Passenger vessel models for fuel efficient fixed-routes

A data-driven approach towards reducing fuel consumption in marine environments using model free reinforcement learning

Master's thesis in Complex Adaptive Systems

BRUCHHAUSEN JAKOB LORENTZON FREDRIK

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Passenger vessel models for fuel efficient fixed-routes

A data-driven approach towards reducing fuel consumption in marine environments using model free reinforcement learning

Bruchhausen Jakob Lorentzon Fredrik

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2022

Passenger vessel models for fuel efficient fixed-routes
A data-driven approach towards reducing fuel consumption in marine environments using model free reinforcement learning
JAKOB BRUCHHAUSEN, FREDRIK LORENTZON

Academic supervisor and examiner: Balázs Kulcsár, Department of Electrical Engineering
Industrial advisors: Joel Ödlund, Cetasol; Simon Johansson, Cetasol

Master's Thesis 2022
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Photograph of the passenger ferry Burö. Source: Pia Lorentzon.

Typeset in LaTeX
Printed by Department of Electrical Engineering
Gothenburg, Sweden 2022

Passenger vessel models for fuel efficient fixed-routes
A data-driven approach towards reducing fuel consumption in marine environments using model free reinforcement learning
JAKOB BRUCHHAUSEN, FREDRIK LORENTZON
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Minimizing fuel consumption of marine vessels has environmental, economical, and health related advantages. Today, many vessels lack software to help operators drive efficiently. We present a complete reinforcement learning framework that models the behaviour of a marine vessel and optimises it with regard to its fuel consumption. The reinforcement learning algorithm consists of an environment and an agent. The environment is built using an LSTM neural network trained on real-life data. The data is analysed to find relevant features, strongly correlated with fuel consumption, and to remove irrelevant ones. The agent is built using a deep Q-learning architecture. Moreover, a Hidden Markov Model was implemented to infer latent variables. It evaluates states at each time-step and feeds its output to both the environment model and the agent. The LSTM and Hidden Markov models are built on data from the passenger vessel Burö, operating in the Göteborg archipelago.

Some models manage to describe the vessel's behaviour relatively well, when evaluated on test data. Implementation of the Hidden Markov Model in one model gave indications of improved model performance. A visual analysis of the obtained latent variable further gave indications of a beneficial implementation of the hidden Markov model. The agent manages to find a good but not optimal policy. In conclusion, the proposed reinforcement learning algorithm is not accurate enough to be implemented into real-life applications. However, we present many important insights to future works, such as the reinforcement learning architecture and the importance of estimating latent variables.

Keywords: machine learning, data-driven, reinforcement learning, LSTM, DQN, NARX, hidden Markov models, Gaussian processes, marine vessel modelling.

Acknowledgements

We would like to direct special gratitude towards our examiner and supervisor Balazs Kulcsár, Dept. of Electrical Engineering at Chalmers, for showing interest and believing in this project from the very first time we met all they way through the end. We will miss the weekly meetings and check-ins, where a positive atmosphere was always to be found. His experience and guidance played a crucial role in this project, without which it would never have been possible to complete. We would also like to extend thanks to the following people: Ethan Faghani, CEO at Cetasol, for having us at Cetasol and making us feel welcome and a part of the team. Joel Ödlund at Cetasol for helping us get started in the beginning of this project and being there to answer our questions. Simon Johansson at Cetasol for the interesting and rewarding discussions, and Björn Karlsson at Cetasol for his expertise in sensors and signals. Rebecka Jörnsten, Dept. of Mathematical Sciences at Chalmers/GU, for your advice on modelling and hidden Markov models, Balazs Varga, Dept. of Electrical Engineering at Chalmers, for revealing to us the wonderful world of Heuristics, Sten Elling Tingstad Jacobsen, Dept. of Electrical Engineering at Chalmers, for his useful advice on Gaussian process regression. All of the above have contributed with good advice and expertise in their own respective fields, all of which has been important towards the completion of this project. We have thoroughly enjoyed meeting you all and wish you good luck in your future endeavors.

Jakob Bruchhausen, Fredrik Lorentzon, Gothenburg, May 2022

Contents

List of Figures

List of Tables

1
Introduction

The maritime industry is a large actor in the global economy. According to the World Bank, in 2017 80% of international trade goods were transported by sea and this figure is expected to quadruple until 2050. Ocean fishing is estimated to contribute more than 270 billion USD to global GPD each year. Ferry passengers in the European Union increased with almost 15% between 2014 and 2019, reaching 419 million passengers during 2019 [1, 2].

The blue economy, defined by the European Commission as all economic activities related to oceans and coastal regions, is growing. In 2021 the European Commission reported coastal tourism, port activities and maritime activities increased with 20.9%, 14.5% and 12% respectively between 2009 and 2018. In the same period coastal tourism also had an increase in employments of 45%. The industry of offshore wind power in 2018 reported an increase in employment of 15% from the year before [3]. To summarise, the blue economy is growing rapidly, and not only due to increase in already existing sectors. New industries dependant on the ocean and coastal regions are also emerging, such as algae production and renewable energy in the form of e.g. wave power. Along with the above reported stable increase comes the pressure of a sustainable future blue economy. In 2018 the total greenhouse gas emissions (carbon dioxide, methane and nitrous oxide) measured $1,076$ CO_2e (carbon dioxide equivalents), an increase of 9.6%. In 2018 this means 2.89% of global emissions could be linked to the shipping industry [4]. Positive correlations has been found between economic growth with maritime transport, air pollution of nitrogen oxides (NO_x) and sulphur dioxide (SO_2) [5]. Increased exposure to these substances can lead to respiratory illness and positive correlations have also been found between increase short-term exposure to SO_2 and mortality due to such illnesses [6]. NO_x is along with particle pollution two of the main exhausts from diesel engines, a common engine type used in marine propulsion. Exposure to particle pollution is also considered one of the major contributions to increased cardiovascular disease along with an increase in metabolic disorders such as diabetes [7]. In addition to the severe health hazards, environmental impacts such as eutrophication and visibility reduction due to smog are also caused by diesel engine emissions [8]. All of the above point towards the need for a decrease in fuel consumption within all sectors of the marine industry.

Machine learning and AI is a thriving field in both industrial applications and in research. Supervised learning is a subcategory of machine learning, where a mathematical model is trained to describe a set of inputs and outputs given real-life data [9]. There are many different types of models which go under the umbrella of supervised learning. Having access to such a model furthermore allows for studying

and optimizing it to a given goal. A second subcategory of machine learning is reinforcement learning, where one attempts to find an optimal policy that maximises a reward function [10]. Reinforcement learning is fundamentally different from other branches of machine learning because it relies on exploration and rewards rather than a defined error. Recently, reinforcement learning has gained publicity for its ability to solve complex tasks, including playing the game of Go to a superhuman level [11] and predict protein folding [12]. In reinforcement learning, an agent and an environment are defined. The agent interacts with the environment, and performs actions in it. The actions result in rewards and the agent's goal is to maximise the long-term reward. This scope makes reinforcement learning useful for performing tasks that require long-term planning. One such example is driving a marine vessel efficiently. The vessel operator needs to get to its target dock in time but should also drive with as low fuel consumption as possible.

In this project we study real-data from Burö, a passenger vessel traversing a fixed route in the archipelago of Göteborg. The project will be carried out in collaboration with Cetasol AB, who are also the providers of the data. Using the data provided, we attempt do define a machine learning based approach for conserving fuel.

1.1 Aim

The large scale aim of the project can be split into two main parts: to create and evaluate three different mathematical models and their ability to accurately describe the dynamics of the vessel, and afterwards attempt to a define a optimal strategy for minimizing fuel consumption on a given trajectory. Such an optimal strategy will be found using reinforcement learning and more specifically deep Q-learning. The goal hereby is to create a starting point of a framework which could be used to conserve fuel in fixed-route marine transport. From the data, features strongly correlated with fuel use will be selected, and irrelevant ones will be removed. Furthermore we analyse the possibility of using a latent variable model to infer more information about the vessel dynamics, and any possible model improvements such inferred information could offer.

1.2 Scope

The goal of this project is not to solve question of minimizing fuel consumption for marine vessels. Rather we seek to create and evaluate different approaches to reducing fuel consumption and compare them to each other. By this we hope to build a initial framework, which in the future can be improved and expanded upon. Focus of the project will be directed towards modelling the internal dynamics of the vessel. By the internal dynamics we more specifically mean parameters such as the engine speed, gas pedal position and the vessel speed. Features which to a larger degree might be considered be of environmental origin will not be considered. The scope of the project is also affected directly by the nature and characteristics of the data. The data is collected from a passenger vessel using a Volvo Penta D13 engine, and as such the results of the project will consequently be biased towards

the specifics of the vessel and its specifications.

For the reinforcement learning part of the project, there will be a limited amount of time spent on querying about different types of reinforcement learning algorithms and the parameters settings of such algorithms.

1.3 Data and ethics

No individual or personal data will be used in the project. No sensitive and private data are used. The method can not be used to discriminate, fairness is by default included (no priority or advantage can be given in the algorithm to discriminate). All constraints will be given externally, no bias can be methodologically included, unbiasedness is built-in. Decisions made by the AI will be made transparent and interpretable, supporting and not constraining human decision-making.

2
Theory

Chapter 2 will discuss the relevant theoretical background to the fundamental concepts applied in the thesis. The theory chapter will not discuss the concepts in relation to the project aim and unique implementations of the work. Instead it will offer a general but detailed description of the theoretical concepts to facilitate further reading and understanding of the report. First the chapter reviews some classical marine vessel dynamics after which we delve further into theory more specific to the project. This includes modelling different aspects of a vessel. Finally the chapter is concluded by describing the means of optimizing a model with respect to some goal.

2.1 Classical marine vessel dynamics

The acceleration $\dot{v}$ of a marine vessel is determined by the engine force, drag, and external forces such as winds and streams. In addition, because a marine vessel operates in a viscous medium and therefore is able to drift, the direction of movement is in part determined by history.

For forward motion, the acceleration in the direction of movement $\dot{v}_{forward}$ follows Newton's second law of motion where $\dot{v}_{forward}$ is proportional to the engine force F_e, factored for the weight of the vessel m

$$\dot{v}_{forward} = \frac{F_e}{m} \quad . \tag{2.1}$$

The negative acceleration $\dot{v}_{backward}$ follows the drag equation

$$\dot{v}_{backward} = \frac{F_d}{m} = \frac{\rho v^2 c_d A}{2m} \tag{2.2}$$

where ρ is the density of the water, v is the velocity of the vessel (for still water), c_d is a dimensionless unity called *drag coefficient*, and A is the reference area (the area relevant for the contact with the water).

Additional forces, such as winds and streams, can be notated ε. The resulting acceleration is hence

$$\dot{v} = \dot{v}_{forward} - \dot{v}_{backward} + \varepsilon \quad . \tag{2.3}$$

Since $\dot{v}_{forward}$ increases independently of v and $\dot{v}_{backward}$ increases with a square relationship to v, the two terms will be equal for a given value of v. This is the speed the vessel will keep for that value of F_e [13] [13].

Equation (2.1) and (2.2) are simplistic marine vessel models. More advanced models strive to also describe propeller and rudder dynamics [14], and the vessel's six degrees of freedom: heaving, swaying, surging, yawing, rolling and pitching [15].

2.2 Moving average filters

Noise in time series data can be the product of different things such as circuit errors, loose connections, etc. If the noise in the signals is deemed to be random it can be filtered out. A method of filtering out noise and in practice smoothing the series is using a moving average filter [16]. A moving average filter calculates the mean in a sampling window of size $2k + 1$ according to

$$MA = \frac{o_{t-k} + o_{t-k+1} + \cdots + o_t + \cdots + o_{t+k-1} \quad , o_{t+k}}{2k+1} \quad . \tag{2.4}$$

By running a moving average process over a set of time series, the series are "smoothed" to remove noise and irregularities.

2.3 Hidden Markov models

Real world events, such as marine vessels traversing a fixed route, generate observable results which can be collected and stored in a database. These observations, e.g. wind speed, vessel course, engine rpm, etc, can later be used to describe the event with a mathematical model. If we assume that the underlying real world process cannot entirely be described by the observed variables alone we find the process must also be a function of a set of latent (hidden) variables which are not observed. To be able to describe the entire real-world process these hidden variables are inferred from existing ones. One class of models applicable to this problem are hidden Markov models [17].

A hidden Markov model (HMM) is a model which aims to describe a Markov process $\boldsymbol{\mathcal{X}}$ with N states that produces outputs that are not observable, i.e. the process is hidden. $\boldsymbol{\mathcal{X}}$ is assumed to influence a second process $\boldsymbol{\mathcal{O}}$ of dimension D which produces observable outputs $\boldsymbol{o}_t$. The connection between $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{O}}$ is assumed to be known and by observing the process $\boldsymbol{\mathcal{O}}$ one attempts to learn the process $\boldsymbol{\mathcal{X}}$ [18]. In relation the modelling of latent variables, the hidden process acts as a latent variable. It affects the observed outputs but can cannot be observed itself.

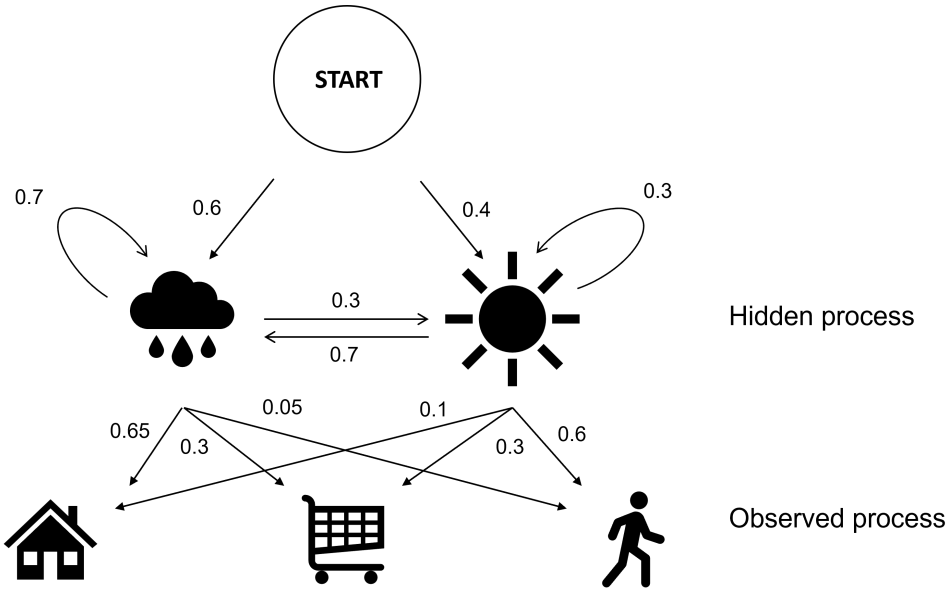


Figure 2.1: Example of a hidden Markov model of $N = 2$ hidden states with three different emission states. Arrows indicate transitions with probabilities. The observed process is whether someone stays at home, goes shopping or goes for a walk. The hidden process is the weather which determines the observed action by a set of emission probabilities B.

Figure 2.1 shows a simple example of a HMM where a behaviour is observed, namely if a person decides to stay at home, go shopping or go for a walk. This observable action is determined by the state of weather, if it is sunny or not, which is hidden from the observer.

The hidden process of the HMM, the weather in the above example, is a discrete-time Markov chain where the number of states N is determined by the number of hidden states in the overarching HMM. As it is a Markov chain it is parameterised by a initial transition probability π which gives the probability of the initial state, as well as a transition probability matrix A containing elements $a_{i,j}$ which are transition probabilities between states i and j for $i, j \in \{0, \ldots, N\}$. In addition to π and A the HMM is further parameterised by the emission probabilities B. $b_i(y_t)$ denotes the probability of observing a given output $\boldsymbol{o}_t$ at time t given state i for $i \in \{1, \ldots, N\}$ [19]. The final parameterization of the HMM can be described by

$$\lambda_{HMM} = \{\pi, A, B\} \quad . \tag{2.5}$$

In the study of HMMs there are three problems which must be solved for the model to be applicable to a real world event. These problems are:

1. Given a sequence of observations $\boldsymbol{O} = [\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_{T-1}, \boldsymbol{o}_T]$ and the model $\lambda_{HMM} = \{\pi, A, B\}$, how does one compute $P(\boldsymbol{O}|\lambda_{HMM})$?
2. Given a sequence of observations $\boldsymbol{O} = [\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_{T-1}, \boldsymbol{o}_T]$ and the model $\lambda_{HMM} = \{\pi, A, B]$. How does one calculate a state sequence $S = [s_1, \ldots, s_T]$, which is optimal?
3. How does one adjust the model parameters λ_{HMM} to maximise $P(\boldsymbol{O}|\lambda_{HMM})$?

A solution to calculating $P(\boldsymbol{O}|\boldsymbol{\lambda_{hmm}})$ can be formulated from its definition. Consider a state sequence of length T, $S = [s_1, \ldots, s_T]$. The probability of the observation sequences given this state sequence can be given as

$$P(\boldsymbol{O}|S, \lambda_{HMM}) = \prod_{t=1}^{T} P(\boldsymbol{o}_t|s_t, \lambda_{HMM}) \quad , \tag{2.6}$$

where $P(\boldsymbol{o}_t|s_t, \lambda_{HMM}) = b_{s_t}(\boldsymbol{o}_t)$ Given the model λ_{HMM} (2.5), the probability of the state sequence S can be written as

$$P(S|\lambda_{HMM}) = \pi_{s_1} a_{s_1 s_2} \cdots a_{s_{T-1} s_T} \quad . \tag{2.7}$$

The product of $P(\boldsymbol{O}|S, \lambda_{HMM})$ and $P(S|\lambda_{HMM})$ is the joint probability $P(\boldsymbol{O}, S|\lambda_{HMM})$ that the observations sequence and the state sequence occur simultaneously. A solution for problem 1 can now be formulated by summing the joint probability $P(\boldsymbol{O}, S|\lambda_{HMM})$ over all possible state sequences S_i of some length T. The final number of possible state sequences is N^T, N being the number of states in the hidden markov model. The resulting equation is

$$P(\boldsymbol{O}|\boldsymbol{\lambda_{hmm}}) = \sum_{i=1}^{N^T} P(\boldsymbol{O}|S_i, \lambda_{HMM}) P(S_i|\lambda_{HMM}) \quad . \tag{2.8}$$

As this computation involves calculating equations (2.6) and (2.7) for each possible state sequence the total number of equations amount to $2TN^T$. For the simple example of $N = 5$ and $T = 100$ this amount to $\approx 10^{72}$ calculations, which of course is very impractical. Instead of this costly computation a solution to be first problem can be calculated quite simply using the forward-backward procedure [20]. The forward procedure is described in more detail in the following equations.

