2022

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)

(confidentiality notice)

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

List of Figures

List of Tables

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

Introduction

In the last few years, the world has seen an exponential increase in technological advancements. This evolution brought with it a new influence of autonomous systems controlled by artificial intelligence (AI). Each of these systems designed with its own goals and characteristics, optimized for its task. More and more of these systems are being deployed as a direct or indirect replacement for tasks humans could do, but also, for tasks humans can't physically do. And because these autonomous systems are optimized for specific jobs, they 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 therefore eliminate the risk of someone's life. On the other hand, they can also be used as a complimentary asset, providing support and aid in logistics. More and more of these autonomous assets such as drones, surface vessels, tanks and reconnaissance vehicles are being deployed around the world for various objectives. But with this increasing amount of autonomous assets, there is need for 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.)

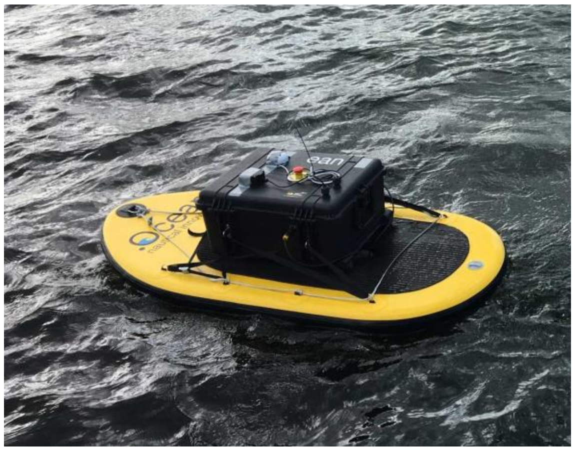


*Figure 1: Prototype of Autonomous Surface Vessel (ASV) with on-board computer and sensors (*MarSur – Robotics & Autonomous Systems*, n.d.)*

The Robotics & Autonomous Systems lab of the Belgian Royal Military Academy is currently working on two autonomous vehicles in two projects named MarSur (*MarSur – Robotics & Autonomous Systems*, n.d.)and MarLand (*MarLand – Robotics & Autonomous Systems*, n.d.). Project MarSur is developing an autonomous surface vessel (ASV) (figure 1) that will interact with a drone that is being developed by project MarLand. The drone needs to be able to take-off and land on the ASV. This proposes a challenge since the ASV is continuously moving due to sea waves and can therefore not provide a stable landing surface. For a smooth landing to be possible, the ASV must be capable to determine its state in a three-dimensional space and predict its movement in the ocean. These predictions must provide an accurate estimation over a future time series to determine the optimal time for the drone to land. This optimal time is a period where the ASV is as stable as possible so that the impact on the drone will be minimized. To facilitate these predictions, the ASV is equipped with an on-board computer and multiple sensors.

ASV sensors

The ASV is equipped with a ZED-mini stereo IMU camera (figure 2) (StereoLabs, Paris, France). This is a multipurpose sensor that can provide both video from its cameras and numeric data from its Inertial Measurement Unit (IMU), to accurately describe the state of the sensor and its surroundings. The ZED-mini 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. These types of movements will be described more in depth in the next part of the introduction.

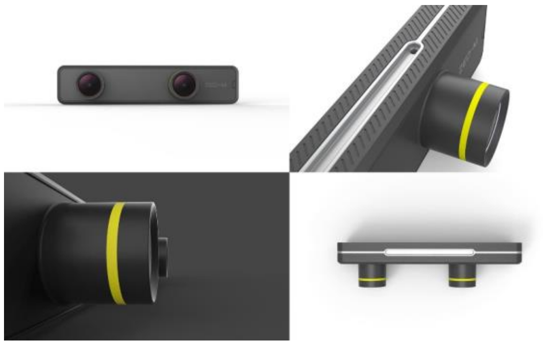


Figure 2: ZED-mini IMU stereo camera (StereoLabs, Paris, France)

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.

The data from the motion sensors and the camera's will be used as input to predict the future movement of the vessel. The deep learning models will digest a sequence of data and images of the past to predict a sequence of data in the future. To do this, the model will try to find trends in the data and continue these trends with regards to the information gathered from the images with incoming waves. Two important parameters will be how much historic data the model takes as input and how much data it can accurately predict in the future.

