

THIS IS A PLACE HOLDER TITLE PAGE

# Ship motion prediction using IMU and wave images - a deep learning approach

LANCE DE WAELE

(empty page)

(second title page)

## Preface

**TODO.** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce ultricies, orci et scelerisque volutpat, nibh metus vestibulum ipsum, quis convallis ex orci ut massa. Curabitur felis dolor, tempor eu interdum nec, mattis quis felis. Vestibulum in nibh sit amet quam porta tristique. Fusce eu tortor tempus, tincidunt tortor hendrerit, sollicitudin elit. Cras a tempor urna. Vivamus vel malesuada purus. Sed feugiat egestas dolor, at feugiat lorem. Aliquam erat volutpat. Ut vel suscipit mi, quis vehicula lacus. Duis vitae libero semper, dignissim risus quis, vulputate augue. Praesent libero mauris, pretium id pharetra eget, malesuada et augue. Donec sed tincidunt augue. Nunc condimentum lectus non augue consequat, eget malesuada felis volutpat. Vestibulum ornare ultricies orci. Nulla sit amet dictum justo, non commodo arcu.

## Abstract

(English)

**TODO** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce ultricies, orci et scelerisque volutpat, nibh metus vestibulum ipsum, quis convallis ex orci ut massa. Curabitur felis dolor, tempor eu interdum nec, mattis quis felis. Vestibulum in nibh sit amet quam porta tristique. Fusce eu tortor tempus, tincidunt tortor hendrerit, sollicitudin elit. Cras a tempor urna. Vivamus vel malesuada purus. Sed feugiat egestas dolor, at feugiat lorem. Aliquam erat volutpat. Ut vel suscipit mi, quis vehicula lacus. Duis vitae libero semper, dignissim risus quis, vulputate augue. Praesent libero mauris, pretium id pharetra eget, malesuada et augue. Donec sed tincidunt augue. Nunc condimentum lectus non augue consequat, eget malesuada felis volutpat. Vestibulum ornare ultricies orci. Nulla sit amet dictum justo, non commodo arcu.

## Extended abstract

(Nederlands)

**TODO** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce ultricies, orci et scelerisque volutpat, nibh metus vestibulum ipsum, quis convallis ex orci ut massa. Curabitur felis dolor, tempor eu interdum nec, mattis quis felis. Vestibulum in nibh sit amet quam porta tristique. Fusce eu tortor tempus, tincidunt tortor hendrerit, sollicitudin elit. Cras a tempor urna. Vivamus vel malesuada purus. Sed feugiat egestas dolor, at feugiat lorem. Aliquam erat volutpat. Ut vel suscipit mi, quis vehicula lacus. Duis vitae libero semper, dignissim risus quis, vulputate augue. Praesent libero mauris, pretium id pharetra eget, malesuada et augue. Donec sed tincidunt augue. Nunc condimentum lectus non augue consequat, eget malesuada felis volutpat. Vestibulum ornare ultricies orci. Nulla sit amet dictum justo, non commodo arcu.

## Table of contents

Preface .....	3
Abstract .....	4
Extended abstract .....	5
Table of contents .....	6
List of Figures .....	9
List of Tables .....	10
List of Abbreviations .....	11
Introduction .....	12
Problem definition .....	12
Ship motion in six degrees of freedom .....	13
Objectives .....	14
Thesis outline .....	15
1 Background and literature review .....	16
1.1 Ocean waves and ship interactions .....	16
1.2 Ship motion prediction methods .....	16
1.2.1 Dynamic models .....	17
1.2.2 Deep learning .....	17
1.2.3 Hybrid models .....	18
1.3 Artificial neural networks .....	19
1.3.1 Gradient descent algorithms .....	19
1.3.2 Activation functions .....	20
1.3.3 Auto-encoders .....	21
1.3.4 RNN and LSTM networks .....	21
1.3.5 Convolutional Neural Networks .....	22
1.4 Tools .....	24
1.5 Expectations and criteria .....	24
1.6 Summary .....	26
2 Data collection and processing .....	27
2.1 Simulated data .....	27
2.1.1 Simulation parameters .....	28
2.1.2 Data augmentation .....	29
2.2 Real data .....	29

2.2.1	Sensors onboard the ASV.....	29
2.2.2	Sensory data challenges.....	30
2.3	Data pre-processing.....	30
2.3.1	Data cleaning.....	30
2.3.2	Data formatting.....	31
2.3.3	Data reduction.....	31
2.3.4	Data normalization.....	32
2.4	Data analysis.....	32
2.4.1	Statistical properties.....	33
2.4.2	Correlation and interactions.....	34
2.5	Data loading.....	34
2.5.1	Sequence creation.....	35
2.5.2	Train-test-validation split.....	35
2.5.3	DataLoader and DataSet.....	36
2.6	Summary.....	36
3	Model designs.....	37
3.1	Single-step stacked LSTM.....	37
3.2	Multi-step models.....	38
3.2.1	Encoder-Decoder LSTM.....	38
3.2.2	Sequential CNN.....	39
3.2.3	CNN LSTM single input.....	41
3.2.4	CNN LSTM dual input.....	42
3.3	Summary.....	43
4	Testing environment.....	44
4.1	Hyperparameters.....	44
4.2	SGD Adam optimizer.....	45
4.3	Performance metrics.....	45
4.3.1	Loss function.....	45
4.3.2	Inference time.....	46
4.4	Summary.....	48
5	Results.....	49
5.1	Hyperparameter optimization.....	49
5.1.1	Number of epochs.....	49
5.1.2	Learning rate.....	50



5.1.3	LSTM hidden size.....	50
5.1.4	Activation functions.....	51
5.2	Single-step model.....	52
5.3	Multi-step models.....	54
5.3.1	Encoder-Decoder LSTM.....	54
5.3.2	Sequential CNN.....	59
5.3.3	CNN LSTM single input.....	60
5.3.4	CNN LSTM dual input.....	61
5.4	Inference time.....	61
5.5	Augmented data.....	61
6	Discussion.....	62
6.1	Future work.....	62
7	References.....	63
8	Appendix.....	67

## List of Figures

Figure 1: Prototype of Autonomous Surface Vessel (ASV) with on-board computer and sensors (MarSur – Robotics & Autonomous Systems, n.d.).....	12
Figure 2: Six degrees of freedom in ship motion(de Masi et al., 2011).....	13
Figure 3: Heave (m) in function of time (s) (Ham et al., 2017).....	14
Figure 4: Neural network structure (left) and a linear neuron (right).....	19
Figure 5: Neurons in a recurrent neural network with additional feed-back connections .....	21
Figure 6: LSTM-cell components (left) and their mathematical notations (right).....	22
Figure 7: Convolution over grayscale image with 3x3 kernel.....	23
Figure 8: Max pooling with 2x2 kernel.....	23
Figure 9: Generated images of incoming waves (chronologically ordered left to right, top to bottom).....	28
Figure 10: Augmented images.....	29
Figure 11: ZED-mini IMU stereo camera (StereoLabs, Paris, France).....	30
Figure 12: Scatterplot of pitch and roll.....	31
Figure 13: Maximum range of rolling motion for ships .....	32
Figure 14: Simulated Pitch and Roll distributions.....	33
Figure 15: Correlation matrix for pitch and roll.....	34
Figure 16: Sequence creation with moving input and output sequence.....	35
Figure 17: First generation LSTM model architectures: single output (left), multi-output (right).....	38
Figure 18: LSTM encoder decoder architecture .....	39
Figure 19: sequential CNN neural network architecture.....	40
Figure 20: CNN LSTM single input architecture .....	41
Figure 21: CNN LSTM dual input architecture.....	43
Figure 22: two different scenarios illustrating the shortcoming of MSE.....	46
Figure 23: inference time over 10.000 prediction with GPU warm-up effect.....	47
Figure 24: training losses for different numbers of epochs.....	49
Figure 25: training loss for different learning rates.....	50
Figure 26: training loss for different LSTM hidden sizes.....	51
Figure 27: predicted (orange) vs. real (blue) values for pitch (top) and roll (bottom).....	53
Figure 28: MSE loss during training of the single-step models .....	54
Figure 29: encoder-decoder LSTM training and validation losses for different IO-ratios.....	55
Figure 30: Loss per frame for different IO-rates with reference line at 3°.....	56
Figure 31: pitch (top) and roll (bottom) predictions for 60/120 configuration.....	57
Figure 33: Prediction (red) vs. real (blue) for pitch (top) and roll (bottom).....	58
Figure 33: Loss per frame for 60/120 configuration.....	58
Figure 34: CNN LSTM single input training and validation loss for 10/60 and 10/120 IO-ratios .....	60
Figure 35: Predictions and LPF of linear CNN model.....	67
Figure 36: Prediction and LPF of single input CNN LSTM for 10/60 IO-ratio .....	68
Figure 37: Prediction and LPF of single input CNN LSTM for 10/120 IO-ratio .....	68

## List of Tables

Table 1: overview of different activation functions.....	21
Table 2: Kaminskyi's results for different models with worst (red) and best (green) model highlighted (Kaminskyi, 2019).....	25
Table 3: prediction error at different future time-steps of Kaminskyi's best model (Kaminskyi, 2019).....	25
Table 4: Statistical information for pitch and roll in simulated dataset .....	33
Table 5: newly introduced notations and their explanations.....	37
Table 6: parameter table for single-step models .....	37
Table 7: parameter table for encoder-decoder LSTM.....	39
Table 8: parameter table for sequential CNN model.....	41
Table 9: parameter table for CNN LSTM single input and dual output (red) .....	42
Table 10: average pitch and roll errors for different activation function configurations.....	52
Table 11: denormalized RMSE for single-output LSTM models in different configurations.....	52
Table 12: Average pitch and roll error per IO-ratio .....	56
Table 13: inference timing and number of trainable parameters for each model.....	61

## List of Abbreviations

AI:	Artificial Intelligence
ASV:	Autonomous Surface Vessel
IMU:	Inertial Measurement Unit
PoV:	Point of View
ANN:	Artificial Neural Network
GPU:	Graphics Processing Unit
LSTM:	Long Short-Term Memory
CNN:	Convolutional Neural Network
NN:	Neural Network
ReLU:	Rectified Linear Unit
PR:	Pitch and Roll
FC:	Fully Connected
RNN:	Recurrent Neural network
GPU:	Graphics Processing Unit
CPU:	Central Processing Unit
RAM:	Random Access Memory

## Introduction

In the last few years, the world has seen an exponential increase in technological advancements. This evolution brought a new influence of autonomous systems controlled by artificial intelligence (AI). Each of these systems being designed with their own unique characteristics, optimized for the desired task. Increasingly more of these systems are being deployed as a direct or indirect replacement for tasks humans could do, but also, for tasks too complex for humans to accomplish. And because these autonomous systems are optimized for specific jobs, they often can be more accurate and faster at it than humans.

Because autonomous systems can replace the position of a human, they are especially useful in military operations. They can take over the role of a human in dangerous environments such as an active warzone and can therefore eliminate the endangerment of someone's life. On the other hand, they can also be used as a complimentary asset, providing support and aid in logistics. An increasing amount of these autonomous assets such as drones, surface vessels, tanks and reconnaissance vehicles are being deployed around the world for various objectives. However, with this increasing amount of autonomous assets, there is need for proper communication between them, to allow them to work together and be aware of the state of each other when they need to interact (de Cubber, 2019).

*"Interoperability is the key that acts as the glue among the different units within the team, enabling efficient multi-robot cooperation."* (MarSur – Robotics & Autonomous Systems, n.d.)



*Figure 1: Prototype of Autonomous Surface Vessel (ASV) with on-board computer and sensors (MarSur – Robotics & Autonomous Systems, n.d.)*

## Problem definition

The Robotics & Autonomous Systems lab of the Belgian Royal Military Academy is currently working on two autonomous vehicles in two separate projects named MarSur and MarLand. Project MarSur is developing framework for autonomous systems to easily interact with each other. In short, they are developing a heterogeneous interoperability and collaboration framework which is seamlessly interoperable with existing infrastructure. This framework will, among others, be used to facilitate the communication and interaction of autonomous surface vessels (ASV) (Figure 1) and other unmanned aerial systems such as drones (MarSur – Robotics & Autonomous Systems, n.d.). Project MarLand proposes research in one of these interactions, namely vertical take-off and landing (VTOL). In short, the MarLand project proposes to provide a proof-of-

concept solution and practical implementation for a helicopter-type drone with the capability to land autonomously on the Belgian Navy vessels (*MarLand – Robotics & Autonomous Systems*, n.d.). The capability for these unmanned aerial drones to automatically take off and land on vessels in all kinds of environmental conditions remains a bottleneck for widespread deployment. Landing a relatively small aerial vehicle - that is inherently very receptive to wind gusts - on the pitching and rolling deck of a moving ship is a very difficult control problem that requires the consideration of the kinematics and dynamics of both the unmanned aerial vehicle and the ship. For a smooth landing to be possible, the target vessel must be capable to determine its state in a three-dimensional space and predict its movement in the ocean. This way, the drone can anticipate the movement off the vessel and avoid collision. In addition, the predictions can be used to find a window of landing opportunity in which the vessel remains in a relatively stable state. In conclusion, this thesis proposes to provide a state-of-the-art solution for ship motion prediction to serve as a landing guidance for drones.

### Ship motion in six degrees of freedom

The motion of a ship or any rigid object in a three-dimensional space can be described in six degrees of freedom. These six degrees can further be divided into two categories: translational and rotational movement. Where translational movement is movement along one of the three axes in a three-dimensional space, rotational movement is the rotation of an object around these same three axes. These three reference axes run through the center of mass of the ship and are oriented as follows:

- Vertical Z-axis runs vertically through the vessel
- Transverse Y-axis runs horizontally across the vessel
- Longitudinal X-axis runs horizontally through the length of the ship

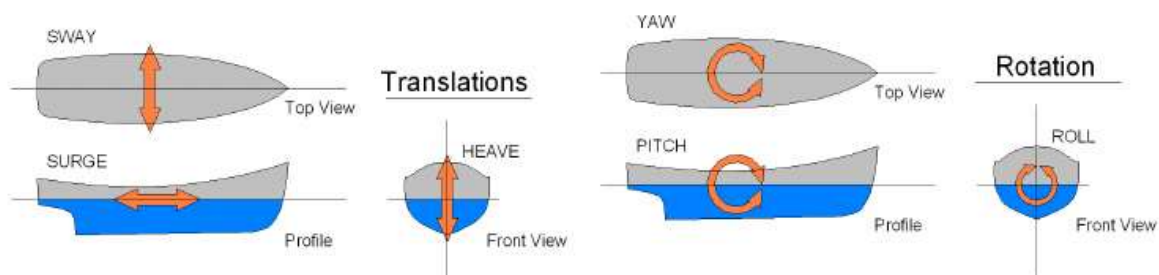


Figure 2: Six degrees of freedom in ship motion (de Masi et al., 2011)

Each type of motion, translation or rotation, among each of the three axes has a different impact on the movement of the vessel (figure 3). The translational movements are expressed in linear units such as meters and are named as followed:

- **Sway:** side to side movement along the transverse Y-axis
- **Surge:** forward and backwards movement along the longitudinal X-axis
- **Heave:** upward and downward movement along the vertical Z-axis

The rotational movements are expressed in angular units and are named as followed:

- **Yaw:** rotational movement around the vertical Z-axis
- **Pitch:** rotational movement around the transverse Y-axis
- **Roll:** rotational movement around the longitudinal X-axis

To predict the motion of a ship, one must differentiate between these different motions. Together they form the complete three-dimensional orientation of a ship. But not all of them need to be predicted. Surge and yaw are controlled by the ASV's autonomous systems and are respectively controlled by the amount of thrust and the rudder position – steering the ship. Surge and yaw will also not change very drastically during the landing or take-off of a drone since this behavior would directly impede our main goal of providing a smooth landing. On the other hand, the sway of a ship, also referred to as drift, is primarily caused by sideways winds or currents in the water and will have minor impact on the stability of the ship whenever the drone needs to take-off or land. If the drone aims for a GPS-tracker present in the ASV, it will follow the vessels movement no matter the sway.

This leaves three main factors remaining which have the most impact on the stability of the vessel: roll, pitch and heave. These three movements have one thing in common, they are all directly caused by the waves in the ocean and are very hard to control. Different methods exist to dampen these movements and keep the vessel as stable as possible such as bilge keels and antiroll tanks. However, most of them are either infeasible or ineffective or do not provide the required stabilization on smaller vessels (Perez & Blanke, 2017). In this case, predicting these movements instead of trying to dampen them, can be an alternative solution. Although it should be noted that using them together, will most likely yield the best performance. Pitch, roll and heave can be divided in two categories based on the effect they have on the landing and take-off of the drone. Pitch and roll are responsible for the stability of the landing surface and heave is responsible for the impact on the drone when landing.

To provide a stable landing zone for the drone, the pitch and roll of the vessel should remain constant and as close to zero as possible. Depending on the characteristics of the drone, the model should be able to analyze its prediction sequence and find a window where the desired circumstances to land/take-off are met. To determine this window of landing/take-off opportunity, different parameters need to be defined such as the maximum difference in consequent prediction values, the length of the window and the interval in which all predicted values should lie. For example, the roll and pitch values should all remain in a  $[-3^\circ, 3^\circ]$  interval, the stable window duration must be at least five seconds and there should be no difference larger than two degrees between consecutive predicted values. Within the objectives chapter, these criteria are discussed in more detail.

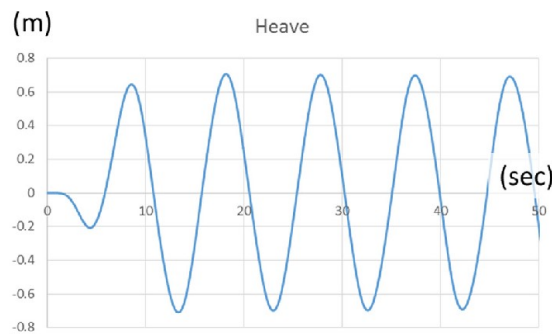


Figure 3: Heave (m) in function of time (s) (Ham et al., 2017)

To minimize the impact on the drone when landing, the heave needs to be constant or decreasing. This means that the vessel is either not moving up or down, or it is slowly moving downwards following the motion of the descending drone. In regular waves, the heave of a vessel follows a wave-like function, alternating between upwards and downwards motion (figure 4). In this case, the window of take-off/landing opportunity can be defined as the points where the vessel transitions from upwards to downwards motion, or vice versa, and thus has an acceleration of zero.

## Objectives

The goal of this thesis is to research and develop a method to accurately predict the motion of a surface vessel using the data captured from onboard sensors. To achieve this, deep learning concepts will be applied to design a neural network architecture capable of achieving this task in an efficient way. The prediction models can use a sequence of values describing the state of the vessel such as pitch and roll together with images of incoming waves taken from a stabilized camera pointing to the front of the vessel. As an output, the model should provide a new sequence of predicted pitch, roll and heave values for every predicted time step. The duration for these predicted sequences should be at minimum thirty seconds at 1Hz to provide the drone with ample time to complete a take-off or landing procedure. Bigger drones will need more time for this procedure so the predicted sequence duration should be maximized within the model's capabilities.

Since the model will be deployed in real-time scenario's and will be making predictions in real-time based on the continuous data stream of onboard sensors, it should be lightweight and not require substantial amounts of computational power. This means that the inference time or latency of the different proposed models should also be considered when comparing different models' performances. In conclusion, both prediction errors and latency should be minimized.

## Thesis outline

The contents of this thesis are divided over eight chapters. In the first chapter, the most important and insightful related studies are discussed together with all background information needed to fully understand concepts used later on. In the second chapter, all aspects related to data are discussed: data collection, processing and analysis. In the third chapter, different solutions are proposed in the form of deep learning neural network architectures. The fourth chapter discusses how the different solutions are going to be tested and evaluated. The results are discussed in chapter five followed by a concluding discussion in chapter six. References and an appendix with complementary documents are found within chapter seven and eight.



# 1 Background and literature review

A lot of research has been performed in the field of motion prediction. It is a topic that has many different applications and is applied to solve or aid in a broad spectrum of problems. For example, with the upcoming trend of autonomous vehicles and self-driving cars, motion prediction is being implemented to avoid collisions and provide a safe experience for the passengers, the vehicles themselves and their surroundings (Ren et al., 2021). Additionally, motion prediction also has applications in human motion prediction for robot cooperation (Tang et al., 2018) and ground motion prediction to anticipate seismic activity (Dhanya & Raghukanth, 2018) to name a few. However, due to its complex and mostly non-deterministic nature, motion prediction remains a very difficult problem to solve. And ship motion prediction is no exception to this. Before looking at some viable solutions however, a basic understanding is needed of ocean wave dynamics and their interaction with ship kinetics and kinematics as these form the foundation of early prediction methods. The following subchapter gently introduces some basic principles before continuing with different ship motion prediction methods.

## 1.1 Ocean waves and ship interactions

Natural ship motions are primarily caused by the motion of ocean waves which are little deterministic. Ocean waves are the result of an accumulation of several types of waves including capillary waves generated from atmospheric pressure, wind waves, and planetary waves (Silva, 2015). The product of this accumulation of waves is the stochastic nature of sea motion, which is often described using a wave energy density spectrum (Abujoub, 2019). According to Perez T., the motion of a ship can be decomposed into three separate motion inducing forces. The superposition or accumulation of these three forces results in the magnitude of the motion.

- **First-order wave-induced forces.** This force is oscillatory. This is commonly modelled as a time series disturbance obtained by combining the wave spectrum with the vessel's Motion Response Amplitude Operators (Motion RAO), which are transfer functions that map the wave elevation or wave slope into force and motion.
- **Slowly varying disturbance forces.** This force is produced by current, wind and second-order wave effects such as wave mean-drift.
- **Control-induced forces.** This is the force induced by the control system, which is usually designed to counteract only the effect of the slowly varying disturbances.

Each one of these forces has profound and proven mathematical foundation which is well documented in Perez T.'s lecture paper (Perez & Fossen, 2005). Besides these forces, the wave encounter frequency spectrum also plays a role in predicting the motion of a ship. According to Dr. Q. Judge, the motions of a vessel can be seen as a three-part black box structure. The input are the ocean waves and their induced forces as discussed above. The black box are the ship dynamics like its inertia, natural frequency and physical form and the output are the motions of the ship in oceanic waves (Q. Judge, 2019).

In conclusion, the motion of a vessel in ocean waves is a complex interplay between the dynamics of the ocean and the dynamics of the ship. Due to extensive research in these fields, all of the above-mentioned concepts have been supported with mathematical equations. These mathematical foundations have allowed researchers to accurately model these dynamics in physics simulations (Ran et al., 2021). These models, together with the ever-evolving technological possibilities by the likes of computer vision and artificial intelligence, present a variety of methods that can be considered when trying to predict the motion of a ship.

## 1.2 Ship motion prediction methods

Ship motion prediction is incredibly useful for several naval operations such as aircraft landing, cargo transfer, off-loading of small boats, and ship "mating" between a big transport ship and some small ships to name a few. With this wide variety of ship motion prediction applications and its long history of research come a wide variety of approaches. However, most of the

approaches found in published research studies can be divided into three categories. They either use a dynamic model based on the above-mentioned principles, artificial neural networks or a hybrid of the first two. In the following sections, each method is discussed.

### 1.2.1 Dynamic models

Dynamic models or state space models are a set of equations mapping inputs to outputs of a given system based all parameters that affect the model state. It is an approach that relies on the mathematical foundations of ship and wave dynamics. A common method that is used is **minor component analysis (MCA)** (Luo et al., 1997). MCA has similar mathematics as the Principal Component Analysis, except that MCA utilizes the eigenvectors corresponding to the minor components. As shown by the work of Zhao, the age of the method is no indication towards its performance, however. He proposed an algorithm using MCA that was able to predict a twenty second sequence from 800 input datapoints with high and consistent accuracy (Zhao et al., 2004). Zhao used a dataset provided by a software simulation from JJMA inc. The data (surge, sway, heave, pitch, roll, yaw) was collected at 8Hz and down sampled to 2Hz. This frequency and input data is very similar to the simulated data that will be used for this research. In his work he compared the method to a neural network, vector autoregression (VAR) (Stock & Watson, 2001) and a Wiener filter (Chen et al., 2006).

