacol.2022.11.253.
- [32] B. B. Schwedersky, R. C. C. Flesch, and H. A. S. Dangui, “Practical nonlinear model predictive control algorithm for long short-term memory networks,” *IFAC-PapersOnLine*, vol. 52, no. 1, pp. 468–473, 2019, Proceedings of the 12th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems (DYCOPS 2019). doi: 10.1016/j.ifacol.2019.06.106.
- [33] K. Zarzycki and M. Ławryńczuk, “Advanced predictive control for GRU and LSTM networks,” *Information Sciences*, vol. 616, pp. 229–254, 2022. doi: 10.1016/j.ins.2022.10.078.
- [34] F. Bonassi, M. Farina, and R. Scattolini, “On the stability properties of Gated Recurrent Units neural networks,” *Systems & Control Letters*, vol. 157, p. 105049, 2021. doi: 10.1016/j.sysconle.2021.105049.

- [35] E. Terzi, F. Bonassi, M. Farina, and R. Scattolini, “Learning model predictive control with long short-term memory networks,” *International Journal of Robust and Nonlinear Control*, vol. 31, no. 18, pp. 8877–8896, 2021. doi: 10.1002/rnc.5519.
- [36] R. Al-Seyab, R. K. Al Seyab, and Y. Cao, “Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation,” *Journal of Process Control*, 2008, doi:10.1016/j.jprocont.2007.10.012.
- [37] H. Yamasaki, I. Maruta, and K. Fujimoto, “Deep Neural Network-Based System Identification for Nonlinear MPC: Enhancements for Massive Multi-Output Systems and Experimental Validation with 1D Camera Image Outputs\*,” *IFAC-PapersOnLine*, 2024, doi:10.1016/j.ifacol.2024.09.042.
- [38] T. Salzmann, J. Arrizabalaga, J. Andersson, M. Pavone, and M. Ryll, “Learning for CasADi: Data-driven models in numerical optimization,” in *Proc. 6th Annual Learning for Dynamics & Control Conf. (LADC), Proceedings of Machine Learning Research*, vol. 242, pp. 541–553, 2024. <https://github.com/Tim-Salzmann/l4casadi> (source code).
- [39] D. T. Doncevic, A. M. Schweidtmann, Y. Vaupel, P. Schäfer, A. Caspari, and A. Mitsos, “Deterministic global nonlinear model predictive control with neural networks embedded,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5273–5278, 2020, Proceedings of the 21st IFAC World Congress. doi: 10.1016/j.ifacol.2020.12.1207.
- [40] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [41] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994, doi: 10.1109/72.279181.
- [42] A. Paszke *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019. [Online]. Available: <https://pytorch.org/>(project

- website), <https://github.com/pytorch/pytorch> (source code), <https://arxiv.org/abs/1912.01703> (preprint).
- [43] M. Diehl, H. G. Bock, and J. P. Schlöder, “A real-time iteration scheme for nonlinear optimization in optimal feedback control,” *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005. doi: 10.1137/S0363012902400713.
- [44] G. Frison and M. Diehl, “HPIPM: a high-performance quadratic programming framework for model predictive control,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020, doi: 10.1016/j.ifacol.2020.12.073.
- [45] The MathWorks, Inc., “Nonlinear grey-box models and estimation (idnl-grey, nlgreyest),” *System Identification Toolbox Documentation*, [Online]. Available: <https://www.mathworks.com/help/ident/nonlinear-grey-box-models.html> (overview).
- [46] The MathWorks, Inc., “Optimization Toolbox and fmincon,” *Optimization Toolbox Documentation*, [Online]. Available: <https://www.mathworks.com/help/optim/ug/fmincon.html> (fmincon).
- [47] E. Koyuncu, *Motion Planning and Control of Underactuated Systems over Optimized Trajectories*, M.S. thesis, Middle East Technical University (METU), 2024. [Online]. Available: <https://open.metu.edu.tr/handle/11511/111305> (record)
- [48] A. Richard, S. Aravecchia, T. Schillaci, M. Geist, and C. Pradalier, “How to Train Your HERON,” *CoRR*, vol. abs/2102.10357, 2021. Available: <https://arxiv.org/abs/2102.10357>.
- [49] Open Robotics / ROS community, “rosbridge\_server — ROS Wiki,” [https://wiki.ros.org/rosbridge\\_server](https://wiki.ros.org/rosbridge_server)
- [50] ROSlibPy Developers, “roslibpy documentation,” <https://roslibpy.readthedocs.io/en/latest/>

