TÉCNICO LISBOA

The application of learning techniques to improve the control of surface vehicles

Victor Luis Gomes Marchesini Fonseca

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Pedro Tiago Martins Batista
Prof. Ricard Marxer

Examination Committee

Chairperson: Prof. João Manuel de Freitas Xavier
Supervisor: Prof. Pedro Tiago Martins Batista
Member of the Committee: Dr. Ricardo Adriano Ribeiro

October 2023

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

Thank you, dad and mom, for your unconditional support, guidance, encouragement and caring. Thank you, brothers, for the help and companionship. Thank you, Grandma, for your cheering messages and constant presence.

I would also like to acknowledge my dissertation supervisors Prof. Pedro Batista and Prof. Ricard Marxer for your dedication, patience, insights, support and sharing of knowledge that shaped and allowed this work to come true.

Last but not least, to all my friends and colleagues who helped me grow as a person and were there for me during the good and bad times in my life. Thank you.

I must also thank what transcends the material perceptions, that some traditions refer to as God, Allah, the Universe, Samadhi, or the Cosmos. I am grateful for this lifetime opportunity and for the chance to learn and give my contribution by means of this thesis.

To each and every one of you – Thank you.

Abstract

This thesis delves into the application of artificial intelligence (AI) techniques for real-time fine-tuning of dynamic models in surface vehicles for control purposes. In the initial stage of this investigation, two baseline nonlinear models were conceived to capture the characteristics of a conventional boat and a hydrofoil, respectively. The first model was designed to be simple, with small nonlinearities, while the second was thought to be challenging with sharp nonlinearities. From those base models, the ground-truth data were generated and used in the learning process of three AI methods for identification. Firstly, the Sparse Identification of Nonlinear Dynamics (SINDy) technique was adopted to identify both the simple and the challenging model. Secondly, a feed-forward Neural Network (NN) was trained to predict a sequence of states of the hydrofoil model, revealing its potential for application in a reinforcement learning structure. Finally, the hydrofoil model was identified offline and adjusted in real-time using a Long Short Term Memory (LSTM), a Gate Recurrent Unit (GRU), and Vanilla Recurrent Neural Networks (RNN) combined with Extended Kalman Filter (EKF). The models were trained offline by applying Backpropagation Through Time (BPTT), and later they were refined on-the-fly by the EKF. This novel approach allowed adaptation and optimization of control systems during operation. Overall, this research offers insights into the use of AI techniques, including SINDy, feed-forward NN, and RNN, to improve control and model identification capabilities in surface vehicles, particularly hydrofoils.

Keywords

RNN; Real-Time Learning; Control; Identification; Marine Vehicle Dynamics.

Resumo

Esta tese investiga a aplicação de técnicas de inteligência artificial (IA) para ajuste fino em tempo real de modelos de controlo em veículos de superfície, com foco em hidrofólios. Inicialmente, dois modelos não lineares foram desenvolvidos com o objetivo de gerar os dados de referência para o treino dos módulos de IA. O primeiro é um modelo mais simples que representa um barco convencional, enquanto o segundo é um modelo mais complexo para identificação, que retrata a dinâmica elaborada de um hidrofólio. A partir dos dados gerados pelos dois modelos básicos, foram testados e comparados três métodos de aprendizagem para identificação. O primeiro método de aprendizagem emprega a Identificação de Sistemas Dinâmicos Não Lineares Esparsos (SINDy) para identificar tanto o modelo simples como o modelo mais complexo com o hidrofólio. O segundo consiste no uso de redes neuronais para a identificação do modelo complexo, revelando o seu potencial de aplicação numa estrutura de aprendizagem por reforço. Por último, este trabalho mostrou que, ao integrar o filtro de Kalman estendido e as redes neuronais recorrentes (LSTM, GRU e Vanilla RNN), é possível refinar, em tempo real, modelos de hidrofólios identificados previamente com *Backpropagation Through Time*. Em geral, esta pesquisa compara e descreve informações importantes sobre o uso de técnicas de aprendizagem, incluindo SINDy, redes neuronais feed-forward e redes neuronais recorrentes, com o objetivo de melhorar o desempenho da malha de controlo e identificação para modelos de veículos de superfície.

Palavras Chave

RNN; aprendizado em tempo real; Controlo; Identificação; Dinâmica de Veículos Marinhos.

Contents

List of Figures

List of Tables

Acronyms

LIDAR	Laser imaging, detection, and ranging
ADCP	Acoustic Doppler current profiler
GPU	graphics processing unit
CEM	Cross Entropy Method
LQG	Linear Quadratic Gaussian
DRL	Deep Reinforcement Learning
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descend
LSTM	Long Short Term Memory
GRU	Gate Recurrent Unit
DDPG	Deep Deterministic Policy Gradient
CNN	Convolutional Neural Networks
DQN	Deep-Q-Networks
RL	Reinforcement Learning
ML	Machine Learning
DL	Deep Learning
USV	Unmanned Surface Vehicle
MIMO	Multi-input multi-output control
NN	Neural Network
MPC	Model Predictive Control
EKF	Extended Kalman Filter
LQR	Linear Quadratic Regulator

PID	Proportional, Integral Derivative Control
SAC	Soft actor critic deep reinforcement learning algorithm
SMC	Sequential Monte Carlo
ODE	Ordinary Differential Equations
PDE	Partial Differential Equations
SINDy	Sparse Identification of Nonlinear Dynamics
DOF	Degree-of-Freedom
ASV	Autonomous Surface Vehicle
AI	artificial intelligence
LASSO	Least absolute shrinkage and selection operator
AUV	Autonomous Underwater Vehicle
SARSA	State, action, reward, state, action
SLAM	Simultaneous localization and mapping
SR3	Sparse relaxed regularized regression

1

Introduction

Contents

1.1 Introduction

Autonomous or unmanned vehicles have been playing an important role in research and production fields. Such devices can be used to perform tedious or dangerous tasks for people, reducing the risk of accidents, human errors, and the cost of processes. Autonomy has been applied to self-driven street cars, flying drones, manufacturing automation, and also to the marine and maritime industry.

For the marine field, specifically, Autonomous Surface Vehicle (ASV) has the potential to improve ocean surveys, transportation, and sea security/defense [3]. The scientific community has been applying multiple efforts to overcome challenges related to Simultaneous localization and mapping (SLAM), trajectory tracking, path following, and performance of manipulators, etc. [3]

The progress of autonomous marine vessels is directly related to the development of Control Theory and artificial intelligence (AI). This master's research is based on the combination of both, and the following section will explore the motivation behind the connection of those two fields.

1.2 Motivation

1.2.1 Challenges in traditional Control Theory

The architecture for marine autonomy can be constituted by several layers, like communication, task planning, mapping, etc. [3]. Three of those modules are important and present even in simpler autonomous crafts, namely: navigation, guidance, and control. The navigation unit collects the sensed information to provide a good state estimation. Guidance uses navigation information and mission objectives to produce references to the control module. The control drives the robot to the desired state. A variety of control approaches may be used in this context. Some examples are:

- Proportional, Integral and Derivative Control (PID);
- Linear Quadratic Regulator (LQR);
- Linear Quadratic Gaussian (LQG);
- Feedback Linearization;
- Model Predictive Control (MPC).

Proportional, integral derivative feedback control has been widely applied and has been proven to be effective, simple, and efficient in terms of computational power. Nevertheless, its simplicity is also the cause of its limitation. For instance, Proportional, Integral Derivative Control (PID)'s are not suited for Multi-input multi-output control (MIMO) problems with highly coupled states.

Linear control methods, such as LQR, handle MIMO systems, balance actuator stress, and output precision through the Q, R matrices. However, LQR would not perform well under certain conditions where linearization with Jacobian matrices yields numeric instability. A car steering system, for example, is highly nonlinear due to trigonometric functions dependency and, when linearized around equilibrium points, it can be non-controllable or non-observable.

Feedback linearization is a technique that allows one to obtain control inputs that precisely reach complex trajectories. However, the requirement of relatively accurate time-invariant models is one of its weaknesses. This fact will be explored more in Section 3.1. Natural phenomena are nonlinear, noisy, time-variant, and uncertain. For some problems, these factors cannot be neglected. An example is a hydrofoil, or water fly-board, that changes its drag, added mass, and Coriolis matrices drastically as it changes its velocity. Similarly, the physical model of a simple ferry boat can be conditioned to the random disturbance ”number of people onboard”. More passengers means, in a nonlinear relationship, more contact with the water, more added mass and drag, and a displacement in the center of gravity of the vehicle [4].

There exist well-established control methods to manage the imperfections of actuators, sensors, plant models, and disturbances. When it comes to practical implementation and control of those methods in marine robots, a lot in terms of tuning and adjusting must be done manually. This is particularly true for low-cost devices and also to carry out complex tasks, such as station-keeping for interventions with a manipulator [3]. Some common manual procedures are:

1. tuning P, I, D coefficients;
2. defining the right equilibrium between cost — Q and R — matrices;
3. preparing meticulously specific tests under very precise (constrained) conditions;
4. executing carefully and repeatedly the tests to find the expert model.

1.2.2 Artificial Intelligence supporting Control Theory

The tuning process requires the work hours of experts, and the application of AI could reduce human labor in those situations. AI and Control are not completely disconnected, though. Adaptive control has been expanding the boundaries of traditional control. This is a ”gray zone” that can be part of either the AI or the control universe [5]. AI is known for extracting information from data (even from what could be considered noise). It has the potential to support Control Theory, enhance its results, and fill some gaps that still have been challenging engineers.

AI has observed a surge of research and applications in engineering within the past 20 years [6]. The exponential growth of integrated circuits powered the advent and practical use of learning-based

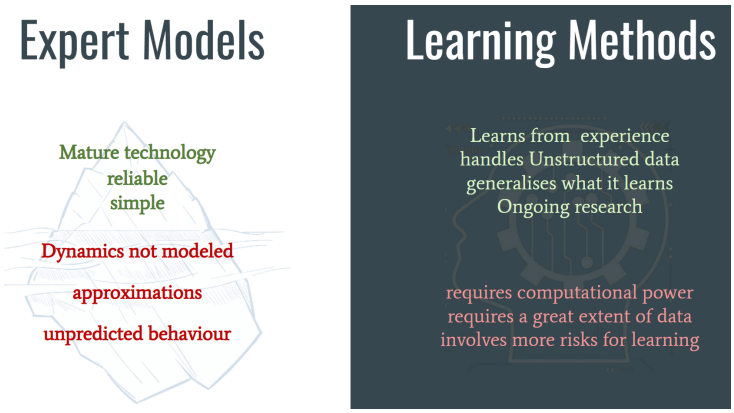

Figure 1.1: Overall comparison between learning and expert-driven methods.

methods. Regressors and classifiers have been improving to support decision-making in a variety of areas. Artificial Intelligence is a broad concept, and in this work, it will be referred to as the most usual learning-based frameworks, which are Machine Learning, Deep Learning, and Reinforcement Learning. There is no strict boundary that says whether an algorithm is classified as Machine, Deep, or Reinforcement Learning, but Machine Learning can be considered a more general area of knowledge that comprises Deep and Reinforcement Learning [1]. Figure 1.2 shows in more detail AI from general to particular cases.

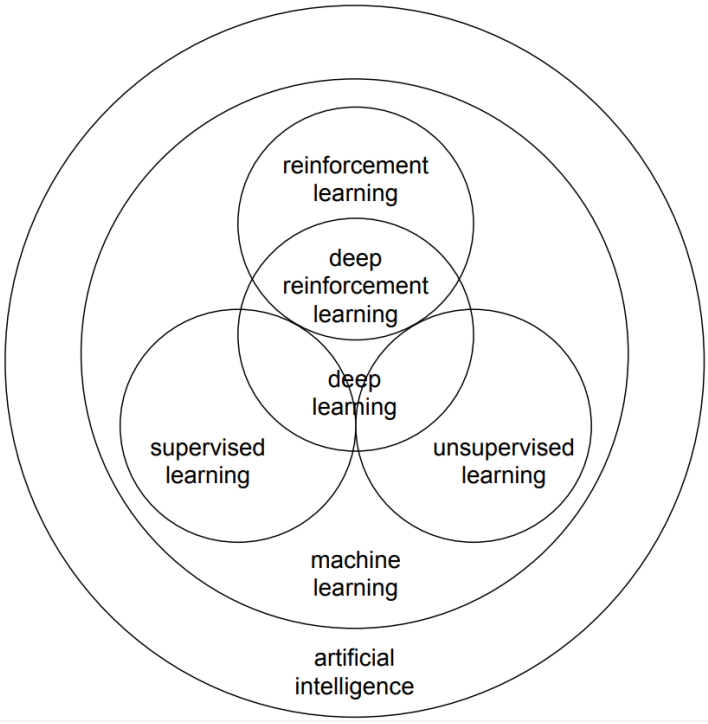

Figure 1.2: Machine Learning, Deep Learning, and reinforcement learning as study areas. [1]

Machine Learning (ML) has been widely used to boost businesses by exploring the statistical connections between different data features [7]. Deep Learning (DL) has been exhaustively applied in speech recognition [8], image processing and classification, etc [9]. Reinforcement Learning (RL) has been successfully applied to teach computers how to outperform humans in chess or go [10]. With RL, Robots learn through experience to execute tasks. In fact, RL can be considered one of the many crossroads where Control Theory and AI intersect.

Considering this scenario, the advantage of managing non-structured data of ML, DL, and RL should be investigated to improve control for marine robots. This may be useful for a reliable identification process or for robust control with real-time self-adjustment. In summary, the application of AI in control can potentially:

- reduces the repetitive and tedious labor of the engineer teams;
- reduces the time spent during the experimental stage;
- improves the performance of the robot;
- improves reliability.

1.3 Thesis outline

The following Chapters will present some of the main academic achievements and challenges to connect AI and control theory, as well as the contribution of this work to the field of study. More specifically:

- Section 1.2 outlines the motivation to use AI as a complementary tool in control of marine vehicles;
- Chapter 2 is a literature review with recent and relevant references for this work;
- Chapter 3 is dedicated to bound the problem tackled, and to define the baseline for the methods presented in Chapters 4 and 5;
- Chapter 4 presents an investigation over Sparse Identification of Nonlinear Dynamics (SINDy) applied to surface vehicle identification, with advantages and drawbacks of this method;
- Chapter 5 is the main focus of investigation in this thesis, it describes how neural nets can be trained to identify dynamical systems and how they can be refined in real-time with Extended Kalman Filter (EKF);
- finally, Chapter 6 summarizes and compare all the approaches presented in Chapters 4 and 5.

2

State of the art

Contents

The achievements and challenges that some of the most recent and praised works involving Marine Robotics, Artificial Intelligence, and Control Theory will be presented in this Chapter 2. In addition, Section 2.5 will provide the background for marine modeling that will be used in Chapters 3, 4, and 5.

2.1 Adaptive and stochastic control

Adaptive control is a versatile way of controlling devices by reacting properly to the uncertainties involved in modeling and unpredictable environmental conditions that inflict the system [10].

MPC takes into consideration the provided model, sensors, and actuators, and runs an optimizer cyclically. The optimizer should converge to a control sequence that minimizes a cost function with terms related to the error along the trajectory and terms related to the stress applied to the actuators. To find the optimized control, MPC predicts what the trajectory should be within the next steps given a sequence of control commands that would be sent to the plant. In other words, MPC continuously uses the current state and available model information to create a prognostic for future states and adjust the sequence of control commands accordingly [11].

Since an optimizer runs frequently, MPC has the inherent advantage of dealing with trajectory, input and output constraints. Due to its advantages, MPC has been widely applied to solve different problems in industrial processes and vehicle autonomy. [12] applied the concept of MPC ally with a set of predefined rules to control the trajectory of a ship and avoid collisions.

[4] successfully applied the MPC concepts to control a surface vehicle. The aim of the authors was to develop a control system for an autonomous transportation boat. The authors built a miniature of an autonomous vehicle, applied nonlinear least squares to identify the model, and used MPC to track the trajectory. Firstly, they used a simulation environment for preliminary tests. Afterward, the surface vehicle miniature was used to conduct their experiments. Through the simulations and the experimental tests, they showed that the prototype followed with time and space accuracy the desired trajectory.

[13] presented a methodology that might be considered in the domains of model predictive control, and also a deep reinforcement learning algorithm. The authors built a data-driven module to control the gait of a four-legged robot. They trained a feedforward NN to output the next 20 state variations $\delta s_t, .., \delta s_{t+H}$, given a sequence of actions $(a_1, a_2, ..., a_H)$ and the state s_t. In other words, it learns how to predict the behavior of the physical model, while the MPC takes care of planning the next optimal sequence of actions to attain the control target. With a model capable of providing the 20 next system states, it is relatively simple to optimize the sequence of actions through a sampling process. Therefore, the MPC works by sampling different sequences of actions and fitting the population of sequences into a distribution of the best sequences. The criterion for defining the best sequence is the Cross Entropy Method (CEM), calculated between the result of the sequence and the desired trajectory.

Other adaptive methods in control, have also been proven to be effective. For instance, [14] uses the Sequential Monte Carlo (SMC) arrangement to deal with uncertainty and to apply ML with Bayesian assumptions. The authors used control concepts to find a proper model for the evolution of the disease "Dengue Fever". SMC is the stochastic framework behind the particle filter concept. The idea is to estimate the state x(t) by optimization of the likelihood among random guesses of the current state. Such guesses are known as particles.

The particles are compared with the data obtained from measurements in order to reach the identified process. The more statistically consistent the trajectory of the particles with the measured trajectory of the controlled system, the greater the chances that this particle is a good guess of the current state. As time passes, the position of the particles converges to what should be the expected real state. The advantage of using this method over Kalman Filter, for example, is that Kalman filters perform well when the noise from measurements and the model errors can be approximated to Gaussian distributions. This is not the case for a problem of simultaneous localization and mapping, for example.

2.2 Machine Learning associated with control methods

As shown in Section 2.1, [14] demonstrated how SMC could be a powerful tool to work with complex models such as the progression of a disease. To some extent, their approach could be classified as ML. Other methods are a more direct application of ML, one example is the methodology presented by [15].

[15] created an identification method based on machine learning, SINDy. The authors explained that other approaches, such as symbolic regression or nonlinear least squares, expanded the capabilities of identifying linear and nonlinear systems, but there are some caveats. For example, symbolic regression is computationally expensive, works poorly with large systems, and is prone to overfitting. On the other hand, ML is more flexible. The procedure presented by [15] tackles the control of sparse problems to identify Ordinary Differential Equations (ODE)'s and Partial Differential Equations (PDE)'s. Sparse means that each state variable derivative is a function of a few state variables. [15] claim that physical problems with high dimensions are commonly sparse. This means that only some of the state variables are dominant when coupled to other state variables. This is true for specific cases of the 6 degree-of-freedom model of a marine vessel that will be presented in Section 2.5. In this case, if a surface or underwater vehicle is symmetrical, slender, and designed to be stable, the cross dependency among different states becomes weak, and the model becomes sparse.