The conclusion was that MCA outperformed all other compared methods and was also suited for real-time implementation. It had the lowest latency based on 500 predictions and the fastest training time. However, only a simple three-layered linear regression neural network was used for comparison. Newer architecture have since been developed better suited time-series predictions. Additionally, 400 seconds were needed to predict only 20 seconds, this is a very high input-output ratio. Finally, but most importantly, the data of the simulation did not show any form of noise. To compensate this, Zhao tested the models with varying percentages of introduced zero-mean Gaussian random noise. This caused the MCA method to quickly lose accuracy with a tenfold decrease in performance at 20% introduced noise.

Another commonly used method is **Kalman filtering**, also known as Linear Quadratic Estimation (LQE) (Kalman, 1960). Initially developed in 1960 and proven effective and reliable by its implementation in the Apollo project (Grewal & Andrews, 2010), Rudolf E. Kalman received the National Medal of Science for Engineering for his research. In theory, the Kalman filter is an algorithm that uses a series of measurements observed over time, including statistical noise and other inaccuracies, and produces estimates of unknown variables. These estimates tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe (Chen et al., 2006).

In the research study of Fossen and Fossen, an exogenous Kalman filter (Johansen & Fossen, 2017) is used for ship trajectory and position estimation based on multiple sensory inputs (Fossen & Fossen, 2018). Another research was done by Peng where Kalman filters are used for estimating the dynamic ship motion states (Peng et al., 2019). The Kalman filter produces an estimate of the state of the system as a weighted average of the system's predicted state and the new measurement.

The Kalman filter has proven its effectiveness and reliability in estimation problems. However, due to the complex interrelation between ocean waves and ship motions, setting up a dynamic model for the Kalman filter can be quite challenging. It is mostly used for theoretical models and is hard to apply to a real-world scenario where not all state parameters are known. This issue of not knowing all parameters to build an accurate dynamic model caused the need for an alternative. As proposed by Zhong-yi Z., one of the alternatives could be to estimate these parameters (Zhong-yi, 2012). However, the complexity of these dynamic models, still remains. For this reason, dynamic modelling was not selected for the purpose of this thesis.

### 1.2.2 Deep learning

Another alternative in trying to determine these parameters was found in artificial intelligence and more specifically, deep learning. The idea is that instead of trying to figure out all necessary parameters to build a dynamic model, a computer is trained to build its own representation which provides the mapping between in- and outputs as accurately as possible. The

computer receives a large set of inputs and their corresponding outputs and learns the relation between the two. This completely eliminates the need to know or estimate parameters for a dynamic model. Because of this, the complexity of the problem is also drastically reduced. This is one of the main reasons why this method was chosen.

The idea of enabling computers to train themselves to solve a problem, dates back to 1958 when the US Navy made a first attempt. However, due to the inability of these early neural networks to learn simple linear decision boundaries like the XOR-function, researchers quickly lost interest (Minsky & Papert, 1969). During the following years, small improvements were made over the next decades that slowly expanded the capabilities of these neural network. Some notable advancements were support for non-linear decision boundaries with multiple layers and the ability to train these multi-layered networks by back-propagating errors (Rumelhart et al., 1986). Nevertheless, they remained inferior to classical methods like Support Vector Machines (SVM) (Boser et al., 1992). It was only around 2010, that their true capabilities came to light when deep neural networks started to outperform all other approaches in computer vision tasks. Ever since, deep neural networks quickly evolved beyond computer vision tasks and have been widely adopted for a plethora of different applications. One of these applications is time-series forecasting problems like ship motion prediction.

Extensive research has been performed in search of optimal deep neural network architectures for time-series prediction and image feature extraction. This presents reliable options today when building a network for ship motion prediction based on images and sensor data. In most cases, Long Short-Term Memory (LSTM) networks are used because they excel in time-series forecasting as will be discussed in 1.3.4. In one of the reviewed studies, a multiscale attention-based LSTM network is proposed to predict ship motion based as an improvement on regular LSTM networks (Zhang et al., 2021). The attention mechanism boosts the sensitivity of the system by paying more attention to significant signals and suppress interference of noise and proves to achieve better performance than other popular methods. In another study, an L1 regularized extreme learning machine is used instead of an LSTM for single-step predictions (predicting only one future value) which resulted in very low near-zero roll prediction errors (°) (Guan et al., 2018). Lastly, in the research of Rashid M., an ensemble model was proposed combining a CNN with an LSTM and a GRU unit (Rashid et al., 2021). The CNN processes two images of incoming waves while the LSTM processes a sequence of pitch and roll values. Both systems would make a prediction from which the average is taken as result. But once again, this study only provided a solution for single value prediction instead of sequences.

While providing good solutions for ship motion prediction, all above mentioned research fails to meet the requirements for this thesis. Only one publication was found by Nazar-Mykola Kaminskyi where both images and sensor data were used to predict a sequence of data (Kaminskyi, 2019). Kaminskyi explored different neural networks that can predict the motion of a vessel based on pitch, roll and incoming wave images. The different model designs all used a combination of CNN, LSTM and linear layers with the ones using images and data showing the best performance. He created a dataset that was used for the majority of this thesis and provided research and results that are directly comparable unlike other studies. However, his research lacked a comprehensive evaluation of the used models, and he also did not perform any latency testing. Both of which will be addressed in this thesis to form a more concise and robust solution that also fully complies with our needs.

In conclusion, deep neural networks provide a state-of-the art solution for ship motion prediction. The fact that they require little knowledge of the underlying physics makes them very accessible. This is clearly visible in the current lay of the land as a majority of found studies on ship motion prediction proposed some form of neural networks. However, due to the virtually unlimited possibilities presented when designing a neural network architecture, finding the optimal one is not unambiguous. This thesis will explore different possibilities in search of finding an optimal architecture for the given simulation data.

### 1.2.3 Hybrid models

Hybrid models are models that combine dynamic modelling and deep learning to achieve better performance. In one paper, a neural net is used to correct the prediction made by a dynamic model (Wei et al., 2022). In another study, a hybrid model is applied for ship trajectory prediction based on current, waves and wind (Skulstad et al., 2021). These studies show that these

hybrid models can also be highly effective in different ship motion prediction applications. However, this approach was not chosen due to its complexity both of the dynamic models and their integration with deep learning concepts.

### 1.3 Artificial neural networks

Neural networks, also referred to as artificial neural networks (ANN) are the building blocks of deep learning problems. These problems are a subset of machine learning problems which are in turn encompassed by artificial intelligence. Neural networks are structures inspired by the human brain and more specifically the neurons within and how they pass signals from one to another. However, similarities end beyond their connected structure.

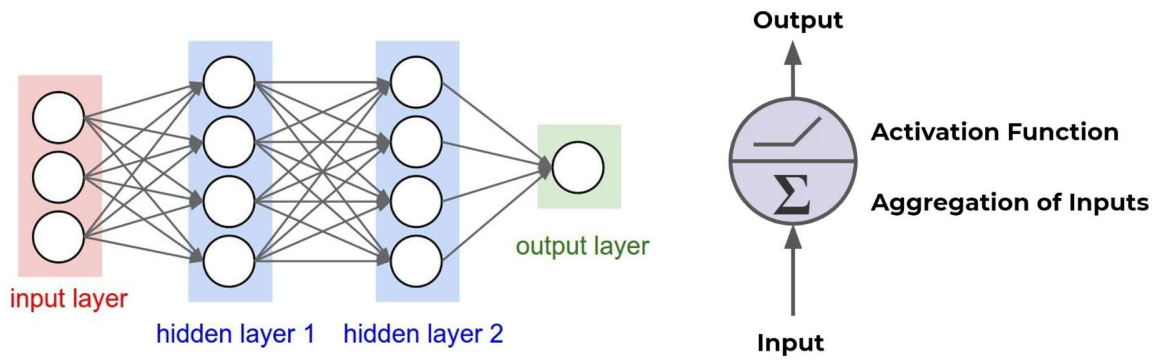


Figure 4: Neural network structure (left) and a linear neuron (right)

A neural network consists of different layers which consist of multiple neurons. These neurons are connected to other neurons in adjacent layers and pass data forward over these connections. The input of one neuron is the output of all its preceding neurons it is connected to. Each neuron applies a weight and adds a bias to all its inputs and passes the aggregated resulting value forward. This way data is fed forward through the network and updated in every neuron. At the end of the network, the resulting value is compared to a ground truth value. After which, every neuron updates its weight and bias via backpropagation to improve its predictions.

There are three main parts in a neural network, the input layer, output layer and hidden layers. The input layer has the same number of neurons as the features in the input data. For example, when simulation data is used and only pitch and roll are used, there are two input neurons. The output layer size equals the number of features one wants to predict. In our case – with the real data – the output features will be pitch, roll and heave.

Different architectures exist for different applications, each with their strengths and weaknesses. In this thesis, sequential data will be used which contains numeric data as well as images. Because of this, different architectures are utilized that are designed to perform best with these kinds of data. Since no single model architecture exists that can handle all data well, different architectures will be combined to form hybrid models. Each of the used architectures and concepts will be discussed more in detail in the following sections.

#### 1.3.1 Gradient descent algorithms



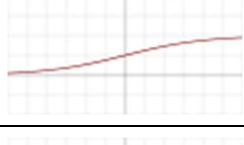
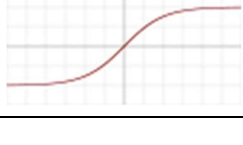
In supervised deep learning problems, each neuron in the network tries to find its optimal weight and bias by comparing the predicted output with the ground truth. The model receives matching inputs and outputs and tries to learn the relation between the two. A metric is used to compare predicted output and ground truth. This metric – commonly referred to as the cost or loss function – is chosen based on the nature of the problem. Some examples are the mean square error (MSE) for regression problems or cross-entropy loss for classification problems (Wang et al., 2022a). During training, the goal is to minimize this error. Each iteration, the model updates its parameters to move, *descent* in the direction of this minimum. The direction in which the minimum lies, is found by calculating the partial derivatives of the loss function for each neuron.

However, since the impact of both weight and bias need to be considered, both partial derivatives are calculated. This is done for all weights and biases. The result is a gradient that can be thought of as a multi-dimensional hyperplane with multiple local minima and one global minimum. To solve for the gradient, the model iterates through all data and computes the gradient in the current configuration of weights and biases. It then utilizes this gradient to find the slope of the cost function and the direction the weights and biases should move towards. Once this direction is known, the parameters of each neuron are updated accordingly starting at the last layer and moving towards the first layer. This process is called backpropagation as the model propagates the errors backwards over its layers (Rumelhart E. et al., 1986).

Different implementations exist for gradient descent with three of them being the most prominent. The main difference between them is how the gradient is calculated. The first is batch gradient descent. This algorithm uses all training data to calculate the gradient before taking a step. This means that the model is updated once per training iteration. The second one is mini-batch gradient descent. Here, the algorithm uses only a subset – a mini batch – of all training data. This allows the model to be updated more throughout training, however some mini batches might not always provide the most optimal gradient. The last one is stochastic gradient descent (SGD). With this implementation, instead of calculating the actual gradient based on all data, an estimate thereof is used (Ruder, 2016). This estimate is calculated through stochastic approximation based on a random subset of the data. Different variants and optimizations exist for each method, one of them being Adam for SGD – which was the chosen algorithm for this thesis. Its concept and implementation will be discussed in more detail in chapter 4.

### 1.3.2 Activation functions

In every neuron the weighted sum of all input is taken, and a bias is added. This aggregation results in a new scalar value that is passed through an activation function before moving to the next layer. The activation function, also referred to as the transfer function, applies a linear or nonlinear transformation to this scalar to limit its output to a certain range. This is especially useful for classification problems where an activation function such as the sigmoid function can be used to limit the output of a neuron to a range of zero to one. The output can then be interpreted as the predicted probability of the input belonging to a specific class. For linear regression problems where a numerical value needs to be predicted like pitch or roll, a linear activation is used. In Table 1, some common activation functions are shown together with their graphs. The sigmoid function  $\sigma(a)$  and hyperbolic tangent  $\tanh(a)$  are both used in LSTM networks while fold functions are used by pooling layers in CNN networks. Fold functions are special activation functions that perform aggregation over the results to take the mean, minimum or maximum.

Linear	$f(a) = a$	
Rectified linear unit (ReLU)	$f(a) = \max(0, a)$	
Sigmoid	$\sigma(a) = \frac{1}{1 + e^{-a}}$	
Hyperbolic tangent	$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$	

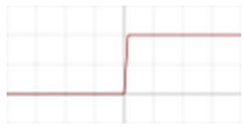
Binary step	$f(a) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$	
-------------	--	--

Table 1: overview of different activation functions

In conclusion, different activation functions are used for different use cases. To predict numeric values such as pitch and roll, a linear or tanh (when input is normalized to  $[-1, 1]$ ) activation is used in the very last layer. However, hidden layers can use different activations to limit the output of their neurons. This way the network can decide whether each hidden neuron's output is important and should be activated. The choice of the activation function can heavily affect a model's performance (Sharma et al., 2020). As a future reference, ReLU and the hyperbolic tangent will both be used in some proposed models in chapter 3.

### 1.3.3 Auto-encoders

Auto-encoders are a type of neural networks that are designed to efficiently copy its input to its output. More specifically, the input gets encoded into a compressed representation, and then decoded or reconstructed based on this encoding. There are two main parts that make up an auto-encoder: the encoder and the decoder. An auto-encoder also holds two main characteristics: the number of neurons in the input is the same as the output and secondly, the hidden layers serve as bottleneck. This bottleneck forces the model to learn only the most prominent features of its input data that are needed to reconstruct it as accurate as possible (Lopez Pinaya et al., 2020).

The encoder part of an auto-encoder is capable of creating a sparse representation of the input data that holds as much information as possible. This property can be used to train an auto-encoder on the images and use the encoder part as a pretrained feature extractor for the images. This concept is applied in the work of Kaminskyi. He used the encoder part of an auto-encoder as a pre-trained feature extractor. The auto-encoder was trained once on the simulation images and could afterwards be used without the need of retraining. However, in this research, the encoder decoder configuration is used in a more liberal approach where two neural networks work together. One is used to encode the input and a second one is used to decode this encoding and make new predictions based on the encoding. This form of auto-encoders is commonly referred to as variational auto-encoders where input is encoded to decode into new outputs instead of reconstructing the input (Kingma & Welling, 2019).

### 1.3.4 RNN and LSTM networks

Recurrent Neural Networks, RNN are special neural networks designed to work with sequential data (Sherstinsky, 2018). Sequential data is data where the order is important, for example time series data, sentences, audio etc. Ship motion data like pitch and roll fall within the time-series data category. The same goes for video footage or consecutive images where frames should be processed in a specific order. The learn this relation between consecutive datapoints, RNNs are introduced.

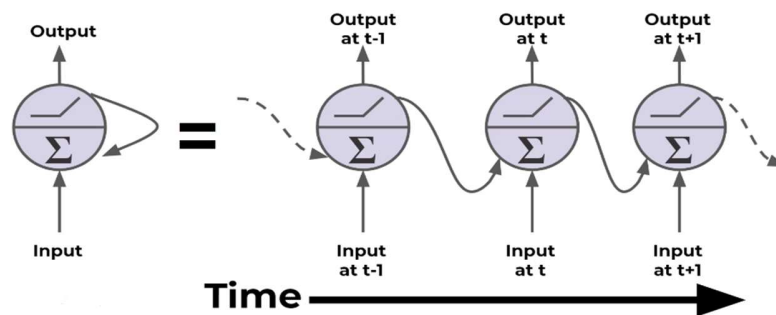
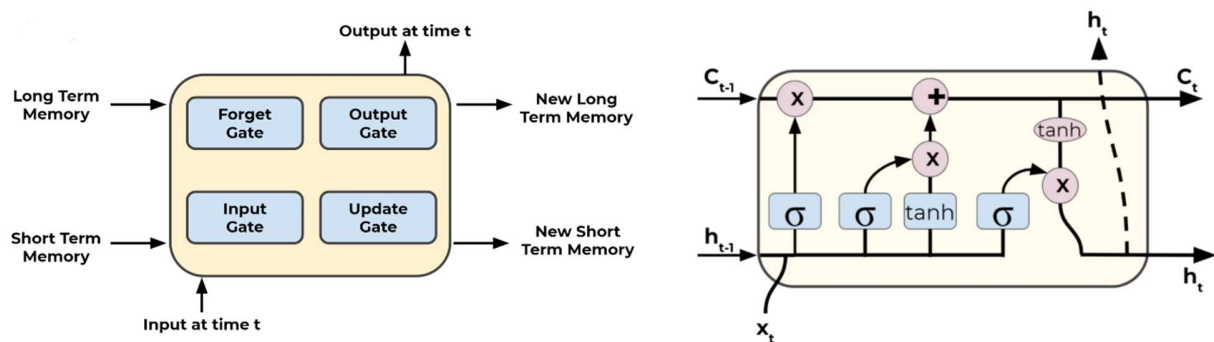


Figure 5: Neurons in a recurrent neural network with additional feed-back connections

To efficiently learn from ordered data, each neuron passes its output forward to the next layer and back into the next neuron in the same layer. Each neuron in an RNN network uses an output that is fed back and a new input to make a prediction. This is the main concept in RNNs and difference between feed-forward networks. However, as illustrated in Figure 5, RNN network neurons only pass the previous output back into the network, therefore it only “remembers” the short-term history in the sequence. This causes the model to “forget” its long-term history or in other words: the model losses perception on the general trend of the sequence.



Because of their design and beneficial characteristics, LSTM networks will be the main building blocks for the deep learning models proposed in this thesis. They should be able to learn the trend of the ship's motion and make accurate predictions based on this trend. However, LSTM architectures are not the only viable option for sequence-to-sequence prediction problems. Different variants exist such as peephole LSTM networks and gated recurrent units, GRU. These will be discussed more in the discussion chapter.

Besides sequential numeric data, there are also images available to aid with the prediction process. However, LSTM networks and standard linear networks are not very efficient when working with image data. Because of the way an image is represented, see below, the number of parameters in these networks ramp up very quickly, even with small images. This causes the networks to train and generalize very slowly. Therefore, a second architecture is introduced: convolutional neural networks, CNN. These networks are highly effective when dealing with images (Wu, 2017).

A convolution kernel or filter is a square  $n \times n$  matrix. Each element within this kernel contains a weight value. During a convolution, the kernel is moved over the matrix representation of the image and applies each of its weights to the corresponding pixel value as illustrated in Figure 7. The result is a new value placed at the center of the current kernel position. Because the kernel is a two-dimensional matrix just like the image, no information is lost, unlike when flattening the image. The kernel is moved across the whole image while repeating the same process. After one convolution, the result is a filtered image with more accentuated features. In typical fashion, multiple convolution layers are connected to allow each consecutive layer to extract more precise features such as wheels, eyes, windows etc. During the training process, the network learns the best values for each weight in the kernels to extract the most prominent features. In the case of incoming wave images, the network is expected to extract the form of the waves in the images. Because the wave images are colored, they contain three channels (red, green and blue) and thus three kernels are used. One for each channel in the image.

Two important parameters are used when dealing with convolutional layers: the kernel size and its stride distance. The first one sets the dimensions for the kernel matrix while the stride determines how many steps forward the kernel moves before

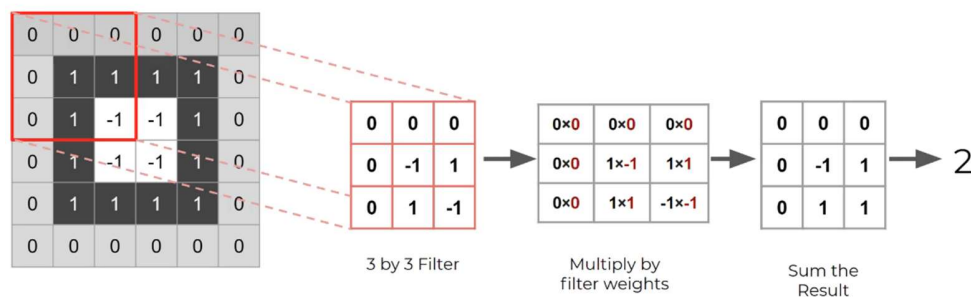


Figure 7: Convolution over grayscale image with 3x3 kernel

calculating a new filter value. Additionally, padding can be added to the border of the image to prevent data loss. This allows the filtered image to remain the same size as the input after a convolution. Otherwise, each convolution would remove one layer of pixels around the border of the image. This is because each resulting value is placed at the center location of the kernel in a new matrix. Padding values are mostly all white or all black pixel values normalized with the method at hand.

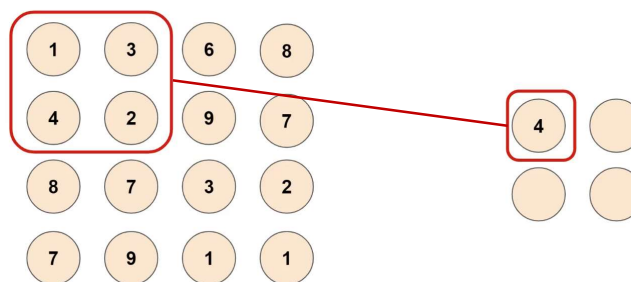


Figure 8: Max pooling with 2x2 kernel

A convolutional neural network still has a very large number of parameters despite being more efficient than linear networks. When dealing with colored images and a lot of filters, tens to hundreds, a method is needed to reduce the number of parameters. Pooling layers can be used to reduce this. A pooling layer samples the filter size down by applying a function to different subsets of data from the filter. The process is very similar to a convolution where a  $n \times n$  kernel is moved across the filter. However, the kernel has no weights. Instead, it takes the maximum of the values in the filter on a  $n \times n$  basis. This method is called Max Pooling. For example, if a 2x2 kernel is used, four data points of the filter will be down sampled to one. When a stride of two is used in this configuration, 75% of the data is reduced. Another method is called Average Pooling where the average is taken from the kernel-sized subset.



## 1.4 Tools

To develop the deep learning neural networks, Python 3.8 will be used. Python is a high-level, general-purpose programming language that is in - most cases - most suited for machine learning applications. To facilitate the development of the deep learning models, different packages were used. The most important of which are briefly discussed below.

Pandas, Seaborn and NumPy were used to process and manipulate data. NumPy provides functionality to perform mathematical functions on large datasets with multiple dimensions. Pandas provides the functionality to visualize and order the data in its Data frames. Seaborn was mainly used to perform data analysis on the numeric data such as pitch and roll, to assess and visualize statistical information between features. To plot training and test results, Matplotlib was used.

For the implementation of the deep learning models, PyTorch was used. PyTorch is an open-source framework for machine learning applications that allows fast development and ease of use. PyTorch was used together with the cudatoolkit extension. This enabled the models to be trained and tested on a dedicated graphics processing unit (GPU) to increase computing performance.

Blender was used to generate a dataset from an ocean wave simulation. Blender is an open-source three-dimensional creation suite that can be used for many purposes. In this case, it was used to simulate incoming waves on a vessel. The data itself will be discussed more in detail below.

All code was written in Jupyter Notebook for its ease of use and simple debugging. Whenever parts of the code in the notebooks were tested thoroughly, they were extracted to standalone Python files and accessed via import statements.

## 1.5 Expectations and criteria

To evaluate the performance of each model and their viability as possible solution based on the requirements discussed in the objectives, all models are subjected to a set of minimum criteria. These criteria set baselines to which the model will be compared. Note that the criteria defined in this subchapter are set for use only in the simulation environment. A model is fit to be used with real data only when it meets the simulation criteria. These criteria were determined based on related work but primarily on the study by Kaminskyi and personal expectations. The reason why Kaminskyi's results were primarily used as a reference is because of the dataset he used. He created and used a simulated dataset of images and corresponding pitch and roll measurements for research. Because of the similarities and the dataset being publicly available, the same dataset was also used for this research. As a result, his research is directly comparable to ours.

In Kaminskyi's research, vastly different performances were measured between different model designs. The LSTM model that processed only pitch and roll as input performed worst, followed by the CNN models that only used images. The best performance was achieved by ensemble models using both pitch and roll and images. The errors of each model are shown in Table 2 with the worst and best performing model highlighted in resp. red and green. These errors were calculated as the average difference between prediction and real values at the tenth predicted second. Models were tested in a configuration where ten seconds were used to predict twelve (resp. 20 and 24 frames at 2Hz). For reference, the model naming convention in his research is a combination of used inputs and architectures. For example, *PR* means that pitch and roll are used as input, *stack* means that a stack of images was used as input. More information on the model designs and naming can be found in his report.