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

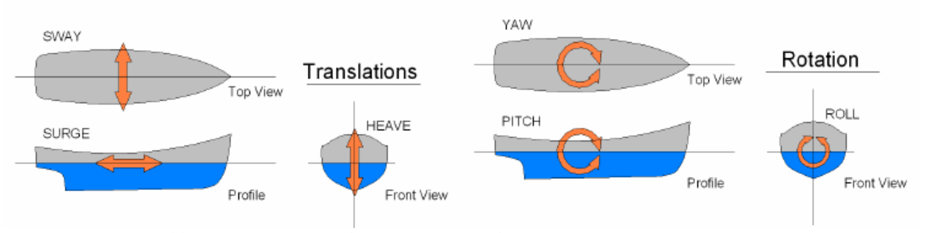


Figure 3: Six degrees of freedom in ship motion(de Masi et al., 2011)

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 don't 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

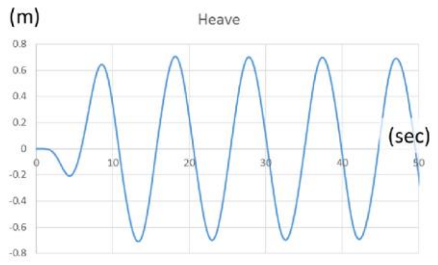


Figure 4: Heave (m) in function of time (s) (Ham et al., 2017)

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°, 3°] 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.

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.

Technologies

To develop the deep learning neural networks, Python[1] 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.

Met opmerkingen [LDW1]: Reference style of these kind of sources?

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.

[1] https://www.python.org/

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 bellow.

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

Simulation data

During most of the development, a simulation 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 very 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. Adjustments such as including passing vessels in the images can make it more realistic. 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.

Neural network architectures

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

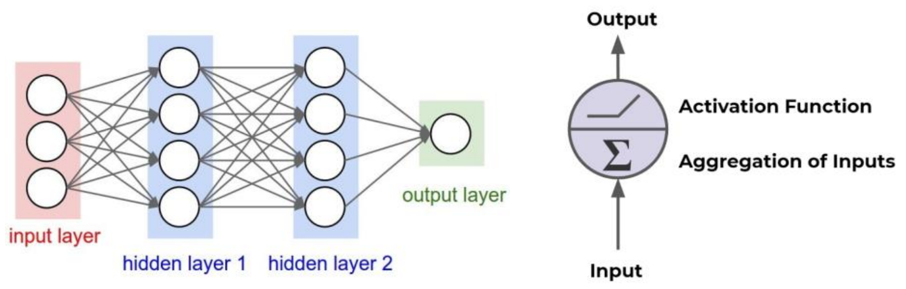


Figure 5: Neural network structure (left) and a linear neuron (right)

A neural network consists of different layers built with multiple neurons. These neurons are connected to 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 a bias to all its inputs and passes the 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.

Gradient descent

TODO mini-batch gradient descent is used

Activation functions

TODO, ReLu and tanh are used

Met opmerkingen [LD2]: Necessary?

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 important features of its input data that are needed to reconstruct it as accurate as possible.

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 related work. 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 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.

RNN and LSTM networks

Recurrent Neural Networks, RNN are special neural networks designed to work with sequential data. This is data where the order is important, for example time series data, sentences, audio etc. Ship motion falls within the time series data category where continuous data is chronologically ordered. The same goes for video footage or consecutive images where frames should be processed in a specific order.

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 6, 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.

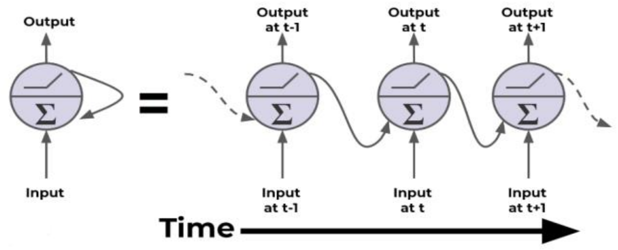


Figure 6: Neurons in a recurrent neural network

LSTM networks provide a solution to this disadvantage by not only using a short-term memory but also saving a long-term memory state. Each neuron of an LSTM network receives three inputs: the input data, the previous output (short-term memory) and the long-term memory state. The short-term memory and long-term memory are called the hidden state and the cell state respectively. Within the cell, four components are used to update these states and produce new states and an output. These four components are gates, each with their own function: input, forget, update or output. The gates use activation functions to decide whether to let information through, “update”, or block information, “forget”. This way, the LSTM-cells gradually learn what information should be kept for long-term memory and which information can be discarded.

Met opmerkingen [LD3]: Provide mathematical functions?