SINDy uses time-stamped measurements to estimate the nonlinear coefficients of the state derivatives with a regularization factor. This approach is equivalent to the machine learning regression analysis known as Least absolute shrinkage and selection operator (LASSO).

The advantages of SINDy are its flexibility to fit nonlinear equations and the availability of python

libraries with ”ready to use” functions [16]. Machine learning techniques are relatively simpler and, therefore, faster than deep learning or reinforcement learning. The caveat is the necessity of adjusting well the regularization hyperparameter λ and the limitations imposed by the terms chosen to compose the nonlinear system. Also, it is still necessary to prepare and clean the input data. The use of this technique will be explored in depth in Chapter 4.

2.3 Deep Learning and vehicle control

Neural network graphs composed of several layers with linear or nonlinear transformations are known as Deep Learning. Authors have been experimenting DL to improve control in complex environments. For example, the work of [13] uses MPC and deep neural networks to drive a legged robot vehicle. More details are explained in Section 2.1.

[17] is a bibliographical review of deep learning applied to control autonomous vehicles. The paper focuses on street cars, but the concepts and conclusions can be extended to marine vessels. It shows the potential and limitations of DL with respect to computational power, control objective, architecture selection, generalization, and reliability. The authors of this paper state that raw control techniques (the ones that do not use learned-based techniques) had to counterweight their stiffness to cope with complexity by using expensive and accurate sensors, such as Laser imaging, detection, and ranging (LIDAR). They explain that many parameters need to be precisely hand tuned in those cases because of the limitations of the controllers based on a predefined set of rules. It is also elucidated in their paper that DL appears as an alternative to those hindrances. Some of the advantages of deep learning are: it self-optimizes through data; it is more flexible to deal with variability; it processes multidimensional large inputs like images. The information generated by deep neural networks can be used, among others, in the context of collision avoidance, SLAM and optimal path. Considering the marine environment, input data coming from sonars, Acoustic Doppler current profiler (ADCP), lasers or cameras can be consumed by deep neural networks to produce information for decision taking and control commands.

To evaluate the performance of the works in autonomous driving carried out with deep learning, [17] divides the problem into two different control objectives: longitudinal speed and position control; lateral (steering) control. The authors point out that most of the results obtained with deep learning applications for robot control originate from simulated environments. There are some factors that justify this trend. Real world experiments can increase the risks of accidents, they are more expensive and it is laborious to assemble the experimental. Finally, the authors list important challenges in deep learning (it also considers RL), some are:

- overfitting the solution;
- inability to test in all possible scenarios;

- high cost of field testing;
- finding the right DL architecture for the problem;
- designing a comprehensive reward function when deep learning is applied with RL;
- safe training in the real-world.

Deep learning can also be trained to replicate the system dynamics. [18] and [19] apply deep neural networks for identification of dynamical systems. [19] explain the similarities between deep nets and control identification. The authors elucidate that both deep neural networks and identification methods can be modeled and tuned with statistical apparatus like variance-bias, cross-validation, and residual analysis. Through practical examples, they demonstrate the effectiveness of NN in playing the role of a canonical control model. [19] also carried out some tests to check which type of deep neural networks would be more suitable to be a replica of a dynamical system. They reached the conclusion that the cascadeforwardnet presents the same performance of more complicated solutions.

In the matters of using Neural Network (NN) as a surrogate for dynamic systems, [20] uses EKF as an alternative to Stochastic Gradient Descend (SGD) method to train Recurrent Neural Networks (RNN)'s with loss functions that are convex. The proposed solution considers regularization terms like L1 and L2. The authors show that the Kalman Filter method has a performance equivalent to SGD in a non-linear system identification benchmark and in training a linear system for classification. Additionally, the authors of the study investigate the application of a data-driven algorithm in nonlinear model predictive control. They analyze its relationship with disturbance models to achieve offset-free closed-loop tracking.

2.4 Reinforcement Learning applied to robotics

Reinforcement Learning and Adaptive Control pursue similar objectives. As a matter of fact, [10] and [21] shows that Adaptive Control is a wide area that could entail learning-based methods, such as Reinforcement Learning. Reinforcement Learning is one of the AI frameworks that allows learning through experience. The idea of RL is having an agent that interacts with the environment and learns through the interactions how to maximize rewards or to reach a goal. Commonly used RL and Deep Reinforcement Learning (DRL) algorithms like State, action, reward, state, action (SARSA), QLearning, Monte Carlo, Deep Deterministic Policy Gradient (DDPG) are modeled with the basic concepts:

- an environment;
- an agent capable of taking actions in the environment;
- the current state (s), that encompasses the agent and environment;

- the actions (a) that the agent can execute to go from state (s) to state (s');
- a final state that can be a goal, failure, etc;
- a reward function that reflects how close the pair (a—s) is approaching the goal;
- a policy that returns the action, given the state;
- a trajectory from the initial state to final state.

[22] is one of the most emblematic applications of Reinforcement Learning. In this work, the authors showed the power of model-based RL to learn from scratch how to play complex games like go, chess, shogi, and even Atari games. Their approach is called MuZero. One of the ingenious features of the proposed method is how the states are processed. Since Atari, shogi, chess, and go have 2D current state representations, Convolutional Neural Networks (CNN)'s were used to synthesize the multidimensional sequence of states before applying the RL algorithm. Their approach outperformed humans by far when playing go, chess, and most of Atari games without knowing the rules previously. The algorithm is built over a Monte Carlo tree search structure where each node is a state representation of the game and the possible actions are the edges of the three. The algorithm should converge to reach two functions — f and g — that describe the dynamics of the problem. The function $s_{t+1} = g(s_t, a_t)$ yields the state transition from the node s_t to the node s_{t+1}, given an action (a). f is equivalent to the policy and outputs the optimal action $a*$ and the reward r. In a further publication, the authors added a new feature to the model of [22] (Muzero). In [23], they included a module called "Reanalyse" capable of learning new experiences online.

Despite the expressive results, RL has limitations that must be addressed whenever it is used for physical systems. The sample inefficiency, issues with formal consensus of stability as well as risks and limitations associated with real hardware defy RL applications for robotics. [17] described in their survey how convergence to policy optimality can be inaccurate, slow, and how it can require a massive number of simulations or real-world training steps. To avoid this difficulty with physical data, most of the references trained their RL for robotics in a simulated reality. Nevertheless, for specific applications, RL is already contributing to robotics.

RL can be applied to many different stages of autonomy of a marine vehicle. Some researchers applied it directly in the control layer, others applied RL to do the path planning with the control. [24] used RL in the guidance decision layer. The authors developed a method that learns to choose among three different guidance modes to keep a safe but effective trajectory. The RL algorithm is trained to decide whether guidance should be in the port collision avoidance mode, starboard collision avoidance mode, or path-following mode. Each one of those guidance strategies supply a PID steering and speed control layers with a target trajectory.

[24] takes advantage of the neural networks power to attain features from images obtained from an onboard camera. The collected surrounding images are inputs for the DRL algorithm. This means that the RL method learns how to identify obstacles in the picture and choose the appropriate strategy. Since the output of the DRL is discrete, the authors could use Deep-Q-Networks (DQN). The state is defined by the position, heading, velocity, and a sequence of 4 images, so the network could recover information regarding the approximation speed to obstacles.

[25] uses reinforcement learning to aim two targets simultaneously: tracking control and collision avoidance. The authors use classical control in an autonomous surface vehicle and reinforcement learning works as a control supervisor that provides intelligence for adaptation in case of nearby obstacles. The authors claim that the method is potentially more efficient than using pure RL and, because of the classical control module, they could also prove mathematically the convergence of the method to the desired trajectory. The method was only tested in a simulated environment.

[25] uses Soft actor critic deep reinforcement learning algorithm (SAC) as the reinforcement learning method. SAC presents similarities with DDPG. However, it is an entropy regularized reinforcement learning method, which means that a certain degree of randomness in the policy is rewarded in the beginning of the training process, and it vanishes as convergence is reached. This procedure is directly related to the trade-off of exploring/exploiting in RL [26]. The reward function used by [25] is based on the current distance and estimated velocity between obstacle and vehicle. In the absence of obstacles, the RL module does not add any adjustments to the thrusters, and solely traditional control is applied.

[27] used neural networks and a RL framework to control an Autonomous Underwater Vehicle (AUV). The researchers developed a method that uses an action and critic network. The critic network manages the long-term decisions to aim trajectory tracking, while the action network is trained to deal with the dynamics. [27] obtained good simulated results, but no real world tests were carried out. The AUV was capable of following a trajectory with better performances than conventional proportional derivative uncoupled controllers. The authors claimed that their method is robust against noise and disturbances.

[28] presented a method with deep reinforcement learning, with a quality table Q, where an Unmanned Surface Vehicle (USV) learns path following and collision avoidance by controlling its position and velocity. The authors define a reward function based on the distance to the obstacle and the distance to the desired path. The state is formed by: current position, current heading, current angular and linear speed, previous propeller commands (previous action) and obstacle distance. [28] uses a sequence of previous states as an input to a CNN that yields the next chosen action. As the action is taken and the next state is achieved, the reward is calculated and the Q table is updated. The authors obtained reasonable simulated results and they highlighted that an environment with disturbances requires much more training episodes.

While all the results obtained by [28] were extracted solely from simulations, [29] simulated and tested

their work with a real boat. [29] proposed a method that uses a DRL architecture derived from DDPG for path following of an USV. The USV had a catamaran shape and was pushed by two parallel thrusters. Since the action and state space in DDPG are continuous, the control commands sent to the propellers were not discrete, as in [28]. [29] comprised guidance and control simultaneously with reinforcement learning. They used the concept of vector field to build a reward function that penalizes the actions that steer away the USV from the desired path. An effective training required thousands of episodes and for practical reasons it was carried out in a simulated environment.

The works referred to so far in this section used RL to train their robots from scratch, without any previous predefined policy. However, it is viable to take advantage of expert knowledge to accelerate the convergence or to improve the performance. Some researchers used expert system strategies as a starting point and the AI to add robustness and flexibility to the control loop.

[30] developed a structure to control robots with proportional-integral control and deep reinforcement learning. In this approach, the PI coefficients were manually set with theoretical values. The agent of the deep reinforcement learning algorithm tuned the PI values as time passed. With this approach, the authors could manage the coupling effects between different states and control a MIMO system. The RL agent ensured that, independently of the state, a well adjusted control would bring the robot to the targeted trajectory. Their method took care of the existent nonlinearities, variable coupling and model deviations. The authors used an adapted DDPG to find a RL policy. The reward funcion was the gaussian $r = \exp -[\frac{v_{req}-v_t}{\sqrt{2}\sigma}]^2 - 1$, where v_{req} is the requested velocity, v_t is the vehicle's velocity and σ is the standard deviation. The algorithm was trained and tested in a simulated environment.

[31] used RL and classical control to steer a robotic arm to place objects in a specific position considering uncertain environmental conditions. The control problem was mathematically addressed as a state that is affected by precedent states and the control action u(t). The mathematical structure adopted (2.1) has common ground with discrete control systems and with the description of RL.

$$\begin{bmatrix} s_m(t+1) \\ s_o(t+1) \end{bmatrix} = \begin{bmatrix} A(s_m,t) & 0 \\ B(s_m,s_o,t) & C(s_o,t) \end{bmatrix} \cdot \begin{bmatrix} s_m(t) \\ s_o(t) \end{bmatrix} + \begin{bmatrix} D \\ 0 \end{bmatrix} u(t). \tag{2.1}$$

In (2.1), s_m is the state component impacted by the control u(t) and that is not dependent on s_o. $A(s_m,t)$ is the function that couples previous states with new states. s_o represents the components of the state that are impacted by s_m but not directly by the action u(t). $C(s_o,t)$ update function for the "hidden" state s_o. (2.1) has a component s_m that is directly affected by the control u(t), and a component s_o that depends on s_m but it cannot be controlled directly. By this approach [31] could use a DDPG to learn through simulated training how to handle unexpected and not explicitly modeled conditions. The

policy space is defined as

$$u(t) = \pi_H(s_m) + \pi_\theta(s_m, s_o) \tag{2.2}$$

$\pi_H(sm)$ is pre-defined by the control law and $\pi_\theta(s_m, s)$ is learned by trial and error.

2.5 Marine robotics modeling

The definition of a model for marine robotics is crucial for the development of good simulators. It is also essential to properly design classical control methods, such as PID, Feedback Linearization, etc. [32] is a comprehensive handbook with theoretical and practical knowledge to describe the dynamics comprised by marine vehicles. The methodologies to obtain a lumped dynamical model from this handbook will serve as a reference in Chapters 3, 4 and 5.

The motion of a marine craft can be depicted by two complementary fields: kinetics and kinematics. Kinematics is related to the studies of the geometrical parameters of the environment and vehicle. For example, the frame of reference is in the domain of kinematics. Kinetics, on the other hand, is the study of the forces that drive the motion of the physical parts involved. For a marine vessel, the representation extracted from [32] that correlates acceleration, forces and torques is given by

$$\tau = M\dot{\nu} + C(\nu)\nu + D(\nu)\nu, \tag{2.3}$$

where: τ are the external forces and torques applied to the vessel; C, M, and D are Coriolis, mass, and drag matrices. ν are the linear and angular velocities of the vessel in the body frame. Considering 6 degree-of-freedom,

$$\nu = [u, v, w, p, q, r]^T. \tag{2.4}$$

In more detail, u, v, and w are the surge, sway, and heave (linear) velocities, while p, q, and r are the roll, pitch, and yaw (angular) velocities. Since the velocity components ν are expressed in the body frame, it is convenient to transform it to the inertial (world) frame η. The transformation can be done with

$$\dot{\eta} = R(\eta)\nu, \tag{2.5}$$

where η are the world frame coordinates and R is a transformation matrix (from body to world).

The mass matrix (M) in (2.3) is formed by two terms, the rigid body mass plus another component known as the added mass. The added mass is the portion of water that is conveyed by a rigid body immersed in movement that contributes to the total mass/inertia properties of the system. The drag coefficients (D) in (2.3) are compounded by skin and pressure drags. The skin component can be

compared to the friction between the robot's surface and the water. The pressure drag occurs due to the pressure differences generated during the movement of a robot part in the water.

With few exceptions, surface vehicles are designed with their center of gravity below their center of buoyancy. This feature yields a natural stability that allows some simplifications in (2.3). [32] mentions that if the impact of roll is negligible and the vessel is designed to be stable, the model can be shortened to a 3 Degree-of-Freedom (DOF), $[x, y, \psi]$. [33] neglected roll, pitch, and heave and their couplings for control purposes and still obtained good results. So, considering the 3 DOF model, η, ν and R in (2.5) and (2.3) are denoted by

$$\eta = [x, y, \psi]^T,\ \nu = [u, v, r]^T, \tag{2.6}$$

$$R = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.7}$$

In the 3 DOF model, considering that the part of the boat that is in contact with the water is a slender body —the size along the x axis is considerably longer than y —and considering the symmetry in the x-z plane, the mutual interactions between x-ψ and x-y can be neglected. So, the coupling occurs only between yaw and sway for mass and drag matrices, as it can be seen in (2.8). In addition, for small velocities, the drag can approximated as a linear function of velocity. Mass, drag, and Coriolis matrices — M, D, C — from (2.3) can be approximated

$$M = \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & m_{23} \\ 0 & m_{32} & m_{33} \end{bmatrix}, \quad D(\nu) = \begin{bmatrix} d_{11} & 0 & 0 \\ 0 & d_{22} & d_{23} \\ 0 & d_{32} & d_{33} \end{bmatrix}, \quad C(\nu) = \begin{bmatrix} 0 & 0 & c_{13} \\ 0 & 0 & c_{23} \\ c_{31} & c_{32} & 0 \end{bmatrix}. \tag{2.8}$$

3

Problem Statement

Contents

3.1 Problem statement

Section 1.2 briefly addressed the problems that might arise in control when a model is not a good representation of the real dynamics. Deviations are prone to occur in highly complex, time-changing environments or when low-cost sensors and actuators are used.

This is the case for surface vehicles and hydrofoils are especially subjected to model complexity and unpredictability. This section is dedicated to describing the problems that arise in hydrofoil modeling and where learning methods could be applied.

3.1.1 The complexity and difficulties of marine robotics models

Modeling errors are particularly relevant in marine robotics because the dynamics occur in a fluid. For robotics applications, water can be considered an incompressible Newtonian fluid. This means that underwater or surface crafts have models with components that depend on the differential equations, so-called Navier-Stokes.

The Navier-Stokes equations are a challenge for engineering because they produce complex phenomena that are hard to predict, and to be completely represented. Consequently, in some cases simplified submarine and boat models can miss important dynamical features. (2.3) extracted from [32] of Section 2.5 is an ingenious way of contouring the complexities of hydrodynamics by inferences of empirical knowledge and applied physics. Despite the solid foundations behind the equations presented by [32] and their vast applicability, the contrast between theory and reality can build up significant errors. One of the differences between reality and theory in surface vehicle models is the calculations of added mass and drag coefficient.

Firstly, added mass and drag are hard to calculate because they depend on geometry — distance from center of buoyancy to center of gravity r_g^b and shape of the vehicle—, change of frame references, physical properties, and numerical partial integrals. Second, these calculations are subject to present differences with the real dynamics even if they are carefully carried out. This occurs because the concept of added mass and drag are inherently an approximation of real-world dynamics.

Similarly, rigid body masses and Coriolis are also strongly related to the shape and will impose uncertainties in the model development. Other model simplifications can also produce differences between model and plant, like the 3 degree-of-freedom simplification cited in Section 2.5.

3.1.2 Problem statement - extreme dynamics of hydrofoils

The estimation of a lumped model with matrices becomes even more complicated for a hydrofoil, as briefly mentioned in Section 1.2. Hydrofoils change the surface of contact with the water, more drastically than conventional boats, as the velocity increases. Figure 3.1 illustrates this behavior. Fitting a

dynamical representation for a hydrofoil is a burdensome task, and it will serve as a baseline for the investigation of this thesis. In summary, the tackled problems are:

1. the limitations and problems for conventional identification with lumped model for hydrofoils;
2. the application of Machine Learning and Neural Networks as an alternative to create a model that adapts well;
3. the restrictions of ML and NN;
4. the application of real-time learning to overcome some of the limitations of ML and NN;
5. concerns and measures when real-time learning is applied.