Model	Pitch at 10 <sup>th</sup> sec [denormalized RMSE]	Roll at 10 <sup>th</sup> sec [denormalized RMSE]
LSTM encoder decoder PR	30.99°	29.44°
CNN stack FC	28.97°	28.94°

CNN stack FC PR	21.18°	22.56°
CNN FC PR	11.36°	11.28°
CNN LSTM image-encoder PR- encoder decoder	3.56°	2.83°
CNN LSTM encoder decoder images PR	2.89°	2.70°
CNN LSTM encoder decoder images	3.42°	2.32°
CNN LSTM decoder images PR	4.22°	4.16°

Table 2: Kaminskyi's results for different models with worst (red) and best (green) model highlighted (Kaminskyi, 2019)

The red and green rows respectively highlight the worst and best performing model. They serve as a reference benchmark to evaluate the models proposed in this research. Below in Table 3, similarly calculated errors are shown of the best performing model after fully optimizing its parameters with the HyperBand algorithm (Li et al., 2016). The error values show the model's capabilities to predict longer future sequences and represent the average error over a series of predictions at different time-steps. For reference, when predicting a thirty second window, the model needs to predict sixty future datapoints at 2Hz.

Parameter	10 <sup>th</sup> second	15 <sup>th</sup> second	30 <sup>th</sup> second
Pitch	2.89°	2.92°	3.06°
Roll	2.71°	3.75°	4.92°

Table 3: prediction error at different future time-steps of Kaminskyi's best model (Kaminskyi, 2019)

Based on these results, a general idea was formed about expected performances on the given dataset and criteria were set accordingly. The following properties of the model must comply with the next requirements. Firstly, the **de-normalized root mean square error (RMSE)** for the predicted angles for pitch and roll at the 10<sup>th</sup> second should be no more than three degrees and no more than five degrees at the 30<sup>th</sup> predicted second. These values are chosen in assumption that the accuracy of any model decreases as it predicts further in the future. The errors should be calculated as the average over all predictions in a dataset of unseen data (data that was not used in training). Secondly, all models will be compared to a hypothetical **zero-predictor**. This is a model that predicts a zero for each frame in the output. Models performing worse than the zero-predictor did not learn at all and can be classified as random generators. Finally, the prediction latency or **inference time** of the model should be as low as possible. The inference time should be measured as the average over ten thousand predictions with a batch size of one. The batch size is chosen at one as the model will not be able to predict more than one sequence at one time when predicting on live data-streams. There was no research found comparing similar neural network designs compared to ours. But some very popular state-of-the art neural networks like ResNet-18 (He et al., 2015) and GoogLeNet (Szegedy et al., 2014) show inference times between 30 and 150 milliseconds with a batch size of one (Canziani et al., 2016). These values can serve as a reference. If possible, the testing on latency should ideally be expanded towards also testing the model's hardware usage as resources will be limited when being deployed.

Whenever a prediction sequence is calculated, it should be analyzed by an algorithm. This algorithm will search the sequence for a window of landing/take-off opportunity. During this window, heave, roll and pitch should remain close to constant over a duration of continuous time. However, different drones have different characteristics and requirements to and. For example, the range of motion for pitch and roll of the hovering drone in which it remains stationary. To compensate for this, the parameters of this window should be easily changeable to the needs of the drone at hand. These are just some starting principles as this algorithm falls out of the scope of this thesis

## 1.6 Summary

In this chapter, the state-of-the-art in motion prediction was briefly discussed. Additionally different methods were introduced to predict wave-induced ship motions. The conclusion was made that for many years, ship motions were predicted based on dynamic modelling of the physics involved. However, due to the non-linear non-deterministic nature of ocean waves and their interaction with ship motions, setting up an agile and accurate quickly becomes extremely complicated. For this reason, deep learning neural networks were chosen as an alternative solution. It is hypothesized that this modern approach can learn the complexity of dynamic models solely based on examples. Their accuracy and effectiveness on an ocean wave simulation will be evaluated in this thesis. The evaluation will be based on criteria that were defined based on results from related studies and expectations. In addition, all used concepts of neural networks were introduced as a reference to future mentioning in the paper. LSTM and CNN networks will be used primarily in the design of proposed neural network architectures due to their efficiency in resp. capturing trends in sequential data and extracting features from images – which are both important given the dataset that will be used.

## 2 Data collection and processing

The first step of any machine learning operation is collecting data. For this thesis, data was collected from two sources as discussed in the introduction. A dataset from simulation and real-world data was used. This poses some challenges as the simulation data is captured in an ideal scenario and contains less motion parameters as the real-world counterpart. This can cause the models to perform differently and have slightly different architectures due to the different inputs and outputs based on the available data. In the following two sections, both datasets will be discussed in more detail. For the remainder of this thesis, the different parameters such as pitch, roll, yaw etc. will be referred to as the *features* of the dataset.

### 2.1 Simulated data

During most of the development, a simulated dataset was used from an ocean wave simulation made in Blender (*Nazotron1923/Ship-Ocean\_simulation\_BLENDER: Python Scripts for Generation Sea States and Ship Motion Using 3D Blender Simulator*, n.d.). This dataset was made by Nazar-Mykola Kaminskyi and is publicly available through GitHub. It was not made by or in cooperation with the research group of this thesis. Simulation data was needed for training and testing purposes during the initial phases of development when real data from the ASV was not yet collected and available.

For this simulation, a standard model of a vessel was used which is floating on the simulated sea surface and moves along with the waves. This presents the issue that all models trained with this data will be biased to this vessel's characteristics. A large, heavy vessel will behave vastly different opposed to a small, lightweight vessel. Therefore, all models should be retrained and re-evaluated with real data from the vessel it will be used on, as this vessel will most likely have different characteristics. However, this issue can also form a way to compare how well one model can predict the motion for different vessels. If there are only minor increases in performance when the model is retrained with data from the specific vessel, it might be more useful to use one general purpose model for all vessels with similar characteristics.

Using a simulation also presents a second issue: perfect conditions. The data taken from the simulation is from a perfect scenario, meaning that there are no obstructions or other objects on the images. The images are also perfectly stabilized on the moving vessel. This can cause models that are trained on the simulation data to perform very well in simulation yet fail to meet desired expectations in a real-world scenario. To minimize this effect, data can be augmented to include more variation, or adjustments can be made to the simulation to make it more realistic such as including passing vessels in the images. Finally, since the data is generated by a computer, a computer might be able to capture the trend a lot better than naturally occurring data. A simulation can be very realistic, but behind the scenes there is still just an algorithm with parameters that a neural network might be able to learn very quickly.

The simulated dataset contains images of incoming waves (figure 10) and the pitch and roll values of a vessel floating on these incoming waves. The data was generated in 540 different episodes, each episode containing 400 frames of data. The frames were captured at two frames per second to minimize data overlap in consecutive images. Each frame contains one image together with the state of the vessel at the time of the image in the form of a pitch- and roll-value tuple. In its totality, this dataset contains 216.000 frames, which translates to thirty hours of simulated data at two frames per seconds. All episodes were structured in the same way. Images were named 1 to 400 and pitch and roll data points were included in a *JSON* file. The pitch and roll couples were numbered 1 to 400 as well to easily identify which image corresponds to which tuple.

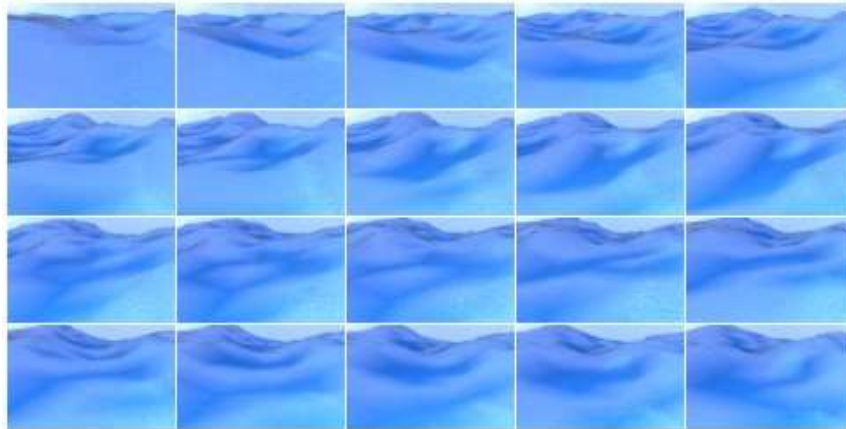


Figure 9: Generated images of incoming waves (chronologically ordered left to right, top to bottom)

With this simulated data we only have access to pitch and roll as input data. This means that during testing in the simulation environment, heave cannot be predicted. However, with the neural networks that will be proposed later, this should not be a big issue for two main reasons. First off, heave follows a somewhat predictable pattern in regular conditions as depicted by figure 4. Because of this, it was assumed not necessary to regenerate the full dataset for just one extra output. It was also assumed that if the model performed well on pitch and roll, it should consequently perform well on heave. Secondly, adding extra input and/or output features to a neural network architecture, is not an expensive operation. It is highly likely that all models will have to be retrained when switching from the simulation environment to a real-world environment to adapt to the new vessels' characteristics and the additional data available from the ZED-mini. The difference in performance of a retrained and non-retrained model remains a topic for future work.

### 2.1.1 Simulation parameters

To simulate this data, different parameters were used to finetune the simulation environment. The effects of these parameters on the simulation were not tested as the used dataset was already generated with fixed parameters. However, assumptions can be made on how they would affect the simulation. The most important parameters will be discussed briefly.

As mentioned above, the images were taken at two frames per second by a virtual camera. The position of the camera on the simulated vessel was set with the following parameters:

- Height: *5 meters*
- Rotation around x-axis: *76 degrees (slightly tilted downwards)*

The images were captured in a low resolution to decrease the memory requirements to load the dataset. The following resolution parameters were used:

- Height: *54 pixels*
- Width: *96 pixels*

### 2.1.2 Data augmentation

When the ASV is operating in open seas, the images of the incoming waves will be mostly free of obstructions. However, when the ASV is close to land, other objects may be in front of the ship like other vessels, shorelines, etc. To better evaluate performance in these real-life environments, a small subset of the simulated images were edited and added to a different separate dataset. Using an image editing software, ink blots were manually drawn on the images to resemble external objects in front of the ship. The shape, size and colors of these blots were randomly chosen. In some cases, they may be very unrealistic. This was done in assumption that an even worse obstruction in the frames is very unlikely and the performance on these images serve as a baseline for the very worst scenarios. Some of the augmented images are shown in Figure 10. To test the performance on this augmented dataset, models were trained on the normal images. Afterwards, the results of the augmented sequence and normal sequence were compared.

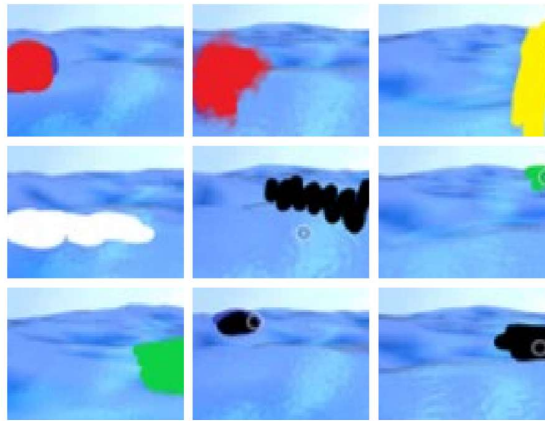


Figure 10: Augmented images

## 2.2 Real data

Eventually, after evaluation on the simulation set, some prediction models will be selected for deployment on the ASV. This means that they will need to transition from simulation data to real data. To provide the real data, the target vessel is equipped with multiple sensors. However, due to the nature of sensory data, using this real data presents some challenges.

### 2.2.1 Sensors onboard the ASV

The ASV is equipped with a ZED-mini stereo IMU camera (Figure 11) (Stereolabs, Paris, France). This is a multipurpose sensor that can capture video from its two cameras and numeric data from its Inertial Measurement Unit (IMU). The combination of these two sensors allows the ZED-mini to accurately describe the state of the sensor and its surroundings. The IMU has two built-in motion sensors: an accelerometer and a gyroscope. These provide a real-time data stream at 800Hz of the movement of the sensor along the rotational and translational axes.

The two forward facing lenses on the ZED-mini provide stereo video. This can be used to map the objects in front of the ZED-mini in a three-dimensional space. A stereo image is a combination of two separate images that are captured from two slightly offset point-of-views (PoV) such as the lenses on the ZED-mini. These two PoV's imitate the left and right human eye and create a perception of depth when the two images are fused together to create one stereo image. This process is called stereoscopy. In computer vision, these two images can be compared to each other to extract three-dimensional information from two dimensional images.

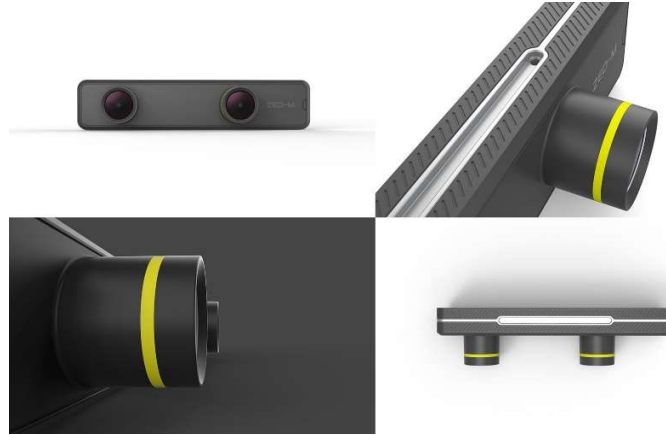


Figure 11: ZED-mini IMU stereo camera (StereoLabs, Paris, France)

### 2.2.2 Sensory data challenges

The real data will be captured and fed into the model at real-time. This means that the data stream will have to be cleaned and filtered to the desired input format according to the prediction model's requirements. As mentioned above, numeric motion parameters such as pitch and roll, are captured by the ZED-mini at 800Hz. This means that every second 800 data points are measured. At this frequency, consecutive measured values contain a lot of overlapping data where minor change is happening between values. Therefore, the data stream will have to be reduced to minimize this data overlap and avoid overloading the prediction model with noisy data. Since the simulation data was generated at two frames per second, 2Hz should be a good starting frequency to reduce down towards to test the models. Later on, this frequency can be increased or further decreased depending on the model's performance and inference time.

Similarly, the video framerate from the stereo camera should be reduced to minimize data overlap between consecutive frames and reduce computational requirements. The video footage will also have to be compressed to reduce the memory and computational requirements needed to process a sequence of images.

*(From this point in the thesis and onward, only the simulation data is used due to unavailability of real data)*

## 2.3 Data pre-processing

Before the images and numerical data can be loaded into the model, they need to be processed. This pre-processing of data is necessary to enhance the performance of the models and ensure that the model operates correctly. Due to a simulation dataset being used, little pre-processing is needed since the data is collected in a predefined format.

### 2.3.1 Data cleaning

The first step of data pre-processing is data cleansing. In this step, data points are checked for inaccuracies and corrupted values. Odd data points are corrected or removed, and unnecessary features are dropped from the dataset.

In the case of the simulated dataset, this task is rather short as the data was generated following a specific format as discussed in 0. To cleanse the simulated data, all episodes were evaluated to contain 400 images and 400 pitch and roll couples. Next, all pitch and roll values were audited to not contain any odd values such as un-numerical values or odd outliers. This was done with `Pandas.isnull().values.any()` and `Pandas.DataFrame.plot.scatter()`. The scatter plot (Figure 12) shows the 2D lay out of pitch and roll. The data is mainly concentrated in the middle with few outliers. Further analysis of the distributions will be conducted in 2.4 data analysis.

When real data is used, comprehensive cleaning is needed to filter out noise from the sensors and excessive outliers from inaccurate readings.

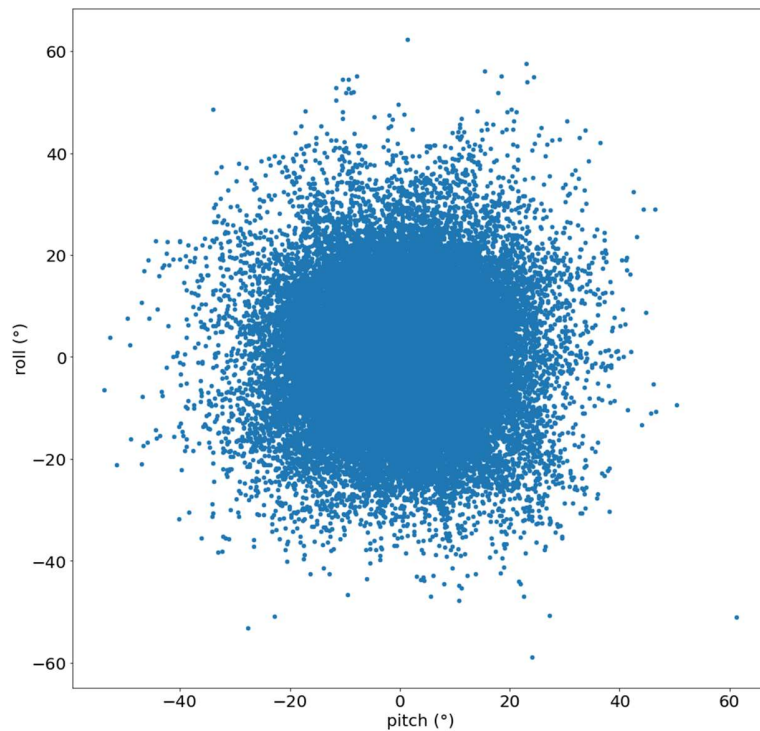


Figure 12: Scatterplot of pitch and roll

### 2.3.2 Data formatting

All data features fed into a neural network must be formatted in the same way. This means that all features containing numerical data should be converted into the right datatype and images should be resized to the same resolution. Textual data should also be pre-processed but this falls outside the scope of this thesis as no textual data will be used. This formatting is particularly important as incorrectly formatted data cannot be used to train and test neural networks.

For the simulation data, little formatting had to be done. All images were generated with the same resolution, so image resizing was not necessary. The pitch and roll values were imported from JSON-files as textual string data. They were subsequently casted to 64-bit floating point values for high accuracy.

### 2.3.3 Data reduction

Because the data was generated by a simulation, values and images are captured as defined by the parameters in a desired format. This means that little to no data reduction is necessary. All images were captured with equal dimensions that are small enough to eliminate the need for additional compression or cropping but large enough to keep some level of detail. The small dimensions also allow for lower computational requirements. The data is captured at two frames per seconds (fps) in assumption that this a good middle ground between minimizing overlapping data without losing to much information. The frequency of the data can be reduced to a lower fps by leaving at a certain number of frames between each loaded datapoint. In this manner, performance can be measured for different fps to find the optimum. Higher fps can also introduce more noise in the form of slight changes or stutters in pitch or roll. These slight changes might have negligible impact on the movement of the ship but may affect the learning process of the prediction model as it tries to learn this noise. As a final note, to find the optimal frequency of the input, not only data overlap should be considered. The inertia of the vessel at hand may play an even more significant role. Large, heavy vessels move a lot slower compared to smaller, lightweight vessels. Because of this



physical property, the latter will probably benefit from a higher data input frequency because it allows for faster movement changes due to its lower inertia.

#### 2.3.4 Data normalization

When one feature contains values in a range of [0, 100] and another feature contains values in a range of [0, 5], the first feature will have a much larger impact on the result if they are used together. For this reason, data needs to be normalized.

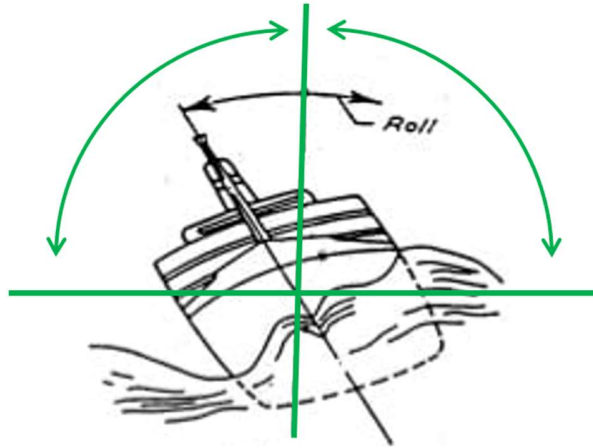


Figure 13: Maximum range of rolling motion for ships

Pitch and roll are normalized with min-max feature scaling. This method rescales values relative to their absolute maximum and minimum within a [-1, 1] range. Since pitch and roll define the tilting of a ship, the absolute theoretical minimum and maximum were chosen to be  $-90^\circ$  and  $90^\circ$  (Equation 2.1). If a ship pitches or rolls beyond these values, it will capsize []. It is also not possible to define the minimum and maximum as the boundaries of the measured values in the dataset, because these numbers are not known when working in real-time.

$$x_{normalized} = \frac{x_{original} - (-90)}{90 - (-90)} \cdot 2 - 1 = \frac{2 \cdot (x_{original} + 90)}{180} - 1$$

Equation 2.1: Min-Max normalization of angular motion parameters pitch and roll

Each images needs to be normalized as well. Each image is three-dimensional matrix containing three values - the red, green and blue channel - for each pixel in the image across the length and width. This means that for every pixel, three values need to be normalized. The three channels indicate the amount of red, green and blue in each pixel. These values are 8-bit integers, meaning that they range from 0 to 255. To normalize them, a similar min-max feature scaling function was used as for the pitch and roll values (Equation 2.2).

$$p_{normalized} = \frac{p_{original} - 0}{255 - 0} \cdot 2 - 1 = \frac{2 \cdot p_{original}}{255} - 1$$

Equation 2.2: Min-Max normalization of RGB-values

## 2.4 Data analysis

The simulation data was briefly analyzed to have a better understanding of the results. Very in-depth analysis of the input data is not necessary for deep learning applications. Deep learning is a form of unsupervised learning, where the model itself determines the importance of each feature. In comparison, supervised learning problems require thorough data analysis to find out which feature(s) have most impact on the desired output feature(s) and should be included in the models' input. The

goal of the data analysis in this thesis is to firstly better comprehend how they behave and secondly the interactions between pitch and roll.

#### 2.4.1 Statistical properties

First off, basic statistical information was extracted from all pitch and roll data points from all episodes. With the built-in function of Pandas `pd.DataFrame().describe()`, this is easily achieved. The results are shown in

Table 4.

	PITCH	ROLL
count	216.000	216.000
mean	0,0678°	0,303°
standard deviation	6,611°	7,022°
minimum	-53,721°	-58,846°
25%	-2,761°	-2,572°
50%	0,0262°	0,230°
75%	2,876°	3,187°
maximum	61,328°	62,217°

Table 4: Statistical information for pitch and roll in simulated dataset

The count refers to the amount of datapoints analyzed which is equal to the 540 episodes, containing 400 frames each. From these values, it can be concluded that pitch and roll both behave similarly on a numerical basis across all fields. However, this does not necessarily mean that they physically behave similar. The minimum and maximum values give a good estimate of the interval in which the predicted values should remain. They also provide a means to normalize the RMSE. An error of three degrees is a 5% error relative to the absolute maximum and thus really good when all data occurs in a  $[-60^\circ, 60^\circ]$  interval. If all data lies in a  $[-10^\circ, 10^\circ]$  interval, an error of three degrees is a 30% relative error which is very high. However, the minimum and maximum don't tell the full story.