For the forward procedure we first define the quantity

$$\alpha_t(i) = P([\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_t], s_t = i|\lambda_{HMM}) \quad , \tag{2.9}$$

which can be described as the probability of the observation sequence $[\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_t]$ up to time t and the state to be i at t. Using the forward algorithm we can now recursively calculate $\alpha_t(i)$ as

1. **Initialization**
   For each i, $1 \leq i \leq N$

$$\alpha_t(i) = \pi_i b_i(\boldsymbol{o}_t) \quad . \tag{2.10}$$

2. **Recursion**
   For t and j, $1 \leq t \leq T-1$, $1 \leq j \leq N$

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right) b_j(\boldsymbol{o}_{t+1}) \quad . \tag{2.11}$$

3. **Termination**

$$P(\boldsymbol{O}|\lambda_{HMM}) = \sum_{i=1}^{N} \alpha_T(i) \quad . \tag{2.12}$$

Equation (2.12) gives the solution to problem 1. $P(\boldsymbol{O}|\lambda_{HMM})$ is also the maximum likelihood function L of the hidden Markov model given the observation sequence $\boldsymbol{O}$.

A solution to problem 2 is the Viterbi algorithm. The Viterbi algorithm is a dynamic programming approach to finding the optimal state sequence $S = [s_1, \ldots, s_T]$ given a observation sequence $\boldsymbol{O} = [\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T]$ [20]. For defining the Viterbi algorithm we need to first define the the following quantity

$$\delta_{t+1}(i) = \max_{s_1, s_2, \ldots, s_{t-1}} P(s_1, s_2, \ldots, s_t = i, \boldsymbol{o}_1, \ldots, \boldsymbol{o}_T|\lambda_{HMM}) \quad , \tag{2.13}$$

where $\delta_t(i)$ as the highest probability along a single path $\boldsymbol{O}$ at time t which describes the first t observations and ends in state i. By induction we have

$$\delta_{t+1}(j) = \max_i \delta_{t+1}(i) a_{ij} b_j(\boldsymbol{o}_{t+1}) \quad . \tag{2.14}$$

To obtain the optimal state sequence we need loop over each time t and each state j and save the argument which maximises (2.14). As we now have defined the above we can describe the Viterbi algorithm as follows:

1. **Initialization**
   For each i, $1 \leq i \leq N$

$$\begin{aligned} \delta_1(i) &= \pi_i b_j(\boldsymbol{o_1}) \quad , \\ \psi_1(i) &= 0 \quad . \end{aligned} \tag{2.15}$$

2. **Recursion**
   For t and j, $2 \leq t \leq T$, $1 \leq i \leq N$

$$\begin{aligned} \delta_t(j) &= \max_{1 \leq i \leq N} (\delta_{t-1}(i) a_{ij}) b_j(\boldsymbol{o_t}) \quad , \\ \psi_t(j) &= \arg \max_{1 \leq i \leq N} (\delta_{t-1}(i) a_{ij}) \quad . \end{aligned} \tag{2.16}$$

3. **Termination**

$$\begin{aligned} P^* &= \max_{1 \leq i \leq N} (\delta_T(i)) \quad , \\ s_T^* &= \arg \max_{1 \leq i \leq N} (\delta_T(i)) \quad . \end{aligned} \tag{2.17}$$

4. **Backtracking**
   The final step of the algorithm consists of backtracking through all the saved arguments of $\psi_t(j)$ for all t and j, to finallty obtain the optimal. For $t = T-1, T-2, \ldots, 1$,

$$s_t^* = \psi_{t+1}(s_{t+1}^*) \quad . \tag{2.18}$$

In equations (2.17) and (2.18) $P*$ denotes the highest probability and s_t^* denotes the optimal state s given a time t. The Viterbi algorithm shares many traits with the forward algorithm used to solve problem 1. However the forward algorithm lacks the final back-tracking step of the Viterbi algorithm, making them distinct from each other.

The Viterbi algorithm gives a solution to problem 2. Problem 3, estimating and tuning the parameters of the HMM can adressed using the Segmental K-means learning algorithm, which makes use of the Viterbi algorithm [21]. This algorithm will be further discussed in section 3.3.2.

2.4 K-means clustering

K-means clustering is a common clustering method. In K-means clustering, K centers are first initialised into some sub-optimal setting. There are many different methods of initialization available. After initialization of centers, data points are reassigned to the cluster with the closest center. The new cluster centers then updated by calculating the mean of the cluster points. The process is repeated until some convergence criteria is met or a predefined number of iterations have been performed [22].

The clustering performed by the K-means algorithm can be evaluated by calculating the within cluster sum of squares (WCSS) defined as

$$WCSS = \sum_{j}^{K} \sum_{i}^{n} (x_i - C_j)^2 \quad , \tag{2.19}$$

namely the squared distance between a data point and its corresponding cluster center summed over all data points and clusters. In equation (2.19), K is the number of clusters, n the number of datapoints, let x_i denote the ith data point and C_j denote the jth cluster. A smaller WCSS indicates more coherent clusters and a better clustering result.

There are several ways to initialise the centers of the K-means clustering algorithm, such as randomly choosing K data points or for example density-based initialization methods. One issue with the K-means algorithm is the algorithms sensitivity to initialization. A way around is the K-means++ initialization algorithm. The first center is chosen randomly among all data points. The next center is chosen among the remaining data points such that the probability of choosing x_i as the next center is proportional to its distance $D(x_i)$ from the previously chosen center. The process is repeated until K centers have been created [23].

2.5 Gaussian mixture models

A Gaussian mixture model (GMM) is a sum of M weighted multivariate Gaussian density functions which can be used to model some data $\mathbf{x}$ with d features and continuous observations. GMM allows for larger flexibility in data modeling by combining several distributions compared to only using one single multivariate Gaussian distribution. A GMM of M components is given by equation,

$$p(\mathbf{x}|\lambda_{GMM}) = \sum_{j=1}^{M} w_j \, p(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma_j}) \quad , \tag{2.20}$$

where w_j are the individual weights such that $\sum_{j=1}^{M} w_j = 1$, $\boldsymbol{\mu}_j$ are the component mean vectors and $\boldsymbol{\Sigma}_j$ are the component covariance matrices. As such a GMM can be parameterised by

$$\lambda_{GMM} = \{w_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\} \quad j = 1, \ldots, M \quad , \tag{2.21}$$

and subsequently fitting the model to some data $\mathbf{x}$ results in the problem of estimating λ_{GMM}.[24]

2.6 Nonlinear autoregressive exogenous model

A nonlinear autoregressive exogenous model (NARX) is a sequential polynomial regression model with exogenous inputs. Exogenous means that predictions are partially dependent on the model's previous predictions. Algebraically, for one endogenous and one exogenous variable we have,

$$y_t = F(y_{t-1}, y_{t-2}, y_{t-3}, \ldots, u_t, u_{t-1}, u_{t-2}) + \varepsilon_t \tag{2.22}$$

where y is the variable of interest (endogenous) and u is a variable originating from outside the model (exogenous). ε is an error term. The number of historical data points that are fed to the model is called the lags n. $F(\boldsymbol{x})$ is a polynomial function with parameters $\beta_0, \beta_1, \ldots, \beta_n$, which are coefficients multiplied with the input vector $\boldsymbol{x}$. Writing equation (2.22) in matrix notation yields

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{2.23}$$

with parameters $\boldsymbol{\beta}$ that can be estimated using the ordinary least squares estimation equation as

$$\widehat{\boldsymbol{\beta}} = \left(X^{\mathsf{T}}X\right)^{-1} X^{\mathsf{T}}\mathbf{y} \quad . \tag{2.24}$$

This equation find the parameters $\widehat{\boldsymbol{\beta}}$ that minimises the error $\boldsymbol{\varepsilon}$.

2.7 Long short-term memory networks

A recurrent neural network (RNN) is a type of feed-forward artificial neural network, which can be used to model varying lengths of time series data. By storing memory about the previous time-step in a inner hidden state, information can be passed on to the next time-step [25]. A simple recurrent network is one which includes one hidden state with a feed-back connection to itself. Figure 2.2 shows a schematic diagram of such a simple recurrent neural network.

The model from 2.2 can be described using the following equations

$$h_t = \tanh(W_{xh}x_t + b_{xh} + W_{hh}h_{t-1} + b_{hh}) \quad , \tag{2.25a}$$

$$y_t = g(W_{hy}h_t + b_{hy}) \quad . \tag{2.25b}$$

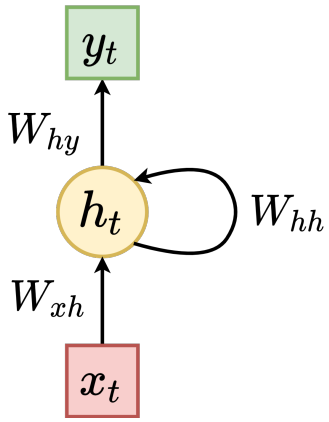


Figure 2.2: Simple RNN model where x_t, y_t and h_t are the model inputs, outputs and hidden state at some time t respectively. W are the different weight matrices of the model. In diagram, model biases b are assumed to be zero.

Where x_t, y_t and h_t denote the input, output and hidden state at time t. W_{xh}, W_{hh}, W_{hy} are the model weight matrices and together with the model biases b_{xh}, b_{hh} and b_{hy} make up the model parameters. g is some activation function.

A long Short-Term Memory Network is a type of recurrent neural network architecture where the hidden neuron is replaced by a module called an LSTM cells. A schematic diagram of a LSTM cell can be seen in figure 2.3. In the LSTM network the hidden state h_t replaces the output of the standard RNN and acts as the output of the LSTM cell, while it also still fed back into the LSTM cell. The internal memory of the LSTM cell is stored in the cell state c_t [26].

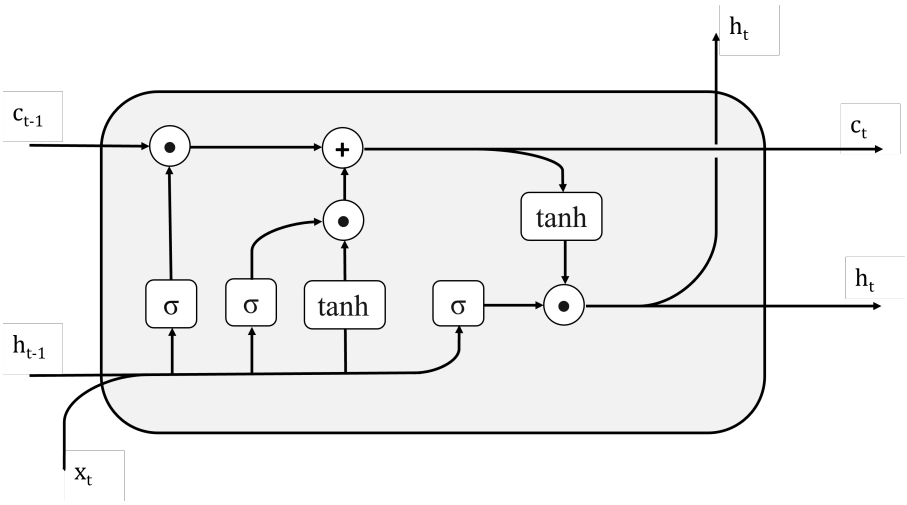


Figure 2.3: Schematic diagram of an LSTM cell. c, h and x denote the cell state, hidden state and input vector respectively. $\cdot$ and $+$ denote the element-wise product and element-wise summation of vectors respectively. The joining of lines in the diagram denotes the concatenation of vectors.

The LSTM cell (figure 2.3) can be separated into four main components. These are the input gate i_t, the forget gate f_t, the output gate o_t and the cell state c_t. The

complete set of equations which describe the different cell gates and a forward pass of a single time-step in time series sequence using a LSTM model is described as follows,

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \quad , \tag{2.26a}$$

$$f_t = \sigma(W_{if}x_t + b_{ii} + W_{hf}h_{t-1} + b_{hf}) \quad , \tag{2.26b}$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \quad , \tag{2.26c}$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \quad , \tag{2.26d}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad , \tag{2.26e}$$

$$h_t = o_t \odot \tanh(c_t) \quad . \tag{2.26f}$$

In equation (2.26), σ is the sigmoid activation function, $\sigma(x) = \frac{1}{1+e^{-x}}$, and $\odot$ is the element-wise product (Hadamard product). Furthermore let W and b denote the weight matrices and bias vectors respectively of each gate. The gates are named by the fact that they are said to have gating properties. More specifically, since the gates all have sigmoid activation functions in the range $[0, 1]$, if the argument to σ is small the gate is more closed and less information is let through and vice versa. The cell state c_t represents the stored memory within the cell and is updated after each time-step t and fed back into the cell. Equations (2.26d) and (2.26e) rule how the internal memory state c_t is updated at time t.

The input gate i (2.26a) governs if information from the input and previous hidden state signals will be stored to the cell state c_t. The parameters of the input gate can therefore be trained to let some signals be stored in cell memory while protecting the cell c_t against perturbations from undesired signals. The output gate o_t (2.26c) controls access to the cell memory c_t then calculating the new hidden state h_t. Training the output gate can allow the model to decide when using memory is useful or not. The cell state c_t tends to grow in a LSTM cell when it is recursively fed back into the cell for each time-step. The forget gate f_t (2.26b) can reset the internal cell memory when the information in c_t is deemed no longer useful, and new useful information can be stored in the memory.

The parameters of a standard RNN and subsequently also a LSTM model can be trained using back propagation. For a full explanation of the back propagation procedure and in particular back propagation through time please refer to [27]. In the training algorithm lies also the primary reason for choosing a LSTM model over the standard RNN. A standard RNN can only take a few time-steps into account when doing predictions. This is due to the back propagated error term which in many cases grow or shrink for each step. As such after a few time-steps the error term tends to either explode or vanish, leading to parameter oscillation during training for exploding gradients or slow or no training for vanishing gradients. The cell state c_t and the gating components of the LSTM architecture deals specifically with the problem of vanishing gradients [28].

LSTM cell units can be combined with other types of artificial neural network architecture, such as fully connected layers, to form larger networks where the LSTM cell is a partial component of the larger model.

2.8 Gaussian processes

A Gaussian process (GP) is a set of random variables $\boldsymbol{X}$, which may be infinite, such that any finite subset $\mathcal{X} \subset \boldsymbol{X}, |\mathcal{X}| < \infty$ can be described as a multivariate Gaussian distribution. The distribution of the GP becomes the joint distribution of $\boldsymbol{X}$. A Gaussian process is said be non-parametric due to the fact of $\boldsymbol{X}$ being a possibly infinite, meaning the GP cannot be described by a finite set of parameters [29]. The GP however has tunable hyper-parameters in the mean function and covariance function, also called the kernel function, both defined in equation (2.27),

$$\mathbb{E}[X] = \mu(x) \quad , \tag{2.27a}$$

$$\text{Cov}(x, x') = k(x, x') \quad , \tag{2.27b}$$

where X denotes a random variables in $\boldsymbol{X}$ and x and x' denote observations of two different random variables X and X' in the Gaussian process model. The mean and kernel function define the prior distribution of the GP. Commonly the mean of the GP is assumed to be zero everywhere and model flexibility is obtained by using the kernel which relates the different random variables to each other. The kernel can be very generic and take many different forms. The kernel function describes the covariance between two observations x and x'. Common kernels include the radial basis function (RBF) kernel and the periodic kernel but several others are available. Figure 2.4 displays two different covariance kernels as a function of the absolute difference between x and x'.

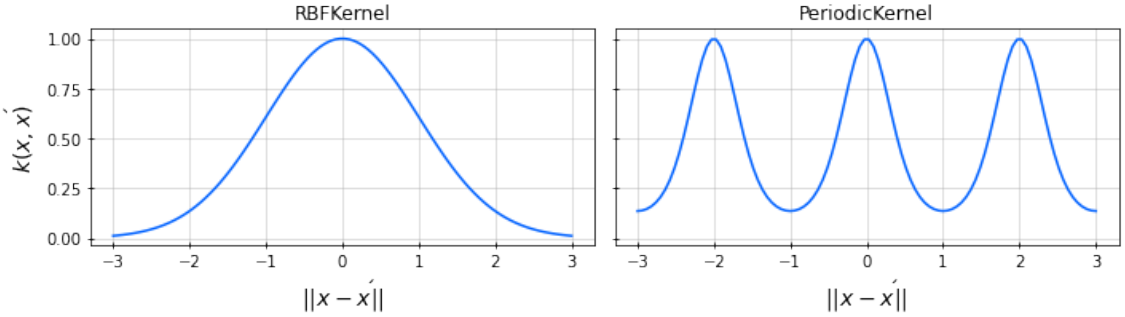


Figure 2.4: The RBG and periodic kernel as a function of $||x - x'||$. For both graphs $\sigma^2 = 1$, $l = 1$ and $p = 2$ for the periodic kernel.

Depending on which kernel is in use the kernel function will have other hyper-parameters parameters such as length-scale for both the RBF and periodic kernel and period time for the period kernel. The RBF and periodic kernel as functions of (x, x') are available in equation (2.28). Here l is the length-scale parameter and p the period time of the periodic kernel, σ_f^2 is the maximum covariance between x and x'.

$$k_{RBF}(x, x') = \sigma_f^2 \exp\left(-\frac{||x - x'||^2}{2l^2}\right) \quad , \tag{2.28a}$$

$$k_{PERIODIC}(x, x') = \sigma_f^2 \exp\left(-\frac{2}{l^2} sin^2\left(\pi \frac{||x - x'||}{p}\right)\right) \quad . \tag{2.28b}$$

From equation (2.28a) and figure 2.4 we find that when using the RBF kernel the covariance $k(x, x') \approx 1$ when $x \approx x'$, and decreases as x and x' move away from each other. Increasing the length-scale parameter l changes the slope at which covariance decreases and as such yields a general increase in covariance between inputs x and x'.

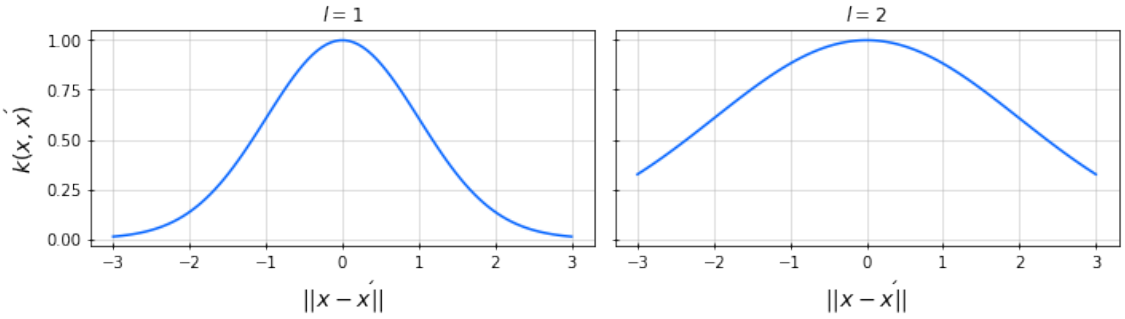


Figure 2.5: Two instances of the radial basis function kernel with two different values for the length-scale parameter l. The left graphs shows a RBF kernel with $l = 1$, for the right graph $l = 2$. For both graphs $\sigma_f^2 = 1$.