Figure 3.1: Two different modes of a hydrofoil. The normal mode is characterized by low-speed/high-drag while the flying is high-speed/low-drag.

3.2 Baseline for the investigation

One way to represent a hydrofoil is by creating 2 different models and establishing a continuous (but steep) transition between the two modes. Each one of the 2 models can follow the modeling defined by (2.3), (2.6) and (2.5). Considering a hypothetical model of a hydrofoil with a propeller and a rudder, the

state variables X are:

$$X = [x, y, \psi, u, v, r, P_f]^T. \tag{3.1}$$

In (3.1): x, y are the coordinates in the world frame; u, v, r are the velocities in the body frame, already defined in (2.4); ψ is the yaw angle; P_f is the actual force provided by the propeller. For a hydrofoil propelled by a DC motor, it is more realistic to consider first-order dynamics between the control signal and the actual force obtained with a time constant in the order of 0.1 to 1s. In this case, 1 s is assumed for the propeller motor.

The control variable μ can be defined as

$$\mu = [PP_{cmd}, RD_{cmd}]^T; \tag{3.2}$$

$$RD_{cmd} = K_{RD}\, sin(RD_{ang}), \tag{3.3}$$

with PP_{cmd} being the control reference for the propeller force, and RD_{cmd} being a variable that yields the rudder torque when multiplied by the surge velocity u. The constant K_{RD} represents the relation between the physical properties of the rudder, such as size and shape, and the torque produced. RD_{ang} is the angular position of rudder that goes from $\frac{-\pi}{2}$ to $\frac{\pi}{2}$.

If the drag is linear, the nonlinear dynamical system of a boat in its canonical form becomes

$$f(X) + g(X)\mu = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{v} \\ \dot{r} \\ \dot{P}_f \end{bmatrix} = \begin{bmatrix} u\,cos(\psi) - v\,sin(\psi) \\ v\,cos(\psi) + u\,sin(\psi) \\ r \\ \frac{(P_f + r\,(m_{22}\,v + m_{23}\,r\,x_g) - d_{11}\,u)}{m_{11}} \\ \frac{-r\,u\,x_g\,m_{23}^2 + (RD_{cmd}\,u - d_{33}\,r - d_{23}\,v)\,m_{23} + d_{23}\,m_{33}\,r + d_{22}\,m_{33}\,v + m_{11}\,m_{33}\,r\,u}{m_{23}^2 - m_{22}\,m_{33}} \\ \frac{m_{23}\,(d_{22}\,v + r\,(d_{23} + m_{11}\,u)) - m_{22}\,[u\,(m_{11}\,v + m_{23}\,r\,x_g) - RD_C md + d_{33}\,r + v\,(d_{23} - m_{11}\,u)]}{(m_{22}\,m_{33} - m_{23}^2)} \\ PP_{cmd} - P_f \end{bmatrix}. \tag{3.4}$$

In (3.4), d_{ij} are m_{ij} are the dissipative and mass terms from matrices D and M in (2.8). x_g is the displacement between the gravity and buoyancy center of the boat in the x coordinate. This term appears in (3.4) due to the Coriolis effect and the added mass. Coriolis also affects the system equations by adding quadratic velocity terms and mass-dependent terms.

To represent the ”flying” and ”normal” dynamics of a hydrofoil, 2 different models based on (3.4) may be combined. One model would have higher drag and added mass coefficients (low-speed mode) whereas the other would have lower drag and added mass coefficients (high-speed or fly mode). The two models should operate separately in different velocities and there should be a continuous transition between them as velocity crosses a threshold. If the combination of the 2 modes is carried out with

sigmoid functions, the system

$$f(X) + g(X)\mu = \sigma(u - u_0)\left[f_{fly}(X) + g_{fly}(X)\mu\right] + \sigma(-u + u_0)\left[f_{low}(X) + g_{low}(X)\mu\right] \tag{3.5}$$

defines the hydrofoil dynamics. In (3.5), the subscript "low" refers to the low-speed mode, while "fly" refers to the hydrofoil flying mode. σ is the sigmoid function and it is given by

$$\sigma(x) = \frac{1}{\exp\left(-12\frac{s}{m}\,x\right) + 1}. \tag{3.6}$$

The greater is the number that multiplies x in the exponential, the steeper is the transition yielded by the sigmoid function. All the simulations that refer to (3.5) in Chapters 4 and 5 use the value of $12\frac{s}{m}$ shown in (3.6).

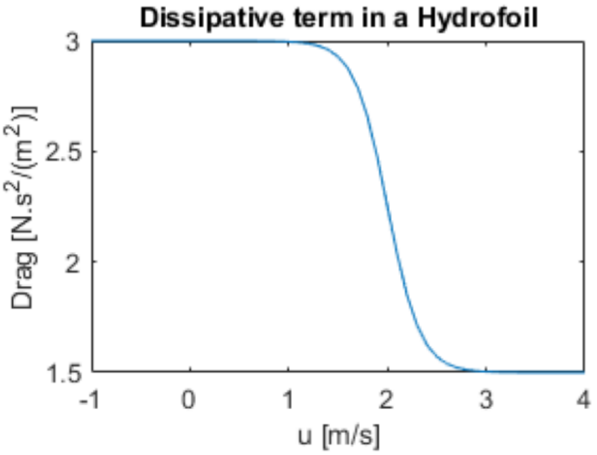

Figure 3.2: Example of how the dissipative terms of a hydrofoil can be modeled. The chart shows the drag as a function of the surge velocity. As velocity increases and the hydrofoil enters into the "fly" mode, the drag reduces.

For the cases where it is necessary to express the non-linear behavior of the drag, it can be more convenient to express it in a quadratic form. In this situation, the function that maps the derivatives of

the states becomes

$$f_2(X) + g_2(X)\mu = \begin{bmatrix} u\,cos(\psi) - v\,sin(\psi) \\ v\,cos(\psi) + u\,sin(\psi) \\ r \\ \frac{P_f - (d_{11}u) + (m_{23}x_g r^2) + (m_{22}rv) - (d_{11a}|u|u)}{m_{11}} \\ \frac{P_f - (d_{11}u) + (m_{23}x_g r^2) + (m_{22}rv) - (d_{11a}|u|u)}{m_{11}} \\ \frac{C_{1r}r - (RD_{cmd}m_{23}u) + C_{1v}v - (m_{11}m_{33}ru) + C_{1|r|}|r|r + C_{1|v|}|v|v + (m_{23}^2 x_g ru)}{(-m_{23}^2 + m_{22}m_{33})} \\ \frac{(RD_{cmd}m_{22}u) + C_{2r}r + C_{2v}v + (m_{11}m_{23}ru) + C_{2|r|}|r|r + C_{2|v|}|v|v - (m_{22}m_{23}x_g ru)}{(-m_{23}^2 + m_{22}m_{33})} \\ PP_{cmd} - P_f \end{bmatrix}, \tag{3.7}$$

with

$$C_{1|r|} = (d_{33a}m_{23} - d_{23a}m_{33}),\ \ C_{1|v|} = (d_{32a}m_{23} - d_{22a}m_{33}),\ \ C_{1v} = (d_{23}m_{23} - d_{22}m_{33});$$
$$C_{1r} = d_{33}m_{23} - d_{23}m_{33},\ \ C_{2|r|} = (d_{23a}m_{23} - d_{33a}m_{22}),\ \ C_{2|v|} = (d_{22a}m_{23} - d_{32a}m_{22}),$$
$$C_{2v} = (d_{22}m_{23} - d_{23}m_{22}),\ \ C_{2r} = (d_{23}m_{23} - d_{33}m_{22}).$$

The subscript ”a” in the drag coefficients of (3.7) is just to differentiate quadratic drag and linear drag coefficients.

3.2.1 Applying Feedback Linearization to control the hydrofoil and generate datasets for the learning methods

To generate data input for the learning methods that will be investigated, a variety of trajectories must be extracted from the fictitious system outlined at the beginning of this section. Specific trajectories, such as a lawnmower, can be generated from the hydrofoil dynamical system (3.5) with Feedback Linearization. To apply this technique, the dynamical model of the surface vessel should be represented in the form

$$\dot{X} = f(X) + g(X)\mu, \tag{3.8}$$
$$Y = h(X), \tag{3.9}$$

with h(X) mapping the state variables X to the desired output Y, f(X) representing the nonlinear dynamics and g(X) the nonlinear relation of the input μ and the state variables.

The linearizing feedback applies a transformation into the system, such that the nonlinear system with nonlinear control feedback from (3.8) can be represented by the transformed state variables vector ξ and transformed feedback control $\tilde{u}$ with linear relations. The transformed model can be represented

as

$$\dot{\xi} = A(X)\xi + b(X)\tilde{u}, \tag{3.10}$$

$$w = C\xi, \tag{3.11}$$

$$\mu(X, e) = A(X)^{-1}(e - b(X)). \tag{3.12}$$

A(X) and b(X) are matrices built from the differential relations between f(X) and Y. μ will be finite if A(X) is non-singular (invertible). e is the feedback error that will be linearized.

The intuitive idea behind Feedback Linearization is to add terms in the feedback that compensate the nonlinear effects of f(X) and g(X) to produce a command μ to track a certain trajectory. (3.10) is the linear representation of the nonlinear system error. If ξ gradually approaches 0, the difference between the desired trajectory τ and Y(t) will also be reduced to zero. Errors in the estimation of functions f(X) and g(X) from (3.8) will propagate to the transformation in (3.10). Hence, model errors will arise as distortions in the control law.

The control objective under analysis will be trajectory tracking. To simplify the problem, the controlled variables will only be the coordinates x and y. The output Y is given by

$$Y = [x,\, y]^T. \tag{3.13}$$

To generate the linearization feedback, it is necessary to find the differential delays matrix with respect to (3.5). The differential delay matrix provides the number of integrals necessary to obtain the controlled state variables starting from the control command. This matrix is used to verify if any output delays would be required to avoid an ill-conditioned feedback system. The column with the smallest derivative degree should be chosen to establish the relations of the feedback control. For this problem, the differential delay matrix is given by

$$R = \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}.$$

Since the number of integrations to reach the two controlled variables $(x,\ y)$ starting from the two control commands $(PP_{cmd},\ RD_{cmd})$ is 3 in all the cases, two columns can be chosen independently to develop the feedback expressions. Hence, it is not necessary to add a delay to control the output. The resulting expression for the feedback is

$$\begin{bmatrix} \dddot{x} \\ \dddot{y} \end{bmatrix} = \begin{bmatrix} a_{11}(X) & a_{12}(X) \\ a_{21}(X) & a_{22}(X) \end{bmatrix} \begin{bmatrix} PP_{cmd} \\ RD_{cmd} \end{bmatrix} + \begin{bmatrix} b_1(X) \\ b_2(X) \end{bmatrix}, \tag{3.14}$$

with a_{ij} being the coefficients of matrix A mentioned in (3.10), b_{ij} being the coefficients of matrix b mentioned in (3.10).

The complete expressions found for A and b are not trivial (more than 20 terms for each matrix element) and they will not be expressed in this document for better readability. The linearization feedback should reduce to 0 the errors of X up to the second derivative and provide the feedback that is dependent on the third derivative. It is given by

$$\mu = A \cdot (\begin{bmatrix} \alpha_0(x_{ref} - x) + \alpha_1(\dot{x}_{ref} - \dot{x}) + \alpha_2(\ddot{x}_{ref} - \ddot{x}) + \alpha_3 \dddot{x}_{ref} \\ \alpha_0(y_{ref} - y) + \alpha_1(\dot{y}_{ref} - \dot{y}) + \alpha_2(\ddot{y}_{ref} - \ddot{y}) + \alpha_3 \dddot{y}_{ref} \end{bmatrix} - \begin{bmatrix} b_1(X) \\ b_2(X) \end{bmatrix}). \tag{3.15}$$

$x_{ref}(t)$ and $y_{ref}(t)$ are the time-variant references applied to the surface vehicle control loop. The α coefficients are the gains of the equivalent linear representation that drives the trajectory error to 0.

If the determinant of matrix A is zero, it characterizes a singularity, where the effectors will not be capable of driving the robot to the desired trajectory. Because the system has many different parameters, it's not a simple task to define all the conditions that might cause singularities, but, for instance, a singularity happens, when all the linear velocities u, v, the angular velocity r, and current force provided by the propeller P_f is 0. So, to start the boat, a forward force command should be applied before the control starts.

The expression (3.15) shows that: the equations for A and B are long and complex; this complexity may lead to a system that is highly sensitive to model errors. The parameters used to determine the nonlinear feedback will influence heavily in the performance of the controller. In some situations, an ill-conditioned matrix A could yield an infinite feedback μ.

This control technique is very powerful for theoretical models, and to generate a baseline dataset to train AI models. Additionally, its vulnerability to inaccuracies highlights again the motivation for applying data-driven methods for model identification.

4

Identification with Machine Learning - SINDy

Contents

This Chapter is dedicated to the identification technique grounded on machine learning a.k.a. SINDy. In this Chapter, it will be discussed:

- the accuracy of SINDy to identify simple USV dynamics;
- the accuracy of SINDy to identify complex USV dynamics;
- limitations and disadvantages SINDy for identification.

As introduced in Chapter 2, SINDy uses the concept of sparse regression to fit a linear combination of functions or terms (that can be linear or nonlinear) in order to represent a dynamical system. The minimization problem

$$\begin{aligned} &y \approx \dot{X}; \\ &\xi = argmin_{(\xi')}(||\Theta(X)\xi' - y||_2 - \lambda||\xi||_1) \end{aligned} \tag{4.1}$$

summarizes the method, with ξ being the matrix of coefficients to be found, and Θ being the feature library. The feature library Θ is a set of functions of the states X that will be multiplied by ξ to form the nonlinear system. Those functions are usually polynomials, but a variety of other functions may be applied, for example, trigonometric terms, exponential functions, and absolute value. $\lambda||\xi||_1$ is the regularization term. The factor λ is a hyperparameter that should be properly set to avoid overfitting. It penalizes an overfitting solution that presents too many coefficients.

To assess Sparse Identification of Nonlinear Dynamics (SINDy) for surface marine vehicles, the Python package "PySINDy" was used. PySINDy has "ready-to-use" sparse identification libraries of features and optimizers to solve the problem stated by (4.1). The built-in feature libraries contain the terms Θ from (4.1) that must be multiplied by coefficients ξ. Some of the built-in optimizers are the least squares regression, LASSO, and a specific method called Sparse relaxed regularized regression (SR3) [34] that allows the user to include equality constraints.

In this section, a 2nd degree polynomial library with 4 features (u, v, r, PP_f), which are the 4 last state variables from 3.1, merged with a control input library with the 2 control commands (PP_{cmd}, RD_{cmd}) defined in (3.2) were used to fit the model. This combination contains 45 coefficients (ξ) per state,

$$\xi = \begin{bmatrix} C_{1-1},\ C_{1-2},\ \dots,\ C_{1-44},\ C_{1-45} \\ C_{2-1},\ C_{2-2},\ \dots,\ C_{2-44},\ C_{2-45} \\ C_{3-1},\ C_{3-2},\ \dots,\ C_{3-44},\ C_{3-45} \\ C_{4-1},\ C_{4-2},\ \dots,\ C_{4-44},\ C_{4-45} \end{bmatrix}, \tag{4.2}$$

that are multiplied by the terms

$$\Theta = [1,\ u,\ v,\ r,\ P_f,\ u^2,\ uv,\ ur,\ uP_f,\ v^2,\ vr,\ vP_f,\ r^2,\ rP_f,\ P_f^2,\ PP_{cmd},\ PP_{cmd}u,\ PP_{cmd}v,\ PP_{cmd}r,$$
$$PP_{cmd}P_f,\ PP_{cmd}u^2,\ PP_{cmd}uv,\ PP_{cmd},\ ur,\ PP_{cmd}uP_f,\ PP_{cmd}v^2,\ PP_{cmd}vr,\ PP_{cmd}vP_f,$$
$$PP_{cmd}r^2,\ PP_{cmd}rP_f,\ PP_{cmd},\ P_f^2,\ RD_{cmd},\ RD_{cmd}u,\ RD_{cmd}v,\ RD_{cmd}r,\ RD_{cmd}P_f,\ RD_{cmd}u^2,\ RD_{cmd}uv,$$
$$RD_{cmd}ur,\ RD_{cmd}uP_f,\ RD_{cmd}v^2,\ RD_{cmd}vr,\ RD_{cmd}vP_f,\ RD_{cmd}r^2,\ RD_{cmd}rP_f,\ RD_{cmd}P_f^2]^T. \tag{4.3}$$

The 2nd order degree was chosen in this case due to the nature of the original dynamical system described by (3.4) and (3.7). The optimizer applied here was the standard least squares.

Tests with SINDy were carried out with 2 different approaches. In the first approach, the data for identification was generated from the reference dynamical system with linear drag (3.4) and data from all the 4 state variables (u, v, r and the P_f) were used to fit the model. The second test was conducted under more challenging conditions, which are:

- the data for identification was extracted from the reference dynamical system with quadratic drag (3.7);
- the model has 2 modes, low and high-speed (hydrofoil) weighted by sigmoid functions;
- Gaussian noise ($\sigma = [1\,cm/s,\ 1\,cm/s,\ 1\,deg/s]$) was added to the data system;
- only data from 3 state variables were considered (it assumes that a direct measurement of the propeller force would not be available);
- the relation between the rudder control variable and the torque produced is a trigonometric function.

Table 4.1 summarizes the 2 test conditions.

Table 4.1: Tests with SINDy

Test	Drag	Presence of Noise	State Representation	Number of modes	Rudder torque
Simplified	Linear	No	u, v, r and P_f	single mode	$u(K_{RD})$
Challenging	Quadratic	Yes	u, v, r	2 modes	$u(K_{RD}\,sin(RD_{ang}))$

Three steps were carried out to achieve both tests results, first, it was necessary to collect the data. The second step consisted of removing spurious data cleansing (control and state variables). Finally, it was necessary to tune the sparsity parameter lambda. Section 4.1 and 4.2 give more details about the methodology and results for both tests.