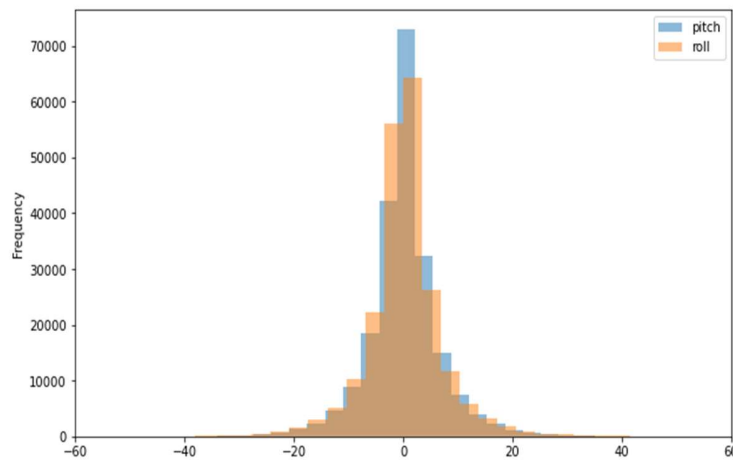


Figure 14: Simulated Pitch and Roll distributions

The distributions of pitch and roll are shown in Figure 14. Both of them are remarkably similar and show remarkably similar characteristics to a normal Gaussian distribution. Intuitively, it can be assumed that the majority of the datapoints are located in a  $[-20^\circ, 20^\circ]$  interval. Compared to the properties of normal distribution, this assumption is justified when the 65%-95%-99.7% rule is applied. This rule determines that given a mean  $\mu$  and a standard deviation  $\sigma$  of a normally distributed

dataset, respectively 65%, 95% and 99.7% of all datapoints lie within a  $\mu \pm \sigma$ ,  $\mu \pm 2\sigma$  and  $\mu \pm 3\sigma$  interval. If this rule is applied with the mean and standard deviation from Table 4, the 99.7% interval can be calculated and are the following (values rounded down for readability):

- Pitch:  $[-19,74^\circ; 19,86^\circ]$
- Roll:  $[-20,70^\circ; 21,30^\circ]$

These intervals confirm that almost all data points – 99.7% – lie within the above intervals. Because of this, the RMSE should be taken relative to these interval boundaries rather than the maxima and minima of the full dataset. The 99.7% intervals represent almost all data whereas the minima and maxima could be two extreme outliers.

#### 2.4.2 Correlation and interactions

Secondly, the correlation of pitch and roll was reviewed to assess the influence of roll on the prediction of pitch and vice versa. This was done to decide if having both motion parameters as input is necessary for the predictions on one or the other feature individually.

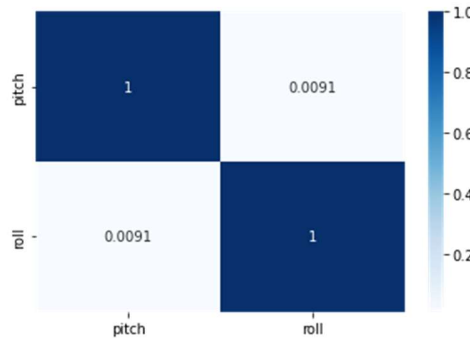


Figure 15: Correlation matrix for pitch and roll

A commonly used method for analyzing correlation is a correlation matrix. This is a symmetric matrix where the number of rows and columns equals the number of features. Each cell in the matrix contains a value equal to the correlation of the features of the corresponding row and column. These values range from -1 to 1 where -1 indicates a strong negative correlation, 1 a strong positive correlation and 0 indicates that there is no correlation. From the matrix in Figure 15, pitch and roll have a correlation close to zero, which indicates little to no correlation.

With this result, it can be assumed that one feature has negligible contribution to the prediction of the other feature. Providing additional features besides the predicted feature can therefore be seen as overhead for the model. However, this assumption is only based on numerical values from a simulation containing only pitch and roll. According to this research (Nayfeh et al., 2012), pitch and roll are coupled nonlinearly when their frequencies are in a ratio of two to one. On top of this, more correlated features could be introduced with the additional input features from the ZED-mini sensor.

With this information, it was concluded that standalone feature correlation was not enough evidence to rule out other possible physical interactions. Because of this, all models were designed to ingest all available features instead of dropping some due to low correlation with the predicted target features. In the case of the simulation data, this meant that models were trained and evaluated to predict both pitch and roll from an input sequence also containing both pitch and roll.

## 2.5 Data loading

Data loading is the process of creating input-output sequences, splitting the data and creating batches for training. For each model, the inputs and outputs will be sequences with varying lengths. Next, these generated sequences will be split into

three subsets for training, testing and evaluating the model. Finally, the data will be wrapped in PyTorch DataSet and DataLoader objects which will convert the data into Tensors and create batches.

### 2.5.1 Sequence creation

The simulated data is generated in episodes. Each episode containing 400 frames that chronologically follow each other as can be seen in Figure 9. However, the episodes themselves do not follow each other chronologically. Different episodes are generated in different simulation sessions and should therefore not be seen as continuous. Because of this, the data must be sequenced correctly to not include data from different episodes when used for deep learning models that ingest and predict chronological sequences. In short, all frames in the input and output sequence, should always be from the same episode.

For this reason, a dedicated function was designed to create sequences from a given dataset. One sequence consists of two parts, an input sequence and an output sequence. The input sequence is the data used to predict the output sequence. The length of the input and output sequence can be passed to the function as a parameter to easily create new sequences. This is necessary as the length of the input and output sequences will have a large effect on the performance of the model and should be thoroughly examined. In addition, the function also accepts parameters to define the input and output features when different datasets are being used.

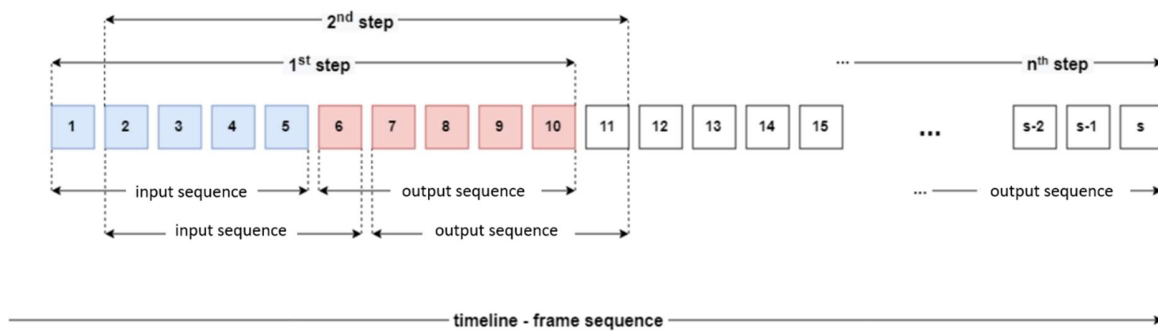


Figure 16: Sequence creation with moving input and output sequence

In Figure 16, the sequence creation process is illustrated with input and output sequences having a length of five. In the first step, the function takes the first five elements of a given episode with length  $s$ . These five elements form the input sequence, which will be used to predict the next five elements: the output sequence. These two sequences are then coupled together and added to a list with all sequences. In the second step, the starting point for the sequences is moved to the next element and the same procedure is conducted, extracting and coupling an input with an output. This process is repeated until the end of the episode is reached. At this point, the function moves to a new episode and repeats the cycle. This way, all input and output sequences can only contain consecutive data from the same episode.

### 2.5.2 Train-test-validation split

After the sequencing is done, the list of all coupled input-output (IO) sequences is split up in three subsets. Each subset will be used for a different purpose: one for training, one for testing and one for evaluating the models. This splitting is necessary to properly test and evaluate the model's performance.

When a machine learning model is trained on a dataset, it will try to optimize its performance as much as possible on this data. Therefore, in order to evaluate how well the model generalizes to new data, a second dataset is needed to test the model. This test dataset contains data that the model has not seen before and is not optimized towards, unlike the training data. So, if the model is tested on this new dataset, the real performance on new data can be evaluated. After this evaluation with the test dataset, the models' parameters can be adjusted in the hopes of further optimizing the generalization to new

data. Finally, a third dataset is used to measure the performance of the model, after being trained on training data and optimized with test data. After this, no changes are made to the model to ensure that the results from the evaluation dataset, represent the model's true core performance on new data.

To achieve this split, elements from the list of coupled IO-sequences are randomly selected and assigned to one of the three subsets. This randomization has positive effect on the model's performance because it mitigates possible trends in the initial dataset. For example, due to pure coincidence it is possible that the first ten IO sequences are from calm seas and the next five are from rough seas. If this dataset is split using standard iteration over the data - adding the first ten IO-sequences to training and the next five to testing - the model will perform very poorly as it never had the chance to learn the rougher sea's trends. Thus, the data is split with random selection. Additionally, since sequential data is used, consequent IO-sequences will have some data overlap. Figure 16 shows this more clearly: the sequences from the 1<sup>st</sup> and 2<sup>nd</sup> step are very similar. When random selection is used, the effect of this data overlap is minimized and IO-sequences in one batch contain the maximum amount of unrelated data.

### 2.5.3 DataLoader and DataSet

Finally, after the pre-processed data is sequenced and split, it is prepared for the model. All neural networks were designed in PyTorch. Because of this, it was most convenient to load the data with a PyTorch DataLoader. The three different datasets – training, testing and evaluation – are wrapped inside a DataLoader object. The DataLoader in turn returns this data as an iterable PyTorch DataSet object and splits it into batches.

The DataSet class acts a data structure which holds the IO-sequences. It is a custom class which inherits from the PyTorch DataSet class and overrides its main method: `__init__()`, `__len__()` and `__getitem__(int)`. The latter of these methods returns the IO-sequence at the given index as a dictionary. This dictionary holds two key-value pairs – one for the input and one for the output sequence - converted into a PyTorch Tensor. A Tensor object is an n-dimensional array that allows functions to be conducted over all elements at once, much like a NumPy array.

The DataLoader class provides methods to load training, test and evaluation data. Part of this loading process involves splitting the data into batches with a fixed batch size. The size of the batches is an important hyperparameter and determines how much data the model will consume before updating its internal parameters. Batch size and its effects will be discussed more in detail in section 3.2.4.

In conclusion, data is wrapped inside a DataSet object, which in turn is wrapped in a data DataLoader object which divides it in batches. The DataLoader can then be called to return either one of the datasets – training, test or validation – and returns this as an iterable object, with each batch of **n** samples being one iteration.

## 2.6 Summary

This chapter discussed all subjects related to data collection and processing. The properties of the simulated data and real data were introduced. Both images and ship motion data were collected. However, due to the real data being unavailable for a majority of the project's duration, there was insufficient time to work with real data and only the simulation data was used in this thesis. Some challenges were discussed when transitioning from simulated data to real data. These will have to be further assessed together with the transition in future work. Besides this, it was concluded that 99.7% of all pitch and roll datapoints lie within an approximate window of  $[-20^\circ, 20^\circ]$ . This is important when calculating relative errors based on minima and maxima. In addition, there was no correlation found between pitch and roll. However, studies have shown that pitch and roll can interact with each other and therefore it was concluded that despite the low superficial correlation, pitch and roll should always be used together as input data. Lastly, the data loading process was discussed. Data is first sequenced into corresponding input-output pairs. These sequences are then split into three separate datasets for training, testing and validation. To divide the data in these different datasets in batches and feed it to the models, a PyTorch DataSet and DataLoader object are used.

### 3 Model designs

In this chapter, all models will be discussed that were created. An iterative process was used where each model design increases in complexity in order to potentially . As discussed in the introduction, eventually hybrid models will be used to obtain optimal performance with the given data and problem definition. To visualize the model designs, illustrations were made and added for each model. In addition, a table is provided with the specific parameters of each model. To identify different models based on their architecture, the following naming conventions are introduced. This nomenclature will also be used in the parameter tables and on the architecture illustrations.

Notation	Explanation
<b>P – R – P   R – PR</b>	Resp. Pitch - Roll - Pitch OR Roll – Pitch AND Roll
<b>LSTM</b>	An LSTM architecture was used
<b>CNN</b>	An CNN architecture was used
<b>FC</b>	Fully connected linear layer
<b>Single-/Multi-step</b>	Single value/sequence of output values
<b>N</b>	Number of frames in the input sequence
<b>M</b>	Number of frames in the predicted output sequence

Table 5: newly introduced notations and their explanations

#### 3.1 Single-step stacked LSTM

First off, a multivariate single-step model was designed and implemented. This model uses a sequence of numeric input data for pitch and roll and predicts one future step - hence 'single-step' - of either pitch, roll or both. This model was designed to get familiar with the workflow of designing, training and testing neural networks with PyTorch. For simplicity, image data was omitted to allow for a simple model structure with has few parameters that can be trained and tested quickly. In addition, since their output is single step, they were also not optimized to their full extend because they don't meet the sequence output requirement. Two variants of this model were trained. They have identical architectures with the only difference being that one variant can predict pitch *and* roll at the same time while the other can only predict *either* pitch or roll. This is reflected by the number of neurons in the output layer shown in Table 6 where *out* is equal one or two neurons for resp. the single or dual output variant.

Input	Sequence of PR at ( $2 \times N$ )					
Name	Layers	Number of layers	Hidden size	Dropout	Input	Output
LSTM	LSTM	2	128	0.2	$2 \times N$	$2 \times 128$
Regressor	Linear				$1 \times 128$	$1 \times \text{out}$
Output	Single value for P R or PR					

Table 6: parameter table for single-step models

The single-output model consists of an LSTM architecture followed by a linear fully connected regressor layer. The general idea behind this design is that the LSTM creates a hidden vector which is then used to calculate the predicted output(s). This hidden vector represents the LSTMs short-term memory of the sequence. To calculate this hidden vector, a two-layered or stacked LSTM is used. In this configuration, the output of the first LSTM is passed to the inputs of a second LSTM. This configuration was chosen based on the research of Cui Z. where performances of both stacked and single layered LSTMs were compared in a similar forecasting context (Cui et al., 2020). The conclusion was made that the stacked LSTM performed better than its single layered counterpart. Both LSTM layers use 128 hidden LSTM cells (neurons) and have a dropout of 0.2. These values were arbitrarily chosen and proved to achieve good performance. Dropout is a function that randomly disables certain

neurons in the network. The value for dropout represents the chance of any given neuron to be disabled. Using dropout helps to reduce overfitting during training and thus improve stability and performance (Srivastava et al., 2014). Dropout is applied when data is passed between the two LSTM layers. The concept of overfitting is discussed later on in 4.1. The last layer is a linear fully connected layer, aggregating all the hidden features into a single output feature. Notice that the hidden output of a layered LSTM is a three-dimensional matrix with the following shape: (number of layers, the batch size, number of hidden neurons). Basically, the two-dimensional hidden vector of each layer is appended one after the other in chronological order, numbered by their index. For the aggregation in the linear layer, only the hidden state from the last layer is used, hence the notation *hidden[-1]*. The single-step stacked LSTM model architecture is illustrated in Figure 17 with the single output variant on the left.

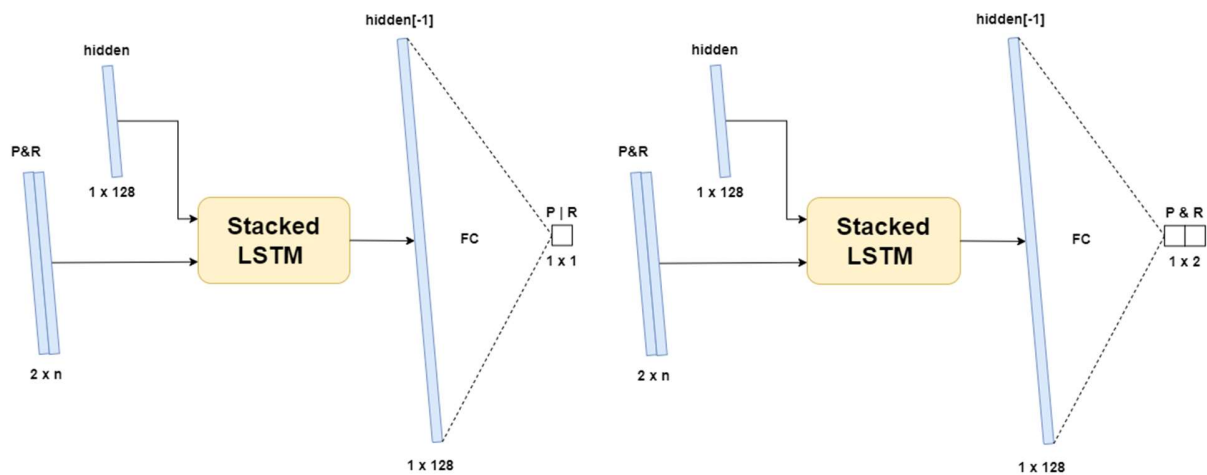


Figure 17: First generation LSTM model architectures: single output (left), multi-output (right)

Due to its simplicity, this model also served as a testing ground to find good values for some important recurring parameters such as the size of the hidden layer and learning rate. The model was trained with different values for these parameters to form a conclusion on what values worked well. From the results of this testing, later model designs could use similar hidden sizes and learning rates in assumption that they will behave similarly. This way, more efforts could be put in testing other parameters such as input and output sequence lengths. A more detailed discussion on these parameters is provided in 4.1.

## 3.2 Multi-step models

The next step in development was to design models capable of predicting a sequence of pitch and roll. In addition, images are also introduced as additional input. The combination of these two expansions results in more complex model architectures. In the next sections, four new multi-step model architectures are introduced.

### 3.2.1 Encoder-Decoder LSTM

The first model that was designed for multi-step predictions was an expanded version of the single-step model. The two-layered LSTM was replaced with two single-layered LSTMs in an encoder-decoder configuration. This configuration was inspired by a similar model of Kaminskyi, and multiple examples found online like the one by Brownlee (Brownlee, 2018). This architecture was chosen to be implemented first because of its relatively simple design and because of its performance. It performed worst out of all tested models in Kaminskyi's research, therefore it should be a good starting point to iteratively improve upon.

When tasked with multi-step prediction, the idea is that the first LSTM encodes a latent vector together with a hidden short-term memory vector which are then decoded by the second LSTM to generate an output sequence. With this model, it was possible to evaluate how well a neural network could predict pitch and roll without having the images of incoming waves. If

this model would perform similarly to one that uses images as an additional input, the conclusion could be made that, if necessary, the images could be omitted. In this case, this model would provide a more lightweight solution, requiring only numeric data.

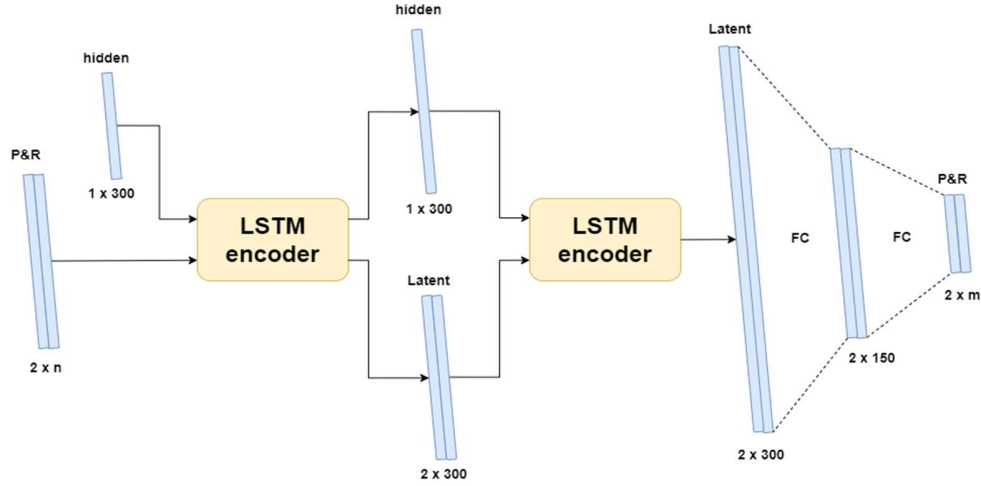


Figure 18: LSTM encoder decoder architecture

Figure 18 shows the architecture of the implemented model. As mentioned above, two LSTMs are used together where the first one acts as an encoder followed by a second decoder LSTM. The first LSTM receives the input sequence of pitch and roll values and has a hidden size of 300. When the data is passed through this network, an encoded latent vector is output together with the hidden state of this first LSTM layer. The second LSTM layer uses this hidden and latent vector and decodes it into a new latent vector. Finally, the latent vector from the decoder is passed through two linear layers that aggregate the vectors into a sequence for pitch and roll.

Input	Sequence of PR at ( $2 \times N$ )					
Name	Layers	Number of layers	Hidden size	Dropout	Input	Output
<b>LSTM component</b>						
Encoder	LSTM	1	300	-	$2 \times N$	$2 \times 300$
Decoder	LSTM	1	300	-	$2 \times 300$	$2 \times 300$
<b>Linear component</b>						
FC 2	Linear				$2 \times 300$	$2 \times 150$
FC 2	Linear				$2 \times 150$	$2 \times M$
Output	Sequence of PR at ( $2 \times M$ )					

Table 7: parameter table for encoder-decoder LSTM

The parameters for the encoder-decoder LSTM network are shown in Table 7. The LSTM hidden vector sizes were chosen equally to Kaminskyi's model at 300. This was done to get a direct comparison between the models on one hand and because the tests with the single step model resulted in better performance with a hidden size larger than its original 128 on the other hand. Lastly, a second linear layer was added to the end of the model. This was also done to keep the model comparable to Kaminskyi's model.

### 3.2.2 Sequential CNN

After the single- and multi-step LSTM oriented models - that only use numeric data - convolutional neural networks were introduced. With their introduction, images could also be efficiently processed and used to make predictions. Firstly, a



sequential CNN was implemented. On one hand, this simple model was created to familiarize with the new CNN network architecture, its parameters and the image data loading process. On the other hand, this model was created to evaluate performances when using solely convolutional and linear layers and no LSTM module. Without an LSTM module, performance of this model is expected to be lower than other models. But it might still be better than previous LSTM models that only use numeric data. Additionally, generally speaking, LSTM networks have a high inference time due to their complexity (Mealey & Taha, 2018). Because of its absence in this model, lower inference time is expected which might make up for possible loss in performance.

Each image of an input sequence is processed individually by a CNN. The resulting  $N$  tensors are then flattened into a one-dimensional vector. These feature vectors are then concatenated into one single large vector which serves as the input layer for the first linear layer. Because the vectors are appended in order, the sequence order is kept. Three consecutive linear layers then aggregate this larger vector into smaller vectors and eventually into an output sequence of pitch and roll. The architecture for this model is illustrated above in Figure 19 together with its parameter table in Table 8.

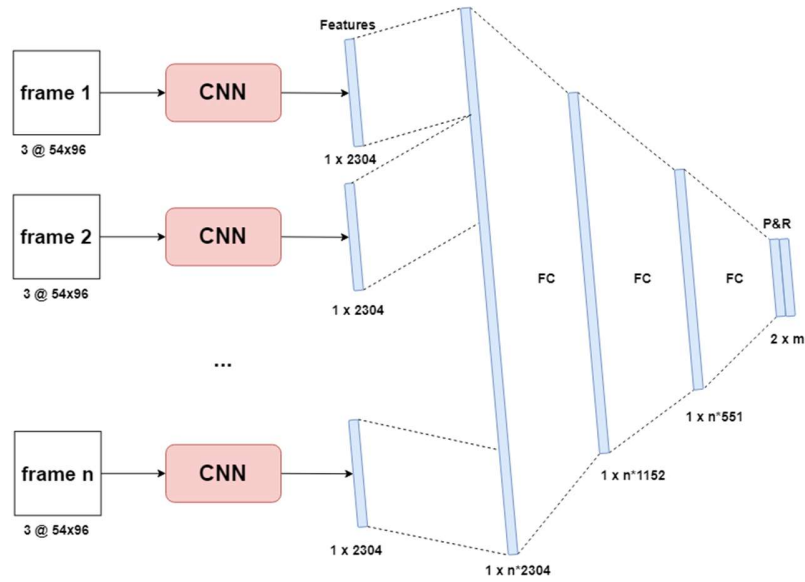


Figure 19: sequential CNN neural network architecture

Three consecutive convolutional layers are used to extract features from the images. The first layer uses a 5x5 kernel to extract larger features. The other two layers use a 3x3 kernel to extract more detailed information. The channels follow a doubling pattern where each layer doubles the numbers of channels by a factor of two. Each layer is followed by a rectified linear activation and max pooling. This was done following the conventions of building CNN networks. A majority of these parameters were also based on the CNN encoder architecture of Kaminskyi's work as it showed good performance and was used in his best model. The size of the linear layers follow a decreasing trend to slowly aggregate the large vector into predicted outputs.