Figure 2.5 shows how the RBF covariance kernel changes with the length-scale parameter l. In most cases the RBF kernel will be the default kernel to use when defining a Gaussian process [29].

2.8.1 Gaussian process regression

Consider the simple regression problem $y = f(x)$ and a dataset containing input and output pairs (x, y). $f(x)$ is a type of simple regression model e.g. polynomial regression of a one-dimensional input where $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n$. Parameter estimation of this type of nonlinear regression model returns a model which by some measure fits the dataset best. There may however be an infinite set of functions that fit the dataset equally well. In contrast to the above regression method the Gaussian process regression (GPR) attempts to describe the distribution of the potentially infinite set of functions $\mathbf{f}$ which best fit a set of data points $\mathbf{X} = [\mathbf{x_1}, \ldots, \mathbf{x_n}]$, we denote this distribution $P(\mathbf{f}|\mathbf{X})$ [29]. From the Gaussian process prior (2.27) we have by assumption that $P(\mathbf{f}|\mathbf{X})$ is a multivariate Gaussian distribution of infinite dimension, with mean function $\boldsymbol{\mu}$ and covariance function $\mathbf{K}$,

$$P(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \mathbf{K}) \quad . \tag{2.29}$$

For regression, the prior is conditioned on the training data $\mathbf{X}$ to obtain the posterior, by which we model the joint distribution of the predictions of the training observations $\mathbf{f}(\mathbf{X})$ and the predictions of new test inputs $\mathbf{f}_* = f(\mathbf{X}_*)$, where $\mathbf{X}_*$ denotes a test sample [30]. We find the joint distribution of $\mathbf{f}$ and $\mathbf{f}_*$ to be

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\Big(\begin{bmatrix} \mu(\mathbf{X}) \\ \mu(\mathbf{X}_*) \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*{}^T & \mathbf{K}_{**} \end{bmatrix} \Big) \quad . \tag{2.30}$$

Where $\mathbf{K} = K(\mathbf{X}, \mathbf{X}), \mathbf{K}_* = K(\mathbf{X}, \mathbf{X}_*)$ and $\mathbf{K} = K(\mathbf{X}_*, \mathbf{X}_*)$. In real world applications observations are often noisy, with ϵ representing some noise parameter. Assuming this noise to be additive, $y_i = f(x_i) + \varepsilon_i$, it is therefore useful to add to the covariance function of the observations y a independent and identically distributed Gaussian noise component $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$. We obtain the updated covariance function $\mathbf{K}' = \mathbf{K} + \sigma_n^2$. Using the updated covariance function for training observations and under the assumption that the mean function $(\mu(\mathbf{X}), \mu(\mathbf{X}_*))$ is zero everywhere (see section 2.8), equation (2.30) can be rewritten as

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} + \sigma_n^2 I & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix} \Big) \quad . \tag{2.31}$$

This is the joint distribution $P(\mathbf{f}, \mathbf{f}_* | \mathbf{X}, \mathbf{X}_*)$. However for useful regression we need the distribution of $\mathbf{f}_*$ conditioned on the training observations $\mathbf{f}$, $\mathbf{X}$ and test inputs $\mathbf{X}_*$ which is $P(\mathbf{f}_* | \mathbf{f}, \mathbf{X}, \mathbf{X}_*)$ [29]. This is necesarry since we want to do prediction of new samples based on the training samples and the test inputs. The needed conditional distribution can be derived from equation (2.30) to get

$$P(\mathbf{f}_* | \mathbf{f}, \mathbf{X}, \mathbf{X}_*) \sim \mathcal{N}(\mathbf{K}_*{}^T (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{f}, \mathbf{K}_{**} - \mathbf{K}_*{}^T (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{K}_*) \quad . \tag{2.32}$$

From the (2.32) we now have final definitions of the predicted means and variances for a given test input $\mathbf{X}_*$,

$$\bar{\mathbf{f}}_* = \mathbf{K}_*{}^T (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{f} \quad , \tag{2.33a}$$

$$\operatorname{cov}(\mathbf{f}_*) = \mathbf{K}_{**} - \mathbf{K}_*{}^T (\mathbf{K} + \sigma_n^2 I)^{-1} \mathbf{K}_* \quad . \tag{2.33b}$$

From equation (2.32) is it clear that each prediction using GPR requires computations using all test inputs as well as all training observations. As all samples are needed for prediction the number computations grow with the size of the training data. Likewise GPR becomes more computationally expensive for high dimensional data due to the increasing number of necessary computations [29].

2.8.2 Inference of hyper-parameters

In order for the predictions described in equation (2.33) to be accurate, the covariance kernel $\mathbf{K}$ needs to fit the data well. The initial step in finding a good covariance kernel is choosing the kernel. In many cases the RBF kernel (2.28a) is considered a de-facto default kernel due to its versatility. Once a kernel is chosen hyper-parameters of the kernel have to be chosen. The hyper-parameters θ are chosen such that they maximise the log marginal likelihood,

$$\theta^* = \arg\max_{\theta} \log p(\mathbf{f} | \mathbf{X}, \theta) \quad , \tag{2.34}$$

where θ^* are the optimal hyper-parameters. Adding the task of fitting the hyper-parameters of the covariance kernel the conditional distribution of the GPR (2.32) can be extended as follows

$$P(\mathbf{f}_*|\mathbf{f}, \mathbf{X}, \mathbf{X}_*) = P(\mathbf{f}_*|\mathbf{f}, \mathbf{X}, \mathbf{X}_*, \theta) \quad . \tag{2.35}$$

After tuning of the hyper-parameters the predicted variance (2.33b) now depends on the training ouputs $\mathbf{f}$, which was not the case before. The optimal hyper-parameters, θ^* can be approximated using gradient descent optimization, or any known modification of such an algorithm such as the Adam optimiser [29].

2.9 Deep Q-learning

Deep Q-Learning (DQN) is a reinforcement learning (RL) algorithm that uses a neural network, called the *Q-network*, to estimate the value function. The Q-network is used to, given a state, decide the next action. The action is executed in an environment, resulting in a reward and a new state, which potentially is terminal. The Q-network is trained to optimise the long-term reward. DQN is *model-free*, meaning that it explores the environment instead of explicitly estimating it [31].

The optimal policy is approached by updating the Q-network according to the *Bellman equation* [31]

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t.A_t) \right] \tag{2.36}$$

where Q is the Q-network, S is the state, A is the action, R is the reward, a is the action with the highest Q-value (according to Q), α is the discount factor, α is the learning rate, and the subscripts indicate which time-step the variables refer to. The Q-value can be thought of as the the value of each action given a state. The Bellman equation (2.36) updates the Q-value based on the immediate reward R_{t+1} and the expected future reward $\max_a Q(S_{t+1}, a)$. The importance of immediate versus future reward is controlled by the $\gamma \in [0, 1]$ parameter. A high γ corresponds to a large weight on future rewards, and vice versa.

Since the Q-network is a neural network, it is most efficiently trained on batches of data. However, since neighbouring data points are likely to be similar, simply using the last n time-steps would result in highly correlated batches. To combat this, each time-step is stored in a so called *experience replay*, from which batches are sampled from [32]. In the beginning of the training, the agent only collects samples to the experience replay and no updates of the Q-network are done. Afterwards, the Q-network is updated at given intervals.

To explore the state-space, an ϵ-greedy policy is used. In this policy, a parameter $\epsilon \in [0, 1]$ is defined. In each time-step the action is decided by the Q-network with probability $1-\epsilon$, and is random with probability ϵ [10]. During training, ϵ is initially large, resulting in a highly random behavior. As training progresses, ϵ decreases linearly, to a threshold > 0. The threshold guarantees that the policy is never deterministic. Because DQN improves by observing its environment, regardless of which policy determined the action, DQN is considered an *off-policy* algorithm [31].

2.10 Adam optimiser

Adam (Adaptive moment estimation) is an optimization algorithm built upon stochastic gradient descent. Let $f(\theta)$ be a noisy stochastic function than is differentiable w.r.t its parameters θ. Let g_t be the gradient vector $\nabla_\theta f_t(\theta)$, and $f_t(\theta)$ be the realisation of the function at some time t. Given the task of minimizing the function $f(\theta)$ the Adam algorithm updates the parameters θ using calculated exponential running averages of the gradient and squared gradient. These exponential running averages are in turn estimations of the mean and variance of the gradient respectively [33]. m_t and v_t are the exponential running averages of the gradient and squared gradient respectively. There are parameters (β_1, β_2) that describe the decay rates of m_t and v_t. Adam is efficient in that is employs only first-order gradients and as such requires only small amounts of memory. As it requires small amounts of memory it is well suited for high-dimensional data as well as large data sets.

2.11 PI controller

The proportional-integral (PI) controller is a control loop mechanism used to dampen time-dependent errors. The PI controller calculates an error $e(t)$ as the different with a desired value and a current, actual one. A corrector is then applied to $e(t)$, creating a new entity $u(t)$. The corrector consists of two terms: the proportional P and the integral I (both dependent of $e(t)$), weighted by two constants K_p and K_i. The correction $u(t)$ is fed to a model that acts in the direction that minimises the error [34]. Algebraically,

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau)\,\mathrm{d}\tau \quad . \tag{2.37}$$

The PI controller is a simplified version of the PID controller which has an additional derivative D term.

A PI controller can be used to test the physical relevance of a model. If the model correlates negative errors with a decrease of the target variable, and vice versa, it's a sign of physical relevance. The PI controller will be used to test physical relevance of the dynamic vessel models.

3
Methods

Chapter 3 holds a review about the project methodology as it was carried out. This chapter does not delve thoroughly into the theoretical framework of the applications and methods but rather give a detailed description of implementations on the research questions. For a more meticulous description of the theoretical concepts applied below, please refer to the previous chapter 2. This being said, minor theoretical concepts previously not discussed in report might be featured. If so, they are explained and given references to. The subsections follow the chronological order of the project. The data was first analysed and subsequently a latent variable model and dynamical models were implemented using the data. Finally the models were evaluated and implemented into a reinforcement learning framework for optimization.

3.1 Data analysis

The data was collected from the passenger vessel Burö, operating in the northern part of the archipelago of Gothenburg. The vessel either operates on trips with four stops (Öckerö-Kalvsund-Framnäs-Grötö) or two stops (Öckerö-Grötö). Figure 3.1 shows samples of some four-stops trips. The colour of the samples represent the current vessel speed. The darkest samples are near or at the docks. Kalvsund and Framnäs are only separated by a narrow strait, making it difficult to distinguish them from one another.

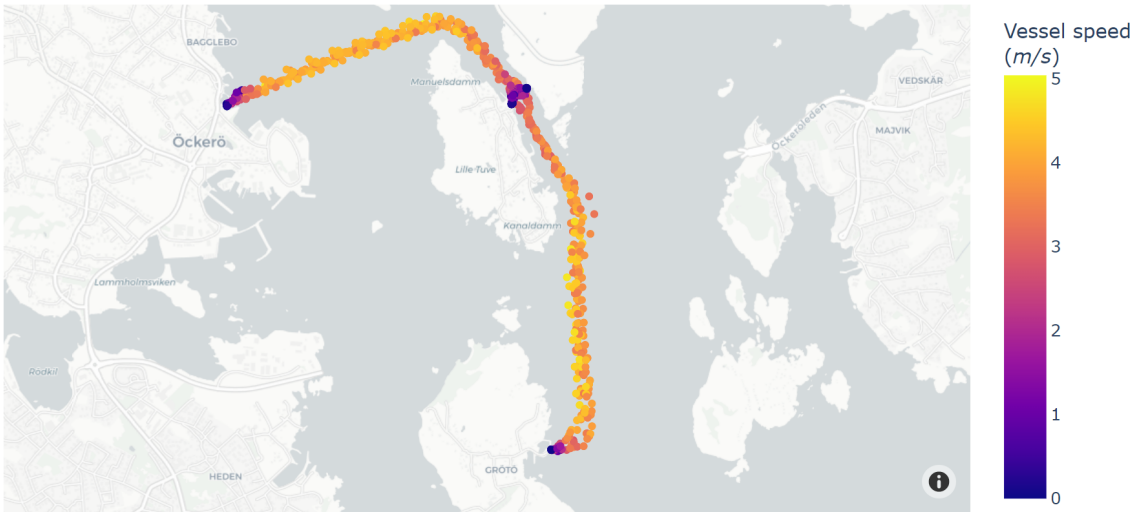


Figure 3.1: 1115 sample points of vessel speed plotted where they were captured.

The data was collected between the 17^{th} of September and 18^{th} of November 2020 was sampled using using a varying frequency and was subsequently interpolated to generate a data set with a sampling frequency of $\frac{1}{0.3}Hz$.

Necessary steps to process the data and make available for use will be taken. Furthermore Pearson's correlation coefficient r between different parameters in the data will be calculated to find relevant parameters and to remove irrelevant ones. This is done to reduce dimensionality of the final problem [35]. r describes the linear correlation between two parameters. If two features are very strongly correlated, $r \approx 1$, one of them is removed to reduce the number of dimensions in the data. The features are then said to be linearly separable. Features showing little correlation to other parameters are also removed [36].

3.2 Markov state enhanced reinforcement learning

The algorithm explored in this work is a reinforcement learning algorithm with a Markov state-enhanced environment. The algorithm consisted of three components: the agent, the environment, and the Markov component. The environment consists of the vessel model, together with a method for calculating the reward. The state and the reward are passed to the agent and the Markov component. The agent outputs its predicted optimal action, that is, a pedal position. The Markov component takes the state as an input and outputs its predicted Markov state, to both the environment and the agent. Figure 3.2 shows a schematic overview of the information flow of the algorithm. For further details of the implementation, see section 3.6.

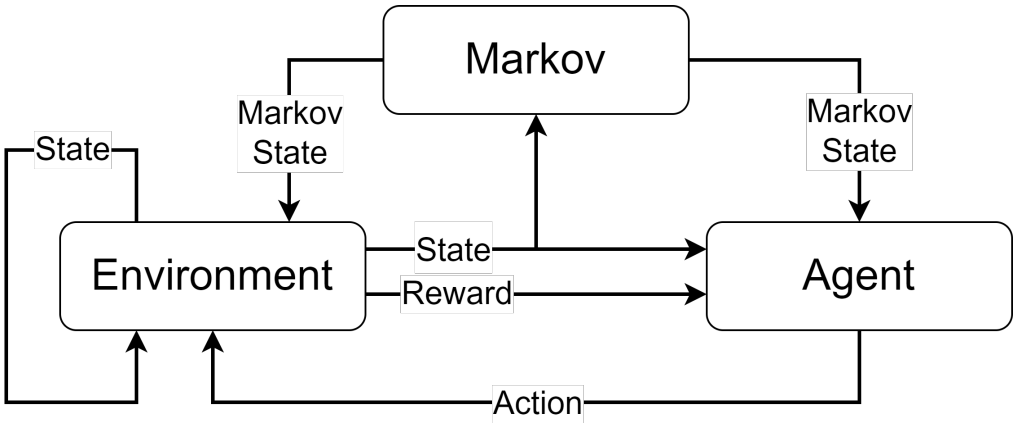


Figure 3.2: Schematic overview of the algorithm.

3.3 Latent variable modelling using HMMs

Given that we find the possibility of the existence of some latent variable(s) we attempt to model the missing information using a hidden Markov model. For modelling of the latent variable a hidden Markov model with Gaussian mixture emissions

(section 2.5) was used (HMM-GMM). This assumes that the data classified to each state $i \forall i \in \{0, \ldots, N\}$ can be modeled using equation (2.20).

As the emission distribution B is defined using a Gaussian mixture distribution (2.20) the parameters of the HMM amount to the combined parameters of equation (2.5) and equation (2.21) where the emission distribution is replaced by the parameters of the GMM. The complete parameterization of the HMM with Gaussian mixture emissions becomes

$$\lambda_{HMM-GMM} = \{\pi, A, w_{ik}, \boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik}\} \quad , \tag{3.1}$$

where $1 \leq i \leq N$, $1 \leq k \leq M$ and N and M are the number of Markov states and number of mixtures in the GMM respectively. The size of $\boldsymbol{\mu}_{i,j}$ and $\boldsymbol{\Sigma}_{i,j}$ is D and $D \times D$ where D are the number of features in the observed process $\boldsymbol{\mathcal{O}}$. The emission probabilities $b_i(y_t)$ can be rewritten using the GMM notation as

$$b_i(y_t) = \sum_k^M w_{ik} \mathcal{N}(y_t | \boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik}) = \sum_k^M w_{ik} b_{ik}(y_t) \quad . \tag{3.2}$$

3.3.1 Initial parameter estimation

Given a set of observations $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n\}$, the initial state probability π was initialised as random numbers from a uniform distribution such that $\sum_i \pi_i = 1$. The transition matrix A was initialised similarly such that $\sum_j a_{i,j} = 1$. The GMM parameters were initialised using the K-means clustering algorithm (2.4). The number of clusters was set to $K = MN$. Cluster centers were initialised using the K-means++ initialization method. As the initialization is stochastic it was ran for 100 iterations and the best initial cluster centers where chosen based on the minimum within cluster sum of squares value (equation (2.19)). K-means clustering was done using the `Scikit-learn` (version 1.0.2) in Python [37]. After running the K-means clustering algorithm the initial parameters for the Gaussian mixture model $\boldsymbol{\mu}_{ik}$ and $\boldsymbol{\Sigma}_{ik}$ were set to the mean and covariance matrix for cluster C_{ik} for $i = 1, \ldots, N$ and $k = 1, \ldots, M$ [21]. The initial component weights w_{ik} were set according to

$$w_{ik} = \frac{n_{ik}}{\sum_k n_{ik}} \quad , \tag{3.3}$$

where n_{ik} denotes the number of observations in cluster C_{ik}.

3.3.2 Model learning and parameter estimation

After initial parameters have been estimated comes the problem of re-estimating the model parameters such as to maximise $P(\boldsymbol{O}|\lambda_{HMM-GMM})$, this is again the task as solving problem 3 in section 2.3. To re-estimate model parameters the segmental K-means learning algorithm was used, allowing to train the model on several sequences of data.

For a set of L sequences, one pass of the algorithm is as follows:

For each sequence $\boldsymbol{O}_l$.