## APPENDICES

### A NMPC formulation and acados solution method (SQP\_RTI + full condensing)

The NMPC problem in this appendix is written in a standard discrete-time optimal control form with a nonlinear least-squares cost. The problem is solved using SQP\_RTI as in the real-time iteration scheme of [43]: at each sampling instant, the dynamics are linearized and a Gauss-Newton QP approximation of the least-squares cost is built and solved. In this thesis, the solver implementation is `acados` [10]. The model and cost functions are written in `CasADi` and differentiated using algorithmic differentiation [9]. With full condensing, the QP becomes a dense problem, which is solved by the primal-dual interior-point method in `HPiPM` [44].

- `cost_type = NONLINEAR_LS` and `cost_type_e = NONLINEAR_LS` (for the velocity-only controller variant, `LINEAR_LS` is used),
- `hessian_approx = GAUSS_NEWTON`,
- `nlp_solver_type = SQP_RTI` with `nlp_solver_max_iter = 1`,
- `qp_solver = FULL_CONDENSING_HPiPM`.

Only input box constraints of the form  $u_{\min} \leq u[k] \leq u_{\max}$  are considered.

**OCP (nonlinear program).** Let  $x[k] \in \mathbb{R}^{n_x}$  and  $u[k] \in \mathbb{R}^{n_u}$  for  $k = 0, \dots, N - 1$ , and let  $N$  denote the prediction horizon length. The discrete dynamics are given by an explicit map

$$x[k + 1] = f(x[k], u[k]), \quad k = 0, \dots, N - 1. \quad (\text{A.1})$$

The input bounds are

$$u_{\min} \leq u[k] \leq u_{\max}, \quad k = 0, \dots, N - 1, \quad (\text{A.2})$$

and the initial condition is fixed to the measured state  $\hat{x}[0]$ :

$$x[0] = \hat{x}[0]. \quad (\text{A.3})$$

For a nonlinear least-squares objective, define the stage and terminal outputs

$$y[k] = h(x[k], u[k]) \in \mathbb{R}^{n_y}, \quad y[N] = h_e(x[N]) \in \mathbb{R}^{n_{y_e}}, \quad (\text{A.4})$$

and references  $y^{\text{ref}}[k]$  and  $y^{\text{ref}}[N]$ . Define residuals

$$r[k] \triangleq y[k] - y^{\text{ref}}[k], \quad r[N] \triangleq h_e(x[N]) - y^{\text{ref}}[N]. \quad (\text{A.5})$$

Let  $W \in \mathbb{R}^{n_y \times n_y}$  and  $W_e \in \mathbb{R}^{n_{y_e} \times n_{y_e}}$  be weighting matrices (typically chosen  $W \succ 0$  and  $W_e \succ 0$ ). The NMPC problem is

$$\min_{\substack{x[0], \dots, x[N] \\ u[0], \dots, u[N-1]}} \sum_{k=0}^{N-1} \frac{1}{2} r[k]^\top W r[k] + \frac{1}{2} r[N]^\top W_e r[N] \quad (\text{A.6})$$

$$\text{s.t. } x[0] = \hat{x}[0], \quad (\text{A.7})$$

$$x[k+1] = f(x[k], u[k]), \quad k = 0, \dots, N-1, \quad (\text{A.8})$$

$$u_{\min} \leq u[k] \leq u_{\max}, \quad k = 0, \dots, N-1. \quad (\text{A.9})$$

**CasADi model (practical note).** The functions  $f(\cdot)$ ,  $h(\cdot)$ , and  $h_e(\cdot)$  are implemented as CasADi symbolic expressions (SX or MX). CasADi provides algorithmic differentiation for the Jacobians required by SQP, and code generation/compilation is used for fast runtime evaluation.