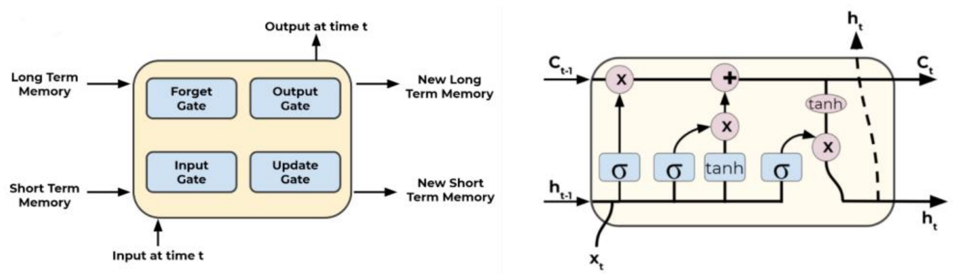


Figure 7: LSTM-cell components (left) and their mathematical notations (right)

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 exists such as peephole LSTM networks and gated recurrent units, GRU. These will be discussed more in the discussion chapter.

Convolutional Neural Networks

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 very effective when dealing with images.

Images are represented as three-dimensional matrices where every element in the matrix represents the intensity of a channel for a specific pixel. The width and height of this matrix are equal to image and the depth is the number of color channels in the image. When this data structure would be used for a linear neural network, the matrix would have to be flattened into a one-dimensional vector to feed into the first layer. This way all two-dimensional information is lost. CNN networks solve this problem by extracting features from the images with two-dimensional convolution kernels.

A convolution kernel or filter is a square $n * n$ 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 8. 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 important 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.

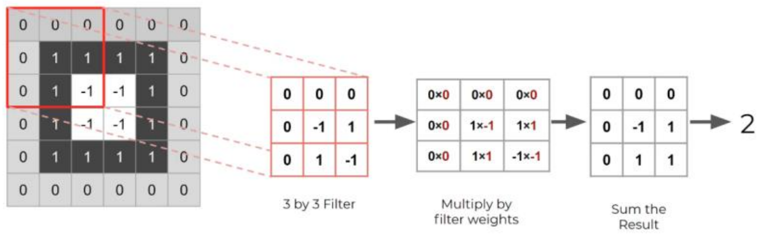


Figure 8: Convolution over grayscale image with 3x3 kernel

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 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.

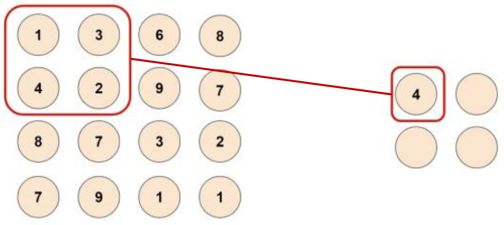


Figure 9: Max pooling with 2x2 kernel

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 $n\ x\ n$ 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 $n\ x\ n$ 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 Related work

A big source of information and inspiration for this thesis was the work of Nazar-Mykola Kaminskyi (Kaminskyi, 2019). Kaminskyi also tried to create different neural networks that can predict the motion of a vessel based on pitch, roll and incoming wave images. He created the dataset that was used for a big portion of this thesis and provided research and results from his research project. His results were used to set up criteria to which our models had to comply. This thesis should not be seen as a replacement for Kaminskyi's work though, but more as continuation and expansion of his work. Kaminskyi never tested his models on real life data and only used pitch and roll as input together with the images. With the ZED-mini, our models will have access to more data and should therefore potentially perform better.

TODO: discuss his results and how this will affect the methods in this thesis.

2 Objectives

The main goal of this research is to develop a neural network capable of predicting the motion of a surface vessel. To do this, the model ingests a sequence of values describing the state of the vessel and images of incoming waves taken from a stabilized camera pointing to the front of the vessel. As an output, the model should provide a sequence of predicted pitch, roll and heave values for every predicted second. The sequence duration for these predictions should be at minimum thirty seconds 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 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 the ZED-mini, it should be lightweight and not require large amounts of computational power. This means that the inference time of the different proposed models should also be considered when comparing different models' performances. In conclusion, both prediction errors and inference time should be minimized.

To test the viability of each model, it should be subjected to a set of minimum criteria in the simulation environment. A model is fit to be used with real data only when it meets these criteria. To make this selection, the following attributes of the model must comply with their respective requirements. The de-normalized root mean square error (RMSE) for the predicted angles for pitch and roll at the 10th second should be no more than three degrees and no more than five degrees at the 30th predicted second. These values are derived from the performance of the best models from Kaminskyi's work. The RMSE of the heave of the vessel should never exceed ten centimeters. Finally, the inference time of the model should be no more than fifty milliseconds.

Met opmerkingen [LDW4]: Cite

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. This window will be different for different kinds of drones. Therefore, the parameters of this window should be easily changeable, to compensate for different drones. During this window, heave, roll and pitch should remain close to constant over a duration of continuous time. To define this constant behavior, the next set of criteria can be used. Pitch and roll angles should remain in a [-3°, 3°] interval and consecutive values should have no difference bigger than two degrees. Consecutive heave values should have no difference bigger than one centimeter at 2 Hz. For heave, there is no interval in which all predicted values should remain since this is highly dependent on the height of the waves.