1. Find the optimal state sequence S using the Viterbi algorithm, given the observation sequence of length T^l.
2. Calculate $\xi_t(i,j)$

$$\xi_t(i,j) = \begin{cases} 1 & \text{if } s_t = i \text{ and } s_{t+1} = j \\ 0 & \text{otherwise} \end{cases} . \tag{3.4}$$

3. Calculate $\gamma_t(i)$

$$\begin{aligned} \gamma_t(i) &= \sum_j^N \xi_t(i,j) \quad , \\ \gamma_T(i) &= \sum_j^N \xi_{T-1}(i,j) \quad . \end{aligned} \tag{3.5}$$

4. Update model parameters according to equation.

In the pass above $\xi_t(i,j)$ is the estimated probability of being in state i at time t and state j at time $t+1$. $\gamma_t(i)$ is the probability of being in state i at time t. The model parameters are after one iteration updated according to the following equations

$$\pi_i = \frac{\sum_l^L \gamma_1(i)}{L} \quad , \tag{3.6a}$$

$$a_{ij} = \frac{\sum_l^L \sum_t^{T_l-1} \xi_t^l(i,j)}{\sum_l^L \sum_t^{T_l-1} \gamma_t^l(i)} \quad , \tag{3.6b}$$

$$w_{ik} = \frac{\sum_l^L \sum_t^{T_l} \gamma_t^l(i,k)}{\sum_l^L \sum_t^{T_l} \gamma_t^l(i)} \quad , \tag{3.6c}$$

$$\boldsymbol{\mu}_{ik} = \frac{\sum_l^L \sum_t^{T_l} \gamma_t^l(i,k) \boldsymbol{o}_t^l}{\sum_l^L \sum_t^{T_l} \gamma_t^l(i,k)} \quad , \tag{3.6d}$$

$$\boldsymbol{\Sigma}_{ik} = \frac{\sum_l^L \sum_t^{T_l} \gamma_t^l(i,k)(\boldsymbol{o}_t^l - \boldsymbol{\mu}_{ik})(\boldsymbol{o}_t^l - \boldsymbol{\mu}_{ik})^T}{\sum_l^L \sum_t^{T_l} \gamma_t^l(i,k)} \quad . \tag{3.6e}$$

In equation (3.6) above, $\gamma_t(i,k) = y_t(i)\frac{w_{ik}b_{ik}(o_t)}{b_i(o_t)}$. The segmental K-means algorithm ran until one of two termination criteria was met. The two criteria were either when 100 iterations had been reached or when model convergence was reached. For model convergence we first let $\tau - 1$ and τ denote two subsequent iterations of the algorithm. Convergence was thereafter defined when $\xi_t^{\tau-1}(i,j) = \xi_t^{\tau}(i,j)$ for $1 \leq t \leq T_l$ for each sequence $\boldsymbol{O}_l$ of length T_l, i.e. when no changes in assigned states occur in equation (3.4) [21]. The convergence criterion is equivalent to the optimal state sequence S^* returned by the Viterbi algorithm not changing after one iteration.

For final predictions of Markov state sequences using the model on a given observation sequence $\boldsymbol{O} = [\boldsymbol{o_1}, \boldsymbol{o_2}, \ldots, \boldsymbol{o_T}]$ the Viterbi algorithm was used to compute the optimal state sequence S^*.

3.3.3 Model topology selection

The topology of the hidden Markov model with Gaussian mixture emissions consists of the number of hidden states in the underlying Markov process N as well as the number of mixtures M in the Gaussian mixture models. For selecting a model topology a set of different candidate models were trained using real data. Candidate models were selected such that $2 \leq N \leq 3$ and $1 \leq M \leq 8$, resulting in a total of 14 candidate models. The models were evaluated and compared against each other using the Aikake information criterion and Bayesian information criterion, which can used to compute some relative model quality through which one can compare different models [38]. The equations for AIC and BIC are

$$AIC = -2\log L + 2p \quad , \tag{3.7}$$

$$BIC = -2\log L + p\log(T) \quad . \tag{3.8}$$

Where L is the likelihood function for the hidden Markov model [39]. p is the number of free parameters of the model. The information criteria punish the model for having a large amount of parameters p. For a hidden Markov model with Gaussian mixture emissions we have,

$$\begin{aligned} p &= N^2 + kN - 1 \quad , \quad \text{where} \\ k &= (M-1) + MD + \frac{D(D+1)}{2} \quad . \end{aligned} \tag{3.9}$$

Here N, M and D is the number of states, the number of components in the Gaussian mixture model and the feature dimension of the input sequence $\boldsymbol{O}$ respectively. The best model parameters selected among the candidate models the one which minimised the two information criteria

$$\begin{aligned} N^*, M^* &= \arg \min_{2 \leq N \leq 3, 1 \leq M \leq 8} AIC(N, M) \quad , \\ N^*, M^* &= \arg \min_{2 \leq N \leq 3, 1 \leq M \leq 8} BIC(N, M) \quad , \end{aligned} \tag{3.10}$$

where N^* denotes the optimal number of states and M^* denotes the optimal number of Gaussian mixture components.

3.4 Vessel modelling

For modelling of the data, three different types of models were considered candidates. These were a Gaussian process model, a recurrent artificial network type model with LSTM architecture, and nonlinear autoregressive exogenous model (NARX). The following subsections describe implementation of the different approaches. Below training iterations are called epochs. In the sections below, for some time t, v_t denotes vessel speed over ground, ρ_t denotes vessel pedal position and η_t denotes the momentary fuel consumption. All values are normalised between [0, 1]. Furthermore, the notation v_{t-1} and v_{t-2} is used to describe the speed over ground shifted backward. For other notations please refer to chapter 2.

3.4.1 Reference model

The reference model is based on differential equations which are not based on the data. It therefore does not model the vessel. Because the model is fully deterministic, it is used for evaluating the reinforcement learning agent.

The guiding equation (3.11) is based on traditional marine vessel dynamics, as discussed in chapter 2.1. The pedal position $\rho \in [0, 1]$ is taken as the forward force. The fuel rate is assumed to be $\eta = v^3$. Combining the equations yields the model as

$$\begin{cases} v_t &= v_{t-1} + 0.1\ (\rho_t - v_{t-1}^2) \\ \eta &= v_t^3 \end{cases} . \tag{3.11}$$

3.4.2 NARX

The NARX model was trained to, given the current and past speeds v_{t-n}, v_{t-n+1}, ..., v_t and a pedal positions ρ_{t-n}, ρ_{t-n+1}, ..., ρ_t, predict the next speed v_{t+1} and the next fuel rate η_{t+1}. n is the number of lags. The training was performed with `Scikit-learn`'s `LinearRegression` package [37], using shifted versions of $[v, \rho]$ as inputs and $[v, \eta]$ as targets. In deployment, a sequence of pedal positions ρ and a starting speed v_0 were used as input, and sequences of predicted speeds v and fuel rates η were outputted. In each time-step, the predicted speed v was fed back as input for the next time-step. The number of lags used were $n = 2$ and the degree for the polynomial regression was 4.

3.4.3 LSTM

A neural network was created by combining a LSTM architecture with several fully connected layers. The model was created in `Python 3.9` using the `PyTorch` package (version 1.10.2) [40, 41]. Fully connected layers are inserted between the input sequence and the LSTM cell and also between the LSTM cell and the output sequence. A schematic diagram of the final architecture can be found in figure 3.3.

The inputs and outputs at one time t are defined below as

$$x_t = \begin{bmatrix} v_{t-1} \\ \rho_t \end{bmatrix}, \quad y_t = \begin{bmatrix} v_t \\ \eta_t \end{bmatrix} . \tag{3.12}$$

As described previously there are intermediate sections of the model before and after the LSTM module. The first fully connected layer $FC1$ encodes the two dimensional input x_t to a higher dimension which is fed on as the input vector to the LSTM module. The output from the LSTM cell is decoded by the five fully connected layers. The complete specifications of the different layers are found in table 3.1. For a more complete description of the equations within the LSTM cell, please refer to section 2.7.

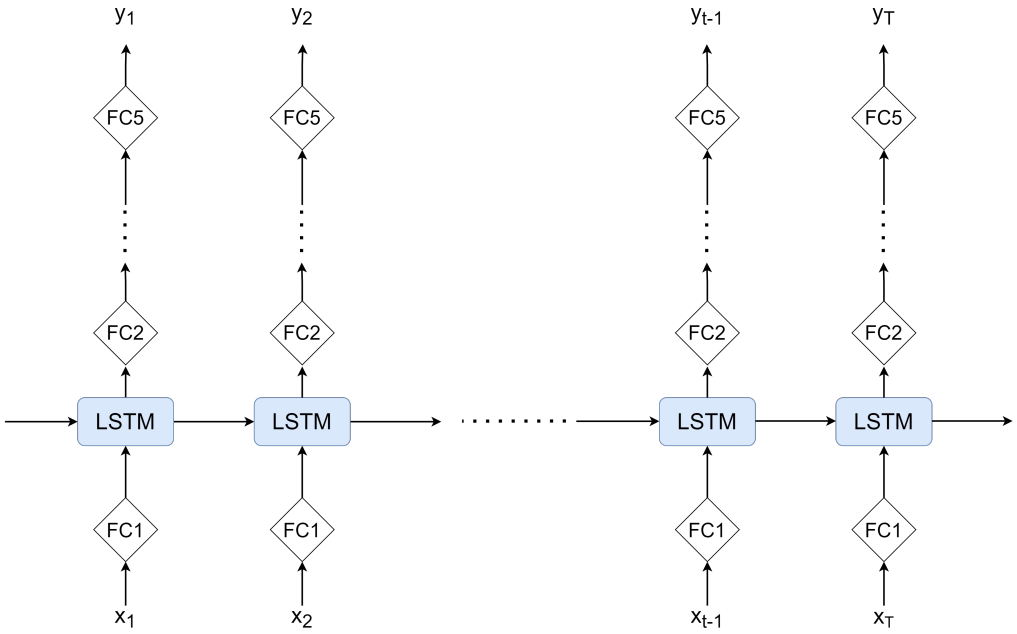


Figure 3.3: Schematic diagram over the LSTM network. Further architecture specifications are found in table 3.1.

Table 3.1: Specifications for all layers in the LSTM model, where FC is a fully connected layers and IN and OUT are the sizes of the input and output respectively. H is the number of features in the hidden state h_t of the LSTM. Here σ denotes the sigmoid activation function.

Layer	**Layer size**
IN	2
FC1	32
LSTM	$L \times H$
FC2	128
FC3	64
FC4	32
FC5	16
OUT	$\sigma(2)$

The parameters L and H are hyper-parameters which determine the specifications of the LSTM component. L is the number of LSTM layers in the model i.e. how many LSTM cells are stacked after each other. If $L > 1$ then the output from one LSTM cell in layer l will be the input to the next cell, $h_t^l = x_t^{(l+1)}$. H is the size of the hidden state vector h_t. Changing the hyper-parameters might influence model performance, therefore we want to find the optimal model given some set of different hyper-parameter pairs (L_i, H_j). To find the optimal model, several different set-ups with varying hyper-parameters were trained and their individual performances compared. We call this a hyper-parameter sweep.

For the sweep the data was split into a training and validation set of 80% and 20%

respectively. The models were trained using for 200 epochs using back-propagation through time and the Adam optimiser with a learning rate $\alpha = 10^{-3}$. The training loss function was the mean square error between the true and predicted observations. MSE was also calculated on the validation set and used as a model evaluation metric. After training, the optimal model hyper-parameters were selected which yielded low values of the mean square error. Other factors such as model size, might also be taken into account if deemed necessary. After selecting an optimal set of hyper-parameters we train the final model. The training setup was near to identical only now the model was trained until the minimum validation error was reached. The final model was also evaluated by calculating the MSE on the validation set.

3.4.4 Gaussian processes

A Gaussian process model was created with a RBF kernel as covariance function. The model was created in `Python` 3.9 using the `GPyTorch` (version 1.6.0) package [40, 42]. The model inputs x_t and outputs y_t where specified as follows

$$x_t = \begin{bmatrix} v_{t-2} \\ v_{t-1} \\ \rho_t \end{bmatrix}, \quad y_t = \begin{bmatrix} v_t \\ \eta_t \end{bmatrix} \quad . \tag{3.13}$$

The input to the model was v_{t-2}, v_{t-1} and ρ_t. The output was v_t and momentary fuel consumption η_t. As the model has multiple inputs and multiple outputs (MIMO), there will in practice be one RBF kernel for each output. Length-scale for both v_t and fuel consumption was initialised to 1 and noise parameters σ^2 was initialised to $\sigma^2 \approx 0.7$.

For training of the models the data was split into a training and validation set of 80% and 20% respectively. The model was trained for 400 epochs using the Adam optimiser (2.10) with a learning rate $\alpha = 0.2$. The loss function for training the parameters of the GP was set to the negative marginal log likelihood [29]. After training the model was evaluated on the validation set by calculating the mean squared error (MSE) between the true and observed predictions.

3.5 Final model evaluation, comparison and improvement

After the implementation and of the three different approaches, the different models will be evaluated. The models will all be evaluated by the calculated mean square error on each respective validation set. This will give an indication of how the different types of models compare in speed and fuel prediction.

The models will also be evaluated on their ability to predict speeds only by recursively feeding the predicted speed back into the model, we call this evaluation of the models performance on deployment. Given an observed sub-leg from the validation set the model is given the observed sequence of pedal positions and the initial velocity. The velocity is thereafter predicted and fed back into the model as the input recursively for each t, $1 \leq t \leq T$. Performance will be measured by

calculating the mean square error between the complete predicted speed sequence and the reference speed sequence from the same observation sequence as the pedal position, and comparing this figure between the different models. The number of steps in time which can be fed back to the model can be changed to include n number of data points $[v_{t-1}, v_{t-2}, \ldots, v_{t-n}]$. We call n the number of lags. The prediction algorithm is described schematically in figure 3.4.

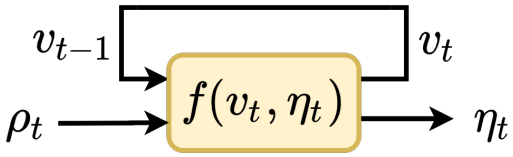


Figure 3.4: Simplified diagram of the recursive speed prediction using the LSTM model. Here the number of lags $n = 1$.

Aside from setting the initial speed v_1, recursive speed prediction can be described by the following equation

$$(v_t, \eta_t) = f(v_{t-1}, \rho_t) \quad , \tag{3.14}$$

where just as in figure 3.4, the number of lags is set to $n = 1$. The process of model evaluation will also take into account the properties of the different model types along with possible advantages and disadvantages that these properties might bring. After the two main evaluation procedures we will attempt to add the latent variables inferred using the hidden Markov model to improve model performance.

By using a hidden Markov model, we attempt to infer latent variables by assigning to each time-step t of a sequence a Markov state. The Markov state contains some information about variations in the observed process. By allowing a regression model to access this data, the intuition is that the model does not need to learn this behaviour in itself. After investigating the three different approaches of modelling the data, the model will be given access to the hidden Markov models a predicted state sequence in a attempt to further increase stability and model performance.

A PI controller was used to evaluate the physical relevance of the model. The hyper-parameters K_p and K_i was tuned such that they approximately balanced the output error in the relevant range [0, 1]. The method for finding the parameters were trial-and-error.

3.6 RL implementation

The RL algorithm was implemented with `Stable-Baselines3` [43] in Python. The agent was implemented with `Stable-Baselines3`'s DQN model. Below is a description of the hyper-parameters and design choices made during the implementation.

3.6.1 Agent

Table 3.2 shows the hyper-parameters used in this work. ϵ linearly decreased from $\epsilon_{initial}$ to ϵ_{final} during the first $\epsilon_{exploration\_fraction}$ of the training length. Table 3.3 shows the specifications of the DQN network. The network was trained with the Adam optimiser with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-9}$.

Table 3.2: Hyper-parameters for DQN. $\alpha = 10^{-5}$ is used for reference environment and $\alpha = 10^{-6}$ is used for Markov-enhanced LSTM.

Hyper-parameter	**Value**
Training length	1.5×10^6 time-steps
Learning rate α	10^{-5} or 10^{-6}
Replay buffer size	10^6
Learning starts after	5×10^4 time-steps
Batch size	32
Discount factor γ	0.99
Training frequency	4 time-steps
Target network update frequency	10^5 time-steps
$\epsilon_{initial}$	1.0
$\epsilon_{exploration\_fraction}$	0.1
ϵ_{final}	0.05
Number of lags for LSTM and HMM	8

Table 3.3: Specifications for all layers in the DQN model network. FC is a fully connected layers and IN and OUT are the sizes of the input and output respectively.

Layer	**Layer size**	**Activation**
IN	8 (7 without Markov)	-
FC1	64	ReLU
FC2	64	ReLU
OUT	3	-

The DQN agent received the following inputs

$$\begin{bmatrix} s_t \\ v_t \\ \rho_t \\ \eta_t \\ \text{traveled distance} \\ \text{traveled time} \\ \text{target distance} \\ \text{target time} \end{bmatrix}. \tag{3.15}$$

The Markov state s_t was omitted when running models that were not connected to the Markov model.

DQN has a discrete action space where each DQN output is connected to an action. In this work, the action was a change to the pedal position, denoted $\Delta\rho$. There were 3 available actions, namely

$$\Delta\rho \in \begin{Bmatrix} +0.1 \\ \pm 0 \\ -0.1 \end{Bmatrix} \quad . \tag{3.16}$$

The pedal position change $\Delta\rho$ was executed whilst keeping the pedal position in the range [0, 1], such as

$$\rho_t = \begin{cases} 0 & \text{if } \rho_{t-1} + \Delta\rho_t < 0 \\ 1 & \text{if } \rho_{t-1} + \Delta\rho_t > 1 \\ \rho_{t-1} + \Delta\rho_t & \text{otherwise} \end{cases} \quad . \tag{3.17}$$

3.6.2 Environment

The environment is a shell for the model, where actions and states are accumulated and connected to rewards. In the environment, the concept of episodes was introduced. In an episode, a target distance, a target time, and a maximum number of time-steps were defined. The episode ended if the agent reached the target distance or if the maximum number of time-steps were reached.

The environment models from section 3.4 were trained on the data to, based on the vessel's speed v_{t-1} and pedal position ρ_t, predict its speed v_t and fuel rate η_t. This is described by,

$$(v_t, \eta_t) = F_{env}\left(v_{t-1}, \rho_t\right) \quad . \tag{3.18}$$

The output (v_t, η_t) is called the *state*. The state is passed to the agent and to the Markov component.