**SQP\_RTI (one SQP step per sampling instant).** Sequential Quadratic Programming (SQP) forms a QP approximation of (A.6)–(A.9) around a current (warm-start) trajectory and solves it to obtain an update. With `SQP_RTI`, `acados` performs one SQP step per sampling instant. Conceptually, RTI is often described in two phases: (i) a *preparation* phase that linearizes the model/cost (and optionally performs condensing/factorization), and (ii) a *feedback* phase that updates the initial state  $x[0] = \hat{x}[0]$ , solves the QP, and applies the first control move.

Let a warm-start trajectory be

$$\{x^{(i)}[k], u^{(i)}[k]\}_{k=0}^{N-1}, \quad x^{(i)}[N], \quad (\text{A.10})$$

and define increments

$$\Delta x[k] \triangleq x[k] - x^{(i)}[k], \quad \Delta u[k] \triangleq u[k] - u^{(i)}[k]. \quad (\text{A.11})$$

For dynamics linearization Linearize  $f$  at  $(x^{(i)}[k], u^{(i)}[k])$ :

$$f(x[k], u[k]) \approx f^{(i)}[k] + A^{(i)}[k]\Delta x[k] + B^{(i)}[k]\Delta u[k], \quad (\text{A.12})$$

where

$$f^{(i)}[k] \triangleq f(x^{(i)}[k], u^{(i)}[k]), \quad A^{(i)}[k] \triangleq \left. \frac{\partial f}{\partial x} \right|_{(x^{(i)}[k], u^{(i)}[k])}, \quad B^{(i)}[k] \triangleq \left. \frac{\partial f}{\partial u} \right|_{(x^{(i)}[k], u^{(i)}[k])}. \quad (\text{A.13})$$

Substituting into  $x[k+1] = f(x[k], u[k])$  yields the linearized constraints in increments:

$$\Delta x[k+1] = A^{(i)}[k]\Delta x[k] + B^{(i)}[k]\Delta u[k] + d^{(i)}[k], \quad k = 0, \dots, N-1, \quad (\text{A.14})$$

with the *defect* (dynamic residual of the warm start)

$$d^{(i)}[k] \triangleq f^{(i)}[k] - x^{(i)}[k+1]. \quad (\text{A.15})$$

Because  $x[0]$  is fixed, the first increment satisfies

$$\Delta x[0] = 0. \quad (\text{A.16})$$

For Gauss-Newton approximation for nonlinear least squares ,

define  $r^{(i)}[k] \triangleq r(x^{(i)}[k], u^{(i)}[k])$  and  $r^{(i)}[N] \triangleq r(x^{(i)}[N])$ .

Since  $y^{\text{ref}}[\cdot]$  is treated as constant w.r.t. the decision variables, the residual Jacobians are equivalently the output Jacobians:

$$J^{(i)}[k] \triangleq \left[ \begin{array}{cc} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial u} \end{array} \right]_{(x^{(i)}[k], u^{(i)}[k])} = \left[ \begin{array}{cc} \frac{\partial h}{\partial x} & \frac{\partial h}{\partial u} \end{array} \right]_{(x^{(i)}[k], u^{(i)}[k])} \in \mathbb{R}^{n_y \times (n_x + n_u)}, \quad (\text{A.17})$$

$$J^{(i)}[N] \triangleq \left. \frac{\partial r}{\partial x} \right|_{x^{(i)}[N]} = \left. \frac{\partial h_e}{\partial x} \right|_{x^{(i)}[N]} \in \mathbb{R}^{n_{y_e} \times n_x}. \quad (\text{A.18})$$

Gauss-Newton uses the Hessian approximation

$$H^{(i)}[k] \triangleq (J^{(i)}[k])^\top W J^{(i)}[k], \quad H^{(i)}[N] \triangleq (J^{(i)}[N])^\top W_e J^{(i)}[N], \quad (\text{A.19})$$

and the gradient terms

$$g^{(i)}[k] \triangleq (J^{(i)}[k])^\top W r^{(i)}[k], \quad g^{(i)}[N] \triangleq (J^{(i)}[N])^\top W_e r^{(i)}[N]. \quad (\text{A.20})$$