4.1 SINDy applied to a simplified USV model

This section details the "Simplified" test mentioned in Table 4.1. The ground-truth model used was based on (3.4) for a boat miniature with a mass of 2 kg and 1 m in length. The ground-truth uses linear drag and 4 state variable model representation. The data provided to fit the model was the expression in (3.2), which is the torque produced by the rudder divided by u (surge). The dynamic system

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \\ \dot{P}_f \end{bmatrix} = \begin{bmatrix} 0.4000\,P_f - 2\,u + 0.0800\,r\,(r + 10\,v) \\ 0.2211\,r - 4.9704\,v - 0.1693\,r\,u + 0.0099\,RD_{cmd}\,u \\ 0.8876\,v - 5.8681\,r - 0.0789\,r\,u + 0.2959\,RD_{cmd}\,u \\ PP_{cmd} - P_f \end{bmatrix} \tag{4.4}$$

was used to generate the input data.

Two different trajectories were selected to verify the sensitivity and robustness of the generated models. Figure 4.1 shows both control sequences. The first trajectory (Control Seq. 1) lasted (60 s) and presented lower variability. The second simulation (Control Seq. 2) extended itself to 200 seconds with a wider range of model excitation.

To find a stable model that fits properly to the collected data, it was necessary to adjust the hyperparameter λ. A too high lambda would disregard too many terms of (4.3) leading to a poor representation, on the other hand, a too low lambda would include overfitting terms. A strategy to adjust λ is to contrast the model with a realistic operation of marine vehicles. It is known that for an USV, if the actuators are subjected to bounded control inputs, the states and derivatives yielded should also be bounded. This means that the SINDy identified model with well-tuned λ should not generate a trajectory with huge values (smaller than 10^3 for this context). From Figure 4.3 it can be seen that values of λ above 0.15 will produce a bounded system for "Control Seq. 1". "Control Seq. 2" produces bounded results for all values tested of lambda.

The sparse regression with the Control sequence 1 led to the dynamical system

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \\ \dot{P}_f \end{bmatrix} = \begin{bmatrix} -1.520\,u + 0.348\,P_f + 0.878\,vr \\ -4.387\,v + 0.659\,uv \\ 26.451\,v + -6.109\,r + -2.431\,u\,v + -0.263\,v^2 + 0.280\,u\,RD_{cmd} \\ -0.993\,P_f + -0.378\,v^2 + 0.994\,PP_{cmd} \end{bmatrix}. \tag{4.5}$$

While the sparse regression with the Control sequence 2 led to

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \\ \dot{P}_f \end{bmatrix} = \begin{bmatrix} 0.4000\,P_f - 2\,u + 0.0800\,r\,(r + 10\,v) \\ 0.426\,r - 4.625\,v - 0.159\,r\,u \\ 0.996\,v - 5.804\,r - 0.075\,r\,u + 0.293\,RD_{cmd}\,u \\ -0.998\,P_f + 0.999\,PP_{cmd} \end{bmatrix}. \tag{4.6}$$

"Control Seq. 2", which excites more the system, produces an identified model much closer to the refer-

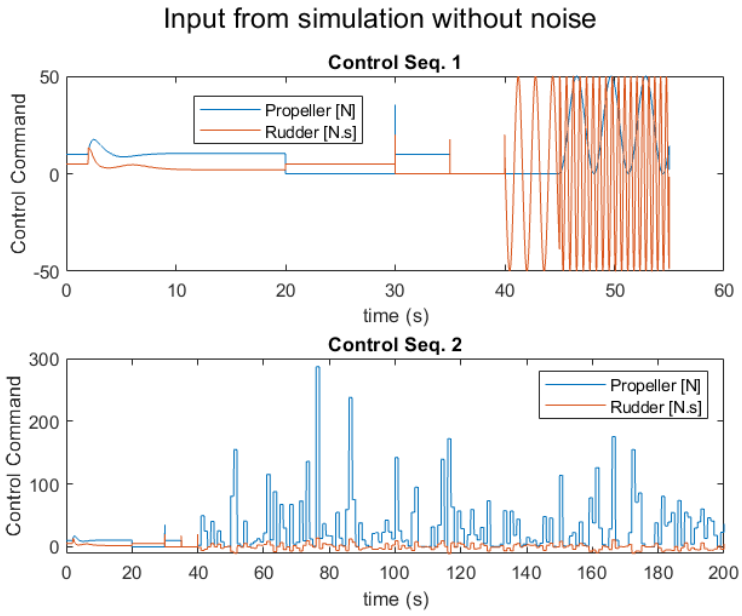

Figure 4.1: Control Sequences used to identify the model. In the simplified model, the multiplication of the rudder command RD_{cmd} (unit N.s) by the surge velocity (unit m/s) yields the yaw torque (unit N.m).

ence than ”Control Seq. 1”. Both identified models were used as a reference for Feedback Linearization with a trajectory of a circle of 10 m radius and period of 40 s. The second sequence presented a smoother control and more accurate trajectory when compared with sequence 1. This can be seen in Figure 4.3 and 4.2.

Both (4.5) and (4.6) presented significant differences between ground truth and identified expressions for the state ”v” (sway). However, these differences did not impose considerable losses on the control process, because the rudder and the propeller do not act directly in this direction.

4.2 SINDy limitations - the method applied to a complex hydrofoil model

In the previous section, the observed results in the simulations showed that for simple dynamical systems, SINDy can be suitable for control identification. This section will investigate whether SINDy performs well under the ”Challenging” conditions summarized in Table 4.1. The tests performed in this section do not use a direct measurement of the propeller force, and the torque yielded by the rudder comes from a trigonometric function of its angle position. Besides that, the drag behaves as a quadratic function, and the system will be composed by the linear combination of two modes weighted by sigmoids as illustrated in the chart in Figure 4.5 and defined by

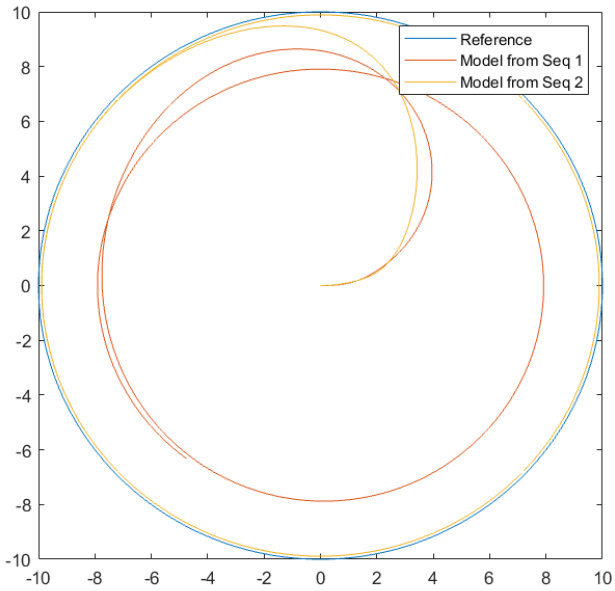

Figure 4.2: Tracking a circular trajectory using Feedback Linearization with the model identified with "Control Seq. 1", (4.5), and model identified with "Control Seq. 2", (4.6).

$$\sigma(u-u_0)\left[f_{low}(X)+g_{low}(X)\mu\right]=\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{r}\\ P_f\end{bmatrix}=\begin{bmatrix}\frac{(0.4000P_f+0.0800r(r+10v)-0.4000u(|u|+3))}{\exp(12u-12u_0)+1}\\ \frac{-(3.9665v-0.2211r+0.1693ru+0.0099|r|r+0.8366|v|v-0.0099RD_{cmd}u)}{\exp(12u-12u_0)+1}\\ \frac{-(5.8681r-1.0060v+0.0789ru+0.2959|r|r+0.0986|v|v-0.2959RD_{cmd}u)}{\exp(12u-12u_0)+1}\\ PP_{cmd}-P_f\end{bmatrix}, \tag{4.7}$$

$$\sigma(u_0-u)\left[f_{fly}(X)+g_{fly}(X)\mu\right]=\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{r}\\ P_f\end{bmatrix}=\begin{bmatrix}\frac{(0.4444P_f+0.0889r(r+10v)-0.2667u(|u|+3))}{\exp(12u0-12u)+1}\\ \frac{-(4.0947v-0.3383r+0.2874ru+0.0045|r|r+0.8578|v|v-0.0076RD_{cmd}u)}{\exp(12u0-12u)+1}\\ \frac{-(6.3206r-1.3683v+0.1208ru+0.3172|r|r+0.0453|v|v-0.5286RD_{cmd}u)}{\exp(12u0-12u)+1}\\ PP_{cmd}-P_f\end{bmatrix}, \tag{4.8}$$

$$f_{hydrofoil}=\sigma(u-u_0)\left[f_{low}(X)+g_{low}(X)\mu\right]+\sigma(u_0-u)\left[f_{fly}(X)+g_{fly}(X)\mu\right]. \tag{4.9}$$

The control sequence applied can be seen in Figure 4.6. For the tests in this section, the control sequence imposed on the system for identification excited it much more than what was used in 4.1. Despite that fact, the model representations found with SINDy were not good in this case. The identified models either did not represent the force applied by the propeller well or they produced trajectories that

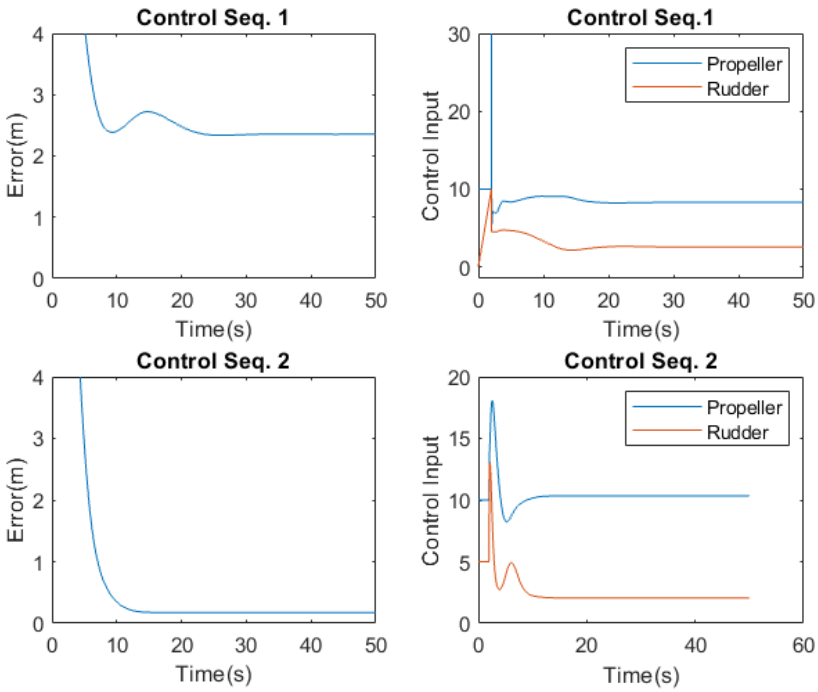

Figure 4.3: Error tracking a circular trajectory using Feedback Linearization with the model identified with "Control Seq. 1", (4.5), and model identified with "Control Seq. 2" (4.6). The data used were generated with the model (4.4). The maximum error at the beginning of both sequences is around 10 m, the value in the chart was clipped in 4 m for better resolution in the steady state error.

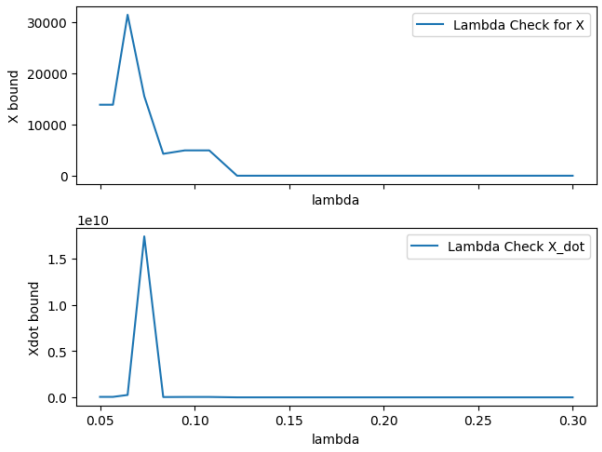

Figure 4.4: Maximum values of X and $\dot{X}$ given a bounded input for different λ, given the state sequence generated by control sequence 1.

escalated to large errors with a few control commands. To exemplify those cases, the system

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} = \frac{1}{100} \begin{bmatrix} 50.9v^2 + 0.4RD_{cmd}uv + 1.5RD_{cmd}v^2 \\ 0.000 \\ -9v - 4.4r + 3.4uv + 2v^2 + (6.5u + 2.3u^2 + 134v^2 - 7vr - r^2)RD_{cmd} \end{bmatrix} \tag{4.10}$$

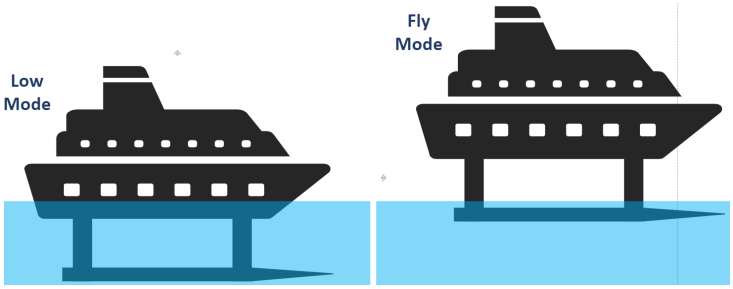

Figure 4.5: Modes described by the expressions (4.7) and (4.8).

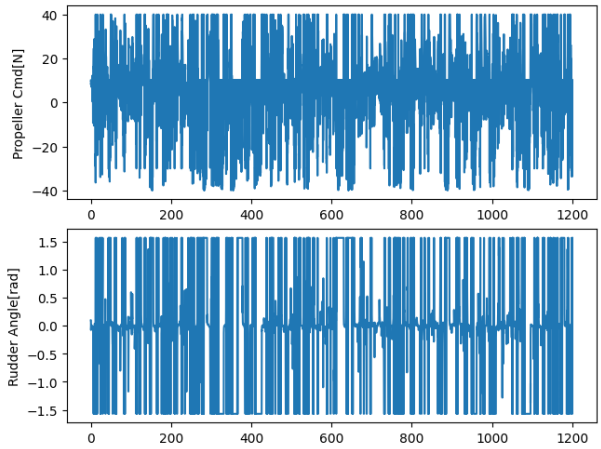

Figure 4.6: Control input sequence applied for model (4.9).

was obtained with $\lambda = 0.6$ (more sparse), whereas

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{r} \end{bmatrix} = \frac{1}{100} \begin{bmatrix} 3 + ... + 84r^2 + (0.7 + ... - 0.8r^2)PP_{cmd} + (-1.5 + ... + -26.5r^2)RD_{cmd} \\ -0.1u... + 1.2r^2 + (3.7v + ... - 0.3vr)PP_{cmd} + (0.1 + ... - 1.7r^2)RD_{cmd} \\ -0.3 + -0.7u + ... + 2.6r^2 + (-20.1v... + -0.2r^2)PP_{cmd} + (-1.5... - 105r^2)RD_{cmd} \end{bmatrix} \tag{4.11}$$

was obtained with $\lambda = 0.06$ (less sparse). The expression in (4.11) is exceptionally long, and some terms were omitted for better readability. (4.10) did not have a direct term of the propeller in the surge equation, and (4.11) produced large errors with few control commands.

The poor performance of the identified models precluded their use for Feedback Linearization with the simple circular trajectory in Figure 4.2. The difficulties to fit a good model with SINDy for the ”Challenging” conditions can be explained by the limited degrees of freedom provided by 2nd order polynomials. A 2nd order polynomial SINDy library is not enough to cover simultaneously:

- the steep nonlinearity devised by the hydrofoil high-low speed transitions;
- the trigonometric, highly nonlinear, relation between the rudder angle and the torque/force applied;
- the quadratic drag.

It is still possible to improve results by using equality constraints or increasing the complexity of functions in $\Theta(X)$ or identifying two different models — one for high and another for low speeds. However, setting up those suggestions can be cumbersome, requires more in-depth knowledge of the problem, and can undermine the flexibility and freedom offered by data-driven methods.

Section 4.1 showed a successful use of SINDy, while this section demonstrated some of the method limitations. In the next Chapter, it will be verified that the identification of the complex dynamical system described in this section can be improved with neural networks. The next Chapter will also demonstrate, with examples, the advantages and disadvantages of this approach for offline or real-time learning of surface vehicle dynamics.

5

Real-time identification with Neural Networks

Contents

In the previous Chapters, it was shown that Machine Learning can be applied to identify dynamic systems. In this Chapter, a similar investigation will be conducted, but instead of using SINDy, feed-forward and recurrent neural networks will be trained for prediction or identification of hydrofoils. Section 5.1 will describe the methodology and results obtained with feed-forward neural networks trained offline with SGD for trajectory prediction. Section 5.2 presents the methodology and the results obtained to train and fine-tune in real time recurrent neural networks using Extended Kalman Filter (EKF) for identification.

5.1 Offline Training of Neural Networks for Identification

To contour the limitations with SINDy described in Chapter 4, an NN f_θ can be trained to identify USV dynamics. With the NN as the representation of the system, it is possible to control the future trajectory by applying an optimizer in an MPC framework, similar to what [13] proposes. Figure 5.1 is a workflow that explains this approach. The tests and simulations conducted in this section are focused on the offline NN training process (the steps inside the yellow square in Figure 5.1).

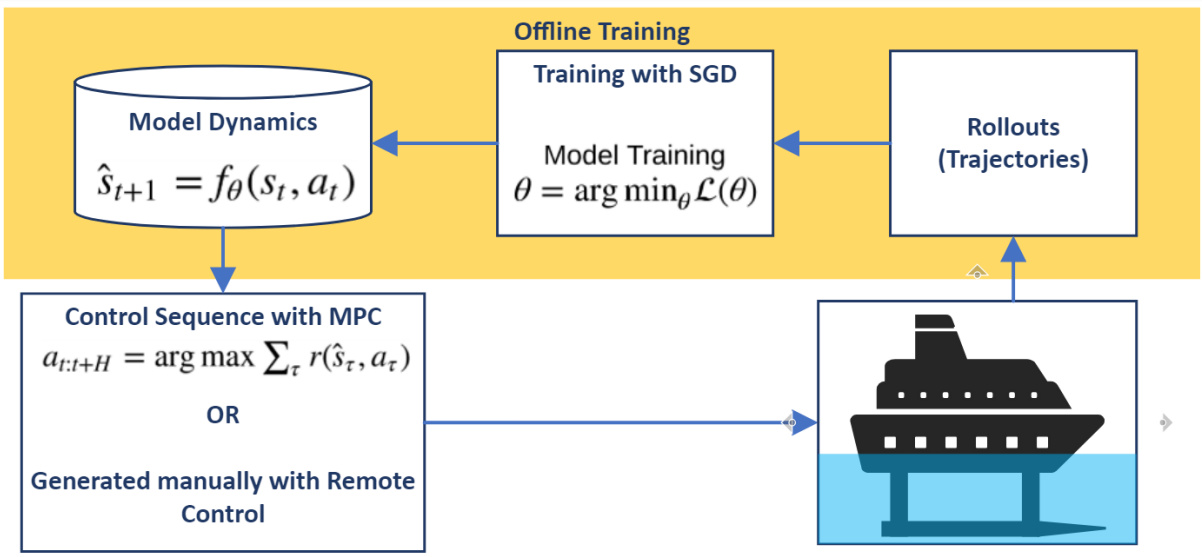

Figure 5.1: A NN plays the role of the dynamic system and forecasts the next 20 steps. With the next 20 steps available, it is possible to apply MPC associated with the Sample-efficient Cross Entropy Method to drive the robot to a desired state.