The agent received a positive reward if it reached (or approached) the target distance, and a negative reward if it had not reached the target distance within the target time. During training, target distances and target times were randomised from a range, to enable the agent to learn different behaviours depending on the conditions. During evaluation, the target distance and the target time were fixed, to make it easier to compare progress. Table 3.4 shows the target parameters used in this work.

Table 3.4: Targets for reinforcement learning.

Target	**Range / value**
Training distance (m)	[50, 250]
Training time (s)	[40, 100]
Evaluation distance (m)	200
Evaluation time (s)	70

Rewards are distributed depending on the performance of the agent. Rewards can be both positive and negative (penalties) and the agent's target is to maximise

its reward over the course of the episode. There are two different types of rewards: *continuous*, distributed at each time-step, and *momentary*, distributed at the end of the episode. Table 3.5 shows the rewards used. The rewards are multiplied with a weight factor.

Table 3.5: Rewards for reinforcement learning.

Name	**Formula**	**Weight**	**Type**
Target reached	1	10	Momentary
Distance traveled (m)	$\frac{\text{[distance traveled current time-step]}}{\text{[Target distance]}}$	50	Continuous
Fuel consumed	$\frac{\text{[fuel rate current time-step]}}{\text{[Target distance]}}$	-50	Continuous
Time delayed (s)	$\begin{cases} 0 & \text{if [traveled time] } < \text{ [target time]} \\ 1 & \text{if [traveled time] } > \text{ [target time]} \end{cases}$	-1	Continuous

3.6.3 Markov component

The Markov state $S_t = [s_{t-n}, \ldots, s_t]$ was inferred at each time-step. The current Markov state s_t was fed to the agent and to the environment. The current time-step's Markov sequence S_t, including the lags, were fed to the environment agent. No Markov state history was stored in the reinforcement learning loop.

4

Results

Chapter 4 contains a complete description of all results obtained after implementing and carrying out the methodology as described in the previous chapter. The objective results are presented along with relevant interpretations and analyses. The results follow in a order similar to the methods section. First we describe our findings of the data, including pre-processing and relevant parameter selection. Later the results of both the latent variable model and the dynamical models are presented and afterwards also the results from the reinforcement learning implementation.

4.1 Data analysis

The data contained information about vessel speed over ground, acceleration, fuel consumption rate, engine speed, engine torque, pedal position, geographical position (latitude & longitude), vessel heading and vessel course. The vessel had two gears, forward and reverse, but the data contained no explicit information about which gear was active. Consequently, a high pedal position could imply a forward acceleration, backward acceleration, forward deceleration, or a backward deceleration. As such we have a partially observed system without observations about gear. In investigating the raw data the logs contained long sequences of missing data. Vessel location before and after these missing sequences was in most cases the same. If the sequences of missing data were longer than 3 time steps, the missing values were assumed to be sections were the vessel did not operate and therefore were removed. After removal of invalid data and outliers, the data consisted of 5,217,285 data points. Those points were classified into 683 so called legs, defined as intact sequences between Öckerö and Grötö, or vice versa. A typical leg can be seen in figure 4.1.

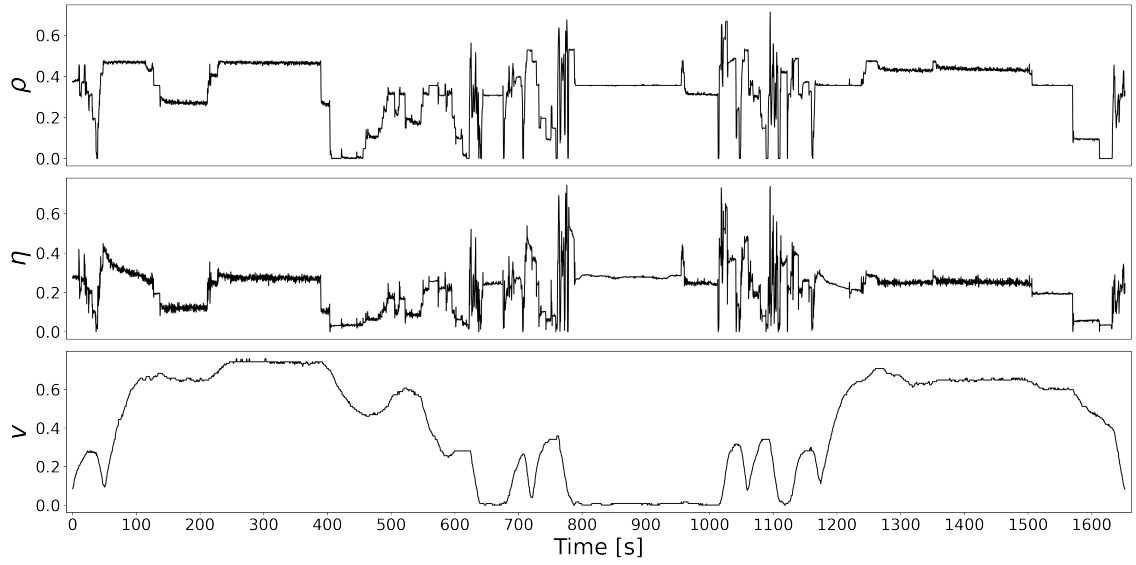


Figure 4.1: Leg going from Öckerö to Grötö with the three parameters: pedal position, fuel consumption and speed over ground.

The range of the parameters in use varied greatly, e.g. speed over ground had a range of approximately [0, 5], while pedal position had a range of [0, 100]. This was dealt with by min-max scaling all variables to the range [0, 1] according to

$$x_i' = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)} \quad , \tag{4.1}$$

where x is the set containing all samples for feature i and x' is the scaled feature.

All legs contain a dormant section, clearly visible in figure 4.1, in the middle while the ferry is docked at Framnäs or Kalvsund, when the vessel is pushing against the dock. Since this meant the vessel was standing still and not moving these sections of the legs were removed. To remove those parts of the data, parts of the large legs were classified as sub-leg. A sub-leg consists of either the trip Öckerö-Kalvsund or Kalvsund-Grötö (vice versa for the route in the opposite direction), where the difference between the direction the vessel was pointing at and the direction the vessel was heading at, was less than $\pi/4$ rad for at least 1500 consecutive time steps.

After processing the data as described above, the Pearson correlation coefficient between the different parameters was calculated. The upper triangular matrix of the calculated correlation matrix is presented below in figure 4.2.

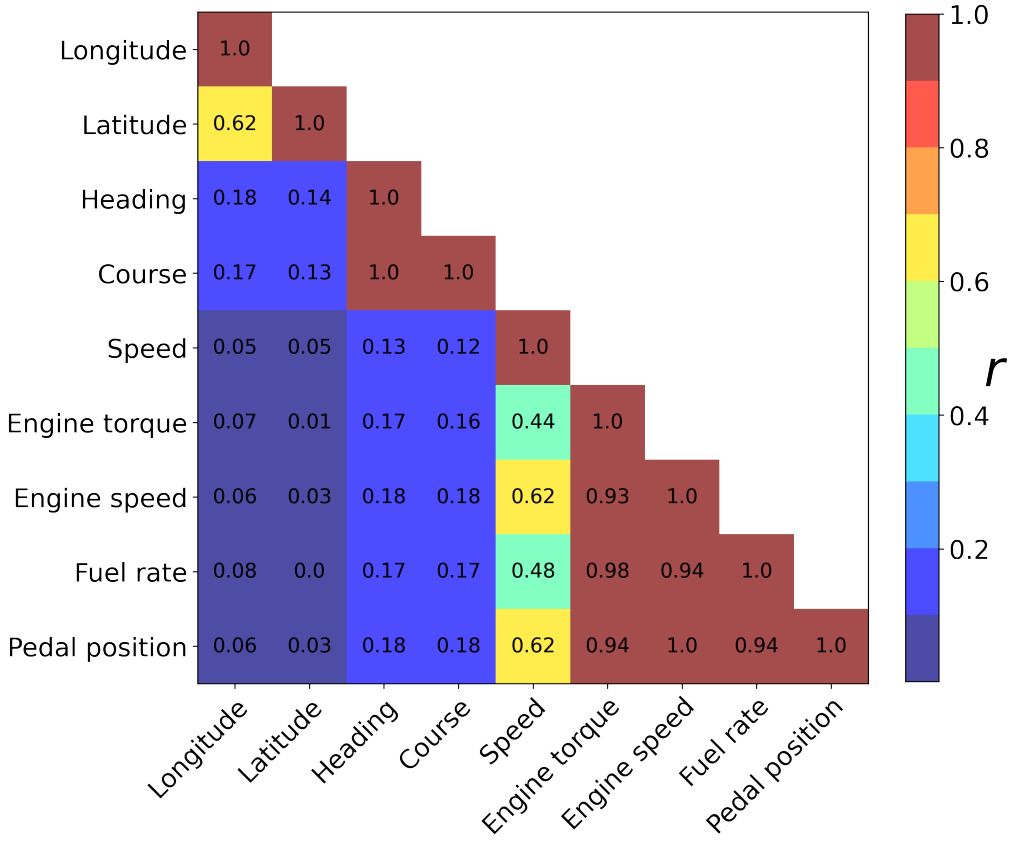
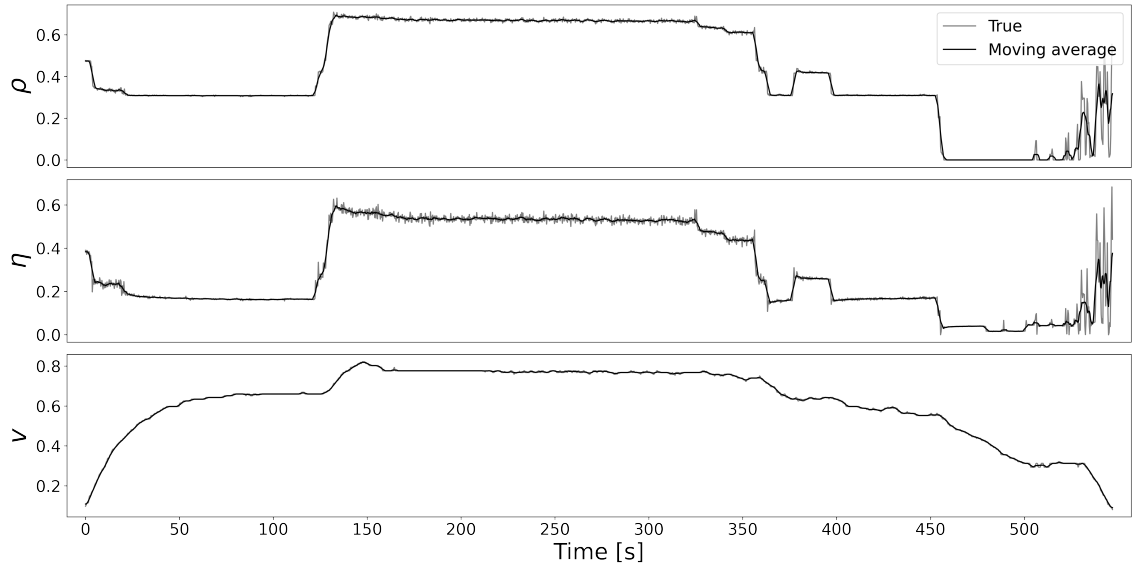


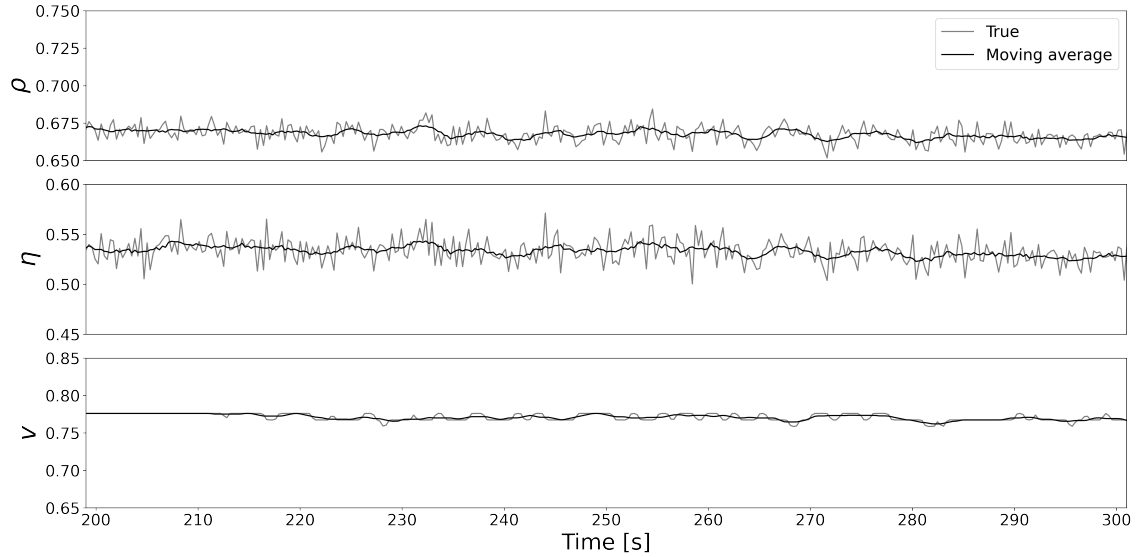
Figure 4.2: Bottom triangular of the correlation matrix between the different features after data-processing.

The correlation matrix show very high correlation between the different parameters pertaining related to the internal parameters of the vessel. As such all of these parameters were removed except pedal position and fuel consumption. Furthermore longitude and latitude were removed to make the model independent of geographic position. The vessel heading and course also had quite a low correlation between fuel rate and pedal position. Therefore these parameters were also removed. After feature removal the remaining features were fuel consumption rate, pedal position and speed over ground.

After feature selection the data was subjected to further analysis and preprocessing. In the sub-legs more constant sections of the trips exhibit noisy behaviour, possibly due to signal noise. A moving average filter using a window size of 10 was applied to the data. Figure 4.3 shows the effect of the moving average filter. The goal of applying the moving average filter was to remove unnecessary noise from the data.



(a) Whole sub-leg including true data and applied moving average.



(b) Same leg as in fig 4.3a but zoomed into the range $t = 200$ to $t = 300$.

Figure 4.3: Entire sub-leg and a zoomed perspective showing the original data and the subsequent smoothed data after applying the moving average filter. The parameters displayed are pedal position, fuel consumption and speed over ground.

At the end of each sub-leg a sequence was observed where there was a rapid deceleration and a high pedal position. This can be seen at the end of the different sub-legs in figures 4.1 and 4.3a. We know from before that we do not have access to which gear, forward och reverse, the vessel is currently operating on. A plausible interpretation for the observed combination of deceleration and high pedal position could as such be that the vessel is operating on reverse gear, spinning the propeller in reverse direction to slow down.

Lastly the lengths of the sub-legs were analysed. Most sequences have a length in the range between 1 and about 2400 while there appears to be some outliers in

leg length of lengths up to 4000. The mean length of all of the sub-legs was 1661 time-steps, or 554 seconds.

4.2 Inference of latent variables

There indeed seemed to exist unobservable features in for example the gear parameter of the vessel. As such the missing information was modelled using a hidden Markov model.

During model parameter estimation using the segmental K-means learning algorithm we are for each iteration calculating a set of emission distributions b_{ik} where $|1 \leq i \leq N, 1 \leq k \leq M$. This gives the requirement that the covariance matrices be symmetric and positive definite [44]. Due to numerical errors the covariance matrix might sometimes falsely be estimated to not be positive definite or contain invalid entries due to e.g. division by zero. These errors were found to be the product of poor initialization of the HMM parameters rather than some algorithmic error. As such the parameter estimation process was restarted on the occurrence of an invalid covariance matrix $\boldsymbol{\Sigma_{ik}}$.

On the problem of selecting model topology, i.e. selecting the model parameters N^* and M^* which minimise the two information criteria AIC and BIC, the models were trained on data separated into sub-legs. Each model was trained on a set of 50 sub-legs. All 14 models converged before 100 iterations were reached. As such all models reached convergence in the sense that no changes in the optimal state sequence occurred. The results from the parameter selection are presented in figure 4.4.

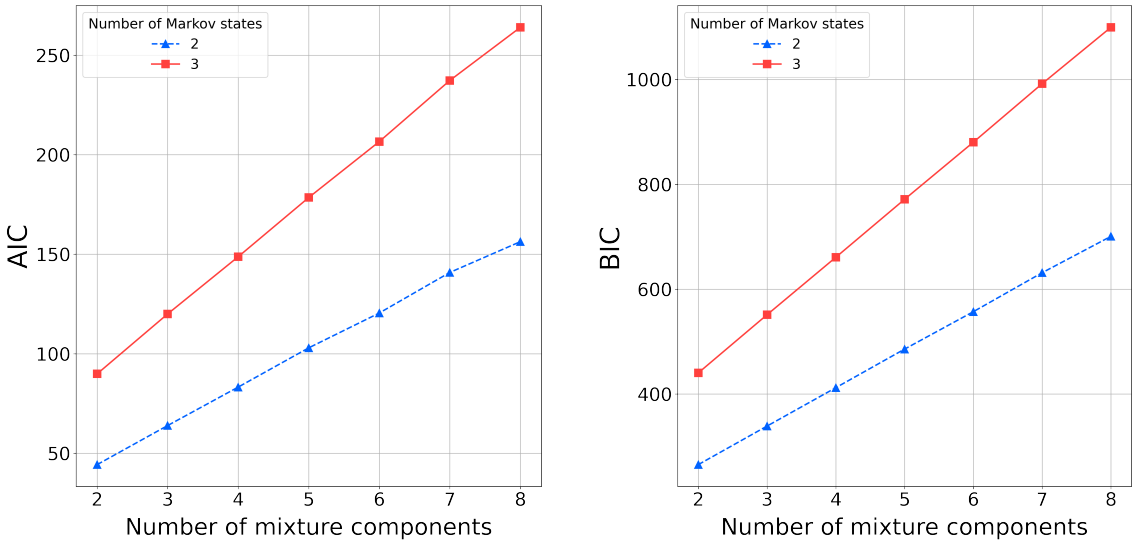


Figure 4.4: AIC and BIC for 14 different models after training. Model parameters N (number of states) and M (number of Gaussian mixture components) ranged from 2 to 3 and 2 to 8.

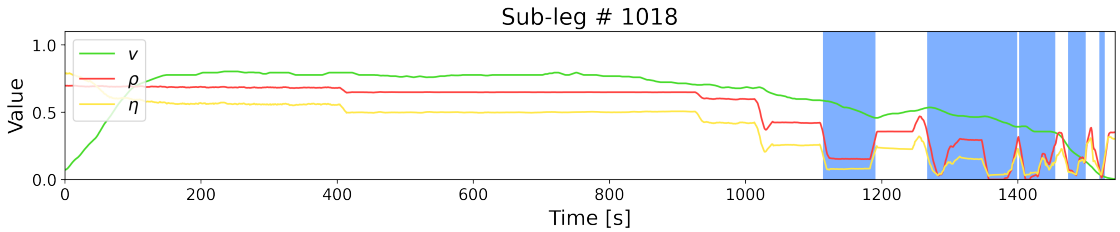
From figure 4.4 it is clear that the smallest model $(N, M) = (2, 2)$ is the one

which minimises both of the information criterion. The increasing model size and the subsequent penalization when calculating the information criteria leads to large increases in AIC and BIC. The model selection yields the results $(N^*, M^*) = (2, 2)$.