Input	Sequence of N images at (N×3×54×96)						
CNN component							
Name	Layers	Out channels	Kernel Size	Stride	Padding	Input	Output
Conv 1	Convolution	8	5x5	1	2	3 x 54 x 96	8 x 54 x 96
	ReLu	-	-	-	-		
Pooling 1	Max Pooling	1	2x2	2	1	8 x 54 x 96	8 x 27 x 48
Conv 2	Convolution	16	3x3	1	1	8 x 27 x 48	16 x 27 x 48
	ReLu	-	-	-	-		
Pooling 2	Max Pooling	1	2x2	2	1	16 x 27 x 48	16 x 13 x 24
Conv 3	Convolution	32	3x3	1	1	16 x 13 x 24	32 x 13 x 24
	ReLu	-	-	-	-		
Pooling 3	Max Pooling	1	2x2	2	1	32 x 13 x 24	32 x 6 x 12
Linear component							
FC 1	Linear					1 x N*2304	1 x N*1152
FC 2	Linear					1 x N*1152	1 x N*576
FC 3	Linear					1 x N*576	1 x M*2
Output	2 x M						

Table 8: parameter table for sequential CNN model

### 3.2.3 CNN LSTM single input

Now that the two main proposed neural network architectures – CNN and LSTM – were both individually implemented in different models, an ensemble model was made composed of both architectures. Many of the design concepts of the previous sequential CNN model were kept the same for this model. The consecutive input images are processed individually by a CNN into  $N$  feature vectors. These different feature vectors are then concatenated into a large single vector. However, this vector is then used as input for an LSTM encoder instead of a linear layer. Since the sequence of the images is kept intact during the concatenation as previously mentioned, the LSTM can use its long- and short-term memory properties and should achieve higher performance than linear layers.

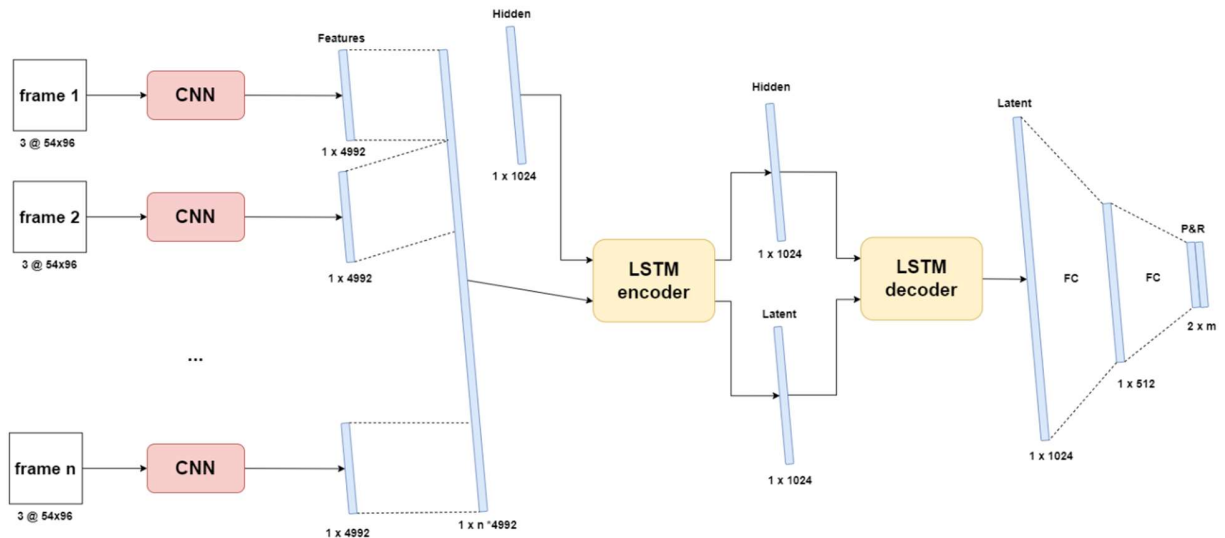


Figure 20: CNN LSTM single input architecture

The first LSTM encodes the large feature vector into a latent and hidden vector. These vectors are then passed to a second decoder LSTM which produces the output. The output of this decoder is then aggregated into a sequence off pitch and roll output by two linear layers. Note the *single input* in the model's name to indicate that only images are used as input for this

model. A second model was designed as well that allows for dual input. This way, the performance of all three input configurations could be measured: pitch and roll only, images only and both inputs.

The parameters for the single output CNN LSTM model are shown in Table 9. The CNN component is identical to the one from previous model. However, the last convolutional, ReLU and Max Pooling layers were removed. This was done in order to reduce the complexity and the number of parameters of the model to provide a more lightweight solution. The LSTM component is set up equal to the encoder-decoder model from 3.2.1, aside from its increased hidden size. This was done to compensate for the much larger input vector of the encoder LSTM.

Input	Sequence of N images at ( $N \times 3 \times 54 \times 96$ )						
CNN component							
Name	Layers	Out channels	Kernel Size	Stride	Padding	Input	Output
Conv 1	Convolution	8	5x5	1	2	3 x 54 x 96	8 x 54 x 96
	ReLu	-	-	-	-		
Pooling 1	Max Pooling	1	2x2	2	1	8 x 54 x 96	8 x 27 x 48
Conv 2	Convolution	16	3x3	1	1	8 x 27 x 48	16 x 27 x 48
	ReLu	-	-	-	-		
Pooling 2	Max Pooling	1	2x2	2	1	16 x 27 x 48	16 x 13 x 24
LSTM component							
Name	Layers	Number of layers	Hidden size	Dropout	Input	Output	
encoder	LSTM	1	1024	-	1 x N x 4992 (+2)	1 x 1024	
decoder	LSTM	1	1024	-	1 x 1024	1 x 1024	
Linear components							
FC 1	Linear					1 x 1024	1 x 512
FC 2	Linear					1 x 512	1 x M*2
Output	2 x M						

Table 9: parameter table for CNN LSTM single input and dual output (red)

### 3.2.4 CNN LSTM dual input

The final model was a slight variation on the previous model which allows it to use both numeric data and images as input. It was built based on the concept of the previous CNN LSTM single input model. The main difference is that pitch and roll values for each frame – a frame being an image and its corresponding PR values – are appended at the end of each image's feature vector before they are concatenated into one large vector. This way, both the image and PR sequence order are preserved for the LSTM. Correct normalization is of utmost importance with this model as one type of data may overshadow the other if they are not normalized to the same window.

Expectations for this model are high as it has all available data at its disposal to make as accurate of predictions as possible. However, this model requires the most amount of processing, so inference time will probably be higher than other models. This is mainly due to the process of building the feature vectors and append the pitch and roll tuples to them. This is an expensive operation in terms of computing resources and might cause this model's inference time to be too high. The illustrated architecture is shown in Figure 21. Since the model's only difference is found within its internal functionality for the feature vector concatenation phase, the parameters of Table 9 still apply with the only change being the extra two datapoints added to each feature vector. This change is marked in red. Additionally, the values of the image feature vector are activated with a tanh. This is done to project them back into the domain of the PR-values that are appended. If this was not the case, the values of the appended PR-pairs might be overshadowed by the potentially larger values in the feature vector.

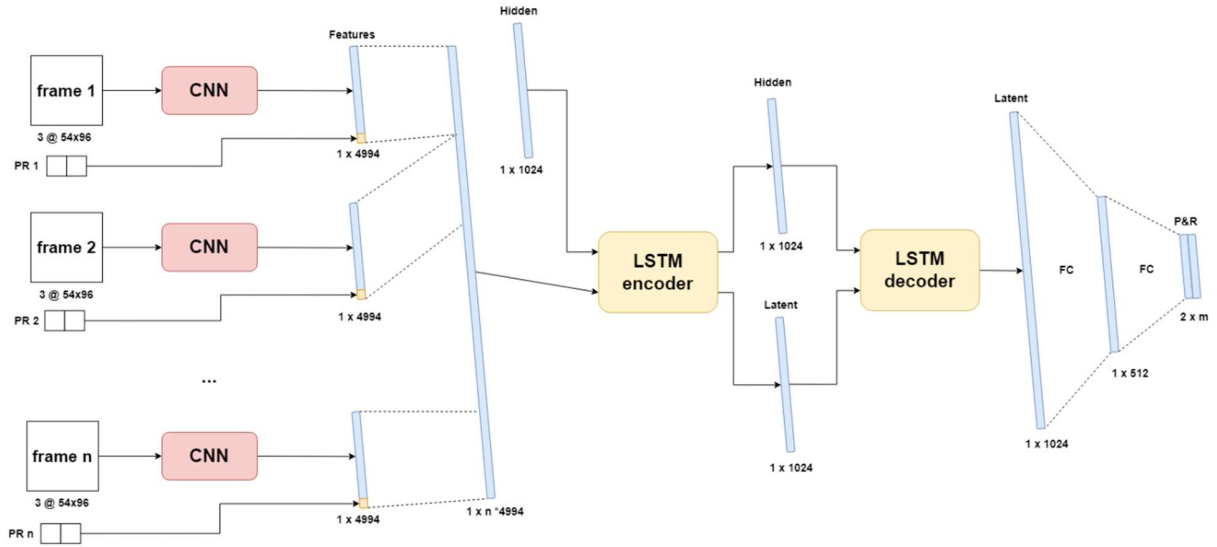


Figure 21: CNN LSTM dual input architecture

### 3.3 Summary

In this chapter, all proposed models and their parameters were discussed. An iterative process was followed where each next model increases in complexity and capabilities in hope of receiving better results. First off, a simple single-step model was implemented to acquire experience with PyTorch and deep learning. Two variants were trained, one for predicting either pitch or roll and one to predict both. They also served as a platform to quickly test recurring parameters to find optimal values for future models. Next, four multi-step models were introduced. These are models that are able to predict ordered sequences of output. Each model represents a different approach based used input data and architectures. One model only uses numeric data, the next two models only use images and the last uses both. Besides this one model was also built with an LSTM module to assess its impact on performance.

## 4 Testing environment

To evaluate each model's performance and match it to the criteria, a testing environment is needed. This environment determines what parameters will be used to train the models and which metrics will be used to measure performance. In the following sections, all components that make up the testing environment are discussed.

### 4.1 Hyperparameters

An important part to any deep learning problem is defining the hyperparameters and finetuning them for optimal performance. A hyperparameter is a parameter that can be manually set by the programmer. They affect the behavior of the model during training and thus have a direct influence on how well the model will perform afterwards. They are different from the internal parameters of the network such as a neuron's weights and bias, whose values are derived during the training process. Many different hyperparameters exist, each having their own function but not all of them need to be used in every model. Below, each hyperparameter that was used in this research is discussed.

- **Train-test-validation ratio** is used to control how much of the original data is allocated to each of the three datasets discussed in 2.5.2. The proportions of each subset are relative to all available data points, i.e., IO-sequences. For most of the research, an **80% - 10% - 10%** split was used for respectively train, test and validation data.
- **Learning rate** refers to the rate that an algorithm converges to a solution. For a deep learning neural network, it determines the size of each step during the gradient descent. High learning rates cause the model to converge very quickly to the local minimum but come with the risk of overshooting the minimum. On the other hand, low rates may cause the model to not reach convergence by the end of training. An initial learning rate of **0.001\*** was used for the Adam optimizer.
- **Batch size** determines how many datapoints are processed at one forward pass during training. A large batch size means that a lot of training data is considered at once. This increases memory usage and lowers the number of mini-batch gradient descent backpropagation steps (Shen Kevin, 2018). Therefore, a higher learning rate should be applied to assure that the local minimum is reached within these fewer steps. A small batch size has the opposite effects. A batch size of **64** was used across all models. This is a relatively low batch size, but it was chosen because of memory limitations when working with images.
- **Number of epochs** or the number of training iterations, determines how many times the model may process all training data. During one epoch, all batches of the training data are processed once to update internal parameters. When the model is trained for more epochs, it has more chances to find the best internal parameters. However, when the model is trained for too many cycles, it might start to learn the detail and noise in the training data to such an extent that it start to negatively impact the performance on new data. By using an unseen validation dataset, overfitting can be limited. When the validation error starts to increase while the training error is still decreasing, the model is overfitting and training should be stopped. All models were trained for **50\*** epochs.
- **Data frequency** defines at which frequency data is fed as input and at which frequency data is predicted by the model. The simulation data was generated at **2Hz** or 2 frames per second as a good balance between having enough detail without having consecutive frames containing too similar data. With up sampling being impossible, the assumption was made that down sampling to a lower frequency was also not beneficial as this would reduce the detail in the predictions by 50%– which is not desired.
- **Input sequence length** sets the number of frames used as input to make a prediction upon. This hyperparameter had no fixed value and was **chosen on a per model basis**.
- **Output sequence length** sets the number of datapoints the model needs to predict. An output length of **60 frames** at two frames per second was chosen as the minimum. This way, performance of each model could be measured over a 30 second prediction window as discussed in the objectives and criteria.

The last two hyperparameters, input and output sequence length, will have the most impact on the performance of the model. They will be tested the most to find the best configuration for each model. Ideally, only a short input sequence is needed to make an accurate prediction over a long sequence length. Due to its importance and for readability, the ratio of input to output sequence length will be defined as the **IO-ratio** (input/output-ratio). The notation of the IO ratio will always be in non-simplified non-fractional form, i.e., a 30/60 IO-ratio will never be written as  $\frac{1}{2}$  or 0.5 to avoid losing sight in the effective sequence lengths at hand.

As a final note on the hyperparameters, finding the optimal value for each hyperparameter is called hyperparameter optimization. Different algorithms exist to perform this optimization (Claesen & de Moor, 2015). However, they are very expensive and time-consuming operations, especially when many different models are at hand. Optimizing all models would be even more time-consuming. An alternative approach was used in this thesis where only some parameters were tested to find optimal values. These parameters were tested on the simple single-step model. Once an optimal value was found, it was then used for all models in assumption that they would behave similarly. This way all models could be compared equally with roughly estimated optimal hyperparameters. The experimentally found values are marked with an asterisk (\*). Complete testing with results will be discussed in the next chapter.

## 4.2 SGD Adam optimizer

As discussed in 1.3.1, different algorithms exist for gradient descent, namely batch, mini-batch and stochastic gradient descent. These algorithms are used to calculate the weights and biases for the neurons in the network by iteratively taking steps in the direction of the minimum on the gradient. The size of this step is determined by the learning rate which is fixed for the standard implementation of the above three methods. However, variants or optimizers exist to dynamically change the value of the learning rate and optimize the training process. In general, the learning rate should start high and decrease as the model gets closer to the local minimum on the gradient. This way, the model will converge quickly without overshooting or diverging from its optimal state.

Adam is an optimization algorithm for stochastic gradient descent (SGD). While normal stochastic gradient descent maintains a single learning rate for all weight updates which does not change during training, Adam applies an adaptive learning rate that is computed for each network based on estimates for the first and second moments of the gradient. It combines the advantages of two other optimization techniques: AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance) (Brownlee, 2017). According to (Kingma & Ba, 2014), Adam is computationally efficient, requires little memory and is invariant to diagonal rescale of the gradients.

The choice of the optimizer is based on which gradient descent algorithm will be used. Within PyTorch, selection of an algorithm is very easy and therefore, the best optimizer was chosen based on the current state-of-the-art. Adam is currently one of the best optimizers and a very popular option. The method is really efficient when working with problems involving a lot of data or parameters (Doshi Sanket, 2019). For this reason and because of its easy implementation in PyTorch, Adam was used as optimizer for all models.

## 4.3 Performance metrics

### 4.3.1 Loss function

To compare the ground truth with the predictions made by the model, a metric is used to calculate the error, i.e., the loss function. Many distinct functions exist to define this loss, each with their own appliances. For continuous data and predictions, the Mean Squared Error (MSE) is a popular option (Wang et al., 2022b). The MSE of two sets of points of  $n$  elements (read number of predicted frames), is defined as the average of the squared errors, where the error corresponds to the difference between ground truth  $Y_i$  and prediction  $\hat{Y}_i$ .

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Equation 3: MSE loss function

During training, the MSE was used as loss function for gradient descent. It calculates the average error over the whole predicted sequence. For example, if sixty values are predicted, the MSE will return one value: the average error over these sixty points. MSE will also be used to compare different models' performances to each other. However, this is quite a general comparison because some interesting information is lost when the difference between two sequences is averaged into one value. For a more detailed comparison and evaluation of the predictions, an extra metric was introduced: loss per frame (LPF). The LPF of two sequences of length  $n$  is defined as a sequence of  $n$  errors where each error is the difference between the two sequences at the corresponding index. Returning to the first example, if sixty values are predicted, the LPF will return an array of sixty elements containing the error between each pair of predicted and real values individually. When this metric is averaged over all predictions made on an unseen test dataset, the result can be used to analyse how the error changes on average throughout the predicted sequence. Additionally, if the root of the MSE is taken – the RMSE – the errors are projected back to the units of the original data and can be de-normalised into their original domain. Using this principle, all MSE values can be translated back into their corresponding pitch and roll angles – which provides a more sensible metric for comparison than MSE.

While being a very simple yet effective solution, MSE does not handle sequential trends very well (le Guen & Thome, n.d.). Since only two points are compared at a time, the overall trend of the prediction and real values is not considered. This shortcoming of MSE was found during testing and is displayed in Figure 22. The two highlighted pairs of datapoints have a similar MSE-error. However, the first pair is a far worse prediction than the second pair since it follows a trend in the opposite direction. Whereas the second pair does follow the trend and should not be considered as an error with the same magnitude of the first pair. As a result of this, the LPF metric can in these cases not be as unambiguous as desired

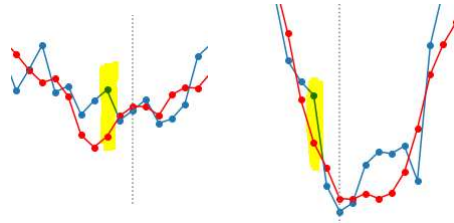


Figure 22: two different scenarios illustrating the shortcoming of MSE

#### 4.3.2 Inference time

In the cases where a deep learning model needs to be deployed in real world scenarios to make predictions on completely new data, the speed at which it can make those predictions can be of significant importance. For example, self-driving cars with limited computational resources need fast, low-latency models. This can be a challenging metric to fully optimize when high demand architectures are used such as LSTM networks (Kouris et al., n.d.). The target vessel will similarly to the self-driving cars have a limited amount of computational resources. Because of this, each model will be subjected to a measurement of its inference time – the time it takes to make one forward pass prediction. Besides having a low error across predictions, having a low inference time is equally important. To measure true inference time, all outside influences should be eliminated. When executing on a GPU, factors like GPU warm-up, asynchronous execution and system hardware specification all need to be considered. As discussed in 1.6, inference time will be measured as the average prediction time over ten thousand single batch predictions.

**GPU warm-up** is a process in which the GPU is initializing. Modern GPUs have multiple power states to reduce their power usage. When the GPU is not used, it will go into a power saving state and potentially completely turn off. In lower power state, the GPU shuts down different pieces of hardware, including memory subsystems, internal subsystems, or even compute cores and caches. A program attempting to interact with the GPU will cause the driver to load and initialize the GPU. It is this driver load that can influence the measurement of inference time. According to (Geifman, 2020), programs that cause the GPU to initialize can be subject to up to three seconds of latency due to the scrubbing behavior of the bit error correcting code (ECC). Memory scrubbing is the process of reading data from each memory location, correcting faulty bits with an ECC, and writing the data back to its original place. Below in Figure 23 the effects of GPU warm-up are clearly visible. A big spike in execution time is measured on the predictions when the GPU is still initializing. When the GPU is initialized, the timings decrease and remain constant. It is important that this initialization period is not included in the inference time calculation because it is inaccurate and does not reflect a production environment where the model is continuously making real-time predictions on an initialized GPU. To compensate for GPU warm-up, a broad estimate of one thousand predictions are executed without time measurements.

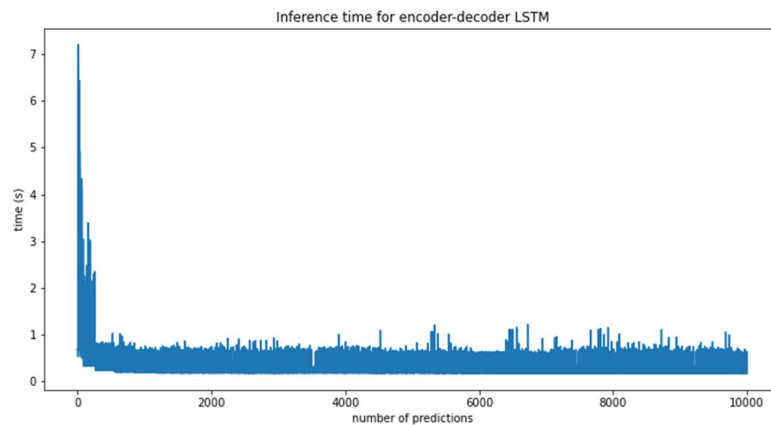


Figure 23: inference time over 10,000 prediction with GPU warm-up effect

**Asynchronous execution** is mechanism that occurs on multi-threaded or multi-device programming. Different tasks are scheduled to different execution units which execute them asynchronously. This can cause a block of code to be executed before the previous one is finished and therefore speeds up execution. However, if a neural network is executed on the GPU – which is asynchronous by default – and inference time is measured on the CPU, the timer will stop before the model is done with its execution. This needs to be taking into consideration when certain popular Python libraries are used like *time* for example, which is executed on the CPU. PyTorch provides a very similar version of the popular *time* library designed for measuring and synchronizing events on a cuda-enabled device.



```

def inference_time(model, dummy_input, repetitions=10000):
    device = torch.device("cuda")
    model.to(device)
    dummy_input.to(device)

    # INIT LOGGERS
    starter = torch.cuda.Event(enable_timing=True)
    ender = torch.cuda.Event(enable_timing=True)
    timings = np.zeros((repetitions, 1))

    # GPU WARM-UP
    for _ in range(1000):
        _ = model(dummy_input)

    # MEASURE PERFORMANCE
    with torch.no_grad():
        for rep in range(repetitions):
            starter.record()
            _ = model(dummy_input)
            ender.record()
            # WAIT FOR GPU SYNC
            torch.cuda.synchronize()
            curr_time = starter.elapsed_time(ender)
            timings[rep] = curr_time

    mean_syn = np.sum(timings) / repetitions
    std_syn = np.std(timings)

    return timings, mean_syn

```

*Listing 1: function definition for inference time measuring*

Lastly, the **system hardware specifications** on which the measurements were recorded are mentioned for reference. All measurements were taken on a system with the following specifications:

- CPU: Intel Core i7 10700k at 3.8GHz base clock and 4.6GHz boost clock
- GPU: Nvidia GeForce RTX 3080 with 8704 cuda cores at 1800MHz boost clock
- RAM: 3200MHz

## 4.4 Summary

In this chapter, the testing environment was discussed. This environment encompasses all relevant variables and methods which have direct influence on the performance and evaluation of the models. By fixing most of these variables and methods across all models, the effect of specific parameters can be accurately measured, and models can be compared fair and equally. In the first section all hyperparameters are discussed in combination with their value. Some of these hyperparameters were one-and-done choices, others were determined based on experimentation on the single-step model. In the second section, the training optimizer was discussed. To optimize the gradient descent process during training, Adam was used. Adam is an optimizer for the stochastic gradient descent algorithm that has high performance compared to other optimizers. It dynamically changes the learning rate to achieve faster and better convergence. Adam is also very easy to implement on PyTorch. Lastly, the error metrics were discussed. To measure performance, MSE loss, LPF and inference time are used. The first two are functions to calculate and visualize errors and their trend across a predicted sequence. The inference time is used to evaluate the latency that occurs between input and prediction. GPU warm-up, asynchronous execution and system hardware specification all play a role when measuring inference time.

## 5 Results

In this chapter, an extensive overview is given of all tests that were performed and their results. In first section, the results from the hyperparameter optimization are discussed. Afterward, each model will be discussed in detail, followed by a section on inference time results. In the last section, a final overview and comparison is provided of the best performing models. To analyze the accuracy of the models, mainly the MSE, LPF and graphs showing the real vs. predicted values were used. As a baseline, each model is also compared to the *zero-predictor* as discussed in the criteria.