Met opmerkingen [LD5]: These are hypothetical values based on my estimations on how much a ship can move during landing. What are the actual values for this?

3 Data

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.

3.1 Simulated data

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.

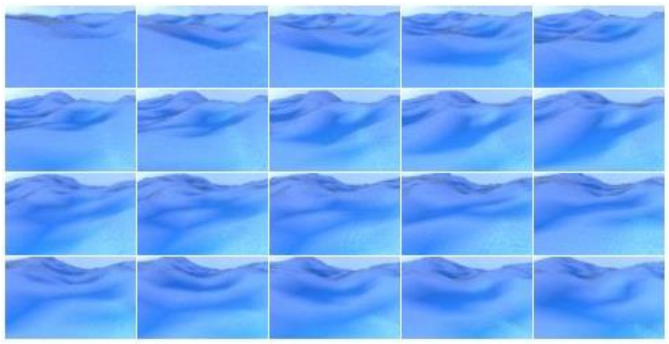


Figure 10: Generated images of incoming waves

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. All models will have to be retrained anyway 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.

3.1.1 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 needed to load the dataset. The following resolution parameters were used:

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

3.2 Real data

At this moment, data has been captured from the ASV but is not yet available.

The real data will be 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 for the model. Numeric motion parameters such as pitch and roll, are output by the ZED-mini at 800Hz. This data stream will have to be reduced to minimize data overlap. Since the simulation data was generated at two frames per second, 2Hz should be a good starting frequency 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 images and reduce computational requirements. The video footage will also have to be compressed to reduce the memory needed to process the images.

(From this point in the thesis onward, only the simulation data is used)

3.3 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 to 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. In the case of the ZED-mini, the data-stream needs to be reduced and cleaned as discussed in section 3.2.

3.3.1 Data cleansing

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 3.1. 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 11) 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 3.4.

Met opmerkingen [LD6]: Is this expected or can it be left out.

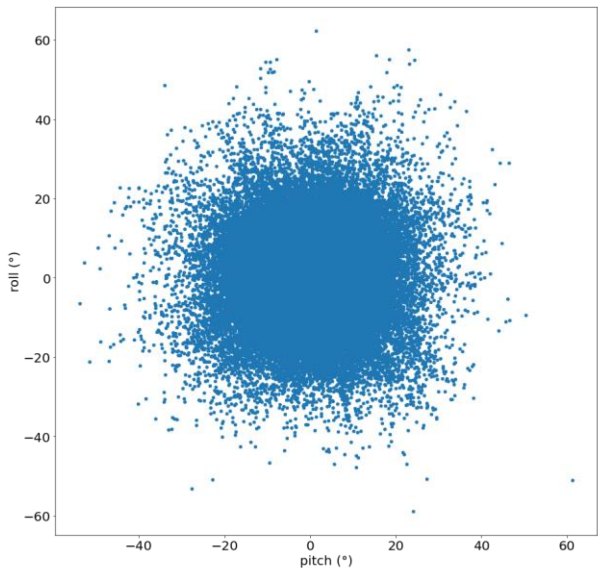


Figure 11: Scatterplot of pitch and roll

When real data is used, comprehensive cleaning is needed to filter out noise from the sensors and excessive outliers from inaccurate readings. *(TODO: add real data cleansing process)*

Met opmerkingen [LD7]: If possible

3.3.2 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 very 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.

3.3.3 Data reduction

TODO: Real data only: resizing/cropping/compressing, filtering 800Hz data and 60fps video to 2Hz

Met opmerkingen [LD8]: If possible

3.3.4 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.

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° and 90° (Equation 3.1). 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 3.1: 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 3.2).

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

Equation 3.2: Min-Max normalization of RGB-values

TODO: normalization for real data: images (cropping), thrust, wind velocity, heave? (m)…

3.4 Data analysis

The simulation data was briefly analyzed to have a better understanding of the results. Very in-depth analysis of the input data in 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.

3.4.1 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 1.

	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 1: Statistical information for pitch and roll in simulated dataset

Met opmerkingen [LD9]: Fix formatting

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 idea 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 and thus really good when all data occurs in a [-60°, 60°] interval. If all data lies in a [-10°, 10°] interval, three degrees is a 30% error which is very high. But the minimum and maximum don't tell the full story.