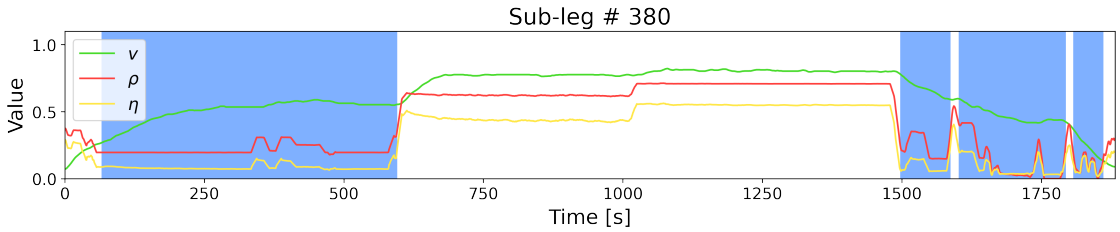
After model selection the final hidden Markov model was trained on the data. The data was pre-processed as previously described, scaling and and splitting into sub-legs. 250 sub-legs were randomly chosen for estimation of model parameters. The input features to the model were

$$\boldsymbol{o}_t = \begin{bmatrix} v_{t-1} \\ v_t \\ \rho_t \\ \eta_t \end{bmatrix} \tag{4.2}$$

where $\boldsymbol{o}_t$ is the observation at time t of a larger sequence $\boldsymbol{O}$. The final estimated parameters for the model can be found in Appendix A. Some examples of model predicted state sequences using the Viterbi algorithm can be found in figure 4.5.



(a) Sub-leg number 1018 including predicted Markov state.



(b) Sub-leg number 380 including predicted Markov state.

Figure 4.5: Predicted Markov State sequences for a selection of input sequences to demonstrate results in assigning Markov states. The two different Markov states are shown by background color, where blue denotes state 1 and white denotes the second state. The three parameters speed v, pedal position ρ and fuel consumption η are also shown.

The examples shown in figure 4.5 are examples of the two ways in which the model assigned latent variable states to observation sequences. The two examples are representations of two different ways in which the hidden Markov model often predicted state sequences. Studying the both examples we find indications of the model assigning the state $s_t = 1$ to parts of the sequence where the pedal position and the fuel consumption exhibit a unstable behaviour. The state $s_t = 2$ instead seems to be assigned to parts of the sequence where the parameters are more constant. One interpretation is the model manages to separate input sequences into

parts where the vessel is at a cruising speed and parts where the vessel is more manually maneuvered with either positive or negative acceleration and changes in pedal position. Important however is to note that this an subjective interpretation made from studying the predictions visually.

In addition to the examples in figure 4.5, there were predicted state sequences which were substantially harder to interpret. One such example is shown in figure 4.6.

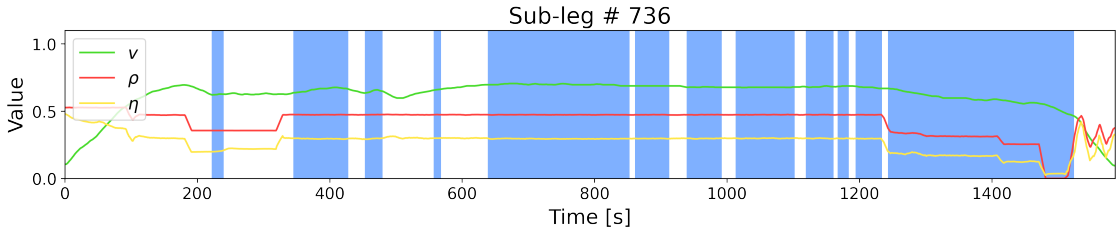


Figure 4.6: Sub-leg number 736 including predicted Markov state. The two different Markov states are shown by background color, where blue denotes state 1 and white denotes the second state. The three parameters speed v, pedal position ρ and fuel consumption η are also shown.

50 sub-legs were selected at random and state sequences predicted to obtain fractions on how many sequences had Markov state assignments with similar appearances to figure 4.5 and figure 4.6 respectively. 84% of the sequences had state sequences which resemble the ones obtained in figure 4.5. This figure was obtained by manually inspecting predictions. Furthermore the run time of the model was assessed. The model will predict states for 21 sequences per second on average.

4.3 Vessel modelling

Below follow descriptions of the results after implementation of the three different approaches to modeling the data described in section 3.4. The model type selection procedure is also described, presenting arguments on which selection was made and what the final selected model was. After model type selection results are presented from the further attempts of improvement and evaluation of the final model.

4.3.1 NARX

The appropriate number of lags and polynomial regression degrees were found using a sweep where MSE values were calculated on validation data. Table 4.1 shows the result from the sweep, lower values are desired. Up to polynomial degree 3, the results improved with increased polynomial degree. After that, numerical instability and over-fitting became apparent, especially for higher lags. For some combinations of high lags and degrees, the result diverged to infinity. This is a characteristic of high degree polynomial regression, where the function can take extreme values between observations. The best combination found was lag $n = 2$ and polynomial degree of 4.

		Lag				
		1	2	3	4	5
Degree	1	8.49	8.21	8.29	8.27	8.29
	2	5.12	5.02	4.99	4.91	4.77
	3	4.15	3.86	3.44	4.31	4.65
	4	4.27	**3.24**	1184.74	713.14	-
	5	5.43	22.30	141.34	-	-
	6	35.67	345.99	-	-	-

Table 4.1: MSE_{deploy} result from sweep over lag and polynomial degree. No result for runs where the numerical instability became too great. Note that the reported MSE values are not directly comparable to other MSE values in this work, because they were found using other settings.

The NARX model was trained on 95% of the data, just over 2 million data points. The model performed $MSE_{deploy} = 4.7535 \times 10^{-2}$ on average over 50 sub-leg runs. Figure 4.12a shows an example run.

4.3.2 LSTM

Model selection was done by running a hyper-parameter sweep. The parameters were varied taking values according to $L_i \in \boldsymbol{L} = \{1, 2, 3, 4\}$ and $H_j \in \boldsymbol{H} = \{8, 16, 32, 64, 128\}$, L_i and H_j being the number of LSTM cells stacked and the number of features in the hidden state h_t respectively. Creating and training a model using every combination of L_i and H_j resulted in training 20 different models.

During training of the LSTM networks the separated sub-legs were used as data. To create sequences of equal length the legs were zero-padded at the tails. To remove outlier legs of long lengths and also to avoid sequences with large amounts of padding a max leg-length of 2250 time steps was set. This cap removed 32 legs from the data set, a fraction of ≈ 0.027. The final data set retained 1135 legs which were split into a training and validation set, containing 80% and 20% of the total number of legs respectively. The results of he sweep are presented in 4.7.

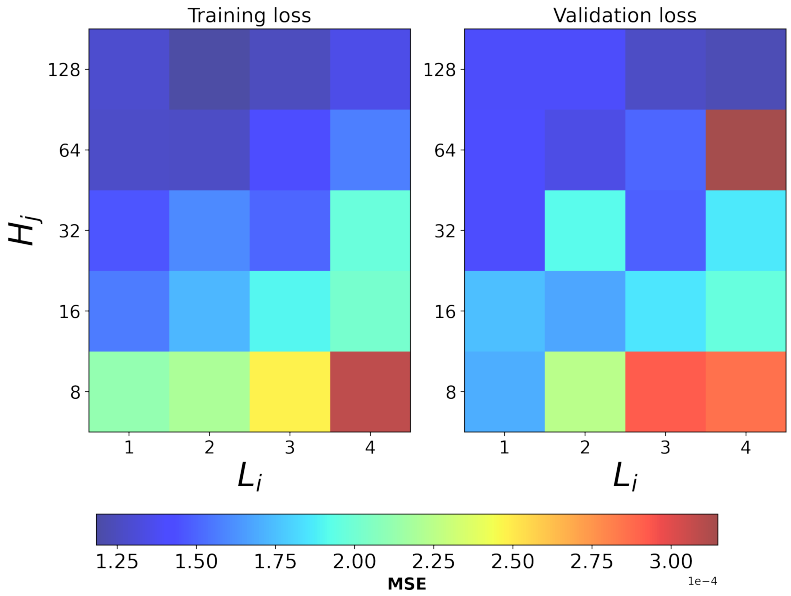
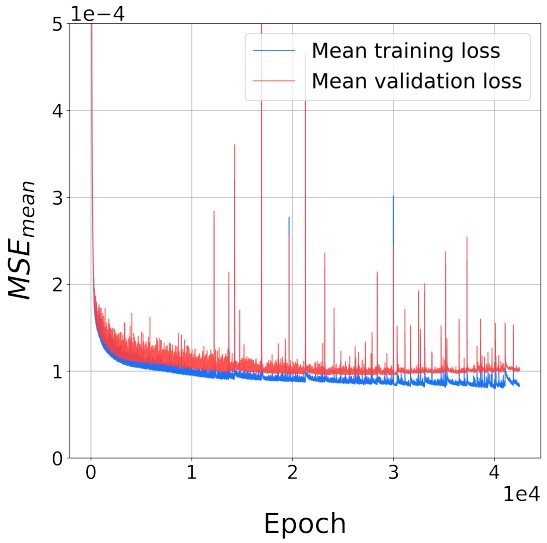


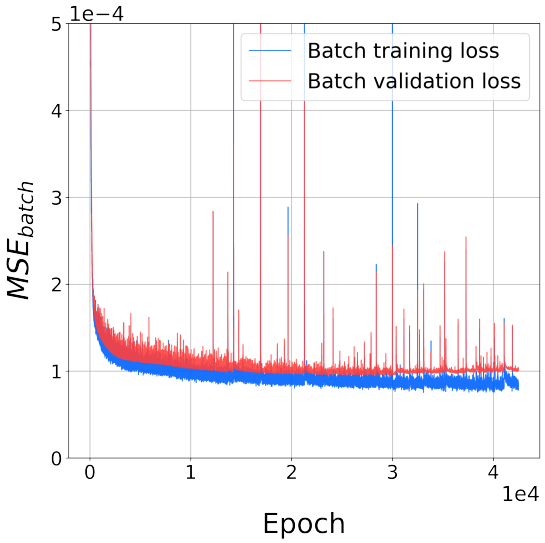
Figure 4.7: Mean square error on both training and validation data-set presented as results of the hyper-parameter sweep

The results of the sweep indicate that a model with a large L will results in a higher MSE, while for an increasing H the MSE decreased. When both L and H were set to large values in the given ranges, the results were not as conclusive. The differences in MSE between the models which yielded low scores were not large. Taking into account also the effect of a larger model, such as substantially longer prediction times and longer training time until model convergence, a smaller model was opted for. The final model architecture selected was $(L, H) = (1, 32)$ with $MSE_{val}(1, 32) = 1.1406 \times 10^{-4}$. The model was as previously described trained until the minimum validation error was reached. This resulted in training for 3×10^4 epochs. After 3×10^4 epochs the validation error leveled out and started increasing slightly, hence training was halted to prevent over-fitting. The model was trained on a `NVIDIA GeForce GTX 1650 with Max-Q Design` and training took approximately six hours. The training progress is visualised below in figure 4.8.

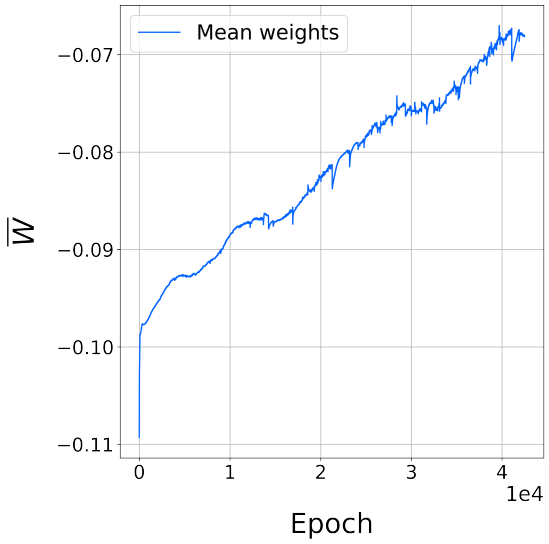
In figures 4.8a and 4.8b we observe the the leveling out of the validation loss. There are spikes in the validation loss, these are most likely due to initialisation of mini-batches. Figures 4.8c and 4.8d shows how the average of the weight and bias parameters changed over time. There are no indications of the model parameters having converged after the allotted amount of training epochs. Although the model did not converge during training, a minimum validation error before over-fitting seems to have been reached, which is why training was stopped. Evaluating the trained model on the validation set yielded av validation mean square error $MSE_{val} = 9.155 \times 10^{-5}$.



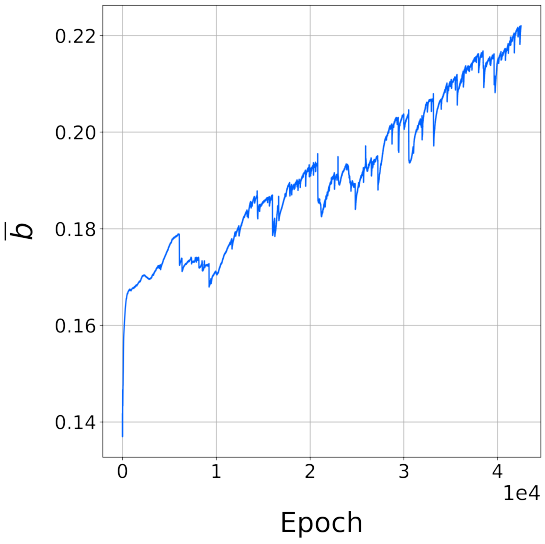
(a) Mean training loss.



(b) Batch training loss.



(c) Mean of all weights parameters $\overline{W}$.

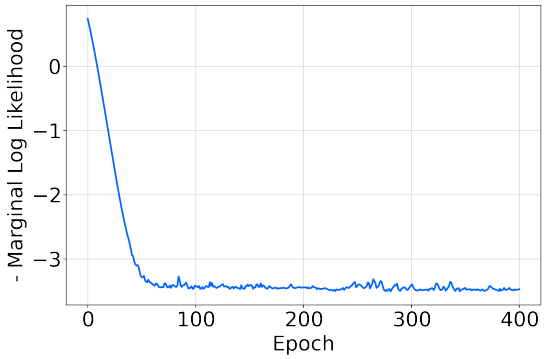


(d) Mean of all bias parameters $\overline{b}$.

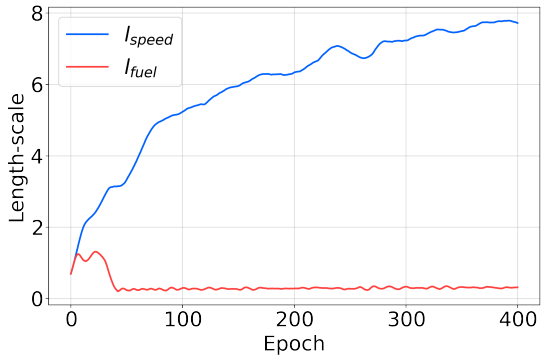
Figure 4.8: Training progress for the LSTM model with $(L, H) = (1, 32)$. Figures show training data for 42500 epochs and the increasing validation error. The spikes in training and validation MSE are believed to be cause of using mini-batch gradient descent.

4.3.3 Gaussian processes

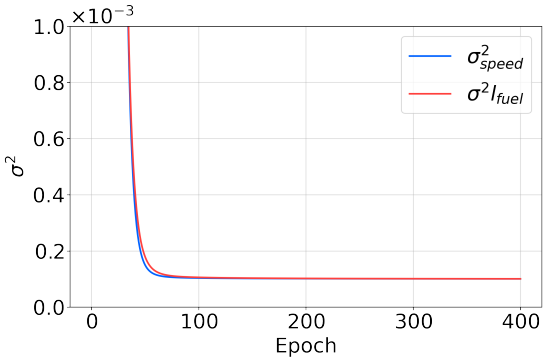
The Gaussian process model was trained on a set of 12500 data points, which were split into training and validation as described in section 3.4.4. Figure 4.9 shows how the loss function as well as length-scale and noise parameters changed during training.



(a) - Marginal log likelihood loss for GP training.



(b) GP model length-scales l during training.



(c) GP noise parameter σ^2 during training.

Figure 4.9: Training process for a Gaussian process model with an RBF kernel and gaussian noise.

Figure 4.9a clearly shows an initial rapid decrease in the negative marginal log likelihood after which training slows down. After 400 epochs there is no longer any apparent increase in model performance, indicating the training algorithm to have reached convergence. The final mean square error calculated on the validation set was $MSE_{val} = 2.385 \times 10^{-4}$. The model was trained on a `NVIDIA GeForce GTX 1650 with Max-Q Design` and training took approximately one hour.

4.4 Model evaluation, comparison and improvement

In an attempt to improve the LSTM model's performance, a combination of the hidden Markov model and the LSTM model was created. All training and validation

data was run through the hidden Markov model, inferring a latent hidden state for each time step t. The LSTM model was extended to include the Markov state as an input. The updated model inputs and outputs were

$$x_t = \begin{bmatrix} s_t \\ v_{t-1} \\ \rho_t \end{bmatrix}, \quad y_t = \begin{bmatrix} v_t \\ \eta_t \end{bmatrix}, \tag{4.3}$$

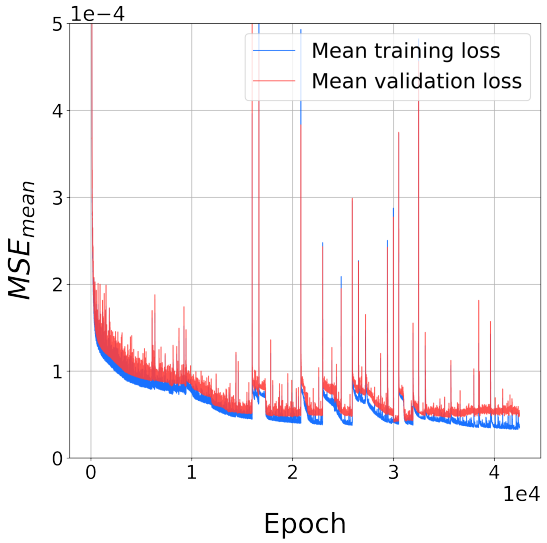
where s_t is the predicted Markov state at time t. The new LSTM model was created using the exact same parameter specifications as the LSTM model without inferred Markov states. All other parameters for training are found in section 3.4.3, no changes were made compared training of the regular LSTM model. The final modes was trained until the minimum validation error was reached, which resulted in training for 3×10^4 epochs. Similarly to the regular LSTM model without Markov state inputs, the validation error leveled out and started increasing slightly after 3×10^4 epochs, hence training was halted. The model was trained on a `NVIDIA GeForce GTX 1650 with Max-Q Design` and training took approximately six hours. Figure 4.10 shows the training procedure, including how the validation error increases late in training.