### 5.1 Hyperparameter optimization

Three hyperparameters were optimized to estimated optimal values: LSTM hidden layer size, number of epochs and learning rate. It was hypothesized that these three parameters would have the most influence on the performance of the models and thus a simple method was applied to optimize them. To find the best parameters, three configurations of each parameter were tested. Firstly, an initial value is tested. This value is chosen to be somewhat in the middle of commonly used values. Then, two others are tested, one higher and one lower value. With the results of these three tests, the optimal value is chosen based on the trend they form. The resulting value is then used for all other models in assumption that they will show a similar behavior. All tests were executed on the single-step LSTM model due to its low complexity and thus fast training time. The dual output variant was used with a 50/10 IO-ratio.

#### 5.1.1 Number of epochs

The number of epochs determines how many times the model can process the full training dataset to optimize its internal parameters. Low training epochs allow for fast training but might lead to the model not reaching its optimum. High epochs cause longer training times but give the model more time to converge. To define the optimal value, the test model was trained for 8, 25 and 50 epochs and the training loss graphs were compared (Fout! Verwijzingsbron niet gevonden.).

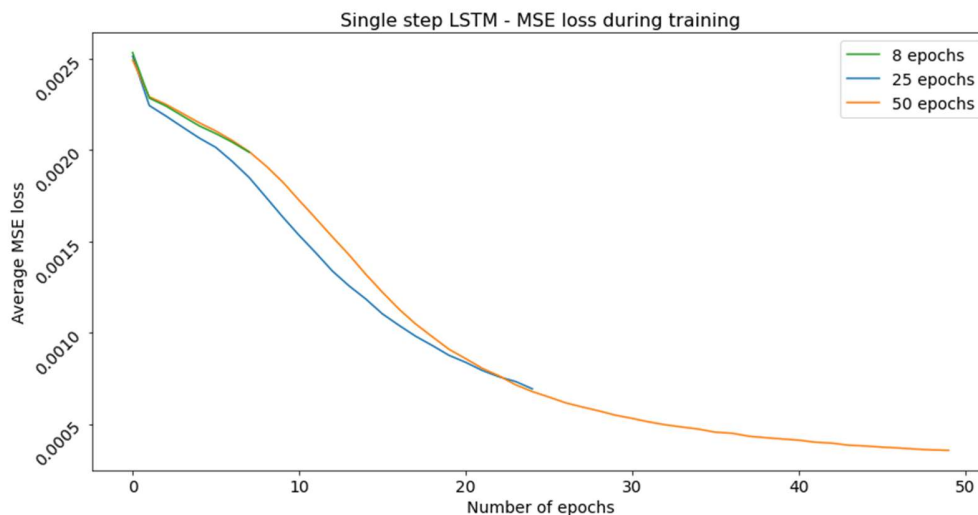


Figure 24: training losses for different numbers of epochs

During training, this loss should rapidly decrease on the first couple of epochs and ease out to a point after which the loss remains constant. At that point, the model has learned the mapping between input and output to the best of its abilities and receives no more benefits from additional epochs. It is clear that the model needs at least 50 epochs to properly train itself and achieve the lowest MSE. However, the model is still marginally decreasing during the final epochs so a higher number of epochs would probably be even better. Nonetheless, 50 was chosen as the optimal number for all future training in

assumption that the increase in performance after 50 would be insignificant compared to the increase in training time. Especially when training the more complex models.

### 5.1.2 Learning rate

Learning rate was the second parameter that was optimized. Learning is a very hard to conceptualize hyperparameter. It is defined as the size of the step that is taken during one iteration of moving down the gradient towards the minimum. During research of this parameter's value in other deep learning applications, most cases used a learning rate of 0.01, 0.001 or 0.0001. Learning rate is very much dependent on the complexity of the problem. Simple problems may get away with high learning rates resulting in very fast conversion, but on complex problems such as this one, a smaller learning rate is recommended (Wilson & Martinez, 2001). The three above mentioned values were tested in order to find the optimal value. Once again, the training loss graph is used to discuss the results (Figure 25). It is important to note that these tested values were used as initial learning rates with the Adam optimizer. As discussed in previous chapter, Adam will dynamically change the learning rates. However this does not mean that the initial value is less important.

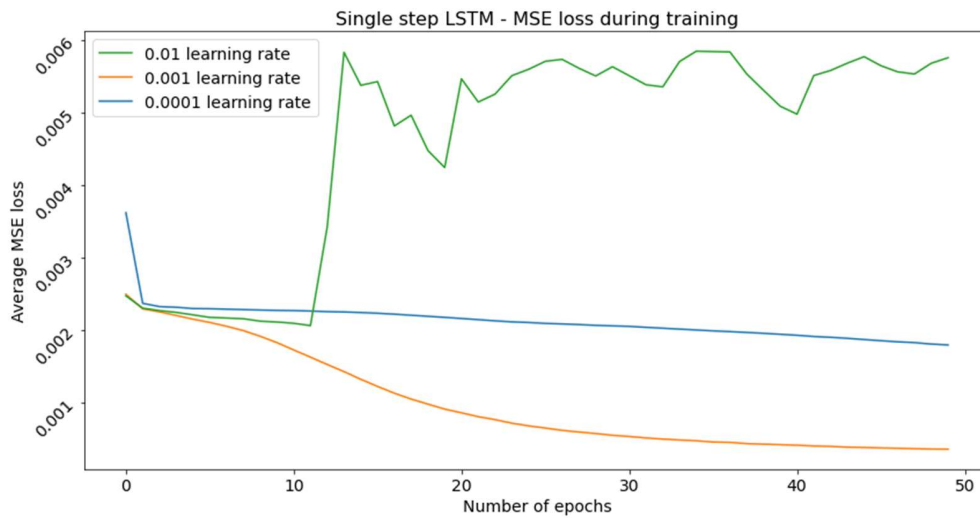


Figure 25: training loss for different learning rates

The model shows very interesting but not unexpected behavior. Based on the mathematical theory for learning rate, a learning rate that is too high will result in very drastic changes to the internal parameters, causing divergent behavior. This is clearly what is happening with the model on the green line of with 0.01 learning rate. The model is basically bouncing around on the gradient without really converging to the minimum at all. The opposite is visible on the blue line. In this case the learning rate is smaller than ideal as the model is converging very slowly compared to the orange line. Due to the smaller steps, the model needs more iterations to reach the minimum. If the current rate is projected forward, convergence will require somewhere between five to ten times more epochs. The model on the orange line and the model on the blue line will probably reach similar performance in the end. However, the orange one will be much more efficient which is why a learning rate of 0.001 was selected as the value for all future models.

### 5.1.3 LSTM hidden size

After determining the learning rate and number of epochs, the hidden size for LSTM modules was assessed. Judging based on the nature of our problem: sequence prediction and the efficiency of the LSTM module for this type of problem, it was concluded that the hidden size of the LSTM modules would be the most important parameter to optimize. The size of linear layers, and parameters of the CNN modules were assumed to be less impactful on performance. In addition, the parameters for the CNN were already tested and proven effective in Kaminskyi's research. Initially, a hidden size of 128 was chosen for

the single step LSTM model. To find the direction of the optimal hidden size, a lower 32 and higher 256 hidden size were tested. The results are shown in Figure 26.

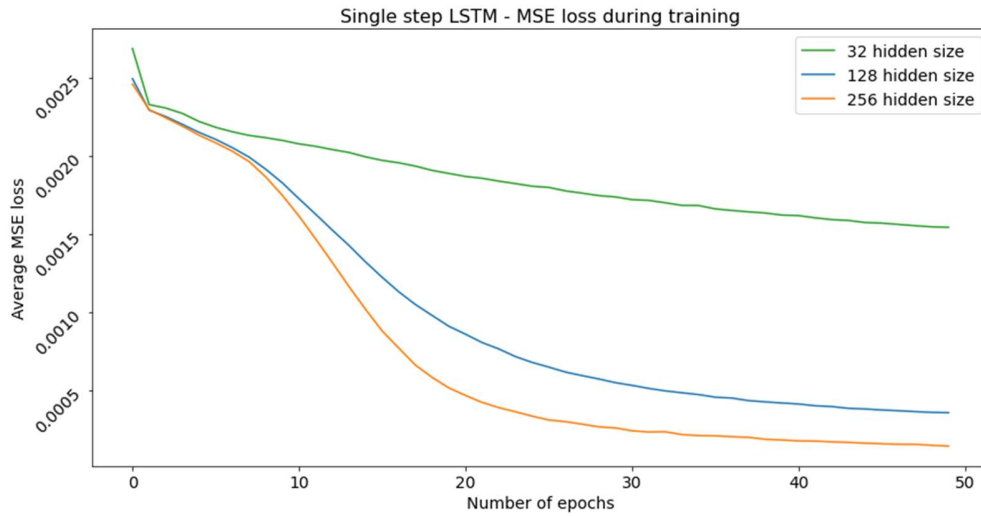


Figure 26: training loss for different LSTM hidden sizes

Once again, the model shows expected behavior. Generally speaking, when increasing the nodes of hidden nodes, the network is able to learn a more powerful function between input and output. In the case of LSTM networks where memory cells are used as hidden nodes, increasing their number allows the model to better remember long term dependencies simply because there are more parameters to represent them. This is visible on the graph where the model converges to the lowest MSE with the highest number of hidden nodes. From this trend, the conclusion was made that larger hidden sizes should be used. For the encoder-decoder LSTM, the hidden size was increased to 300 following this trend. For the CNN LSTM ensemble models, the hidden size was increased even further to remain in line with the much larger input vector.

#### 5.1.4 Activation functions

As a final general-purpose optimization, the effect of activation functions was assessed on the single-step model. The single step model has three logical points in its architecture where activation functions can be placed: one to activate the input before the LSTM module (1), one after the LSTM (2) and one after the linear layer (3) which basically activates the output. Since the model trains with normalized data between  $[-1, 1]$  and is expected to return values in this domain, a tanh is the only possible activation usable on the output layer. The activation on the input should consider this domain as well. A ReLU here would be ineffective as it nullifies all datapoints lower than zero. In the case of ship motion with angular data, negative and positive data are equally important. This leaves the second option as the most interesting point for an activation function.

Two configurations were tested on the single step model: one with a ReLU at (2) and one with a ReLU at (2) and a tanh at (3). The results are shown in Table 10. Each configuration was trained for 50 epochs with a 0.001 learning rate. The results of this experiment were very similar, almost identical. The introduction of activation functions resulted in no measured benefit. From this experiment it was concluded that activation functions were not beneficial to performance and should not be used as they potentially could introduce additional latency.

Configuration	Average Pitch error [denormalized RMSE]	Average Roll error [denormalized RMSE]
Barebones model	2.18°	2.06°
ReLU (2)	2.15°	2.06°
ReLU (2) + Tanh (3)	2.22°	2.04°

Table 10: average pitch and roll errors for different activation function configurations

## 5.2 Single-step model

In this subchapter, the two single-step model will be analyzed on prediction accuracy. Three different variants were trained: one single-output variant for each pitch and roll and one dual-output variant for simultaneous pitch and roll predictions. These models should perform very well as they only need to predict one value. The three models were trained with an input sequence length of both fifty and ten PR-values to identify their optimal IO-ratio. Below in Table 11, the results are shown for each model in each configuration. Note that error values in the table are calculated as the denormalized RMSE averages over all predictions made on the unseen test data set.

Variant	Epochs	IO-ratio	Average Pitch error [denormalized RMSE]	Average Roll error [denormalized RMSE]
Pitch	50	50/1	1.85°	/
Roll	50	50/1	/	1.91°
Dual	50	50/1	2.17°	2.03°
Pitch	50	10/1	2.28	/
Roll	50	10/1	/	2.25°
Dual	50	10/1	2.34°	2.09°
Zero prediction			6.43°	7.30°

Table 11: denormalized RMSE for single-output LSTM models in different configurations

On the first three grey rows, the results are shown for the models that were trained with an input of fifty frames. Overall, the performance is very good. The dual-output variant performs slightly worse compared to the single-output ones shown on row one and two. This could be due to the network only having to optimize to one parameter. The bottom row in red shows the average error for the hypothetical zero-predictor model. As expected, these values are significantly worse than the trained models, meaning that the models have indeed captured most of the underlying trend in the simulation data and are making logical predictions. The same variants were also tested with a lower number of frames in the input sequence. This was done to find out whether or not it is necessary to provide fifty frames for just a single prediction. With an input sequence length of ten, the single step models still perform very well, even though the model only has a fifth of the input of the previous configuration. Only a marginal decrease in performance is shown compared to the 50/1 IO-ratio models.

A recurring trend was noticed among all these results. Roll prediction errors are overall lower than pitch prediction errors. This could be explained by roll having a somewhat more predictable behavior than pitch. For example, pitch could inherently have a more random behavior which causes it to have a lot more micro changes that are hard to learn and consistently predict. In contrast, when looking at the zero predictions, the roll prediction error is higher than the pitch prediction's error. Based on the data analysis which showed that pitch and roll have very similar distributions, this difference means that roll in general has more points that are further away from zero compared to pitch.

To further analyze the results from the single-step model, only the dual output variant will be considered as it shows similar prediction behavior as the single output models while predicting both pitch and roll. The Table above provided a good general idea of prediction accuracy; however, some prediction behavioral aspects can't be derived from a single number. In Figure 27 the prediction results are shown for pitch (top) and roll (bottom) across the whole test dataset. The predicted values are shown in orange on top the real values which are shown in blue. The dark sand-colored area represents overlapping values between predictions and real values while orange or blue areas mean that predictions are respectively exceed or underrun the real values. For readability, the dark sand-colored area will be referred to as the *dessert*. The top graph for pitch shows a constant blue area around its dessert. Judging by the shape of the blue outline and the shape of the dessert, the predicted pitch values seem to follow the trend well but consistently fall short of the real values by a couple of degrees. When comparing this to the roll graph, the effect is much less present which indicates that roll is being predicted with greater

accuracy. This confirms the hypothesis made above - that was solely based on the RMSE averages of Table 11 - that pitch must be more random and thus harder to predict.

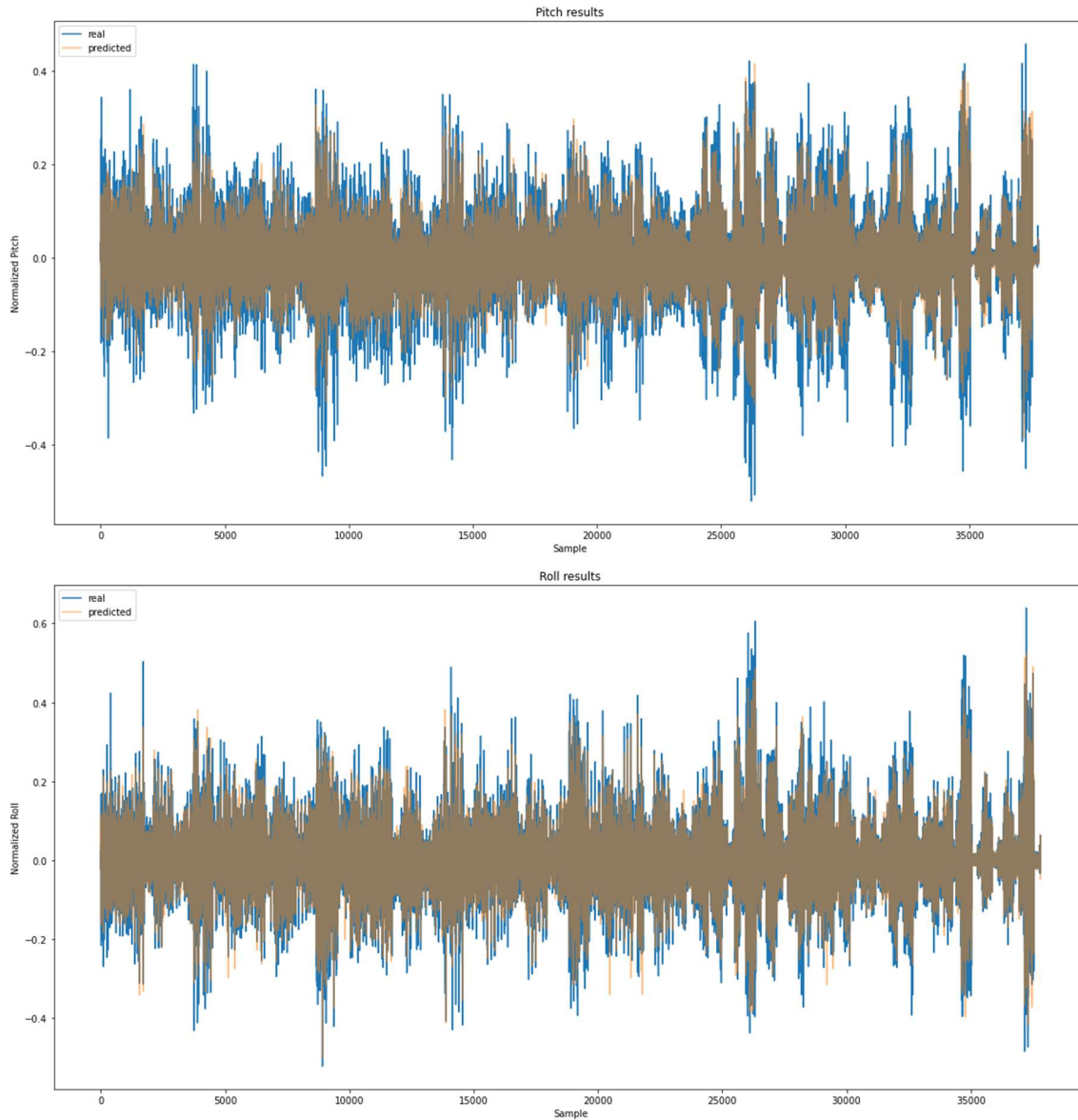


Figure 27: predicted (orange) vs. real (blue) values for pitch (top) and roll (bottom)

Lastly, the training generalization of the three single-step models is analyzed in the 10 input 50 epoch configuration. This is done by plotting the average MSE per epoch of the models for both the training and test dataset. This is shown in Figure 24. Each full line represents the training dataset loss while the dotted line represents the validation dataset loss. Generally speaking, the training curve should be lower than the validation loss curve because the model always tries to optimize towards the training data and does not know the validation data.

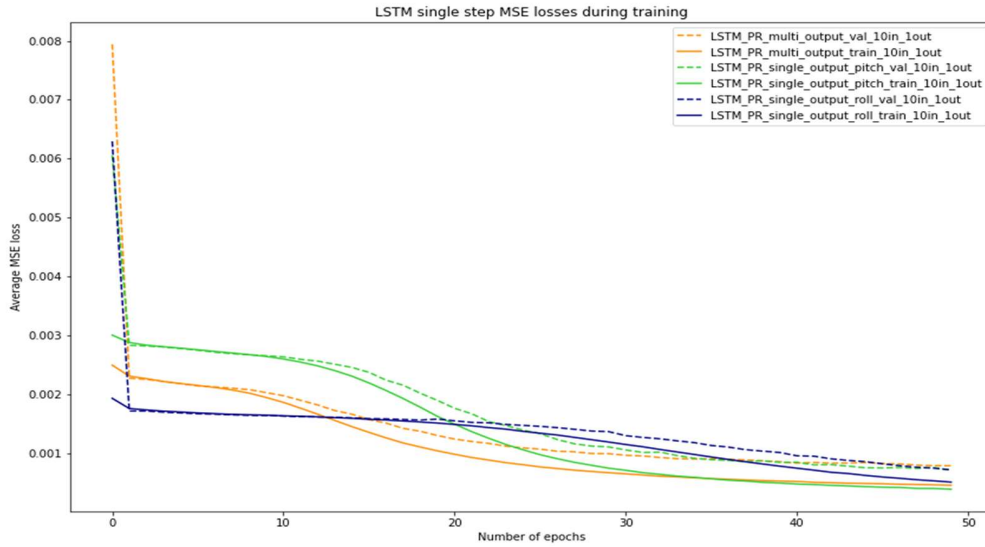


Figure 28: MSE loss during training of the single-step models

With this representation, generalization and overfitting can be visualized and assessed. The first of which is the rate at which the losses minimize and reach a state of very low change per epoch. The single step roll prediction model has by far the worst convergence out of the three, followed by the multi-output model. This means that it needs to most time to find its optimal internal state. Eventually, all models converge to a similar average error. However, the roll prediction model does not look to have reached its optimal form yet and might perform even better after more extensive training. The same constellation as above can be made here once again that roll has the most accurate predictions. Its validation loss (blue dotted line) is minimal on the last epoch. Overfitting occurs when the validation loss starts to increase again after reaching a certain point. Based on these graphs, little to no overfitting has happened during training.

In conclusion, even with roll having more spread-out values, it is still easier to predict than pitch. All models performed really well and produced results that can be later used as an additional benchmark. This model and its variants prove that pitch and roll can be predicted with average error of approximately  $2^\circ$ . However, they are not capable of sequence predictions.

### 5.3 Multi-step models

#### 5.3.1 Encoder-Decoder LSTM

The LSTM numeric data model is expected to perform the worst out of all proposed models according to Mykola's research. However, when looking at the results below, it becomes clear that this is not the case, and this model actually performs very well. Mykola's research of this model architecture resulted in an average error of approximately  $30^\circ$  of for both pitch and roll on the 10<sup>th</sup> predicted second (20<sup>th</sup> frame at 2Hz). He used a 20/24 IO-ratio configuration and trained for 50 epochs. This error is far worse than the zero-predictor. In addition, based on the nature of the data and its distribution, a  $30^\circ$  error is catastrophic when most data lies within a  $[-20^\circ, 20^\circ]$  interval. Based on these two reasons, it was assumed that Mykola had made an error during the calculation of this error or that the model had a too low learning rate which caused it to not have converged by the end of training.



This model was extensively trained for a multitude of different input sequence lengths for a fixed length output of 60. This was done to discover the optimal I/O ratio for accurate predictions without creating too much overhead of unnecessary long input sequences. Based on the results from the single-step models, the number of epochs for all training was fixed at 50. The goal in mind was to find the limits of this model, how far in the future can it predict and how much data is minimally needed for accurate results. The training results are shown Figure 25. The same layout as above is used where each color represents a different configuration, and the training and validation losses are respectively the full and dotted lines.

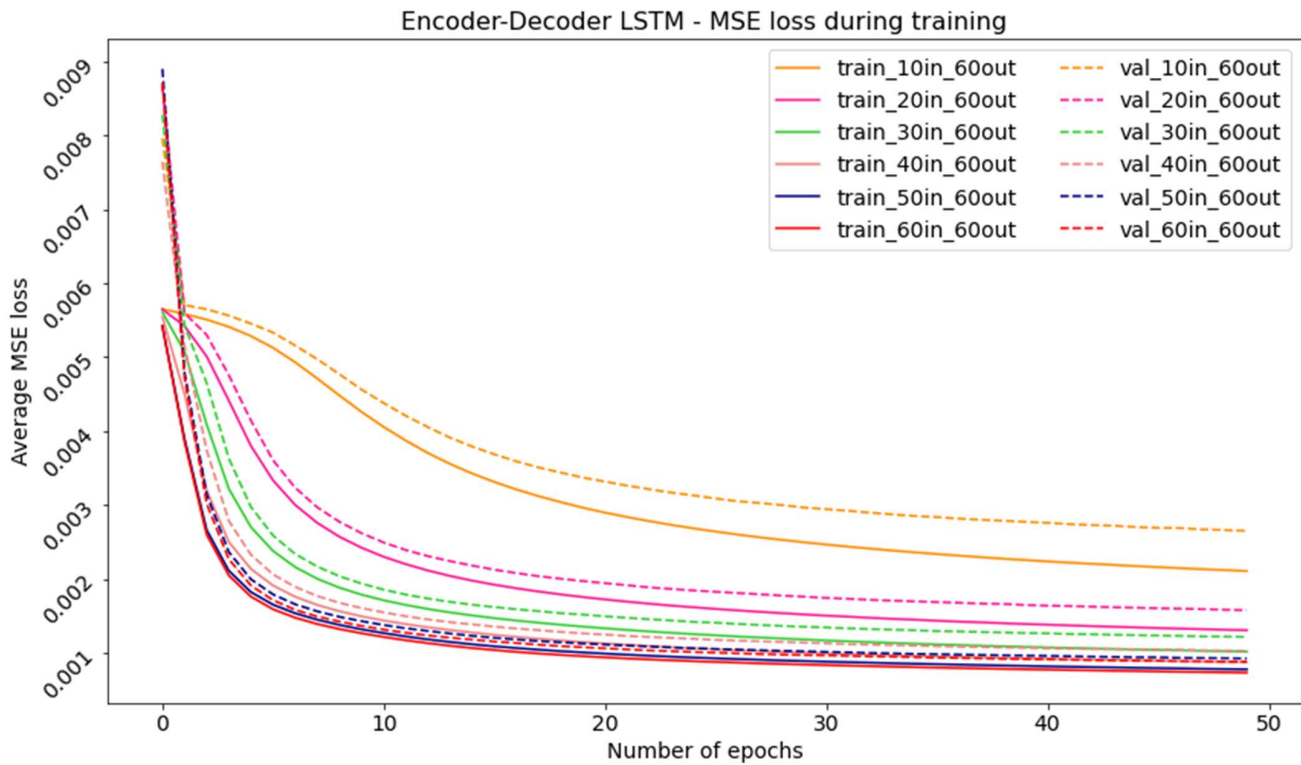


Figure 29: encoder-decoder LSTM training and validation losses for different IO-ratios

A general trend is present among the configurations: the error and convergence gets better the more input frames are used. Intuitively, this behavior is to be expected. The first model with a 10/60 IO-ratio (orange) performs the very worst and has very poor convergence. A little bit of overfitting is also happening during the final epochs where the training loss is still decreasing while the validation loss is almost stagnant. The next three ratios of 20/60, 30/60 and 40/60 (magenta, green and pink) are all improving upon themselves and converge to a lower and lower minimum. However, the best results are from the 50/60 and 60/60 ratio models (navy, red) with the latter being the very best by a small margin.