In this work, the inputs of the NN predictor f_θ were specified as the 4 latest states, expressed by

$$s_t(k) = \{X(k-3),\ X(k-2),\ X(k-1),\ X(k)\}, \tag{5.1}$$

alongside the latest 3 command steps concatenated with the next 20 command steps, expressed by

$$a_t(k) = \{\mu(k-3),\ \mu(k-2),\ ...\ \mu(k+19)\}. \tag{5.2}$$

It is assumed that the next 20 commands are known in advance. For instance, they can be extracted from the Data Efficient CEM to drive the vehicle to a desired trajectory of 20 steps ahead in time. As seen in Figure 5.1, the neural network and its state predictions are defined as

$$\hat{s}_{t+1} = f_\theta(s_t, a_t). \tag{5.3}$$

The output of $f_\theta(s_t, a_t)$ are the estimated states of the future 20 steps, given by

$$\hat{s}_{t+1}(k) = \{\hat{X}(k+1),\ \hat{X}(k+2),\ ...\ \hat{X}(k+20)\}. \tag{5.4}$$

The reason to consider the 4 past states instead of just 1 is to cope with noise. If trained properly, an NN is able to handle noisy measurements from a set of past inputs. As it was done in the challenging conditions for SINDy in Section 4.2, the state X is composed only by the velocities (u, v, r) from (3.1), and the control inputs μ are composed by PP_{cmd}, RD_{ang} from (3.2). This means that the learning process will have to account internally for the propeller dynamics and the trigonometric relations between the rudder angle and the actual torque yielded.

To improve the learning capabilities of neural networks, the data consumed in the training and testing stages should be normalized. This step is distinctly important for the dynamic systems defined by (3.4) and (3.8) due to the differences in magnitude among surge, sway, yaw, PP_{cmd} and RD_{ang}. Consequently, Z-Score normalization was applied to the training and testing data, as described by

$$\begin{aligned} \mu_{norm}(k) &= [\frac{PP_{cmd}(k) - \overline{PP}_{cmd}}{\sigma_{PP_{cmd}}},\ \ \frac{RD_{ang}(k) - \overline{RD}_{ang}}{\sigma_{RD_{ang}}}]^T, \\ X_{norm}(k) &= [\frac{u(k) - \overline{u}}{\sigma_u},\ \ \frac{v(k) - \overline{v}}{\sigma_v},\ \ \frac{r(k) - \overline{r}}{\sigma_r}]^T. \end{aligned} \tag{5.5}$$

The variables accompanied by the overwritten bar in (5.5) are the mean values of each feature from the training dataset. For example, considering that the training dataset has N terms, the mean value of u would be

$$\overline{u} = \frac{1}{N}\sum_{k=1}^{N} u(k).$$

On the other hand, σ refers to the standard deviation of the training dataset. For example, the standard

deviation of the feature u is

$$\sigma_u = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(u(k) - \overline{u})^2}.$$

This way, the mean of each feature from the training dataset becomes centered in 0, and the variance becomes 1.0. After Z score normalization, before forwarding the data to the neural net, the inputs are flattened and concatenated. The NN is formed by linear layers followed by hyperbolic tangent (tanh) activation functions. Figure 5.2 shows the arrangement adopted. For training, the loss function chosen was the mean square error, given by

$$\mathcal{L} = \sum_{n=1}^{20} ||X_{norm}(k+n) - \hat{X}_{norm}(k+n)||_2^2, \tag{5.6}$$

and Adam Optimizer was selected for the Stochastic Gradient Descent process.

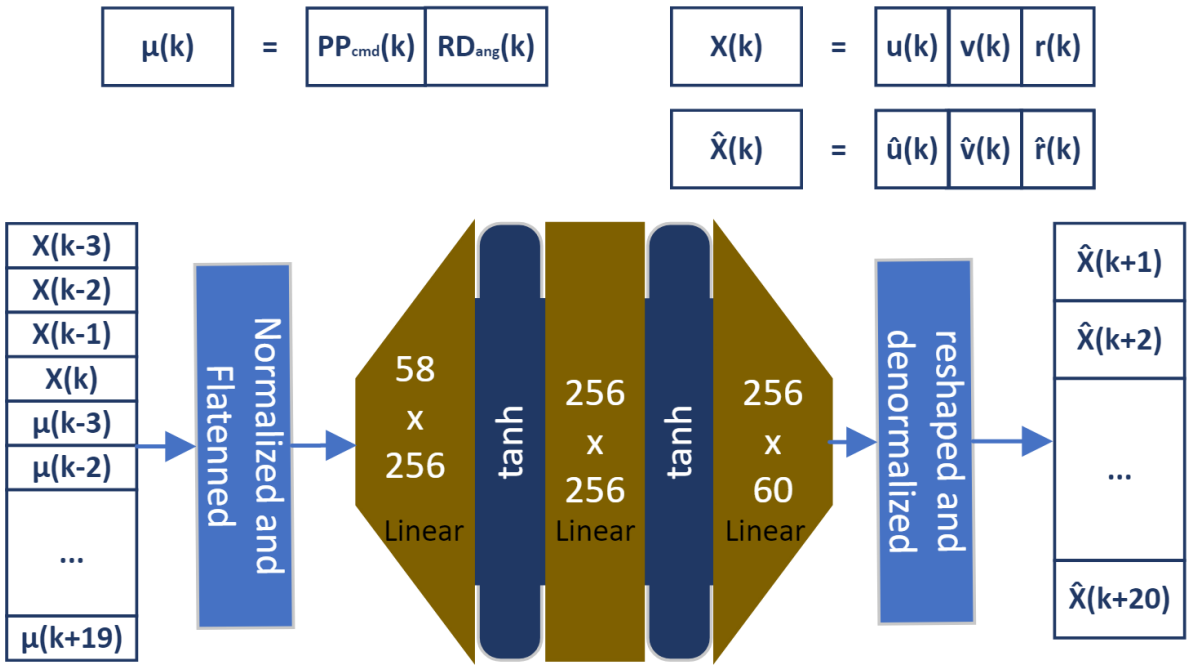

Figure 5.2: Neural Network design for system identification and MPC.

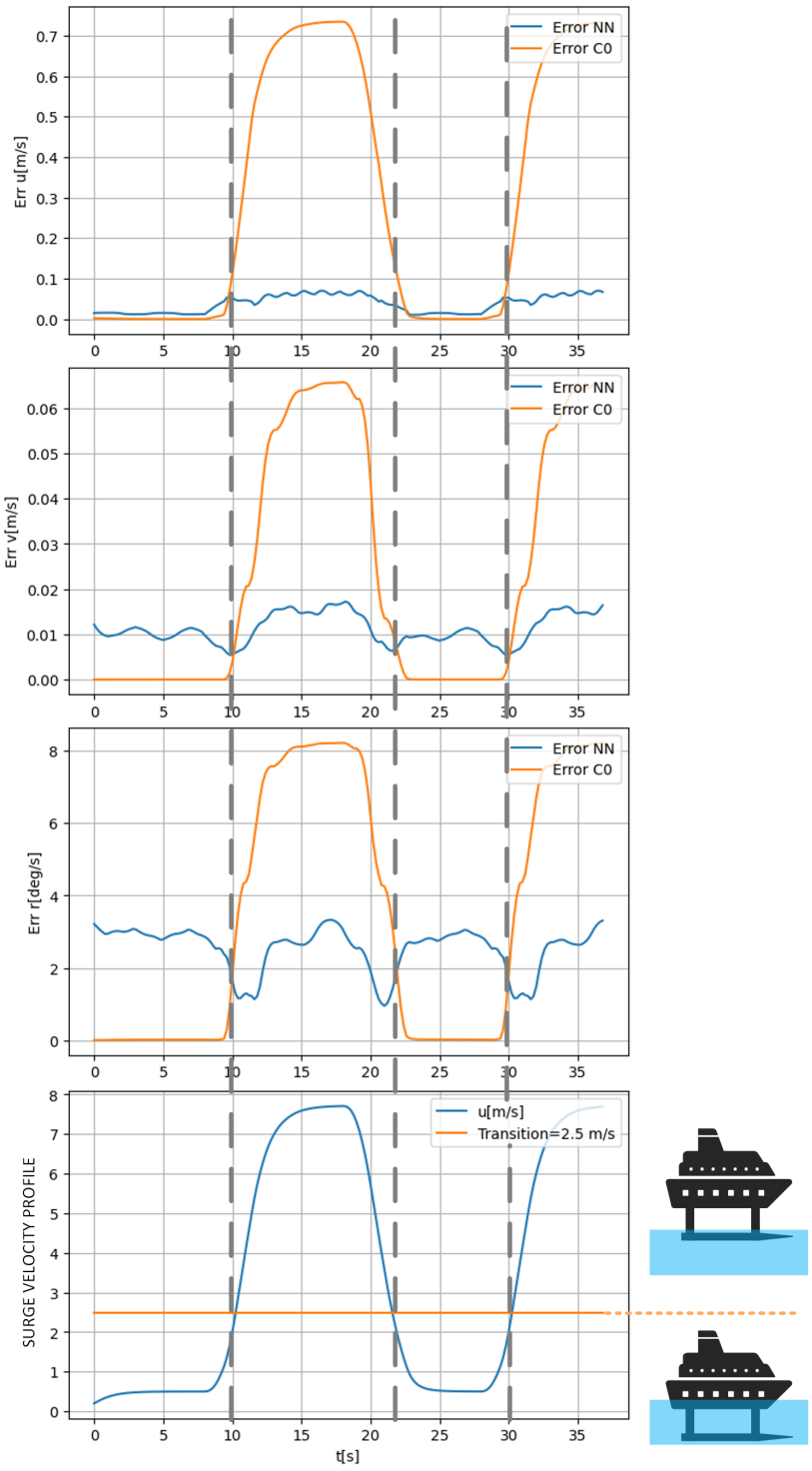

Figure 5.3: ”Error NN” is the test dataset error between the NN prediction for either u, v, or r and the hydrofoil model. Similarly, ”Error C0” is the error between an ideal low-speed-only model and the complete hydrofoil model. ”u[m/s]” is the surge velocity of the boat for the test dataset, and ”Transition” indicates the borderline between low and high-speed modes.

The dataset summed up approximately 2 hours of simulation with a step of 10 ms (around 72000 steps). The training data was generated with Matlab Simulink simulations. The inputs for training were carefully chosen to excite the system dynamics in a variety of ways. To fulfill this objective, the commands sent to the actuators were a composition of: random variables; constant 0.0 values (to train the natural decaying effect, and excite independently each actuator); chirp functions; triangular functions, etc. The system excitation was also designed in a way that the hydrofoil would operate in low and high-speed modes. In this case, the transition between low and high-speed modes of the hydrofoil occurred when u crossed 2.5 m/s.

To verify the accuracy of the NN, its results were compared with an ideal model using only the low-speed mode of the hydrofoil ("C0" in Figure 5.3). With this, the test dataset error between the ground-truth and "C0" is expected to increase every time the surge speed exceeds 2.5 m/s (from where the hydrofoil enters the flying mode). Conversely, the test dataset error between the ground-truth and the trained NN should remain stable and smaller than "C0" in all velocity conditions.

Figure 5.3 shows the test dataset error after training the NN. As expected, the errors of the reference "C0" grow as the speed goes above the transition point of 2.5 m/s, while for the neural networks, the error remains very small, regardless if the surge velocity is below or above the threshold of 2.5 m/s. This result shows that the NN effectively learned to predict the next 20 states of the hydrofoil. The error seen in Figure 5.3 is calculated using

$$\begin{bmatrix} Error_u(k) \\ Error_v(k) \\ Error_r(k) \end{bmatrix} = \begin{bmatrix} \frac{1}{200}\sum_{m=0}^{9}\sum_{n=1}^{20} ||u(k+20m+n) - \hat{u}(k+20m+n|s_t(k+20m), a_t(k+20m))||_1 \\ \frac{1}{200}\sum_{m=0}^{9}\sum_{n=1}^{20} ||v(k+20m+n) - \hat{v}(k+20m+n|s_t(k+20m), a_t(k+20m))||_1 \\ \frac{1}{200}\sum_{m=0}^{9}\sum_{n=1}^{20} ||r(k+20m+n) - \hat{r}(k+20m+n|s_t(k+20m), a_t(k+20m))||_1 \end{bmatrix}, \tag{5.7}$$

with $[\hat{u}(k+n|s_t(k), a_t(k)) \;\; \hat{v}(k+n|s_t(k), a_t(k)) \;\; \hat{r}(k+n|s_t(k), a_t(k))]^T$ being the velocities predicted by the neural network or the reference model "C0", given the inputs $s_t(k)$ and $a_t(k)$.

Since the dynamical system to be identified is not presented in the canonical form of $\dot{X} = f(X) + g(X)\mu$, with the control variables μ only multiplying $g(X)$, Feedback Linearization or Backstepping cannot be used with the NN. Nevertheless, as mentioned at the beginning of this section, the identified system can be associated with an optimization method in order to apply MPC. [13] and [35] provided comprehensive evidence that the MPC trajectory optimization can be achieved in real-time problems with the Sample-Efficient CEM.

The NN predictor presented in this section has proven capable of learning dynamics very accurately when a rich training dataset is provided. However, its results can be deficient when real conditions are not covered by the training data. For example, if unexpected disturbances, measurement problems, or physical damages occur, the trained neural networks can present considerable inaccuracies. Section 5.2 will describe a methodology with EKF and RNN's that can be explored to overcome this difficulty.

5.2 Real-time Learning of Neural Networks for Identification with EKF

In the previous section, a feed-forward neural network was used to learn the dynamics of a hydrofoil. In this section, instead of traditional feed-forward NN's, RNN's will be employed for a similar purpose. Additionally, this section will cover the real-time (online) training of recurrent neural nets. By means of a technique proposed by [20], EKF may be applied to adjust or fine-tune recurrent neural networks "on-the-fly".

RNN's are appropriate for sequential data, where it is necessary to find a chain of outputs that depends on previous states, and on external inputs. This is the case for the context of dynamical systems. Figure 5.4 elucidates how data flows through recurrent neural networks.

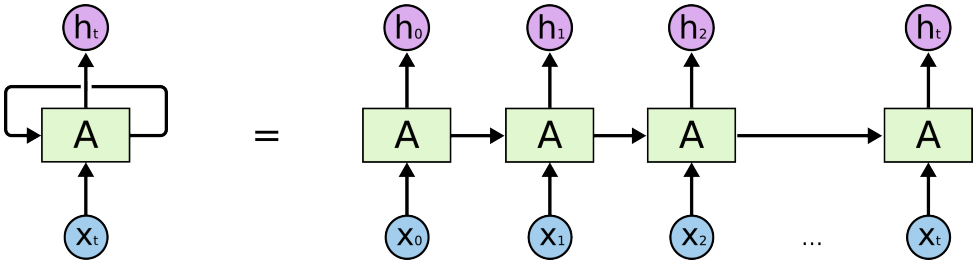

Figure 5.4: The recurrent neural network, represented by A, uses the previous hidden state h_{t-1} and input x_t to yield the current hidden state h_t. Image extracted from [2].

Following the presentation of [20], the state transition and output prediction with RNN are expressed by

$$\begin{aligned} h(k+1) &= f_h\left(h(k), \mu(k), \theta_h(k)\right) + \xi(k) \\ y(k) &= f_y\left(h(k), \mu(k), \theta_y(k)\right) + \zeta(k) \\ \theta(k+1) &= \theta(k) + \eta(k), \quad \theta(k) \triangleq \begin{bmatrix} \theta_h(k) \\ \theta_y(k) \end{bmatrix}, \end{aligned} \tag{5.8}$$

where f_h and f_y are recurrent neural networks, and θ_h and θ_y are, respectively, their parameters. [2] and [20] use different notations, $h(k)$ for [20] is equivalent to the hidden state h_t in Figures 5.4, 5.13, 5.10, and 5.8, given a time step k. $\mu(k)$ is equivalent to x_{t-1} in Figures 5.4, 5.13, 5.10, and 5.8. $\zeta(k)$, $\xi(k)$, $\eta(k)$ are Gaussian noises, and $y(k)$ is the estimated output state. For The hydrofoil dynamic system with 2 modes defined by (3.5), $y(k)$ are the surge, sway, and yaw velocities,

$$y(k) = X(k) = [u(k),\ v(k),\ ,r(k)]^T. \tag{5.9}$$

Initially, the RNN hydrofoil model can be trained offline with Backpropagation Through Time (BPTT),

similarly to what has been done in Section 5.1 with SGD. Afterward, the model learned offline can be put into operation and improved or fine-tuned by the EKF technique. EKF is a modification of the traditional Kalman Filter used for estimating states and process parameters of a dynamic system. It's particularly helpful to handle nonlinearities. Besides h, y, f_h, f_y and θ, the terms that appear in an EKF are the state covariance P, the process noise covariance Q, the measurement noise covariance Q_y, and the Kalman Gain M.

The state Covariance (P) represents the uncertainty associated with the state estimate. It expresses quantitatively how precise is the estimate. This matrix is impacted by Q and is weighted by A, a matrix composed of Jacobians. Process Noise Q is the uncertainty or noise associated with transitions of the state. It considers the distortions between the real system and the predicted state transition function. It depends on terms related to h (Q_h) and terms related to θ (Q_θ). Measurement Noise (Q_y) represents the noise or uncertainty related to the y. It accounts for imprecision in the sensors and other sources of measurement error. The Kalman Gain (M) defines how much relevance to give to the state estimate and the most recent observation when updating the state estimate. It's calculated based on P and Q_y and weights measurements and predicted states.