**QP solved in the RTI feedback step.** The SQP QP in increments is

$$\begin{aligned} \min_{\{\Delta x[k], \Delta u[k]\}} \quad & \sum_{k=0}^{N-1} \left( \frac{1}{2} \begin{bmatrix} \Delta x[k] \\ \Delta u[k] \end{bmatrix}^\top H^{(i)}[k] \begin{bmatrix} \Delta x[k] \\ \Delta u[k] \end{bmatrix} + \begin{bmatrix} \Delta x[k] \\ \Delta u[k] \end{bmatrix}^\top g^{(i)}[k] \right) \\ & + \frac{1}{2} \Delta x[N]^\top H^{(i)}[N] \Delta x[N] + \Delta x[N]^\top g^{(i)}[N] \end{aligned} \quad (\text{A.21})$$

$$\text{s.t.} \quad \Delta x[0] = 0, \quad (\text{A.22})$$

$$\Delta x[k+1] = A^{(i)}[k] \Delta x[k] + B^{(i)}[k] \Delta u[k] + d^{(i)}[k], \quad k = 0, \dots, N-1, \quad (\text{A.23})$$

$$u_{\min} - u^{(i)}[k] \leq \Delta u[k] \leq u_{\max} - u^{(i)}[k], \quad k = 0, \dots, N-1. \quad (\text{A.24})$$

After solving, the SQP update is

$$x^{(i+1)}[k] = x^{(i)}[k] + \Delta x^*[k], \quad u^{(i+1)}[k] = u^{(i)}[k] + \Delta u^*[k]. \quad (\text{A.25})$$

With SQP\_RTII and one SQP step per sampling instant, the applied control is

$$u_{\text{applied}} = u^{(i)}[0] + \Delta u^*[0]. \quad (\text{A.26})$$

**Full condensing and HPIPM.** The QP (A.21)–(A.24) has coupled variables due to (A.23). Full condensing eliminates the state increments by forward substitution of the linearized dynamics. Define the stacked input increment

$$\Delta U \triangleq \begin{bmatrix} \Delta u[0] \\ \Delta u[1] \\ \vdots \\ \Delta u[N-1] \end{bmatrix} \in \mathbb{R}^{Nn_u}, \quad (\text{A.27})$$

and stack  $\Delta x[1], \dots, \Delta x[N]$  into  $\Delta X$ . There exist a matrix  $S$  and vector  $s$  (constructed from  $A^{(i)}[k], B^{(i)}[k], d^{(i)}[k]$ ) such that

$$\Delta X = S \Delta U + s. \quad (\text{A.28})$$

Substituting (A.28) into the objective yields a dense QP in  $\Delta U$ :

$$\min_{\Delta U} \quad \frac{1}{2} \Delta U^\top H_c \Delta U + g_c^\top \Delta U \quad (\text{A.29})$$

$$\text{s.t.} \quad \underline{b} \leq \Delta U \leq \bar{b}, \quad (\text{A.30})$$

with bounds

$$\underline{b} \triangleq \begin{bmatrix} u_{\min} - u^{(i)}[0] \\ \vdots \\ u_{\min} - u^{(i)}[N-1] \end{bmatrix}, \quad \bar{b} \triangleq \begin{bmatrix} u_{\max} - u^{(i)}[0] \\ \vdots \\ u_{\max} - u^{(i)}[N-1] \end{bmatrix}. \quad (\text{A.31})$$

HPIPM solves (A.29)–(A.30) using a primal–dual interior-point method (IPM). The KKT conditions for the bound-constrained dense QP (written here without barrier perturbation) are

$$H_c \Delta U + g_c + \lambda^+ - \lambda^- = 0, \quad (\text{A.32})$$

$$\underline{b} \leq \Delta U \leq \bar{b}, \quad (\text{A.33})$$

$$\lambda^+ \geq 0, \quad \lambda^- \geq 0, \quad (\text{A.34})$$

$$\lambda^+ \odot (\Delta U - \bar{b}) = 0, \quad \lambda^- \odot (\underline{b} - \Delta U) = 0, \quad (\text{A.35})$$

where  $\lambda^+, \lambda^-$  are Lagrange multipliers for the upper and lower bounds, respectively, and  $\odot$  denotes elementwise multiplication. (In the IPM iterations, complementarity is typically perturbed and driven to zero as the barrier parameter decreases.)

**Summary of one control update.** At each sampling instant, `acados` takes a warm start, linearizes the discrete model and the least-squares residuals, builds one Gauss–Newton QP, condenses it to a dense input-only QP, solves it with HPIPM, and applies the first input move (A.26).