Table 3 shows the average error for pitch and roll per IO-ratio. These errors were calculated as the average error over the full predicted sequence. Once again, the zero-predictor results are appended as the bottom row for comparison. The trend found within the single step models' results appears once again for the LSTM model: pitch prediction errors are consistently higher than those of roll. This further affirms that predicting pitch is harder than predicting roll. Another thing to note is the small improvement in errors between a 50/60 and 60/60 IO-ratio. Only a negligible decrease in error is measured. Because of this, the optimal ratio for this model is approximately one-to-one. If the model were to be applied in real life, this ratio would be the starting point for further testing.

IO-ratio	Average Pitch error [denormalized RMSE]	Average Roll error [denormalized RMSE]
----------	--	---



10/60	5.25°	4.64°
20/60	3.95°	3.36°
30/60	3.39°	2.96°
40/60	3.19°	2.65°
50/60	2.95°	2.53°
60/60	2.89°	2.45°
zero-prediction	6.43°	7.30°

Table 12: Average pitch and roll error per IO-ratio

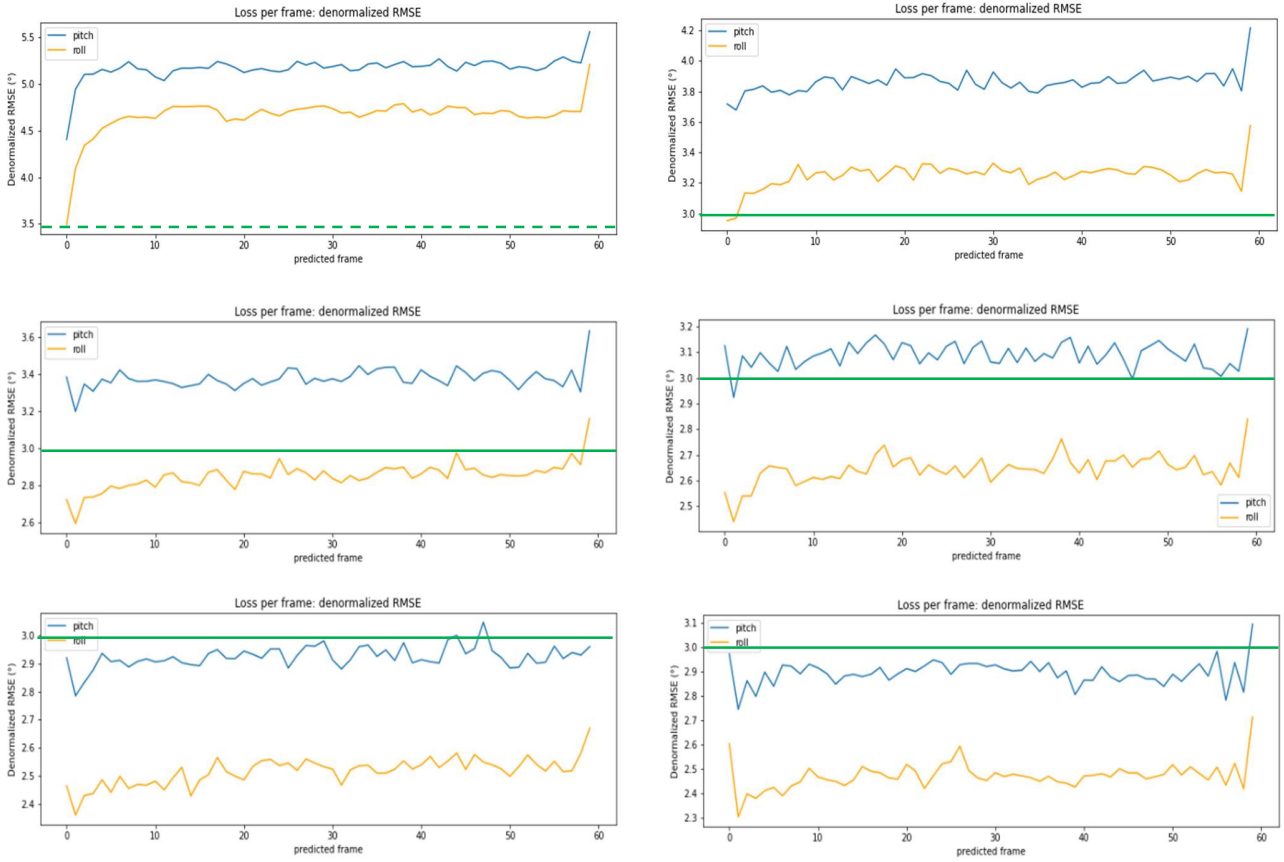


Figure 30: Loss per frame for different IO-rates with reference line at 3°

For each of the different configurations, the average loss per frame was calculated as well. This a custom algorithm calculates the average error per frame over all test data. For example, when dealing with an output sequence of 60 frames, the algorithm will calculate the error between the predicted sequence and the real sequence frame per frame over the 60 frames. The results is an array of 60 values that holds the difference between real and predicted for each predicted frame. This array calculated for each IO-sequence in the test dataset and the average is taken of each frame over the whole data set. In Figure 26, the six graphs are shown for each of the above listed configurations. The graphs are ordered on increasing input sequence length per graph from left to right, top to bottom. Here, the difference in pitch and roll prediction difficulty is very apparent. In each configuration, the blue pitch plot is higher than the roll plot in its entirety. Besides this, a remarkable and counterintuitive property of this model is visual. Only a minor increase in prediction error happens when moving along

further into the future. Instead of having an increasing error as the model predicts values further in the future, only a slight increase is noticeable. The models predict with a close to constant error across the whole sequence and the magnitude of the error is mainly affected by the length of the input sequence. This is similar behavior to Zhao's results where a consistent error was also found. Predictions at the 5<sup>th</sup> second and 20<sup>th</sup> second had equal magnitudes (Zhao et al., 2004).

Figure 27 shows a prediction for pitch (top) and roll (bottom) from the 60/60 model on one of the IO-sequences of the test dataset. On the x-axis the number of frames is shown in the sequence, on the y-axis, both parameters are shown in their normalized form.

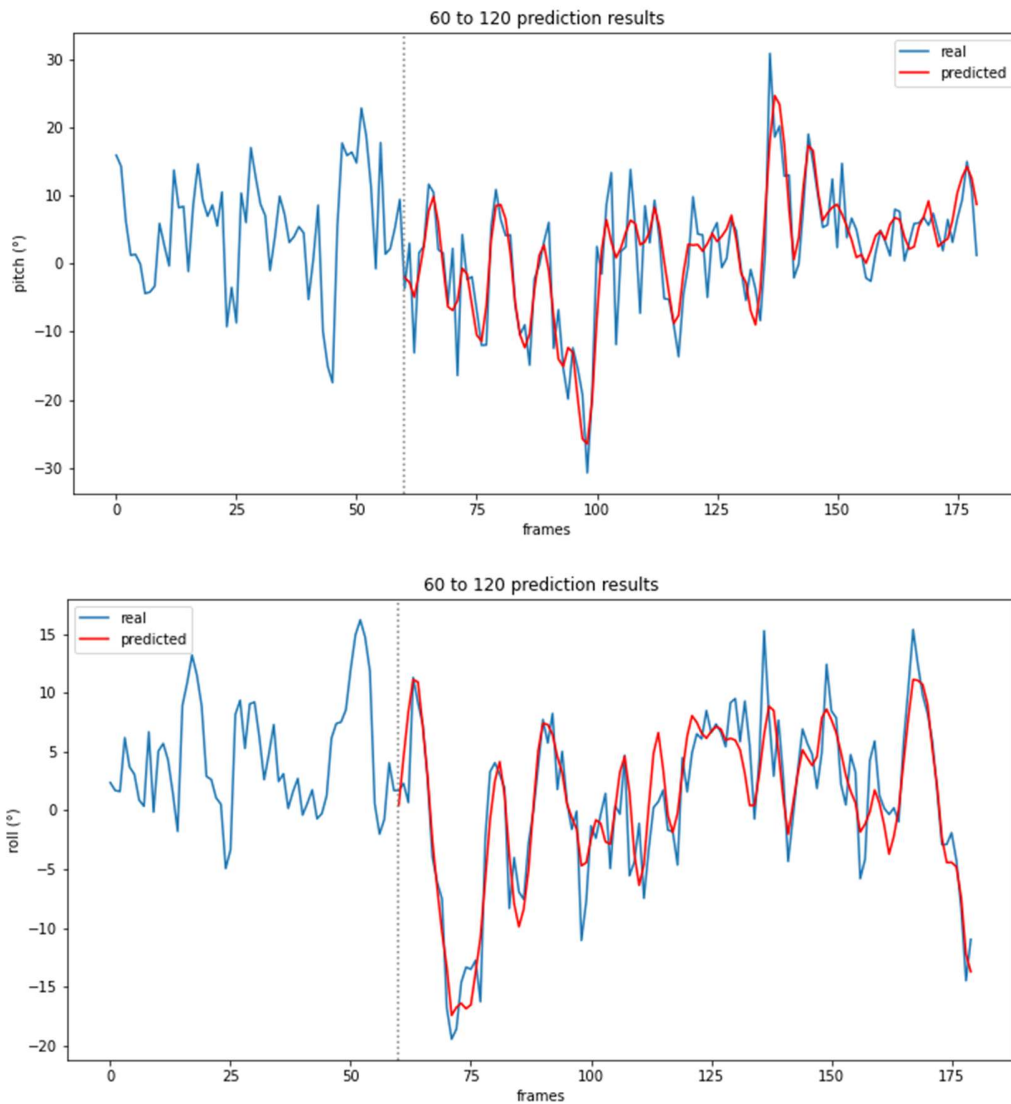


Figure 31: pitch (top) and roll (bottom) predictions for 60/120 configuration

Lastly, a final test was performed to push this model to its limits. A new configuration was trained using a 60/120 IO-ratio. At two frames per second or 2 Hz, this model uses 30 seconds of data to predict one minute. This window size provides a lot of possibilities to find landing opportunities. The results are shown below. In conclusion, this configuration performed better than expected. Compared to the 60/60 model, the average loss per frame is also not much higher even though ramping up slightly after the 60<sup>th</sup> frame. This indicates that the optimal IO-ratio for a model of this type, might not be influenced by the

output size and thus only input size should be considered. In this case, the optimal input size is approximately 60. At this value, the model performs best and receives no more benefits from more input, even when prediction long sequences.

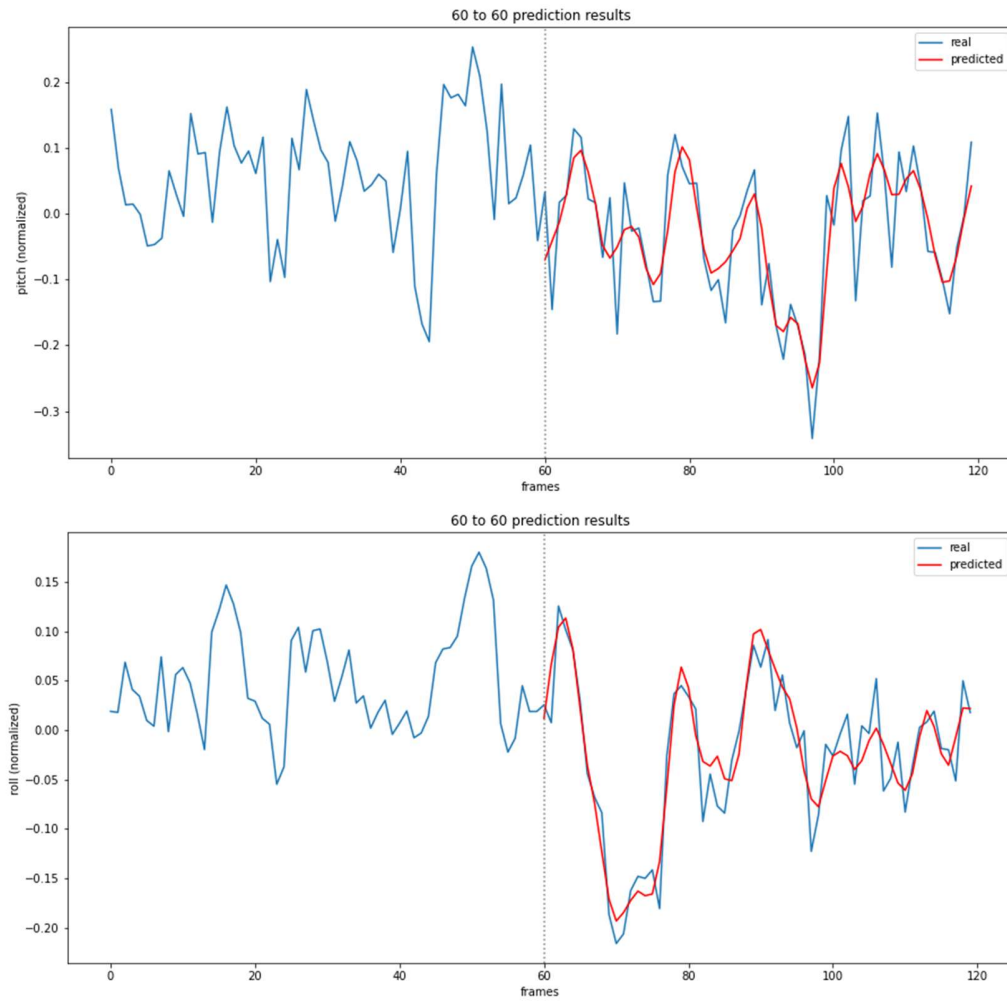


Figure 33: Prediction (red) vs. real (blue) for pitch (top) and roll (bottom)

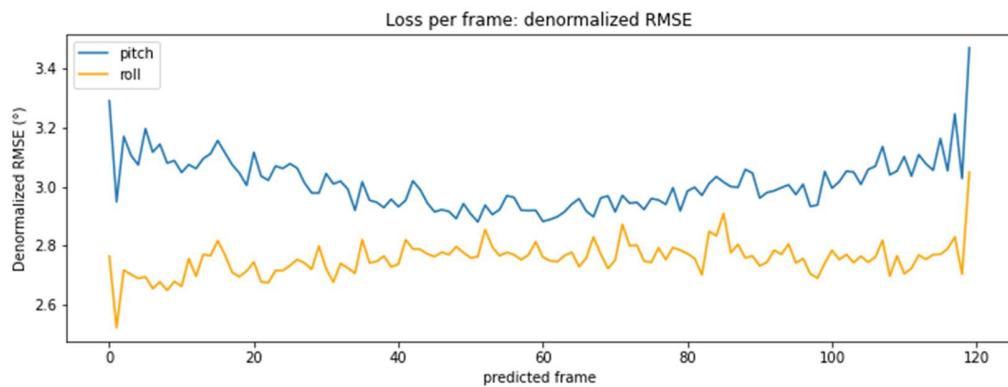
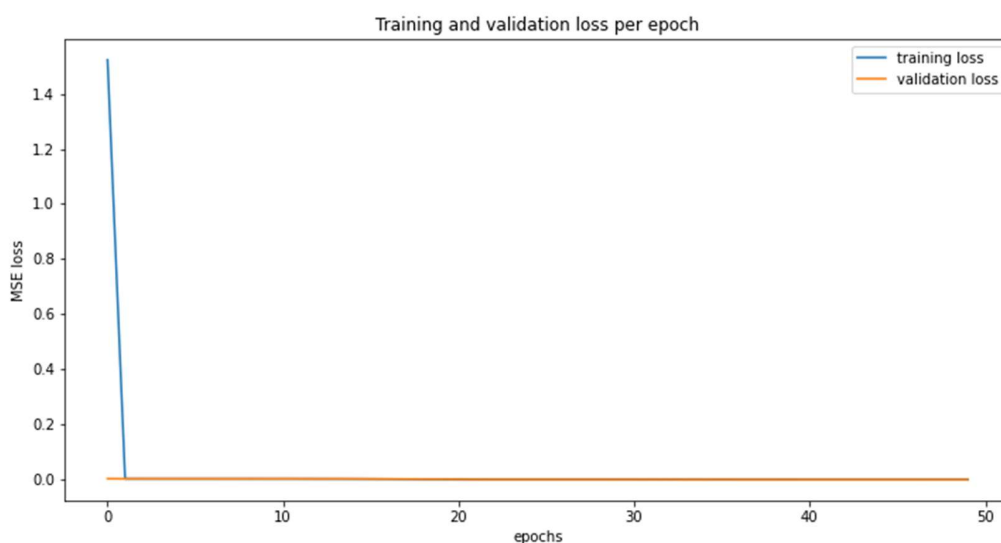


Figure 33: Loss per frame for 60/120 configuration

### 5.3.2 Sequential CNN

The CNN to linear model is the only model to not use an LSTM architecture. Because of this, it should have the highest difficulty learning the trend of the data and should perform worse than other models. This was definitely felt during training and testing of this model as it showed very instable behavior. Only one training attempt returned good results while all other attempts resulted in very abnormal behavior. This abnormal behavior was noticed by visually analyzing the predicted vs. real graphs and comparing it to the average RMSE over all predictions in the test dataset. For some IO-sequences, the model would produce really good predictions whilst for others it would produce seemingly random outputs. Even though showing good performance on some sequences, the model would eventually end up with an average denormalized RMSE of  $20^\circ$  which is really bad compared to other models. In most cases, model would also not converge very well in a sense that the MSE training and validation loss slowly decrease but after some epochs, they start to diverge. Training loss might go up again for a couple of epochs and eventually the training would finish with the model having a higher MSE than some iterations earlier in training.



The model was trained with an input sequence size of ten and an output size of sixty. The lower input sequence length was chosen in assumption that less images would be necessary compared to numerical datapoints, to predict with the same accuracy as the encoder-decoder LSTM model.

**TODO:** describe results, abnormally high errors occur during the first epoch of training: (800+ mse errors while average first epoch errors are 0.008-0.005). This is very odd behavior as the same data and same training loop is used as the CNN LSTM image model which does perform good

### 5.3.3 CNN LSTM single input

**TODO:** describe results, dotted image

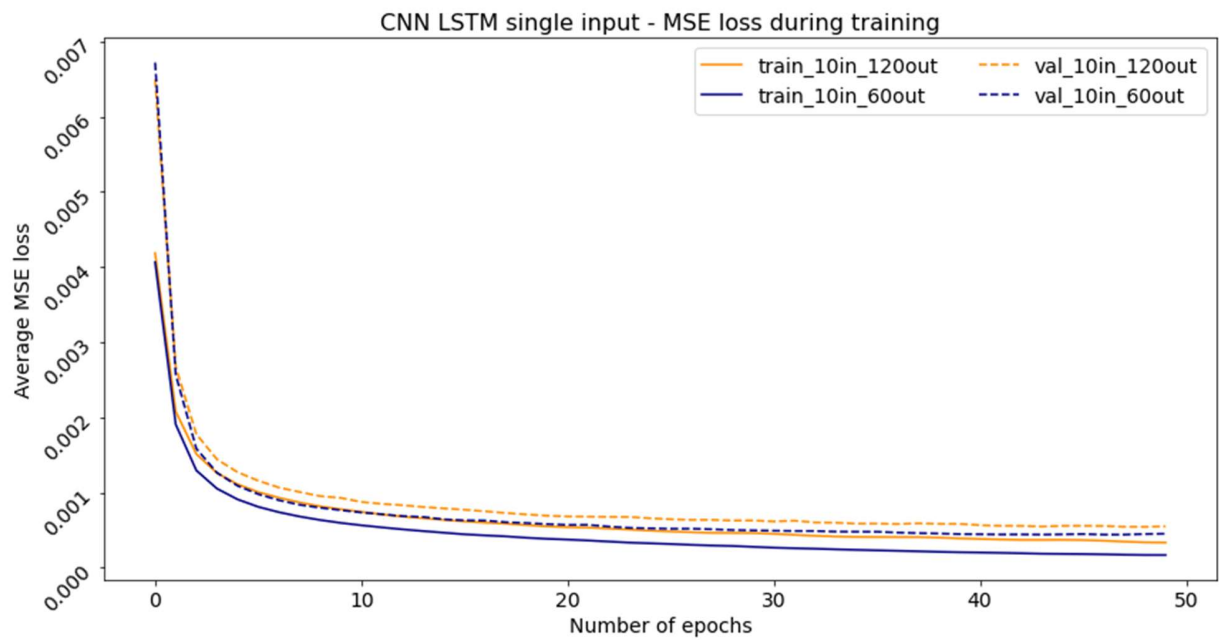
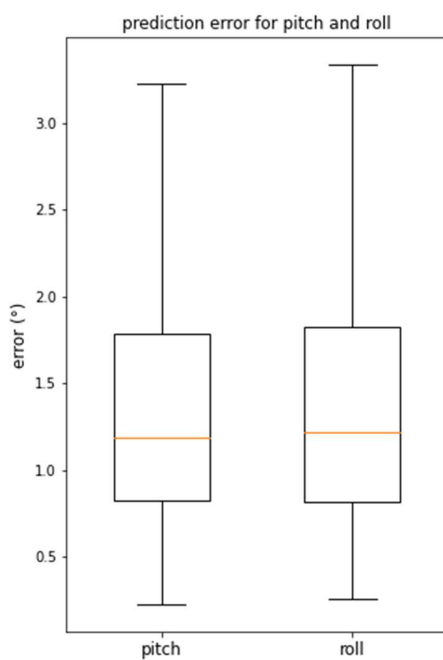


Figure 34: CNN LSTM single input training and validation loss for 10/60 and 10/120 IO-ratios



#### 5.3.4 CNN LSTM dual input

**TODO**, describe results, takes very long to make one forward pass (approximately 10 seconds). The torch.concat() method to append the two values of PR to the multidimensional (batch size, 1, flattened features size) Tensor is very inefficient on a cuda device and different solutions are still being experimented with.

### 5.4 Inference time

Inference time measured over 10.000 predictions and averaged.

Model name	IO-ratio	Inference time	Trainable parameters
Single-step LSTM (dual)	10/1	950ms	199.938
Encoder-Decoder LSTM	60/60	155ms	1.211.010
Sequential LSTM	10/60	<b>TODO</b>	<b>TODO</b>
CNN LSTM single input	10/60	4155ms	217.659.752
CNN LSTM dual output	10/60	4453ms	217.741.672

*Table 13: inference timing and number of trainable parameters for each model*

**TODO**, impact of IO-ratio, discuss findings, single step really slow despite simple architecture and low params. Could be because of stacked LSTM complexity

### 5.5 Augmented data

**TODO** dotted images test on CNN models

## 6 Discussion

**TODO**, compare all models to one another based on errors, complexity, requirements, inference time and propose one final architecture: encoder-decoder LSTM, images cause too much overhead and have too high memory and hardware requirements to process at real time. Lower frequency needed for image models but that reduces quality and detail of predictions.