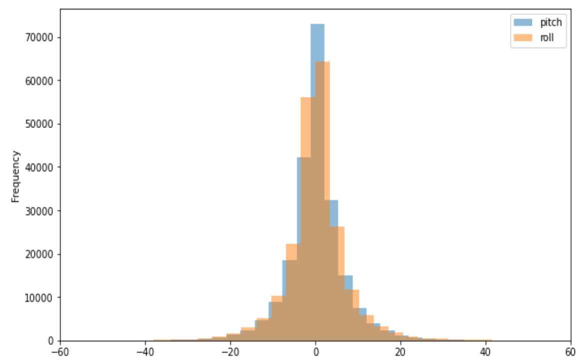


Figure 12: Simulated Pitch and Roll distributions

The distributions of pitch and roll are shown in Figure 12. Both of them show remarkably similar characteristics as a normal distribution. Intuitively, it can be assumed that the majority of the datapoints are located in a [-20°, 20°] 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 **μ** and a standard deviation **σ** of a normally distributed dataset, respectively 65%, 95% and 99.7% of all datapoints lie within a **μ** ± **σ, μ** ± **2σ** and **μ** ± **3σ** interval. If this rule is applied with the mean and standard deviation from Table 1, the 99.7% interval can be calculated and are the following (values rounded down for readability):

- Pitch: *[-19,74°; 19,86°]*
- Roll: *[-20,70°; 21,30°]*

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.

3.4.2 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.

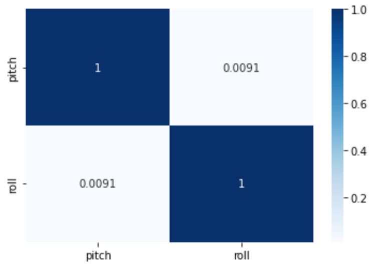


Figure 13: Correlation matrix for pitch and roll

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 13, 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 tested to predict both pitch and roll from an input sequence also containing both pitch and roll.

3.5 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.

3.5.1 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 10. 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.

Met opmerkingen [LD10]: Provide code?

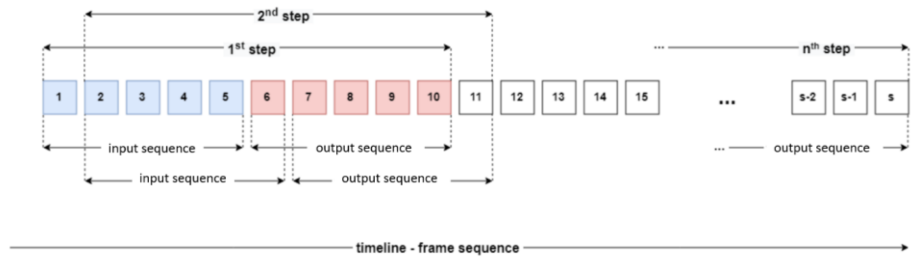


Figure 14: Sequence creation with moving input and output sequence

In Figure 14, 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 carried out, 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.

3.5.2 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 hasn't 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 5 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 14 shows this more clearly: the sequences from the 1st and 2nd 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.

3.5.3 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 carried out 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 4.4.

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

4 Model designs

Met opmerkingen [LD11]: Add parameter tables for input and output size, hidden size, kernel size, stride, padding, …

In this chapter, all models will be discussed that were created. An iterative process was used where each model design tries to improve on its predecessor. As discussed in the introduction, eventually hybrid models will be used to obtain optimal performance with the given data and problem definition. To identify different models based on their architecture, the following notations are introduced.

Notation	Explanation
P, R, P|R, PR	Resp. Pitch - Roll - Pitch or Roll - Pitch
LSTM	An LSTM architecture was used
CNN	An CNN architecture was used
FC	Fully connected linear layer
Single-/Multi-step	Single value output/sequence of values output
n	Number of frames in the input sequence
m	Number of frames in the predicted output sequence

4.1 Single-step LSTM models

First off, three multivariate single-step models were created. These models use a sequence of numeric input data for pitch and roll and predict one future step, hence ‘single-step’. They were designed to get familiar with the workflow of designing, training and testing neural networks with PyTorch. For simplicity, image data was also omitted to allow for simple model structures with few parameters that can be trained and evaluated quickly. In addition, since their output is single step, they were also not optimized to their full extend.

TODO: parameter table

4.1.1 Stacked single-output LSTM

The first two models that were created, were stacked LSTM architectures. These models use a sequence of pitch and roll as input and predict one value of *either* pitch or roll. The design parameters of these models were arbitrarily chosen. The architecture is show in Figure 15 on the left.