In the Prediction Step, the function f_x predicts a new state, and its associated uncertainty P is refreshed. In the Update Step, the predicted output $\hat{y}$ is obtained from the predicted state $\hat{h}$ and f_y. The difference between the actual observations and $\hat{y}$ is weighted by the Kalman Gain (M) and added to the state estimate $\hat{h}$. P is also adjusted with M.

Every iteration, the EKF predicts and updates the state, as new observations are detected. The method continuously adjusts the results and their uncertainty. The most challenging step in EKF is coping with nonlinearities in f_x and f_y, which requires linearizing these functions regularly around the current estimate. This procedure is implicitly executed when matrices A and C are calculated and used to update P and M.

The Linearization combined with the Update and Prediction Step allow the EKF to operate even under nonlinear conditions, making it a solid resource in different areas, such as marine robotics.

The EKF is summarized by the sequence

$$
\begin{aligned}
C(k) &= \left[\begin{array}{ccc} \frac{\partial f_y}{\partial h} & 0 & \frac{\partial f_y}{\partial \theta_y} \end{array} \right] \Big|_{\hat{\theta}(k|k-1),\hat{h}(k|k-1),\mu(k)} \\
M(k) &= P(k \mid k-1)C(k)' \left[C(k)P(k \mid k-1)C(k)' + Q_y(k)\right]^{-1} \\
e(k) &= y(k) - f_y\left(\hat{h}(k \mid k-1), \mu(k), \hat{\theta}_y(k \mid k-1)\right) \\
\left[\begin{array}{c} \hat{h}(k \mid k) \\ \hat{\theta}(k \mid k) \end{array} \right] &= \left[\begin{array}{c} \hat{h}(k \mid k-1) \\ \hat{\theta}(k \mid k-1) \end{array} \right] + M(k)e(k) \\
P(k \mid k) &= (I - M(k)C(k))P(k \mid k-1) \\
\left[\begin{array}{c} \hat{h}(k+1 \mid k) \\ \hat{\theta}(k+1 \mid k) \end{array} \right] &= \left[\begin{array}{c} f_h\left(\hat{h}(k \mid k), \mu(k), \hat{\theta}_h(k \mid k)\right) \\ \hat{\theta}(k \mid k) \end{array} \right] \\
A(k) &= \left[\begin{array}{ccc} \frac{\partial f_h}{\partial h} & \frac{\partial f_h}{\partial \theta_h} & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{array} \right] \Bigg|_{\hat{\theta}(k|k)\hat{h}(k|k),\mu(k)} \\
P(k+1 \mid k) &= A(k)P(k \mid k)A(k)' + \left[\begin{array}{cc} Q_h(k) & 0 \\ 0 & Q_\theta(k) \end{array} \right].
\end{aligned}
\tag{5.10}
$$

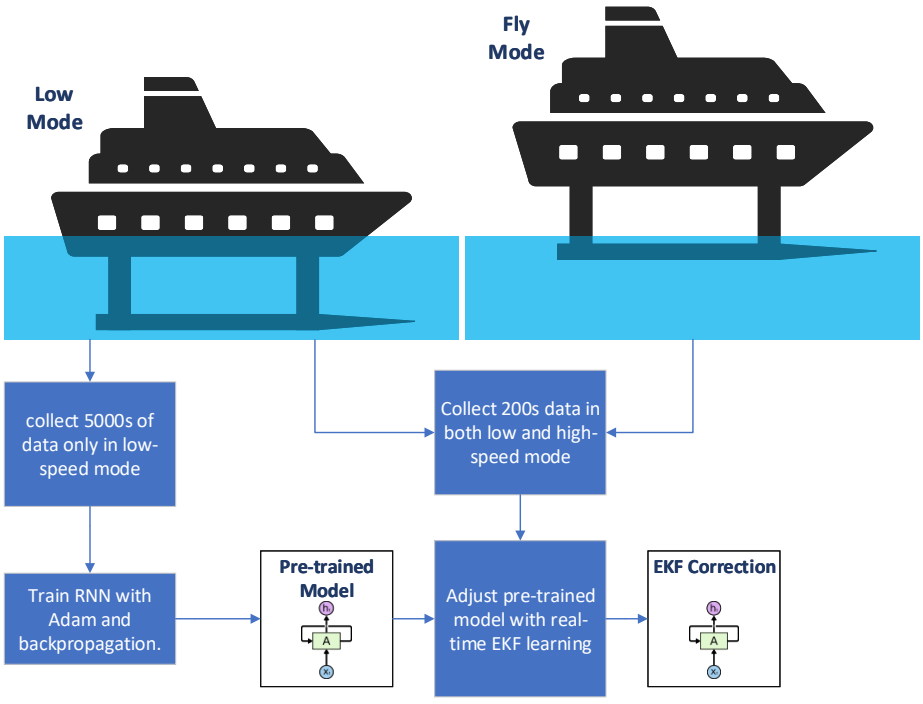

Figure 5.5: Process to verify the performance of the EKF real-time learning in comparison with offline learning.

To avoid numerical distortions and computational inefficiencies, the Jacobian's of the RNN with relation to the parameters θ and the hidden states $h(k)$ should be calculated analytically in advance with the chain rule. In this work, the Python library PyTorch was used to train the RNN's offline with SGD

and to adjust them in real time with EKF. With PyTorch, matrix and gradient operations can be carried out in graphics processing unit (GPU)'s. Moreover, with PyTorch, Jacobians are easily and efficiently calculated through graph gradients.

To assess the performance of the EKF method to adapt to unknown conditions in real time, the RNN models were initially trained offline through backpropagation with Adam Optimizer. The offline training dataset had only trajectories in low speed. This means that the models are expected to perform poorly when exposed to high-speed hydrofoil mode. For real-time learning, the dataset used contained both low and high-speed modes. Figure 5.5 has a visual description of this process. Three different variants of RNN were tested, which are:

- Vanilla Recurrent Neural Networks (RNN);
- Long Short Term Memory (LSTM);
- Gate Recurrent Unit (GRU).

In Sections 5.2.3, 5.2.2, 5.2.4, the results for GRU, LSTM and Vanilla RNN configuration will be presented. For a better understanding, Figure 5.6 depicts the notation used in Figures 5.13, 5.10 and 5.8. In the next 3 situations, the models were trained with noisy data. The same noise used in Section 4.2, which is a Gaussian noise with $\sigma = [1\,cm/s,\ 1\,cm/s,\ 1\,deg/s]$, will be used here. Moreover, the same time step used in Section 5.1 and in Chapter 4 (10 ms) will be used in the tests with GRU, LSTM, and Vanilla RNN. The tests were obtained from a vehicle traveling along a trajectory that resembles a lawnmower with different turning angles, as shown in Figure 5.7.

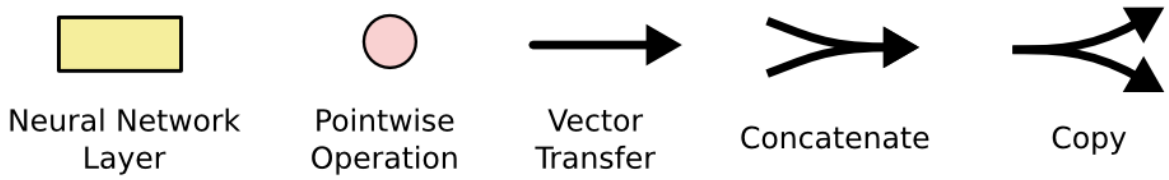

Figure 5.6: Notation to describe the RNN structures in Figures 5.13, 5.10 and 5.8. Image extracted from [2].

5.2.1 Vanilla RNN applied for system identification in real time

The results presented in this section were produced by a Vanilla RNN formed by 2 subsequent layers. This is the most basic structure of RNN's, and each layer consists of a linear combination followed by a hyperbolic tangent (tanh) activation function as shown in Figure 5.8. It reduced slightly the average and maximum error with the EKF for all the steps ahead, except the average and maximum prediction error for the sway velocity. As explained in Chapter 4, this error in the sway is less critical than surge and yaw for the hydrofoil dynamics studied here.

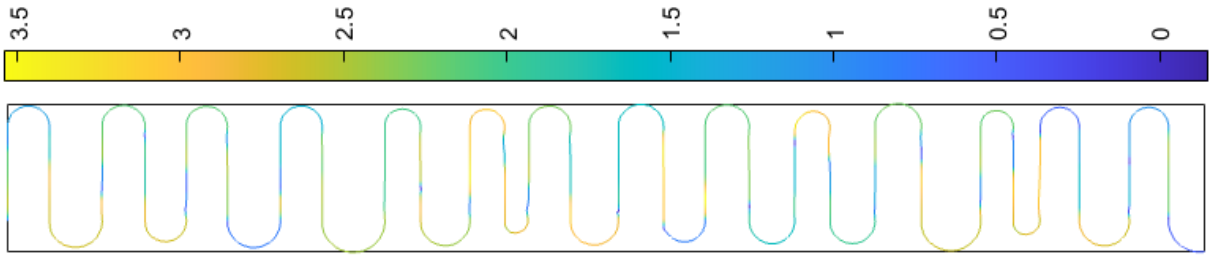

Figure 5.7: Example of a lawnmower trajectory with different turning angles and different velocities. The horizontal color bar is the surge velocity profile in m/s.

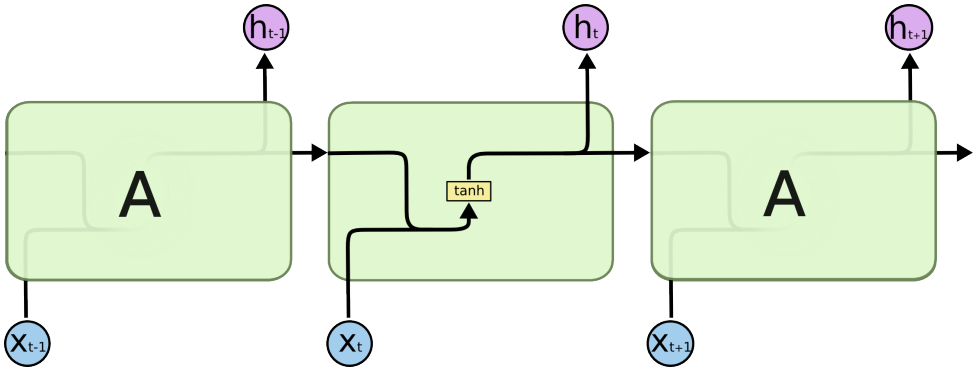

Figure 5.8: RNN structure with a linear layer followed by a hyperbolic tangent (tanh) activation function. Source [2].

In spite of the modest improvement achieved with the EKF corrections in real time, when compared with GRU and LSTM, Vanilla RNN took 4 times longer to be trained offline. Moreover, Sections 5.2.2 and 5.2.3 will show that GRU and LSTM present much better accuracy for offline identification, and much better performance for corrections in real time with EKF.

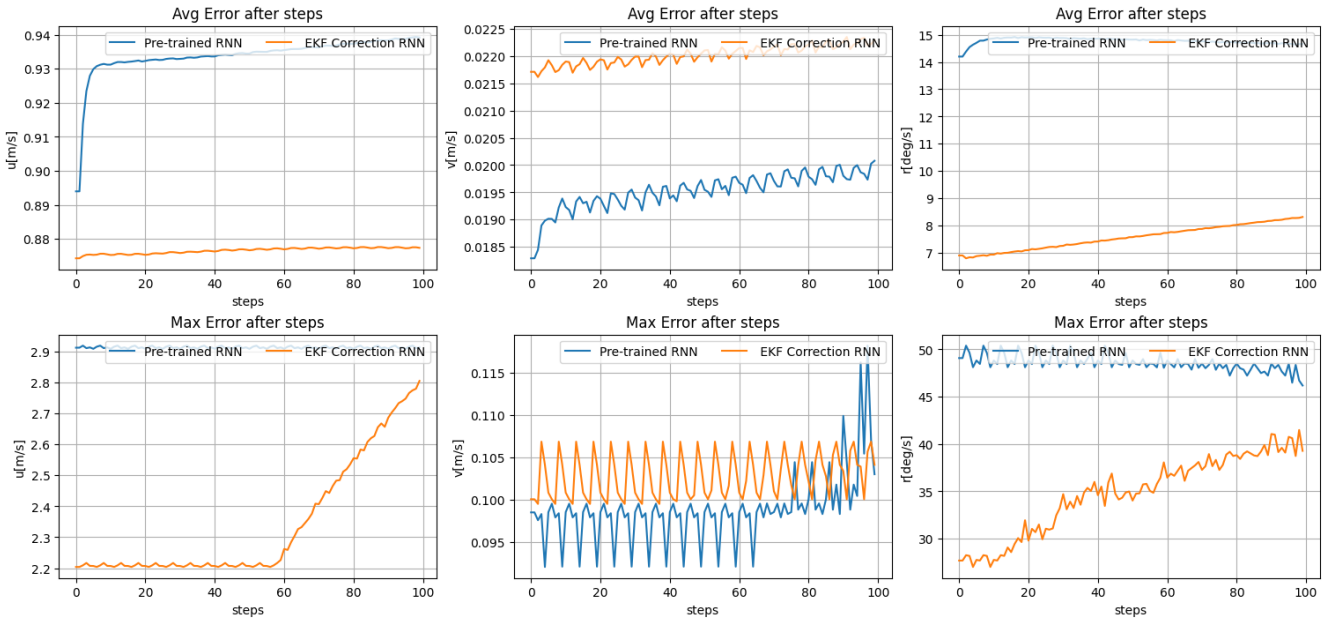

Figure 5.9: Error comparison between the pre-trained and the EKF real-time adjusted model for the Vanilla RNN predicting 20 steps ahead the reference.

5.2.2 LSTM applied for system identification in real time

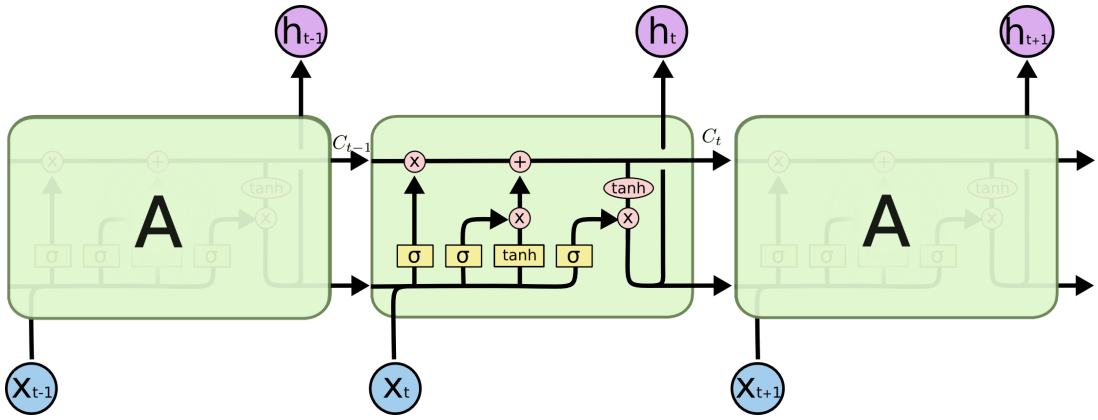

Figure 5.10: LSTM architecture. σ represents a layer with a sigmoid activation function. In this work, x_t is equivalent to $\mu(k)$ and h_t corresponds to $\hat{h}(k)$ in (5.8). Image extracted from [2]

.

In this section, an LSTM was applied to identify the hydrofoil dynamics. As seen in Figure 5.10, the LSTM has a cell state C_t and a hidden state h_t. The cell state is less affected by the inputs x_t, thus its output is more resilient against changes and works as a memory structure. Conversely, the hidden states h_t are more flexible because they are directly affected by the inputs. The tests in this section were carried out with a cell state of size 18 and a hidden state of size 3. The hydrofoil velocities (u, v, r) coincide with the hidden state h_t.

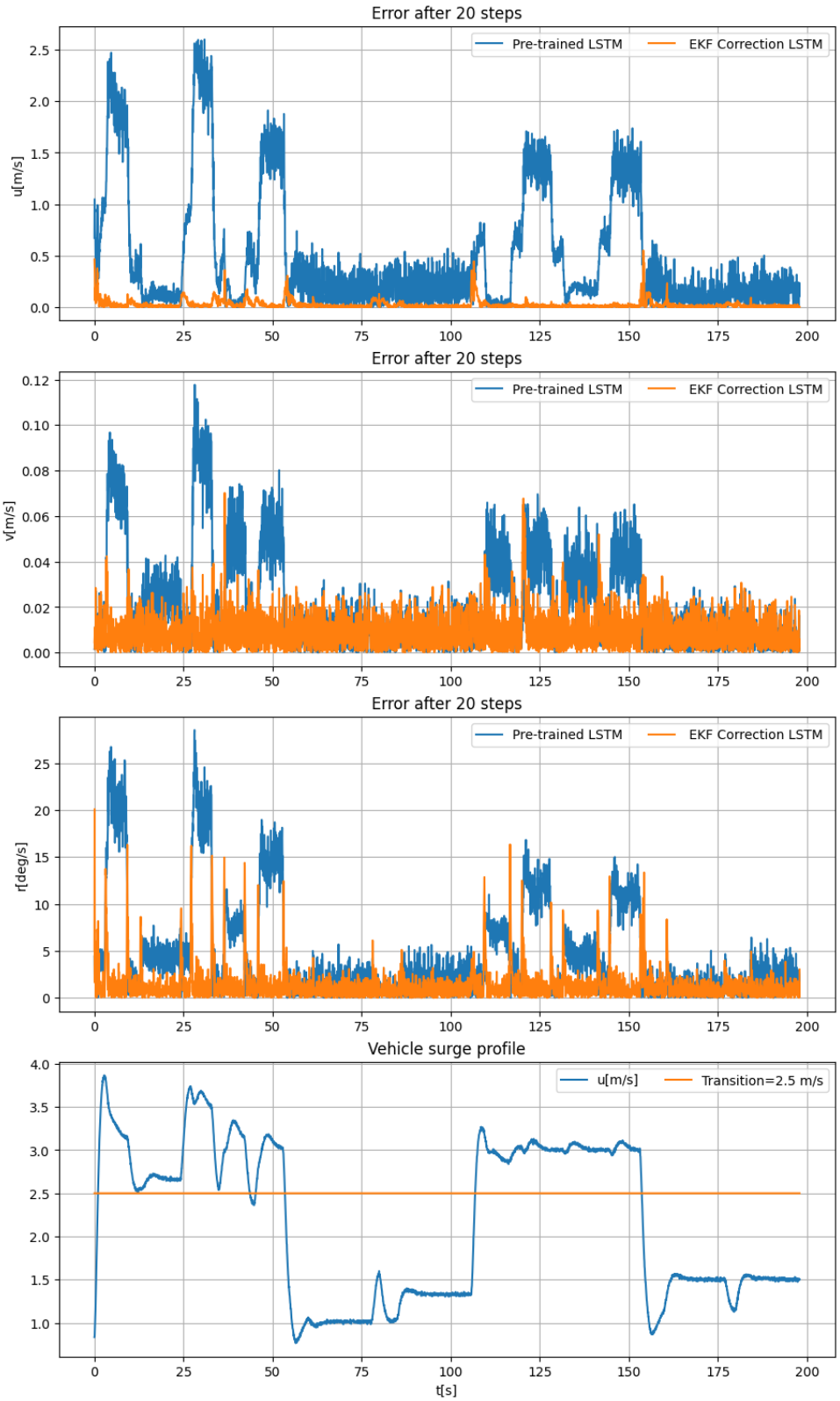

Figure 5.11: Comparison of the errors for a prediction of 20 steps ahead, between the pre-trained LSTM and the EKF real-time adjusted LSTM. The model had a cell size of 18 and a hidden size of 3.