### 6.1 Future work

**TODO**

Optimizing CNN LSTM dual input design: optimize slow concat operation on GPU (fast on CPU)

Optimize models for latency

HyperBand optimization

Real data transition

Angular velocity and heave velocity can also be important for determining window for landing opportunity

## 7 References

- Abujoub, S. (2019). *Development of a Landing Period Indicator and the use of Signal Prediction to Improve Landing Methodologies of Autonomous Unmanned Aerial Vehicles on Maritime Vessels*. Carleton University.
- Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). Training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, 144–152. <https://doi.org/10.1145/130385.130401>
- Brownlee, J. (2017). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Machinelearningmastery. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>
- Brownlee, J. (2018). *Multi-Step LSTM Time Series Forecasting Models for Power Usage*. Deep Learning for Time Series. <https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/>
- Canziani, A., Paszke, A., & Culurciello, E. (2016). *An Analysis of Deep Neural Network Models for Practical Applications*. <https://doi.org/10.48550/arxiv.1605.07678>
- Chen, J., Benesty, J., Huang, Y., & Doclo, S. (2006). New insights into the noise reduction Wiener filter. *IEEE Transactions on Audio, Speech and Language Processing*, 14(4), 1218–1233. <https://doi.org/10.1109/TSA.2005.860851>
- Claesen, M., & de Moor, B. (2015). *Hyperparameter Search in Machine Learning*. <https://doi.org/10.48550/arxiv.1502.02127>
- Cui, Z., Ke, R., Pu, Z., & Wang, Y. (2020). Stacked bidirectional and unidirectional LSTM recurrent neural network for forecasting network-wide traffic state with missing values. *Transportation Research Part C: Emerging Technologies*, 118. <https://doi.org/10.1016/j.TRC.2020.102674>
- de Cubber, G. (2019). *OPPORTUNITIES AND SECURITY THREATS POSED BY NEW TECHNOLOGIES*.
- de Masi, G., Gaggiotti, F., Bruschi, R., & Venturi, M. (2011). Ship motion prediction by radial basis neural networks. *IEEE SSCI 2011 - Symposium Series on Computational Intelligence - HIMA 2011: 2011 IEEE Workshop on Hybrid Intelligent Models and Applications*, 28–32. <https://doi.org/10.1109/HIMA.2011.5953967>
- Dhanya, J., & Raghukanth, S. T. G. (2018). Ground Motion Prediction Model Using Artificial Neural Network. *Pure and Applied Geophysics*, 175(3), 1035–1064. <https://doi.org/10.1007/S00024-017-1751-3/FIGURES/22>
- Doshi Sanket. (2019). *Various Optimization Algorithms For Training Neural Network*. Towards Data Science. <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>
- Fossen, S., & Fossen, T. I. (2018). EXogenous Kalman filter (XKF) for Visualization and Motion Prediction of Ships using Live Automatic Identification System (AIS) Data. *Modeling, Identification and Control*, 39(4), 233–244. <https://doi.org/10.4173/MIC.2018.4.1>
- Geifman, A. (2020). *How to Measure Inference Time of Deep Neural Networks / Deci*. Deci.Ai. <https://deci.ai/blog/measure-inference-time-deep-neural-networks/>
- Grewal, M. S., & Andrews, A. P. (2010). Applications of Kalman Filtering in Aerospace 1960 to the Present. *IEEE Control Systems*, 30(3), 69–78. <https://doi.org/10.1109/MCS.2010.936465>
- Guan, B., Yang, W., Wang, Z., & Tang, Y. (2018). Ship roll motion prediction based on  $\ell_1$  regularized extreme learning machine. *PLoS ONE*, 13(10). <https://doi.org/10.1371/JOURNAL.PONE.0206476>



- Ham, S.-H., Roh, M.-I., & Zhao, L. (2017). *Integrated method of analysis, visualization, and hardware for ship motion simulation*. <https://doi.org/10.1016/j.jcde.2017.12.005>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-December*, 770–778. <https://doi.org/10.48550/arxiv.1512.03385>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/NECO.1997.9.8.1735>
- Johansen, T. A., & Fossen, T. I. (2017). The eXogenous Kalman Filter (XKF). *International Journal of Control*, 90(2), 177–183. <https://doi.org/10.1080/00207179.2016.1172390>
- Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME--Journal of Basic Engineering*, 82(Series D), 35–45.
- Kaminskiy, N.-M. (2019). *Deep learning models for ship motion prediction from images*.
- Kingma, D. P., & Ba, J. L. (2014). Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. <https://doi.org/10.48550/arxiv.1412.6980>
- Kingma, D. P., & Welling, M. (2019). An Introduction to Variational Autoencoders. *Foundations and Trends in Machine Learning*, 12(4), 307–392. <https://doi.org/10.1561/22000000056>
- Kouris, A., Venieris, S. I., Rizakis, M., & Bouganis, C.-S. (n.d.). *Approximate LSTMs for Time-Constrained Inference: Enabling Fast Reaction in Self-Driving Cars*. Retrieved May 20, 2022, from [www.imperial.ac.uk/intelligent-digital-systems/approx-lstms/](http://www.imperial.ac.uk/intelligent-digital-systems/approx-lstms/)
- le Guen, V., & Thome, N. (n.d.). *Deep Time Series Forecasting with Shape and Temporal Criteria*.
- Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18, 1–52. <https://arxiv.org/abs/1603.06560v4>
- Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Autoencoders. *Machine Learning: Methods and Applications to Brain Disorders*, 193–208. <https://doi.org/10.48550/arxiv.2003.05991>
- Luo, F. L., Unbehauen, R., & Cichocki, A. (1997). A Minor Component Analysis Algorithm. *Neural Networks*, 10(2), 291–297. [https://doi.org/10.1016/S0893-6080\(96\)00063-9](https://doi.org/10.1016/S0893-6080(96)00063-9)
- MarLand – Robotics & Autonomous Systems*. (n.d.). Retrieved March 29, 2022, from <https://mecatron.rma.ac.be/index.php/projects/marland/>
- MarSur – Robotics & Autonomous Systems*. (n.d.). Retrieved March 29, 2022, from <https://mecatron.rma.ac.be/index.php/projects/marsur/>
- Mealey, T., & Taha, T. M. (2018). Accelerating Inference in Long Short-Term Memory Neural Networks. *Proceedings of the IEEE National Aerospace Electronics Conference, NAECON, 2018-July*, 382–390. <https://doi.org/10.1109/NAECON.2018.8556674>
- Minsky, M., & Papert, S. A. (1969). Perceptrons: An Introduction to Computational Geometry. In *Perceptrons*. The MIT Press. <https://doi.org/10.7551/MITPRESS/11301.001.0001>
- Nayfeh, A. H., Mook, D. T., & Marshall, L. R. (2012). Nonlinear Coupling of Pitch and Roll Modes in Ship Motions. <https://doi.org/10.2514/3.62949>, 145–152. <https://doi.org/10.2514/3.62949>

- Nazotron1923/ship-ocean\_simulation\_BLENDER: Python scripts for generation sea states and ship motion using 3D Blender simulator. (n.d.). Retrieved March 29, 2022, from [https://github.com/Nazotron1923/ship-ocean\\_simulation\\_BLENDER](https://github.com/Nazotron1923/ship-ocean_simulation_BLENDER)
- Peng, X., Zhang, B., & Rong, L. (2019). A robust unscented Kalman filter and its application in estimating dynamic positioning ship motion states. *Journal of Marine Science and Technology (Japan)*, 24(4), 1265–1279. <https://doi.org/10.1007/S00773-019-00624-5>
- Perez, T. ;, & Blanke, M. (2017). Ship Roll Damping Control. *Annual Reviews in Control*, 36(1), 129–147. <https://doi.org/10.1016/j.arcontrol.2012.03.010>
- Perez, T., & Fossen, T. I. (2005). Kinematics of ship motion. *Advances in Industrial Control*, 9781852339593, 45–58. [https://doi.org/10.1007/1-84628-157-1\\_3](https://doi.org/10.1007/1-84628-157-1_3)
- Q. Judge, C. (2019). Ship motion in waves. In *Seakeeping and Maneuvering*. [https://www.usna.edu/NAOE/\\_files/documents/Courses/EN455/AY20\\_Notes/EN455CourseNotesAY20\\_FrontMaterial.pdf](https://www.usna.edu/NAOE/_files/documents/Courses/EN455/AY20_Notes/EN455CourseNotesAY20_FrontMaterial.pdf)
- Ran, T., Tong, S., Yang, Y., & Zhang, H. (2021). Research and Application on Mathematical Model of Ship Manoeuvring Motion under Shallow water effect. *IOP Conference Series: Earth and Environmental Science*, 643(1). <https://doi.org/10.1088/1755-1315/643/1/012127>
- Rashid, M. H., Zhang, J., & Minghao, Z. (2021). Real-Time Ship Motion Forecasting Using Deep Learning. *The 2nd International Conference on Computing and Data Science*, 5(2021). <https://doi.org/10.1145/3448734>
- Ren, X., Yang, T., Erran Li, L., Alahi, A., & Chen, Q. (2021). Safety-aware Motion Prediction with Unseen Vehicles for Autonomous Driving. *IEEE Explore*. <https://github.com/>
- Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. <http://caffe.berkeleyvision.org/tutorial/solver.html>
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 1986 323:6088, 323(6088), 533–536. <https://doi.org/10.1038/323533a0>
- Rumelhart E., D., Hinton E., G., & Williams J., R. (1986). Learning representations by back-propagating errors. *Letters To Nature*.
- Sharma, S., Sharma, S., & Athaiya, A. (2020). ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology*, 4, 310–316. <http://www.ijeast.com>
- Shen Kevin. (2018). *Effect of batch size on training dynamics*. Medium. <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e>
- Sherstinsky, A. (2018). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. *Physica D: Nonlinear Phenomena*, 404. <https://doi.org/10.1016/j.physd.2019.132306>
- Silva, M. (2015). Ocean Surface Wave Spectrum. In *Physical Oceanography*. [https://www.researchgate.net/publication/283722827\\_Ocean\\_Surface\\_Wave\\_Spectrum](https://www.researchgate.net/publication/283722827_Ocean_Surface_Wave_Spectrum)
- Skulstad, R., Li, G., Fossen, T. I., Wang, T., & Zhang, H. (2021). A Co-Operative hybrid model for ship motion prediction. *Modeling, Identification and Control*, 42(1), 17–26. <https://doi.org/10.4173/MIC.2021.1.2>
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56), 1929–1958. <http://jmlr.org/papers/v15/srivastava14a.html>

- Stock, J. H., & Watson, M. W. (2001). Vector Autoregressions. *Journal of Economic Perspectives*, 15(4), 101–115. <https://doi.org/10.1257/JEP.15.4.101>
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going Deeper with Convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 07-12-June-2015*, 1–9. <https://doi.org/10.48550/arxiv.1409.4842>
- Tang, Y., Ma, L., Liu, W., & Zheng, W. S. (2018). Long-Term Human Motion Prediction by Modeling Motion Context and Enhancing Motion Dynamic. *IJCAI International Joint Conference on Artificial Intelligence, 2018-July*, 935–941. <https://doi.org/10.48550/arxiv.1805.02513>
- Wang, Q., Ma, Y., Zhao, K., & Tian, Y. (2022a). A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science*, 9(2), 187–212. <https://doi.org/10.1007/s40745-020-00253-5>
- Wang, Q., Ma, Y., Zhao, K., & Tian, Y. (2022b). A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science*, 9(2), 187–212. <https://doi.org/10.1007/s40745-020-00253-5/TABLES/8>
- Wei, Y., Chen, Z., Zhao, C., Chen, X., Tu, Y., & Zhang, C. (2022). Big multi-step ship motion forecasting using a novel hybrid model based on real-time decomposition, boosting algorithm and error correction framework. *Ocean Engineering*, 256, 111471. <https://doi.org/10.1016/J.OCEANENG.2022.111471>
- Wilson, D. R., & Martinez, T. R. (2001). The need for small learning rates on large problems. *Proceedings of the International Joint Conference on Neural Networks*, 1, 115–119. <https://doi.org/10.1109/IJCNN.2001.939002>
- Wu, J. (2017). *Introduction to Convolutional Neural Networks*.
- Zhang, T., Zheng, X. Q., & Liu, M. X. (2021). Multiscale attention-based LSTM for ship motion prediction. *Ocean Engineering*, 230. <https://doi.org/10.1016/J.OCEANENG.2021.109066>
- Zhao, X., Xu, R., & Kwan, C. (2004). Ship-motion prediction: Algorithms and simulation results. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 5. <https://doi.org/10.1109/ICASSP.2004.1327063>
- Zhong-yi, Z. (2012). An adaptive ship motion prediction method based on parameter estimation. *Journal of Ship Mechanics*.

## 8 Appendix

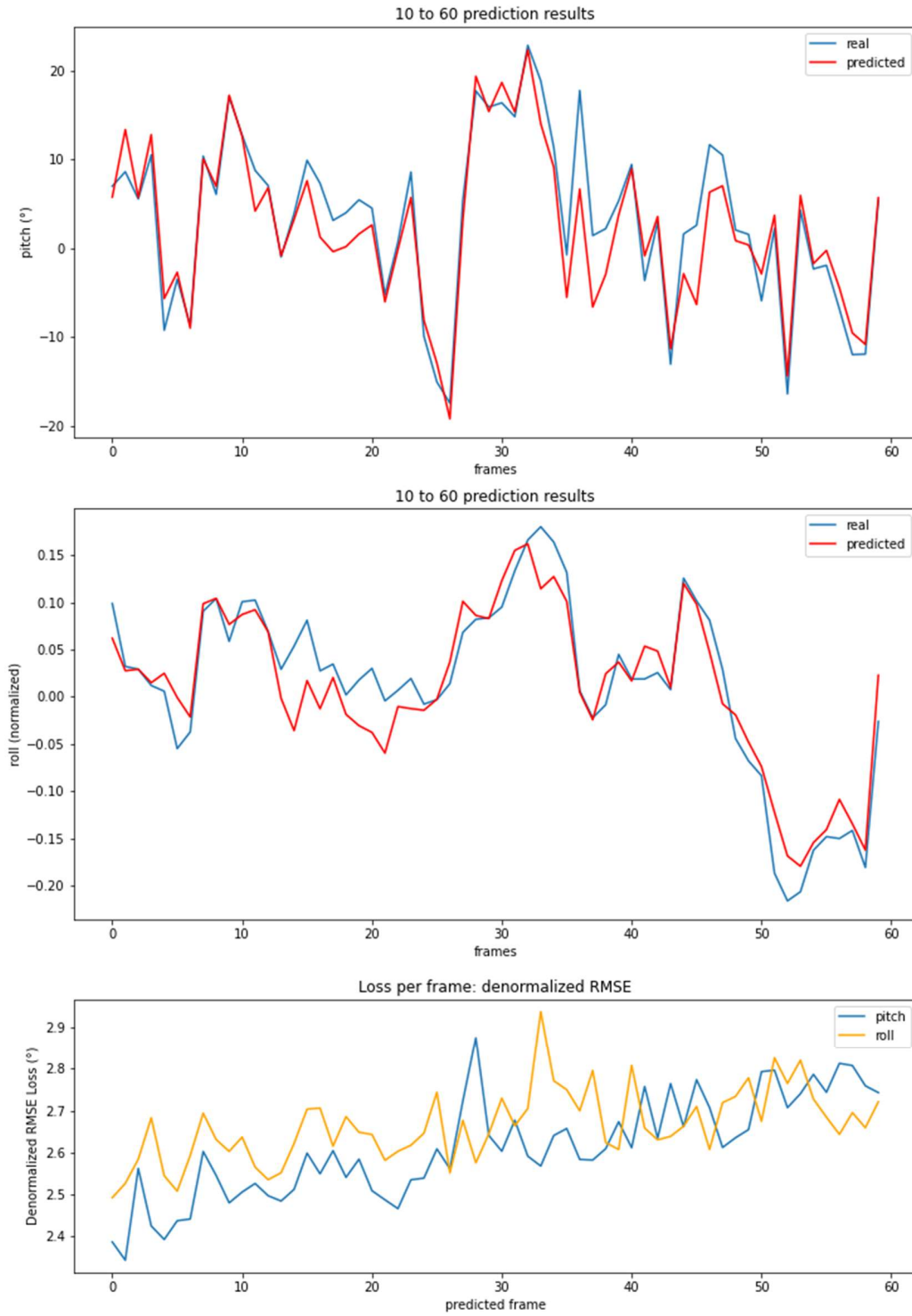


Figure 35: Predictions and LPF of linear CNN model

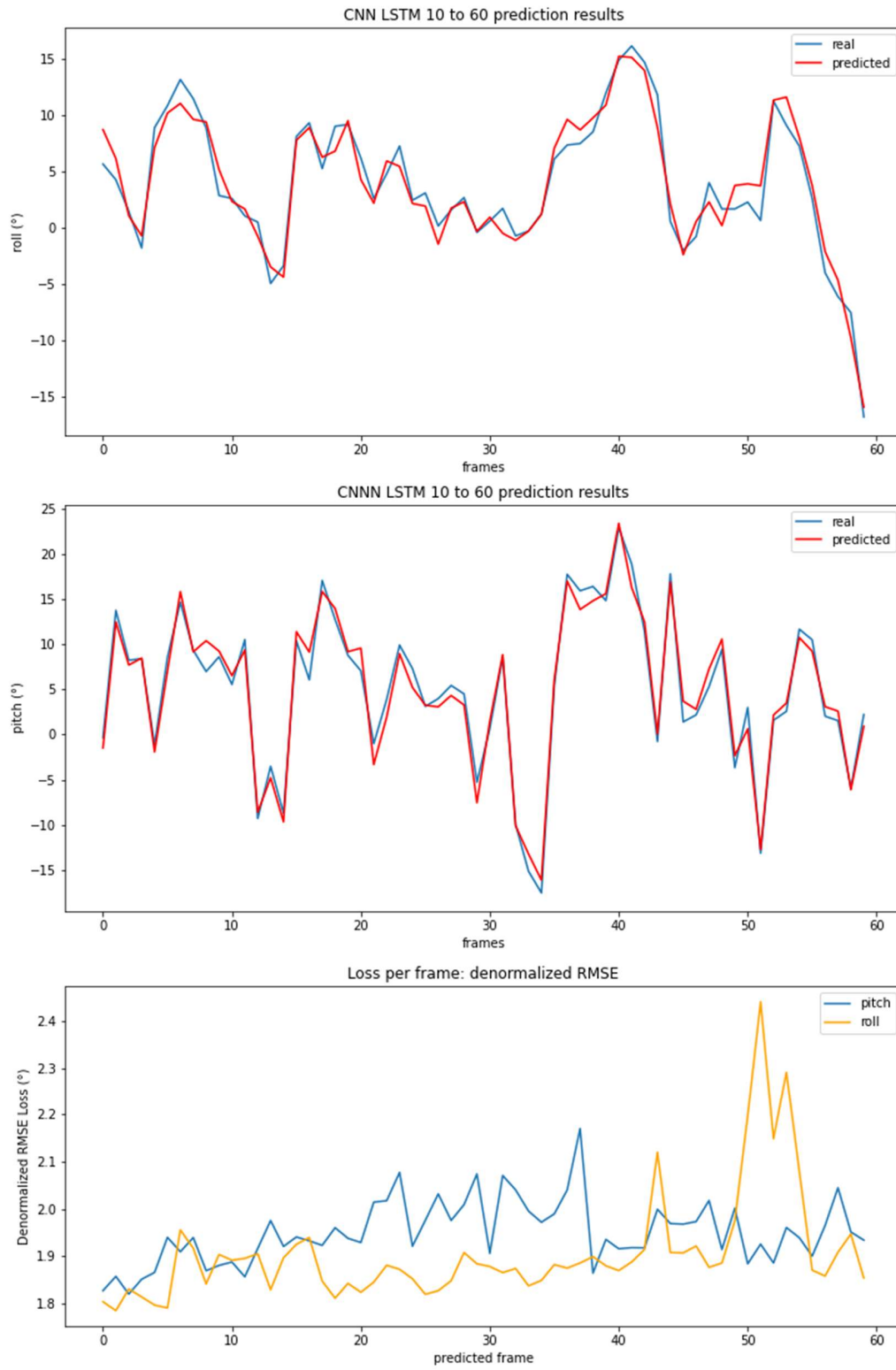


Figure 36: Prediction and LPF of single input CNN LSTM for 10/60 IO-ratio

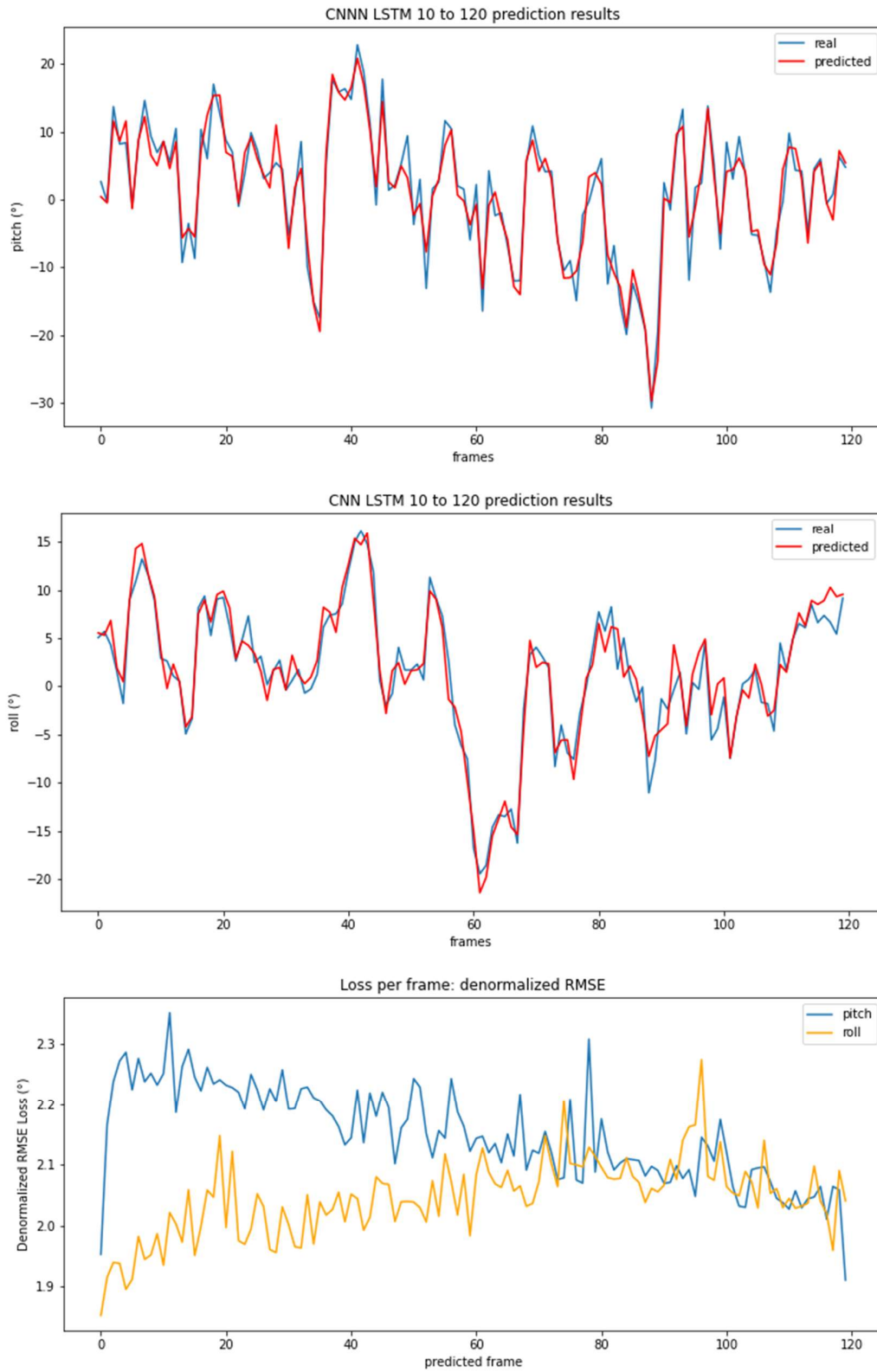


Figure 37: Prediction and LPF of single input CNN LSTM for 10/120 10-ratio