A two layered or stacked LSTM configuration was used. In this configuration, the output of the first LSTM is used as input by a second LSTM. Both LSTMs use 128 hidden neurons and have a dropout of 0.2. The last layer is a linear fully connected layer, aggregating the hidden features into a single output feature. Notice that the hidden output of a layered LSTM is a three-dimensional matrix where the first dimension is equal to the number of layers, and they are ordered by index. For the aggregation in the linear layer, only the hidden state from the last layer is used, hence the notation *hidden[-1]*.

4.1.2 Stacked multi-output LSTM

For this model, a slight adjustment was made to the above models. The number of output neurons in the linear layer was increased to two. This way, one frame of pitch and roll could be predicted simultaneously from a sequence of pitch and roll. The architecture is show in Figure 15 on the right.

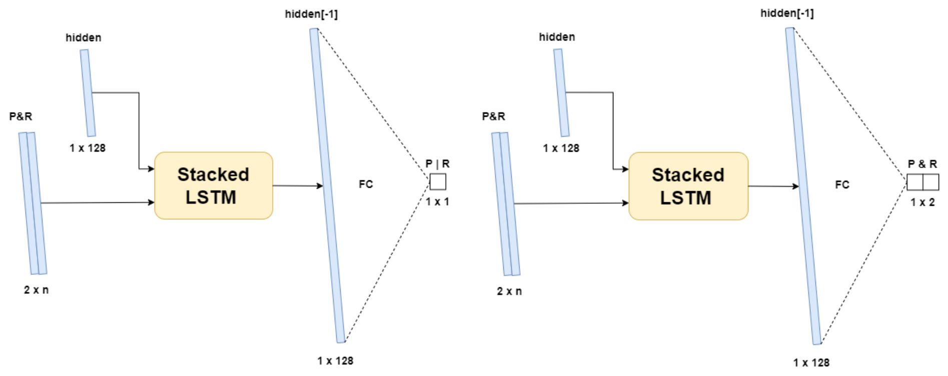


Figure 15: First generation LSTM model architectures

4.2 Multi-step encoder decoder LSTM

The next step in development was to expand the capabilities of previous models to be able to predict a sequence of pitch and roll. 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.

The architecture for this model was based on the baseline model from Mykola's research. He defined a similar model that only used numeric input data in an encoder-decoder LSTM configuration. This architecture was chosen to be implemented first because of its simplicity and because of its performance. It performed worst out of all tested models in his research, therefore it should be a good starting point to iteratively improve upon during this research.

Figure 16 shows the architecture of the implemented model. Two LSTMs are used together where the first one acts as an encoder followed by a second decoder LSTM. The first cell receives the sequential pitch and roll values and uses a hidden size of 300 neurons. 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. Finally, the latent vector from the decoder is passed through two linear layers that aggregate the vectors into a sequence for pitch and roll.

TODO: parameter table

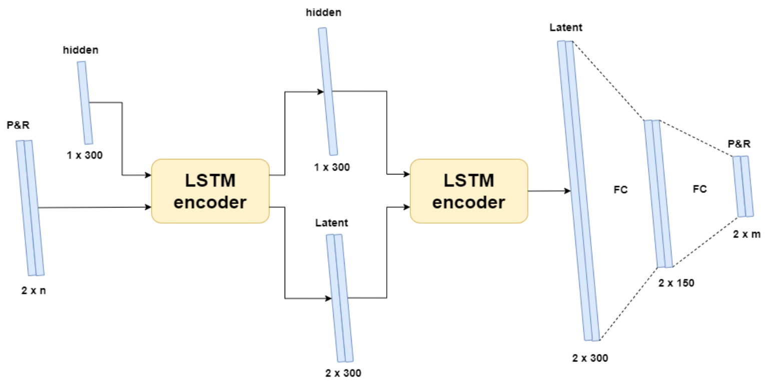


Figure 16: LSTM encoder decoder architecture

4.3 Multi-step CNN image-only models

After the LSTM models that only use numeric data, convolutional neural networks were introduced. This way, images could also be processed and used to aid with the predictions. Firstly, a simple CNN was created to familiarize with the new network architecture, its parameters and the image data loading process. After which, a more complex hybrid model was designed using both CNN and LSTM for more accurate predictions.