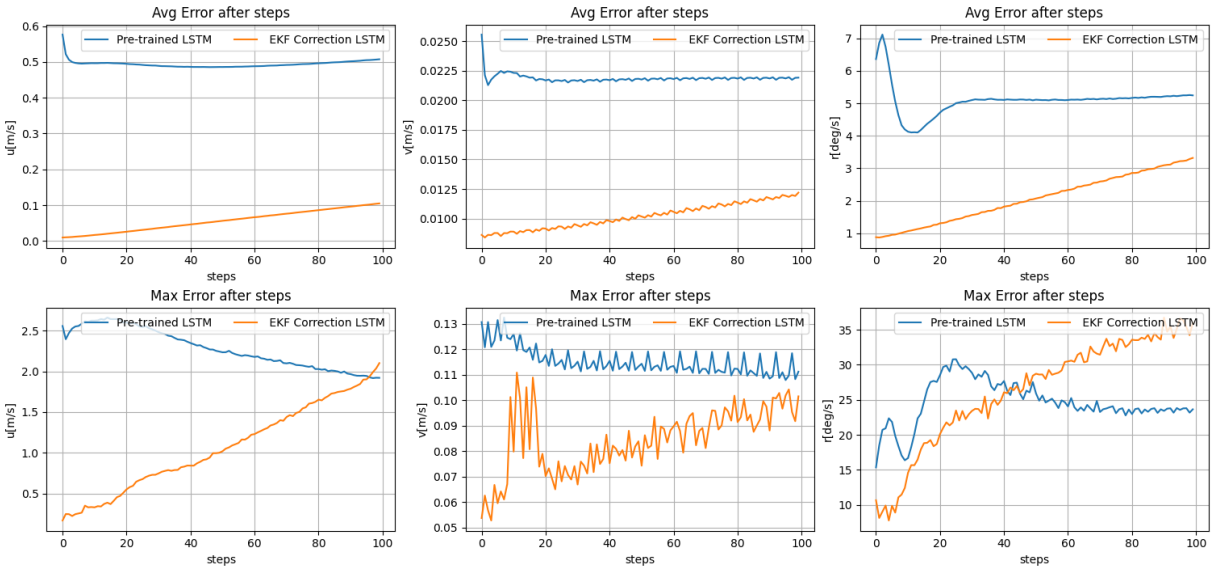

Figure 5.12: Overall results for an LSTM with a cell state size 18. The first row of charts shows the average prediction error as more simulation steps are given. The second row shows the maximum prediction error obtained as the model progresses without a reference.

The results of the LSTM are shown in Figures 5.11 and 5.12. The transition velocity from low-speed mode to high-speed mode was 2.5 m/s. With the EKF real-time correction, the average error was reduced in all the cases, for all velocities.

When compared with the pre-trained, the maximum error in the whole trajectory was also reduced in all predictions tested for u and v velocities, except from step 97 to 99 for u. For yaw velocity, the maximum prediction error for the real-time was below the pre-trained up to 42 steps ahead, after 40 steps the model presented a greater maximum error than the pre-trained. The explanation for this comes from the fact that, during more dramatic velocity changes, peaks of error pop out, indicating that it takes some time to adapt to the new conditions. This behavior is seen in Figure 5.11 when t is close to 115 s, for example. Near this instant of time, the EKF error is clearly higher than the pre-trained.

Despite the errors, the LSTM trained in real time can be used to predict the next 20 steps and seems to be reasonable to be applied as a predictor in an MPC framework.

5.2.3 GRU applied for system identification in real time

As seen in Figure 5.13, the GRU architecture is a modified version of the LSTM, where the cell state is agglutinated with the hidden state. In this section 2 different arrangements of GRU were applied to identify the hydrofoil dynamics. The first arrangement has a hidden layer with size 18 and the 3 vehicle velocities (u, r, v) are obtained from a linear combination of those 18 internal state variables, f_y in (5.10) is a matrix. The second arrangement is similar to the first but with a hidden layer of size 9. The training set used here was a lawnmower trajectory similar to the one used with the LSTM, the transition velocity

from low-speed mode to high-speed mode was 2.0 m/s.

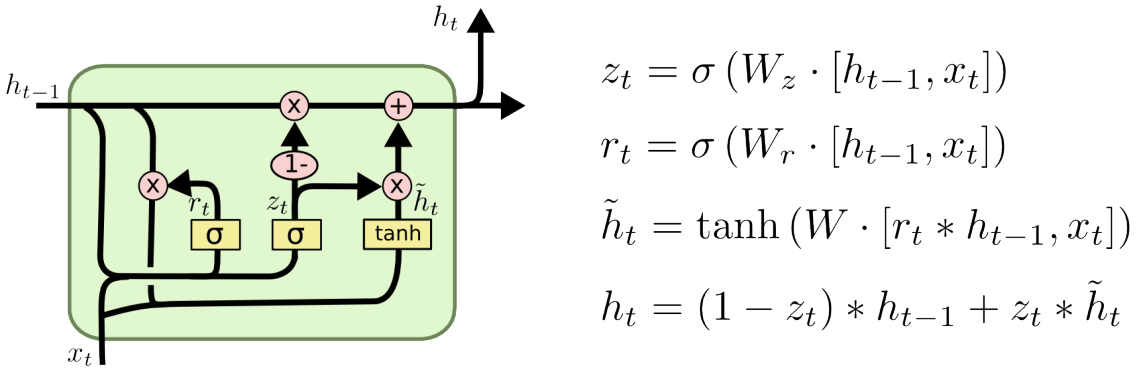

Figure 5.13: GRU architecture. σ represents a layer with a sigmoid activation function. In this work, x_t is equivalent to $\mu(k)$ and h_t corresponds to $\hat{h}(k)$ in (5.8). Image extracted from [2]

.

The results of the GRU arrangement with a hidden layer of size 18 and 9 are displayed in Figures 5.14, 5.15, 5.16. The accuracy was improved with EKF real-time learning. As happened with LSTM, in regions with sharper nonlinearities, errors increased, indicating that time is necessary to adapt to new conditions. GRU models also presented more difficulties in correcting the yaw velocity. If the model is used to predict the next 20 steps and apply MPC, the learned model seems to be capable of choosing a good control trajectory. The GRU model with a hidden layer with dimension 9 is simpler and lighter than the model with size 18. However, the accuracy and adaptation capacities are higher for the 18-sized model.

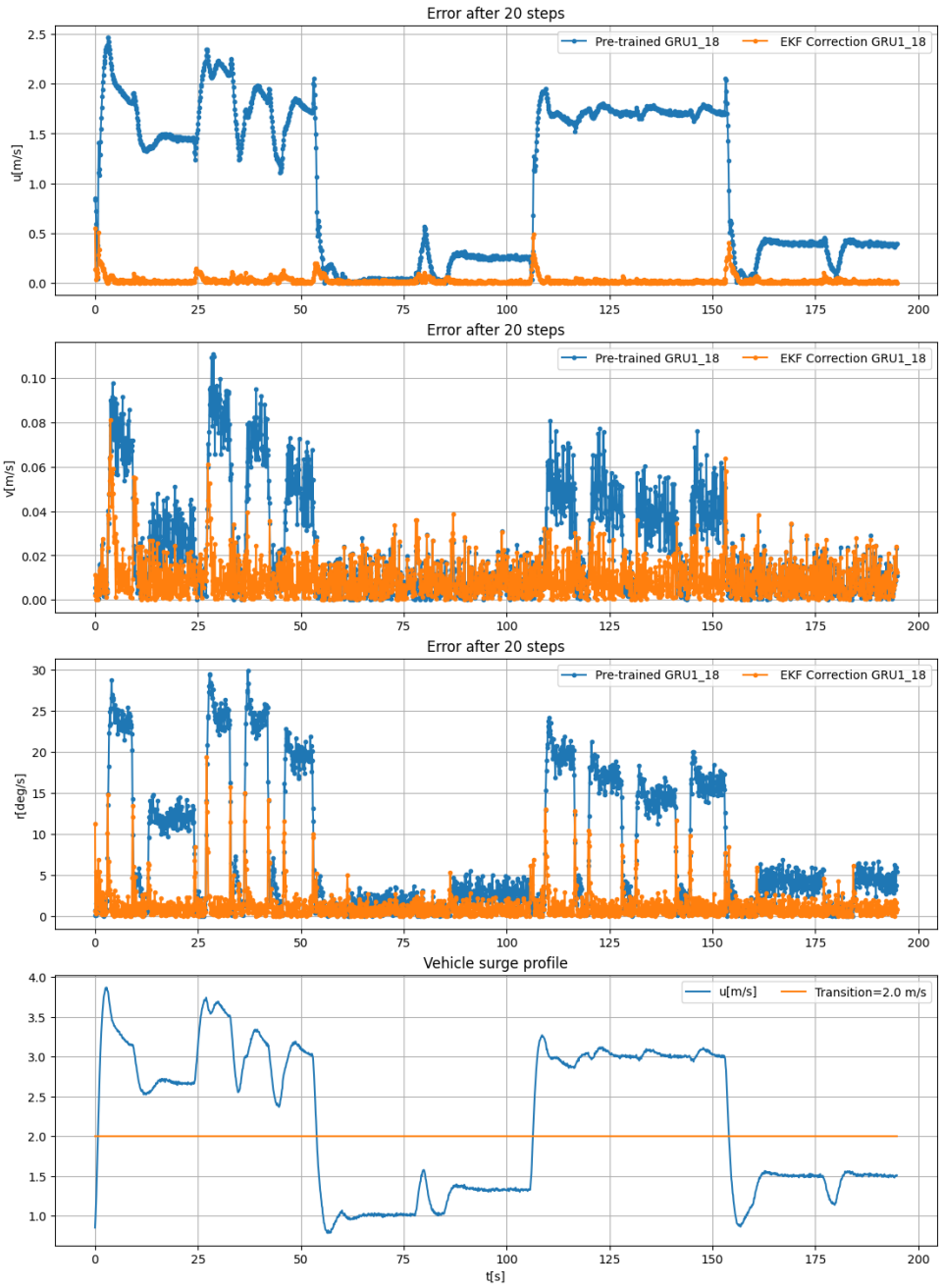

Figure 5.14: Comparison between the pre-trained and the EKF real-time corrected model for the GRU, with a hidden layer of size 18, predicting 20 steps ahead the reference.

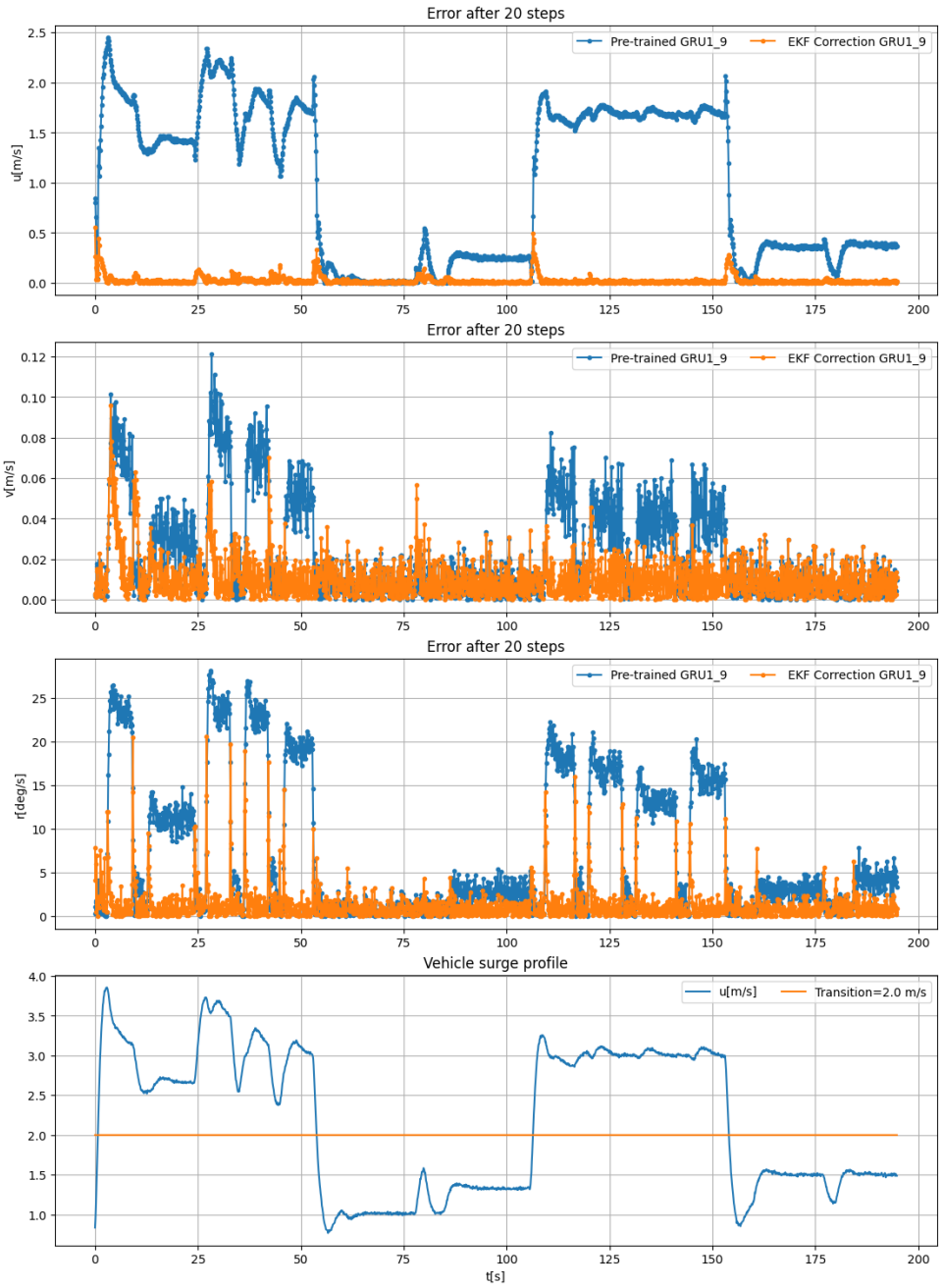

Figure 5.15: Error comparison between the pre-trained and the EKF real-time corrected model for the GRU, with 9 hidden states, predicting 20 steps ahead the reference.

5.2.4 Challenges to apply real-time training with EKF

There are 3 challenges with the EKF real-time learning that must be highlighted.

Firstly, [20] clarifies that the EKF method does not guarantee a convergence to global optima. Since this method inherits properties from Newton's optimization, it works better in functions that have a strictly

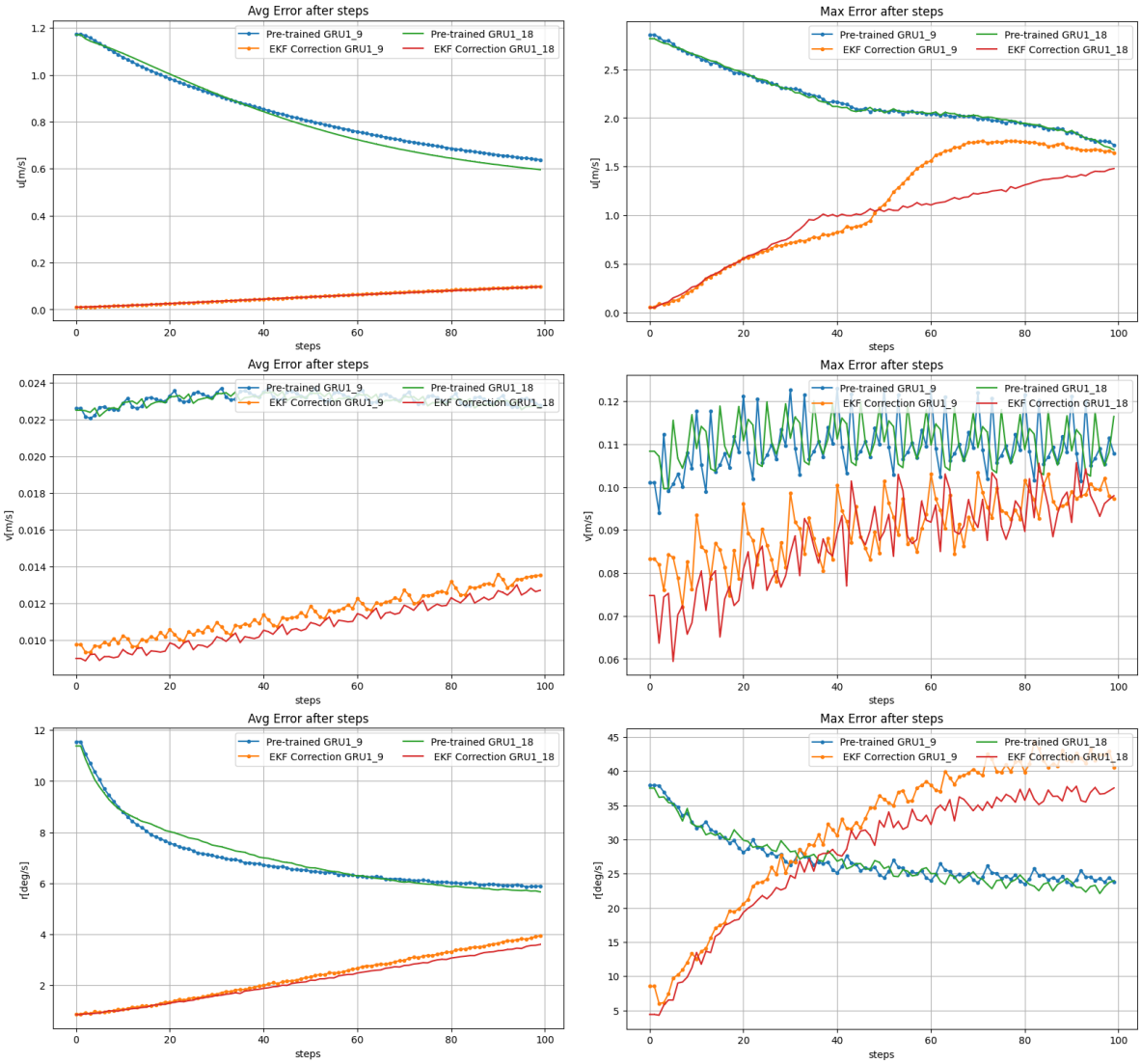

Figure 5.16: Overall results for a GRU's with hidden layers of size 18 and 9. The first column of charts shows the average prediction error as more simulation steps are given. The second column shows the maximum prediction error obtained as the model progresses without a reference.

convex loss. If the dynamical system associated with the regularization factors yields a loss that does not resemble strictly convex, this method is expected to present a poor performance.