Just as for the LSTM model without inferred Markov states as an additional input (see section 4.3.2) the average weights $\overline{W}$ and bias $\overline{b}$ parameters do not converge. The final LSTM model was evaluated by calculating the MSE on the validation set was $MSE_{val} = 4.1821 \times 10^{-5}$. Compared to the regular LSTM model this was a decrease of $\approx 54.3\%$ in validation loss.

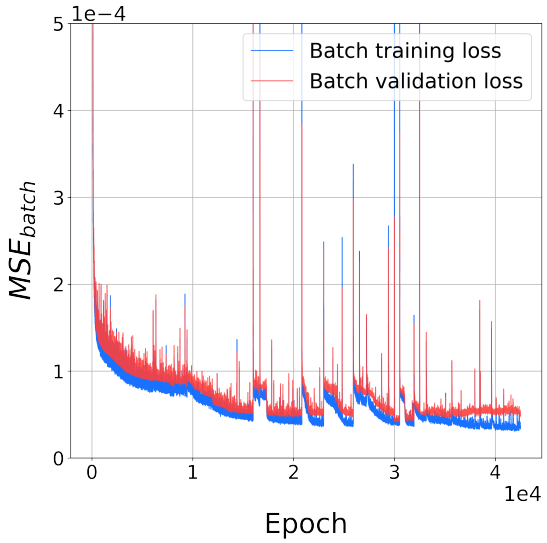
After the different model types were trained they were subject to two main ways of comparison. First models were compared by the obtained validation error from the training, these figures are presented in table 4.2. All the models were evaluated on the same amount of sub-legs. By evaluating all of the models using the MSE metric, they can objectively be compared to one another.

Table 4.2: Table showing the MSE_{val} for the three different approaches to modelling the dynamics of the vessel.

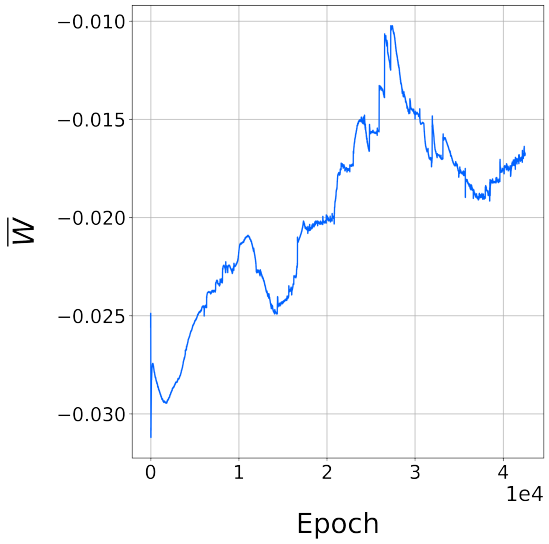
Model	$\mathbf{MSE_{val}}$
GP	2.3850×10^{-4}
LSTM	9.1550×10^{-5}
$\text{LSTM}_{\text{Markov}}$	4.1821×10^{-5}
NARX	3.8674×10^{-5}



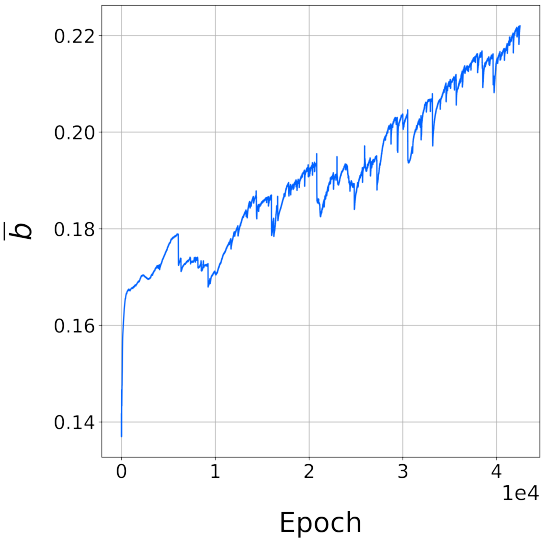
(a) Mean training loss.



(b) Batch training loss.



(c) Mean of all weight parameters $\overline{W}$.



(d) Mean of all bias parameters $\overline{b}$.

Figure 4.10: Training progress for the LSTM model with Markov states included as an input parameter. Figures show training data for 42500 epochs and the increasing validation error. The spikes in training and validation MSE are believed to be cause of using mini-batch gradient descent.

We find that the NARX model has the lowest mean square error of the three approaches. As such we find that in the training environment, the NARX model is the one with best performance.

Secondly we evaluate models by recursive speed prediction, the algorithm previously defined as deployment. The LSTM model with hidden Markov states as inputs was included in the evaluation. The evaluation of the LSTM models is dependant on the number of lags n, which for the LSTM model we define as length of the input sequence. By varying the sequence length we control the amount of *memory* the model has access to. We want to find the optimal length sequence since as sequence length increases, so does the number of computations and run-times. Each LSTM model was evaluated on a set of 50 sub-legs from the validation sets for lags such that $1 \leq n \leq 50$. The results are presented in figure 4.11.

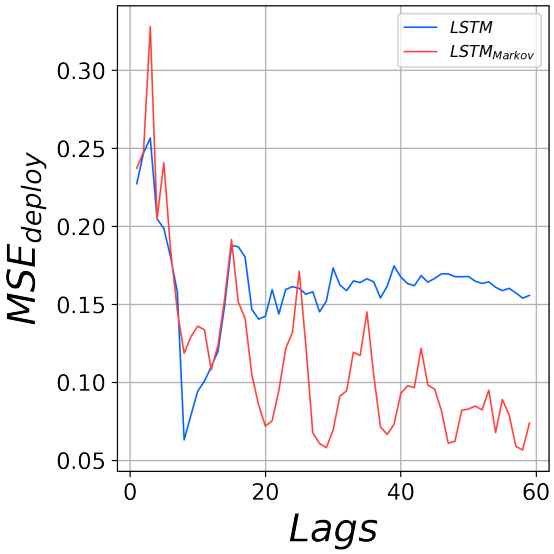


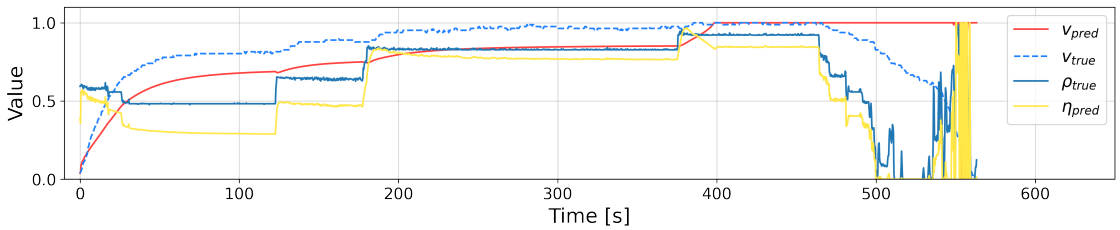
Figure 4.11: LSTM model evaluation for different lags n (sequence lengths). Graph shows both the results of the regular LSTM model and the LSTM model with Markov inputs.

For the LSTM and LSTM with Markov inputs number of lags n, was determined to be 58 and 8 respectively. After determining the optimal lags the different model types were evaluated on deployment. As before each model was evaluated on the set of 50 sub-legs. The comparison between the mean square error of the different models is presented in table 4.3.

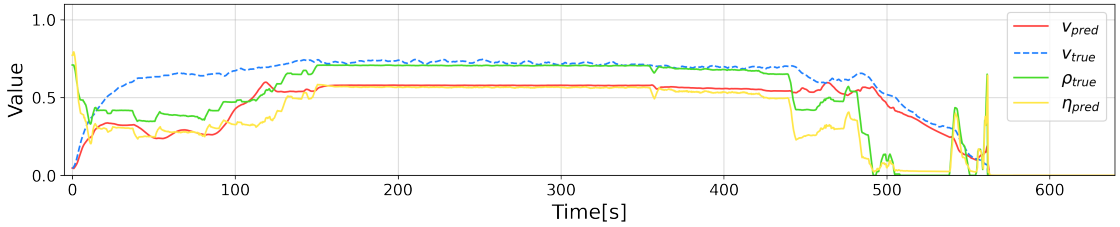
Table 4.3: Table showing the MSE_{deploy} for the three different approaches to modelling the dynamics of the vessel.

Model	$\mathbf{MSE_{deploy}}$
GP	2.0300×10^{-1}
LSTM	4.5760×10^{-2}
$\text{LSTM}_{\text{Markov}}$	6.3118×10^{-2}
NARX	4.7535×10^{-2}

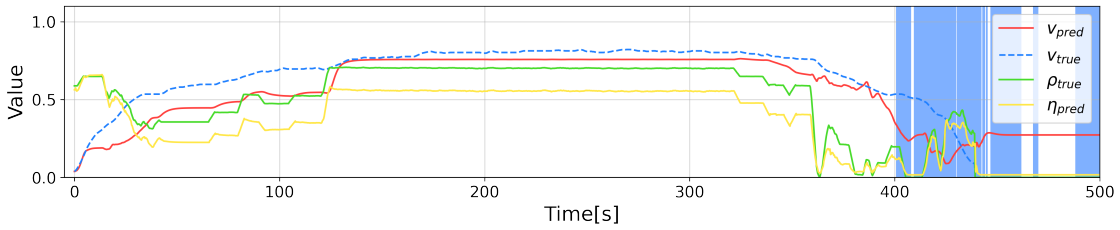
A quick review of table 4.3 quickly tells us that the Gaussian process model clearly has the highest error and is removed as a candidate for a good model. Among the other models the LSTM model has the lowest mean-square error. Compared to NARX and $\text{LSTM}_{\text{Markov}}$, MSE_{deploy} for the LSTM model was 3.7% and 27.5% lower respectively. Examples of recursive speed predictions using NARX, LSTM and $\text{LSTM}_{\text{Markov}}$ are presented in figure 4.12. For predictions the respective optimal lags previously determined for the two LSTM models were used.



(a) Recursive speed prediction using the NARX model.



(b) Recursive speed prediction using the LSTM model.



(c) Recursive speed prediction using the $\text{LSTM}_{\text{Markov}}$ model. Binary Markov states shown by background color.

Figure 4.12: Examples of recursive speed prediction (deployment) for the different models.

It is clear from figure 4.12 that the predicted speed does not match the true values. The general shape of the speed curve however is retained somewhat well in the prediction using the different models. From figures 4.12b and 4.12c we find the pattern of deceleration in combination with a high pedal position in the true data, something which was earlier assumed to describe breaking the ship with reversed gear. The predicted speed however does not decrease, but instead increases and resulting in in a terminal speed $v_T \geq 0$ for $\text{LSTM}_{\text{Markov}}$. Comparing the latter part of the prediction, where deceleration occurs, the NARX model does not reduce speed. Instead speed remains at the maximum. Both the LSTM models exhibit deceleration when pedal position is lowered.

4.5 PI controller

The PI controller was applied to the non-Markov LSTM model. The parameters were $K_p = 10$ and $K_i = 2$. 100 lags were fed to the LSTM model and the I term integrated over the previous 10 time-steps. The requested speed curve was a parabola which was 0 at the start and the end, and peaked on a speed of 0.3. The simulation ran for 1000 time-steps and all parameters were bound in the interval $[0, 1]$.

Figure 4.13 shows the outcome of the simulation. The red speed curve followed the blue, dashed requested speed curve closely. This was accomplished by rather drastic pedal movements, with spikes when the speed needed to increase and zero pedal when the speed needed to decrease. This behaviour is similar to what in control theory is known as bang-bang controlling [45].

The fact that the PI controller was able match the requested speed indicates that the LSTM model is physically relevant up to the investigated speed $v = 0.3$.

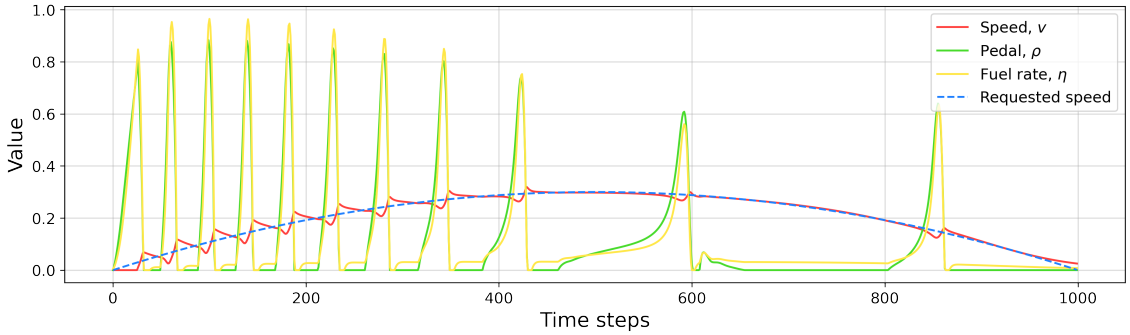


Figure 4.13: Performance of non-Markov LSTM model with PI controller.

4.6 RL training

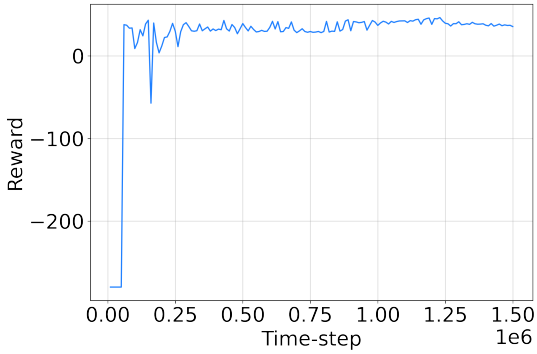
This chapter present the progress of the trainings. The performances are measured on validation rewards and training rewards.

4.6.1 Reference environment

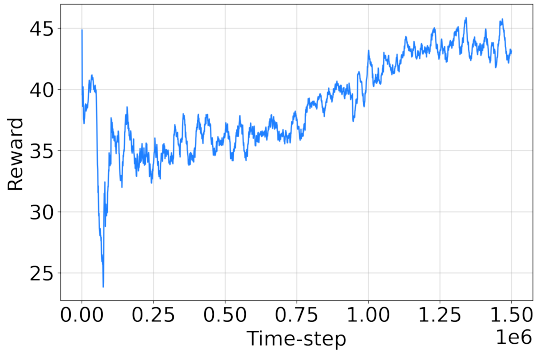
Figure 4.14 shows the development of some parameters and metrics during training in the reference environment. The validation reward quickly reaches positive values and is thereafter slowly increasing and stabilizing. The theoretical (but not in practice reachable) maximum reward is 60 and the validation reward stabilises around 43. The training reward initially dropped but then recovered and increased until approximate convergence. The gradients of the weights in the first layer of the DQN network decreases as training progresses. The weight gradients had occasional spikes which could be due to a high learning rate.

The rewards have a steady, positive trend for the entirety of the training. This shows that the DQN agent is capable of learning and that the used parameters are relevant.

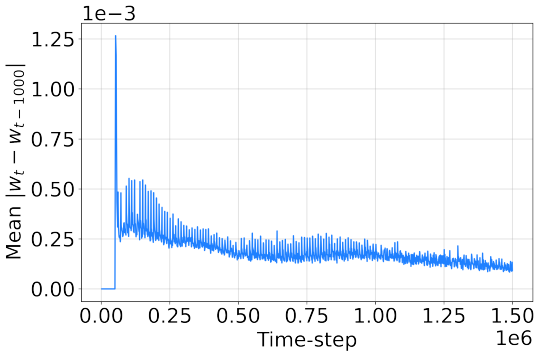
The training ran for 1.5×10^6 time-steps on an `Intel Core i5-8250U CPU`. The training took 41min.



(a) Rewards on validation runs (static targets).



(b) Rewards on training runs (randomised targets).



(c) Mean absolute difference between weights in the first layer of the DQN agent.

Figure 4.14: Training of the DQN agent in the reference environment. Training and validation runs are not from the same distributions.

4.6.2 Markov-enhanced LSTM environment

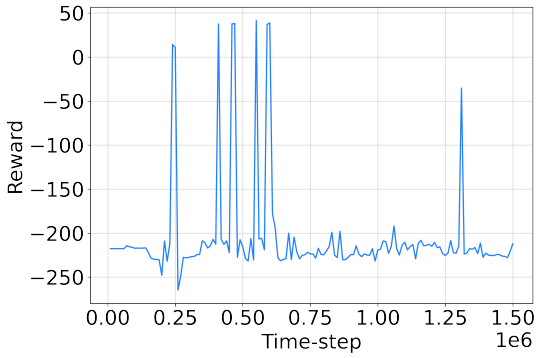
Figure 4.15 shows the development of some parameters and metrics during training in the Markov-enhanced LSTM environment. The validation reward is relatively stable around -200 with occasional spikes that reach positive values. The spikes are increasingly rare as training progresses. The training reward decreases during the first 2×10^5 time-steps. After this, the rewards show a positive but slow and noisy trend. The gradients of the weights in the first layer of the DQN network decreases as training progresses.

To reduce the risk of decreasing reward after a period of training, the learning rate was set to the low value of $\alpha = 10^{-6}$ and the rewards were multiplied by a factor 10^-3 (only during training, results presented here are re-scaled to the original scale). The efforts had limited impact, the rewards decreased regardless after a period of training.

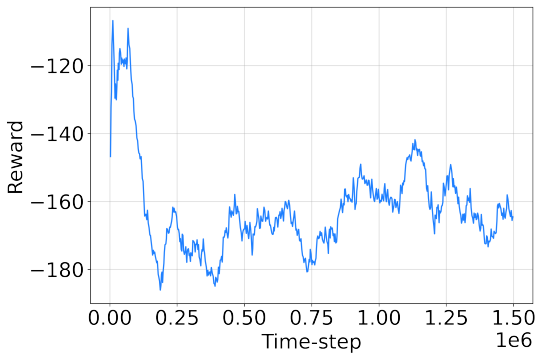
From figure 4.15, it is clear that the agent failed to converge the optimal policy.

Though positive rewards were possible during every evaluation run, the agent only occasionally reached them, and more training did not make those occasions more frequent.