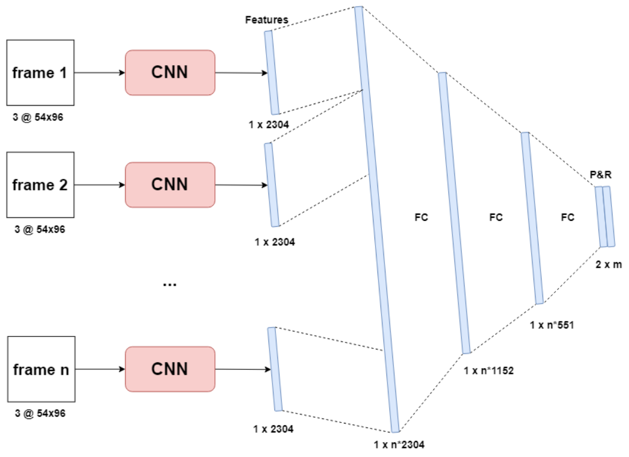


Figure 17: CNN to linear neural network architecture

4.3.1 CNN to linear

This model was created to test performances when using only CNN and linear layers. Without an LSTM model, 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. Each image is processed individually by the CNN and flattened to a feature vector. These feature vectors are then concatenated into one single large vector which serves as the input layer for the first linear layer. Three consecutive linear layers then aggregate this vector into smaller vectors and eventually into an output sequence of pitch and roll. The architecture for this model is illustrated above in Figure 17 together with its parameter table in Table 2.

Met opmerkingen [LD12]: Fix numbering

Input	Sequence of n frames at $(Nx3x54x96)$						
Name	*Layers*	*Kernels*	*Kernel Size*	*Stride*	*Padding*	*Input*	*Output*
Conv 1	Convolution ReLu	8 -	5x5 -	1 -	2 -	3 x 54 x 96	8 x 54 x 96
Pooling 1	Max Pooling	1	2x2	2	1	8 x 54 x 96	8 x 27 x 48
Conv 2	Convolution ReLu	16 -	3x3 -	1 -	1 -	8 x 27 x 48	16 x 27 x 48
Pooling 2	Max Pooling	1	2x2	2	1	16 x 27 x 48	16 x 13 x 24
Conv 3	Convolution ReLu	32 -	3x3 -	1 -	1 -	16 x 13 x 24	32 x 13 x 24
Pooling 3	Max Pooling	1	2x2	2	1	32 x 13 x 24	32 x 6 x12
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 2: parameter table for CNN to linear model

4.3.2 CNN LSTM for images

Secondly, a CNN-LSTM hybrid model was created. Many of the design parameters of the previous model were kept the same. The consecutive images in the input are processed individually by a CNN into 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 the linear layers from the

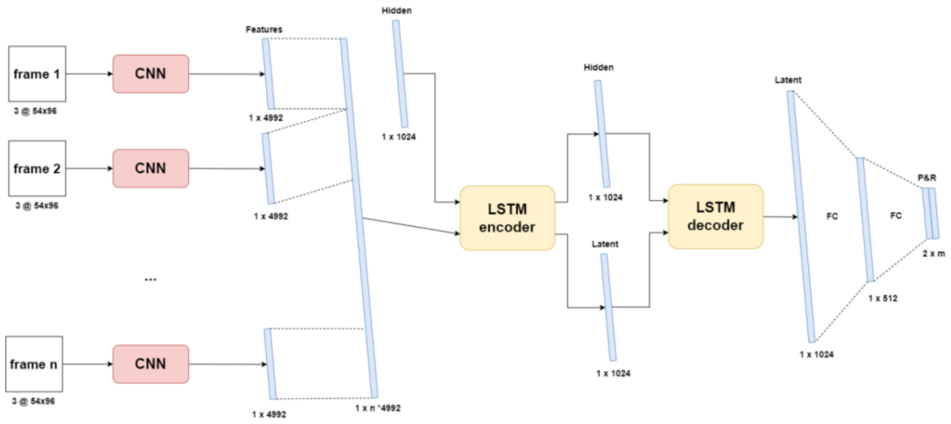


Figure 18: CNN LSTM hybrid model architecture

previous model. This LSTM encodes the large vector and produces a latent vector. The latent vector is then passed to a decoder LSTM together with the hidden state. The output of this decoder is then aggregated down in two linear layers to a sequence off pitch and roll output.

TODO: parameter table

4.4 Dual input models

4.4.1 CNN LSTM for images and PR

Lastly a model was designed to use both numeric data and images as input. It shares a lot of characteristics with the previous CNN LSTM hybrid model. The main difference between both is that in this case, the pitch and roll values for each frame are appended at the end of each frame's feature vector before they are concatenated into one large vector. This way, the sequence of the PR-values is still kept intact. Correct normalization is of utmost importance with this model as one type of data may overshadow the other if they aren't normalized to the same window.