Secondly, from Figures 5.14 and 5.11, it can be seen that the EKF can lead the model to be "greedy" with respect to the current conditions. It can "forget" what was learned previously, especially if the NN does not have enough degrees of freedom to represent the actual system. Due to the greediness of EKF in real time, the model is subjected to become degraded by the method. Thorough tests carried out with the same networks from Sections 5.2.3, 5.2.2, 5.2.4 showed that if the EKF training is repeated for several epochs, the model can diverge. To avoid degeneration over time, strategies to preserve stability

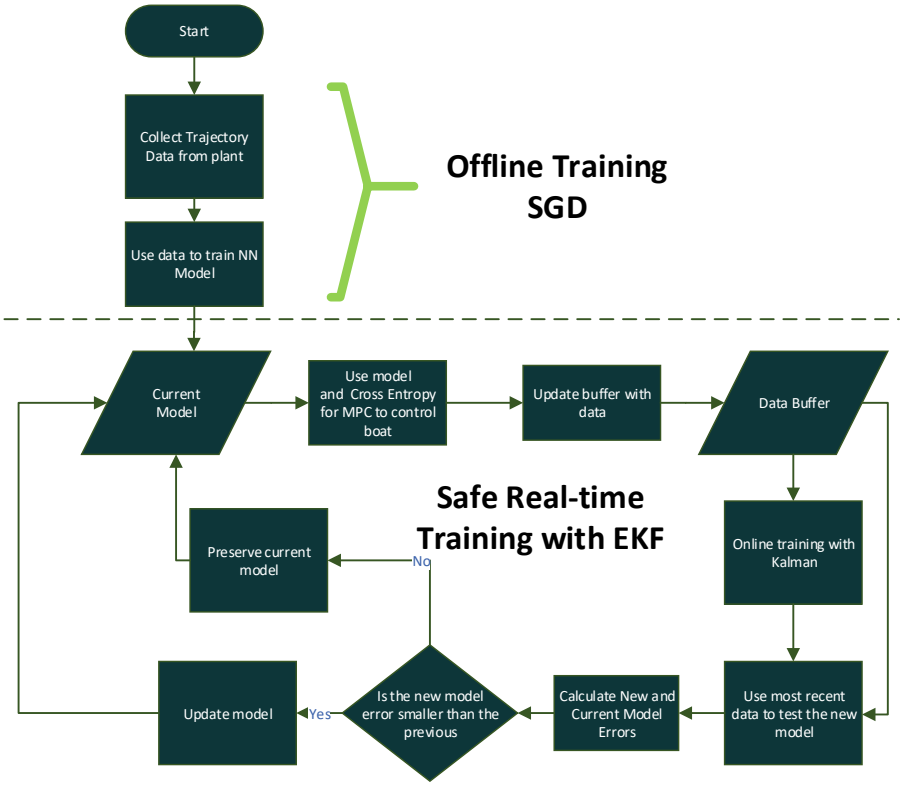

Figure 5.17: Workflow for a safe self-learning process with RNN models.

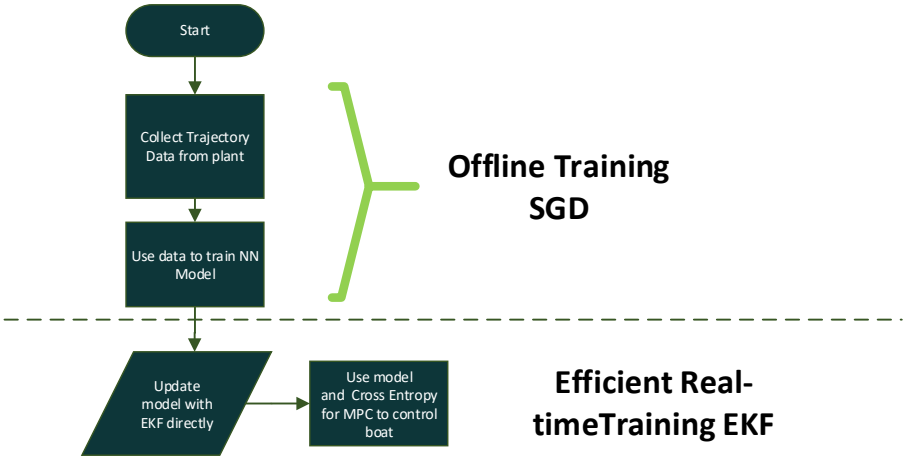

Figure 5.18: Workflow for an efficient self-learning process with RNN models.

can be applied. Figure 5.17 shows a workflow with a procedure to have a safer but also heavier and slower online learning process. If memory constraints and speed of convergence are more critical than maintaining a safe convergence, the EKF can be applied directly, as shown in Figure 5.18. It has the

disadvantage of being a ”greedy” strategy that can lead the model to degenerate or become fit only locally.

Finally, as the third challenge, this method requires extra attention to memory usage. It is necessary to be careful with the depth and width of the RNN's fine-tuned with the EKF. This occurs because matrices P, A, and Q from (5.10) grow quadratically with the number of parameters θ. In addition, P and A are multiplied in (5.10), which requires even more memory to be allocated during this operation. Thus, models that are lean in parameters, like the LSTM or GRU, are advantageous over deep and fully connected NN's.

6

Conclusions

Contents

6.1 Overview and final considerations for the identification learning methods

The learning methods studied in this research can facilitate or contribute to control identification for marine dynamic systems. Nevertheless, it is necessary to comprehend under which conditions their maximum potential can be exploited. The outcome from Chapters 4 and 5 provides the basis to understand how those approaches can be harnessed. Table 6.1 sums up a comparison among SINDy, Offline NN training, Vanilla RNN, GRU and LSTM.

Table 6.1: Relative comparison between methods studied in Chapters 4 and 5.

						Accuracy	
	Training Time	**Data Required**	**Memory Required**	**Real-time Learning Capabilities**	**Flexibility with different models**	**Simple systems**	**Complex systems**
SINDy	0.1~0.5 h	Low	Low	High	Low	High	Low
Offline NN	5~12 h	High	High	Low	High	High	High
Vanilla RNN	5~12 h	Medium	Medium	Medium	Medium	Medium	Low
GRU	1~5 h	Medium	Medium	High	High	High	Medium
LSTM	1~5 h	Medium	Medium	High	High	High	Medium

Chapter 4 showed that SINDy is suitable for identifying nonlinear marine vehicle dynamics. This method is the simplest learning method, when compared with Offline NN training, Vanilla RNN, GRU and LSTM. It learns quickly, does not require a large amount of memory, and consumes less data for training. Because the identified model is in the canonical form, Feedback Linearization or Backstepping can be applied directly with this method. The Python library PySINDy allows a design with polynomial terms or other nonlinear terms that can be easily customized. The more it is known about the nonlinear system, the more SINDy can be shaped with constraints and proper nonlinear terms to fit better the marine craft dynamics. In opposition to this, if the system to be modeled has sharp nonlinearities, like the hydrofoil model (3.5), or if not much information is available beforehand, the performance can be poor. For the case where SINDy fails, Chapter 5 showed that neural networks can be trained to predict the dynamic states of marine vessels.

Section 5.1 showed that with enough data, the identification with NN's do not require much knowledge from the control designers beforehand and the predictions are very accurate when it is properly trained. This is the most complex learning method, when compared with SINDy, Vanilla RNN, GRU and LSTM, though. It learns slowly, requires a large amount of memory, and consumes more data for training. It is possible to use EKF real-time learning with this method, and the arrangements for that were left as

future work.

Unlike the Offline Predictor presented in Section 5.1, Section 5.2 showed that RNN's are apt to be trained in real-time with EKF. This learning structure is simpler, and learns much quicker, when compared with the offline NN predictor. However, it is slightly slower than SINDy. The predictions can be very accurate when it is properly designed and trained, which makes it appropriate to identify complex systems, such as the hydrofoil method in (3.5). With enough data, the identification does not require much knowledge from the control designers beforehand. Feedback Linearization or Backstepping can be applied if the RNN structure is designed to be in the canonical form $f(X) + g(X)\mu$. This was left as a future exercise.

6.2 Limitations for real-time learning with EKF

The focal point of this research was harnessing the use of NN for identification and control. From Table 6.1, it can be drawn that LSTM's and GRU's, from Section 5.2 present advantages over the other Offline NN, presented in Section 5.1. Perhaps the most prominent advantages are derived from the capacity to be easily fine-tuned in real time with EKF. In spite of that, there are 2 major limitations in the real-time adjustment.

The first limitation is the risk of degrading the model when real-time learning is applied. The EKF real-time fine-tune can lead to a degenerated RNN model because the dynamics are nonlinear and the error (loss function) is not necessarily strictly convex. This means that some real-time adjustments can diverge the identification instead of improving the accuracy. Figure 5.17 is a workaround for this issue, but not a final solution.

The second limitation comes from the high computational power required to effectuate operations with big matrices that are dense. This is distinctly relevant in the robotics environment, due to the limited resources used in embedded systems. To work with the EKF it is necessary to multiply matrices and extract gradients. The number of terms in P, Q and A matrices are equal to the square of their dimension, which is $(n_h + n_{\theta h} + n_{\theta y})$. This means that the memory consumption grows quadratically with the dimension of the inner state h, and the model parameters. Analogously, the deeper and wider the NN the heavier the calculations to get the gradients.

Being aware of those two major issues, real-time learning seems to be a promising approach for control identification.

6.3 Future work

Chapters 4 and 5 showed evidence that SINDy or NN's model-based can be used for identification. Chapter 5 also showed that real-time adjusting can be applied for marine surface vehicles with EKF. Nevertheless, this research can be improved, extended, or even completely reinvented. Considering this, five possible future works are suggested:

1. designing and testing a structure with real-time learning for the NN predictor from Section 5.1;
2. applying RNN real-time learning to predict the n^{th} step ahead, instead of the first;
3. designing and testing RNN structures that support Backstepping or Feedback Linearization;
4. applying RNN real-time learning under the RL framework;
5. applying regularization terms to improve performance for real-time learning.

The first suggestion is a combination of the work presented in Section 5.1 and Section 5.2. The EKF can be adapted to train the predictor described in Section 5.1. The NN shown in Section 5.1 estimates 20 steps ahead, given 19 future command steps, 4 past command steps, and 4 past states. This arrangement could be replaced with an RNN, where the previous states could be the last 20 states acquired, and the inputs could be $a_t = \{\mu(k),\ \mu(k+1),\ ...\ \mu(k+19)\}$. For 20 estimates ahead, the corrector should be on standby during the first 20 steps. The second suggestion is very similar to the first, except that instead of predicting the entire trajectory for n steps ahead, just the ultimate one would be predicted. This strategy can reduce the complexity and size of the RNN. Due to the higher magnitude of the changes between step k and step $k+n$, the impact of noise would be less relevant.

The first and the second propositions, as they are presented, still depend on an MPC framework to find a control sequence from the identified dynamics. The third and fourth suggestions were thought to eliminate this requirement. To carry out the third suggestion, an RNN architecture can be built with a structure similar to $x_{k+1} = f_x(x_k) + g_x(x_k) \cdot \mu$, instead of encapsulating the control μ and X in a single nonlinear RNN $f_x(X, \mu)$. The implementation should be with an architecture composed by two separate NN, the first being $f_x(X, k)$, and the second being $g_x(X)$. $f_x(X, k)$ and $g_x(X, k)$ could be a simple Vanilla RNN model or a customized arrangement. The third suggestion can be applied in conjunction with the first and second suggestions, or it can be applied independently. Conversely, the fourth suggestion eliminates the necessity of a MPC optimizer by employing an RNN directly as a controller, instead of as an identifier. In this case, the inputs of the RNN, x_{t+1} (see Figure 5.4), should be the current dynamic state, $x(k)$ concatenated with the future desired dynamic state $\hat{x}(k+n)$ and h_t could be the sequence of commands μ to reach $\hat{x}(k+n)$. This modification puts the RNN with EKF real-time learning under the RL paradigm. Because of the frequent update of the RNN, this approach could yield smooth control sequences.

The last suggestion can be applied in conjunction with all the others. It is feasible because the EKF enables the application of regularization terms. Regularization terms have the potential of maximizing the accuracy and enhancing the capacity of NN to generalize. In this thesis, regularization functions were not explored for real-time NN adjustments and it could be further investigated.

Bibliography

[1] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, vol. abs/1701.07274, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274

[2] "Understanding lstm networks – colah's blog 2015," Aug 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[3] P. Ridao, M. Carreras, D. Ribas, P. J. Sanz, and G. Oliver, "Intervention auvs: the next challenge," *Annual Reviews in Control*, vol. 40, pp. 227–241, 2015.

[4] W. Wang, L. A. Mateos, S. Park, P. Leoni, B. Gheneti, F. Duarte, C. Ratti, and D. Rus, "Design, modeling, and nonlinear model predictive tracking control of a novel autonomous surface vehicle," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6189–6196.

[5] N. Matni, A. Proutiere, A. Rantzer, and S. Tu, "From self-tuning regulators to reinforcement learning and back again," in *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 2019, pp. 3724–3740.

[6] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto *et al.*, "The ai index 2021 annual report," *arXiv preprint arXiv:2103.06312*, 2021.

[7] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[10] D. Bertsekas, “Lessons from alphazero for optimal, model predictive, and adaptive control,” *arXiv preprint arXiv:2108.10315*, 2021.

[11] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[12] T. A. Johansen, T. Perez, and A. Cristofaro, “Ship collision avoidance and colregs compliance using simulation-based control behavior selection with predictive hazard assessment,” *IEEE transactions on intelligent transportation systems*, vol. 17, no. 12, pp. 3407–3422, 2016.

[13] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, “Data efficient reinforcement learning for legged robots,” vol. 100, pp. 1–10, 30 Oct–01 Nov 2020. [Online]. Available: https://proceedings.mlr.press/v100/yang20a.html

[14] A. Wigren, J. Wågberg, F. Lindsten, A. G. Wills, and T. B. Schön, “Nonlinear system identification: Learning while respecting physical models using a sequential monte carlo method,” *IEEE Control Systems Magazine*, vol. 42, no. 1, pp. 75–102, 2022.

[15] S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Discovering governing equations from data by sparse identification of nonlinear dynamical systems,” *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[16] A. A. Kaptanoglu, B. M. de Silva, U. Fasel, K. Kaheman, J. L. Callaham, C. B. Delahunt, K. Champion, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton, “Pysindy: A comprehensive python package for robust sparse system identification,” *arXiv preprint arXiv:2111.08481*, 2021.

[17] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, “A survey of deep learning applications to autonomous vehicle control,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, 2020.

[18] A. Ayyad, M. Chehadeh, M. I. Awad, and Y. Zweiri, “Real-time system identification using deep learning for linear processes with application to unmanned aerial vehicles,” *IEEE Access*, vol. 8, pp. 122 539–122 553, 2020.

[19] L. Ljung, C. Andersson, K. Tiels, and T. B. Schön, “Deep learning and system identification,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1175–1181, 2020.

[20] A. Bemporad, “Recurrent neural network training with convex loss and regularization functions by extended kalman filtering,” *IEEE Transactions on Automatic Control*, 2022.

[21] F. L. Lewis and D. Vrabie, “Reinforcement learning and adaptive dynamic programming for feedback control,” *IEEE circuits and systems magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[22] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[23] J. Schrittwieser, T. Hubert, A. Mandhane, M. Barekatain, I. Antonoglou, and D. Silver, "Online and offline reinforcement learning by planning with a learned model," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 580–27 591, 2021.

[24] J. Woo and N. Kim, "Collision avoidance for an unmanned surface vehicle using deep reinforcement learning," *Ocean Engineering*, vol. 199, p. 107001, 2020.

[25] Q. Zhang, W. Pan, and V. Reppa, "Model-reference reinforcement learning for collision-free tracking control of autonomous surface vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 8770–8781, 2022.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[27] R. Cui, C. Yang, Y. Li, and S. Sharma, "Adaptive neural network control of auvs with control input nonlinearities using reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 6, pp. 1019–1029, 2017.

[28] Y. Cheng and W. Zhang, "Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels," *Neurocomputing*, vol. 272, pp. 63–73, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217311943

[29] Q. Zhang, J. Lin, Q. Sha, B. He, and G. Li, "Deep interactive reinforcement learning for path following of autonomous underwater vehicle," *IEEE Access*, vol. 8, pp. 24 258–24 268, 2020.

[30] I. Carlucho, M. De Paula, and G. G. Acosta, "An adaptive deep reinforcement learning approach for mimo pid control of mobile robots," *ISA transactions*, vol. 102, pp. 280–294, 2020.

[31] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6023–6029.

[32] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[33] M. Caccia, M. Bibuli, R. Bono, and G. Bruzzone, "Basic navigation, guidance and control of an unmanned surface vehicle," *Autonomous Robots*, vol. 25, no. 4, pp. 349–365, 2008.

[34] K. Champion, P. Zheng, A. Y. Aravkin, S. L. Brunton, and J. N. Kutz, “A unified sparse optimization framework to learn parsimonious physics-informed models from data,” *IEEE Access*, vol. 8, pp. 169 259–169 271, 2020.

[35] C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius, “Sample-efficient cross-entropy method for real-time planning,” vol. 155, pp. 1049–1065, 16–18 Nov 2021. [Online]. Available: https://proceedings.mlr.press/v155/pinneri21a.html