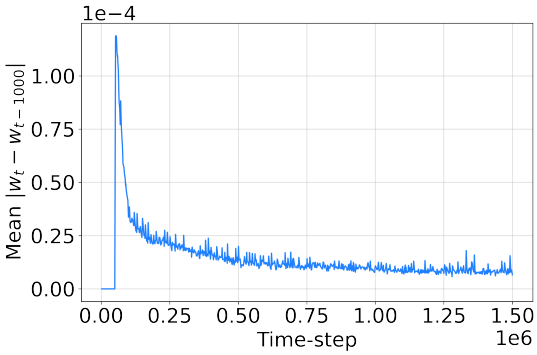
The training ran for 1.5×10^6 time-steps on an `Intel Core i5-8250U CPU`. The training took 3h 30min.



(a) Rewards on validation runs (static targets).



(b) Rewards on training runs (randomised targets).



(c) Mean absolute difference between weights in the first layer of the DQN agent.

Figure 4.15: Training of the DQN agent in the Markov-enhanced LSTM environment. Training and validation runs are not from the same distributions.

4.7 RL performance

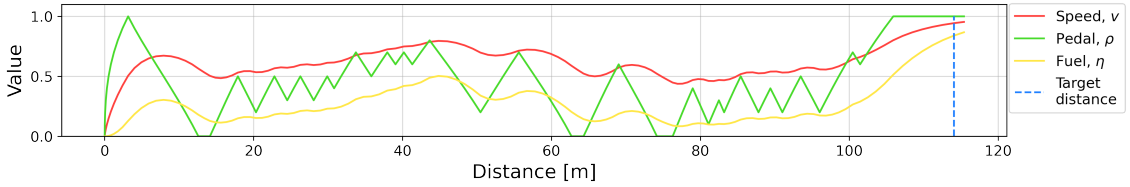
The performances of the agents in different environments were systematically tested during training. Here, sample runs are presented. During the sample runs are deterministic, meaning $\epsilon = 0$.

4.7.1 Reference environment

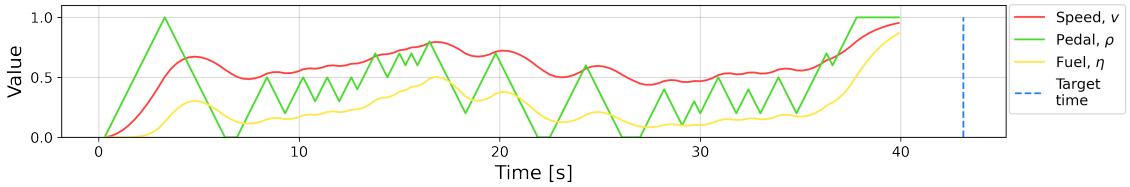
Figure 4.16 shows the development of some parameters during a run in the reference environment. The speed and the fuel rate is by definition correlated as $\rho = v^3$. The agent behaves such that it reaches the target distance before the target time, but

only slightly before so that it does not burn unnecessary fuel. The agent constantly changes its pedal position.

The behaviour of the agent is approximately what's intuitively the most efficient behaviour, and also the desired behaviour. This shows that the rewards are appropriate for reaching the desired behaviour (although results are dependent on the parameter values that the different environment models output).



(a) Distance on x axis.



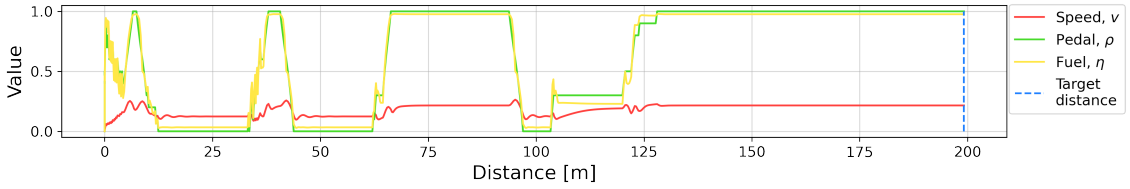
(b) Time on x axis.

Figure 4.16: Performance of DQN agent in reference environment. The two subfigures show the same simulation but with different x axes.

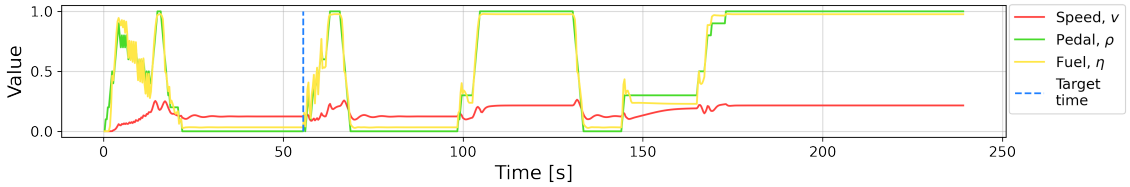
4.7.2 Markov-enhanced LSTM environment

Figure 4.17 shows the development of some parameters during a run in the Markov-enhanced LSTM environment. The pedal positions and fuel rates appear very closely correlated. They both shift quite violently between 0 and 1. The speed is much more stable and does not increase above $v = 0.3$ when the pedal position is $\rho = 1$ for a sustained period (at the end).

This behaviour shows that the LSTM model does not behave physically relevant to the actions proposed by the agent. This likely introduces unpredictability that could explain the lack of stable training.



(a) Distance on x axis.



(b) Time on x axis.

Figure 4.17: Performance of DQN agent in Markov-enhanced LSTM environment. The two subfigures show the same simulation but with different x axes.

5
Conclusion & future work

Data was segmented into sub-legs which were said to have useful information in that missing values and docked time periods were removed from the data. Upon analysis we found many parameters which were strongly correlated with fuel consumption and speed over ground, from which the conclusion could be made that predicting fuel and speed would be possible.

Regarding the inference of latent variables using hidden Markov models, reasonable interpretations of the predicted sequences can be made to some extent. The model seems to classify sequences into sections where the vessel speed is near constant and sections where the input varies and the vessel is either accelerating or slowing down. Analysis on random set of data sequences yielded that 84% of sequences had Markov state assignments suitable for this type of interpretation. There are also examples, e.g. figure 4.6, where predictions are not as easily interpreted. Table 4.2 shows a decrease in mean square error (MSE_{val}) between the first LSTM model and the LSTM model which included predicted Markov states as inputs. This indicates that the added Markov state contains some information about the data distribution, which the LSTM model subsequently can take advantage of. Furthermore the model was able to classify 21 sub-legs per second, which would not be possible were the sequences to be classified manually.

Upon final assessment of the models, we look to table 4.3 and figure 4.12. The regular LSTM model was the best model based on MSE_{deploy}, however both the NARX and the $\text{LSTM}_{\text{Markov}}$ models yielded results in the same order. The Gaussian process model had a substantially higher error and was deemed ill-suited for solving the problem of modelling the data. The predictions using the remaining models in figure 4.12, although only examples, indicate none of the models predict speed curves with exact precision. A general likeness of the shape of the curve however is retained. We can make some assessment of different model strengths: Both of the LSTM models show dynamic deceleration when the pedal position is lowered, while the NARX does not. They also show increase in speed when the true data suggests that the vessel is indeed breaking. This flaw may originate from the fact that no data exists about the gear. The ambiguity of the interpretation of the pedal position is not captured within the model dynamics. Selecting comprehensively the superior model is difficult. While the NARX model had a low error, the observed inability to predict deceleration possibly reduces its overall performance.

The reinforcement learning agent was able to learn, according to figure 4.14. However, when using the Markov-enhanced LSTM environment, training was unstable and did not lead to accurate results. According to figure 4.17, the model had difficulties reaching higher velocities than $v = 0.3$ when controlled by the agent. This is

unexpected since figure 4.12c shows stable speeds of $v = 0.5$. A possible explanation is that the agent's actions were unlike the true pedal position sequences observed in the training data, so that the model could not accurately predict useful Markov state sequences. Another explanation is that the Markov component self-reinforced errors in the model. Because Markov states are fed as inputs to the environment, a faulty Markov state prediction could lead to faulty parameter predictions in the subsequent time-step. This could lead to a vicious loop where the model gets stuck in a Markov state. Up to $v = 0.3$, the model is physically relevant, as demonstrated in figure 4.13 with the PI controller. The PI controlled pedal position has the appearance of a bang-bang controller with smooth transitions between high and low pedal positions. What's more, we know from the data analysis and modelling that we are working with a partially observable system. Applying reinforcement learning on partially observed environments have been known to be very demanding, with requirements on the number of samples being very large [46]. The inferred latent Markov state does not appear to reconstruct the information lost in data collection.

We have presented a reinforcement learning framework that we have showed is capable of solving the research question. However, the examined models are not accurate enough to make it useful in practice. Future work should focus on making the environment model more accurate. In addition, they should be predictable in the sense that a high pedal position should correlate with higher velocities.

A possible solution would be to have access to data on the vessel gear state, to reduce the ambiguity of the interpretation of the pedal position. If this information should be explicitly collected from the vessel or inferred from other variables, is a question open for others to solve. It could however be interesting to analyse whether a hidden Markov model could be created to infer such a parameter. Additional means of improving model performance could include to use a combination of a data-driven model and classical vessel dynamics, or to adjust the loss function when training the models. Since the data provided was collected during summer/autumn 2020, it could also be interesting to research the weather trends during this time to investigate any potential biases in the different models towards specific weather patterns.

More work could be put into finding an optimal agent model. The one used in this work, DQN, is able to find an strong policy, but the architecture was not extensively examined and is likely not the optimal one. Using agent models with continuous action outputs, such as the Soft Actor-Critic (SAC) [47], is an option, tough their main advantage is that they handle high dimensional action spaces well, and we only have one dimension. The role of the Markov model should also be further investigated. Perhaps, the Markov model is not suited well for running inside the reinforcement learning loop but should rather be used to label training data?

Our work focuses on the correlation between the acceleration pedal position, the speed and the fuel rate. An extended model could take more factors into account, such as direction of movement, influence of wind, or vessel load. In addition, a more complete vessel optimizing model would take the route of the vessel into account.

In conclusion, we have initialised a work, proposed a scaffold upon which models

can be implemented, and solved some of the initial questions. To make this into a product that is useful in real-life applications, additional work is needed. Specifically, two main areas are of particular interest. The first is to improve the model's ability to accurately model the behaviour of the vessel. The second is to extend the number of factors that the model takes into account.

Bibliography

[1] C. De Fontaubert and M. Viarros, "The potential of the blue economy: Increasing long-term benefits of the sustainable use of marine resources for small island developing states and coastal least developed countries," Jun 2017.

[2] "Maritime passenger statistics," Dec 2021.

[3] European Commission and Directorate-General for Maritime Affairs and Fisheries, A. Addamo, A. Calvo Santos, N. Carvalho, J. Guillén, D. Magagna, S. Neehus, A. Peralta Baptista, S. Quatrini, and Y. Schinasi Romeu, *The EU blue economy report 2021*. Publications Office, 2021.

[4] International Maritime Organization, "Fourth imo greenhouse gas study 2020," tech. rep., London, UK, 2021 [Online].

[5] A. Fratila (Adam), I. A. Gavril (Moldovan), S. C. Nita, and A. Hrebenciuc, "The importance of maritime transport for economic growth in the european union: A panel data analysis," *Sustainability*, vol. 13, no. 14, 2021.

[6] P. Orellano, J. Reynoso, and N. Quaranta, "Short-term exposure to sulphur dioxide SO_2 and all-cause and respiratory mortality: A systematic review and meta-analysis," *Environment International*, vol. 150, p. 106434, 2021.

[7] R. B. Hamanaka and G. M. Mutlu, "Particulate matter air pollution: Effects on the cardiovascular system," *Frontiers in Endocrinology*, vol. 9, 2018.

[8] J. E. Jonson, J. Borken-Kleefeld, D. Simpson, A. Nyíri, M. Posch, and C. Heyes, "Impact of excess NO_x emissions from diesel cars on air quality, public health and eutrophication in europe," *Environmental Research Letters*, vol. 12, p. 094017, sep 2017.

[9] Q. Liu and Y. Wu, *Supervised Learning*, pp. 3243–3245. Boston, MA: Springer US, 2012.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2015.

[11] D. Silver and A. Huang, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, p. 484–489, 2016.

[12] J. Jumper and R. Evans, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, p. 583–589, 2021.

[13] G. Batchelor, *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

[14] K. Rawson and E. Tupper, "10 - powering of ships: general principles," in *Basic Ship Theory (Fifth Edition)* (K. Rawson and E. Tupper, eds.), pp. 365–410, Oxford: Butterworth-Heinemann, fifth edition ed., 2001.

[15] M. Taz Ul Mulk and J. Falzarano, "Complete Six-Degrees-of-Freedom Nonlinear Ship Rolling Motion," *Journal of Offshore Mechanics and Arctic Engineering*, vol. 116, pp. 191–201, Nov 1994.

[16] R. J. Hyndman, *Moving Averages*, pp. 866–869. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[17] D. Jurafsky and J. Martin, *Speech and Language Processing*, ch. Appendix A: Hidden Markov Models. 3rd edition ed., Dec 2021.

[18] M. Awad and R. Khanna, *Hidden Markov Model*, pp. 81–104. Jan 2015.

[19] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[20] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[21] T. Bhowmik, J.-P. Oosten, and L. Schomaker, "Segmental k-means learning with mixture distribution for hmm based handwriting recognition," vol. 6744, pp. 432–439, Jun 2011.

[22] X. Jin and J. Han, *K-Means Clustering*, pp. 563–564. Boston, MA: Springer US, 2010.

[23] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, (USA), p. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[24] D. Reynolds, *Gaussian Mixture Models*, pp. 827–832. Boston, MA: Springer US, 2015.

[25] S. A. Marhon, C. J. F. Cameron, and S. C. Kremer, *Recurrent Neural Networks*, pp. 29–65. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec 1997.

[27] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[28] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM - a tutorial into long short-term memory recurrent neural networks," *CoRR*, vol. abs/1909.09586, 2019.

[29] J. Wang, "An intuitive tutorial to gaussian processes regression," 2020.

[30] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, Nov 2005.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[32] L.-J. Lin, "Reinforcement learning for robots using neural networks," *Technical report, DTIC Document*, 1993.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[34] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library, Princeton University Press, 2015.

[35] E. C. Blessie and E. Karthikeyan, "Sigmis: A feature selection algorithm using correlation based method," *Journal of Algorithms & Computational Technology*, vol. 6, no. 3, pp. 385–394, 2012.

[36] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proceedings, Twentieth International Conference on Machine Learning* (T. Fawcett and N. Mishra, eds.), Proceedings, Twentieth International Conference on Machine Learning, pp. 856–863, Dec. 2003. Proceedings, Twentieth International Conference on Machine Learning ; Conference date: 21-08-2003 Through 24-08-2003.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] H. Bozdogan, "Model selection and akaike's information criterion (aic): The general theory and its analytical extensions," *Psychometrika*, vol. 52, pp. 345–370, Feb 1987.

[39] W. Zucchini and I. Macdonald, *Hidden Markov Models for Time Series: An Introduction Using R*. Apr 2009.

[40] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[42] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," in *Advances in Neural Information Processing Systems*, 2018.

[43] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[44] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1. Wiley, Jan 1968.

[45] T. Glad and L. Ljung, *Control Theory*. Control Engineering, Taylor & Francis, 2000.

[46] Q. Liu, A. Chung, C. Szepesvári, and C. Jin, "When is partially observable reinforcement learning not scary?," 2022.

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

A

Hidden Markov model parameters

The final estimated parameters $\lambda_{HMM-GMM}$ of the hidden Markov model with Gaussian mixture emissions and optimal parameters N^* and M^* are

$$\pi^* = \begin{bmatrix} 0.020000000000 & 0.980000000000 \end{bmatrix} \tag{A.1a}$$

$$A^* = \begin{bmatrix} 0.994304737636 & 0.005695262364 \\ 0.004052898479 & 0.995947101521 \end{bmatrix} \tag{A.1b}$$

$$w^* = \begin{bmatrix} 0.151828367009 & 0.848171632991 \\ 0.990335007817 & 0.009664992183 \end{bmatrix} \tag{A.1c}$$

$$\boldsymbol{\mu}_1^* = \begin{bmatrix} 0.686157904401 & 0.474185856784 & 0.686196485658 & 0.292015267448 \\ 0.533019992043 & 0.218989920188 & 0.532459538430 & 0.111895982931 \end{bmatrix} \tag{A.1d}$$

$$\boldsymbol{\mu}_2^* = \begin{bmatrix} 0.646207951699 & 0.535275692354 & 0.646460922843 & 0.377306966358 \\ 0.755970800113 & 0.588032658397 & 0.756016138059 & 0.391249855380 \end{bmatrix} \tag{A.1e}$$

$$\boldsymbol{\Sigma}_{1,1}^* = \begin{bmatrix} 0.000200500569 & -0.000000431596 & 0.000199329646 & -0.000011070698 \\ -0.000000431596 & 0.000000408412 & -0.000000417564 & 0.000000402416 \\ 0.000199329646 & -0.000000417564 & 0.000198537610 & -0.000010841375 \\ -0.000011070698 & 0.000000402416 & -0.000010841375 & 0.000004617569 \end{bmatrix} \tag{A.1f}$$

$$\boldsymbol{\Sigma}_{1,2}^* = \begin{bmatrix} 0.012196416806 & 0.008880107851 & 0.012226371659 & 0.003105412404 \\ 0.008880107851 & 0.014733888869 & 0.008940802419 & 0.005937198377 \\ 0.012226371659 & 0.008940802419 & 0.012257777840 & 0.003130096466 \\ 0.003105412404 & 0.005937198377 & 0.003130096466 & 0.002613973827 \end{bmatrix} \tag{A.1g}$$

$$\boldsymbol{\Sigma}_{2,1}^* = \begin{bmatrix} 0.024658422441 & 0.008422824931 & 0.024623499852 & 0.003812614407 \\ 0.008422824931 & 0.009020147774 & 0.008481529729 & 0.008264714401 \\ 0.024623499852 & 0.008481529729 & 0.024593163736 & 0.003878384104 \\ 0.003812614407 & 0.008264714401 & 0.003878384104 & 0.008888215543 \end{bmatrix} \tag{A.1h}$$

$$\boldsymbol{\Sigma}^{*}_{2,2} = \begin{bmatrix} 0.000104594766 & -0.000000005371 & 0.000103875838 & -0.000004024312 \\ -0.000000005371 & 0.000000010008 & -0.000000005786 & 0.000000008615 \\ 0.000103875838 & -0.000000005786 & 0.000103520441 & -0.000003765701 \\ -0.000004024312 & 0.000000008615 & -0.000003765701 & 0.000007166349 \end{bmatrix} \tag{A.1i}$$