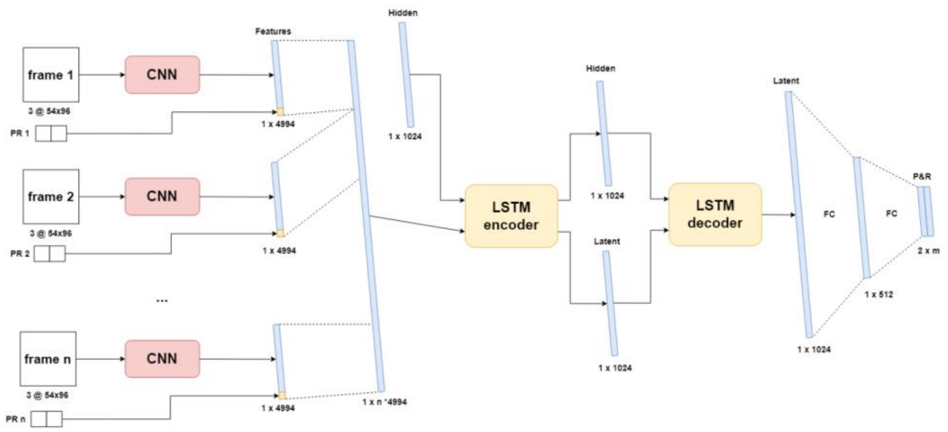


Figure 19: CNN LSTM image PR model architecture

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 to build 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 to high.

TODO: parameter table

5 Hyperparameters and optimizers

5.1 Hyperparameters

An important part to any deep learning problem is setting the hyperparameters and finetuning them for optimal performance. These hyperparameters affect the behavior of the model during training and thus have a direct impact on how well the model will perform afterwards. Hyperparameters are the parameters that can be manually set by the programmer. They are different from the internal parameters of the network such as the weights and biases whose values are derived during the training process. Many different hyperparameters exist, each with 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 3.5.2. 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 to model to not reach convergence by the end of training. Different optimizers exist to dynamically update the learning during training. In this research, ADAM was used as optimizer with an initial learning rate of **0.001**. ADAM will be discussed in greater detail in the next subchapter.
- **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. During training of all models, **50** epochs were used to train. This number was experimentally found to be sufficient to allow for proper convergence (Figure 20).
- **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.

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.

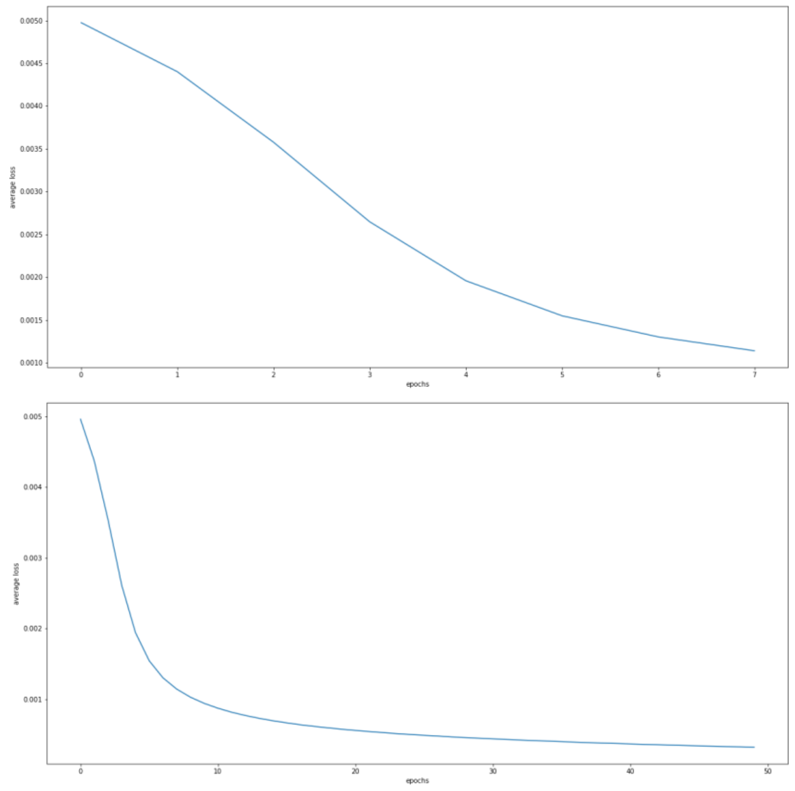


Figure 20: Average MSE loss during single step model training for 8 epochs (top) and 50 epochs (bottom)

5.2 Optimizer

As mentioned in the previous subchapter, different optimizer methods exist to dynamically change the value of the learning rate. 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. Adaptive Moment Estimation, Adam, is currently one of the best methods and therefore a very popular option (Doshi Sanket, 2019). The method is really efficient when working with problems involving a lot of data or parameters. For this reason and because of its easy implementation in PyTorch, Adam was used as optimizer for all models.

Adaptive moment estimation or ADAM is an optimization algorithm for stochastic gradient descent. Stochastic gradient descent maintains a single learning rate for all weight updates. This learning rate also does not change during training. With Adam, an adaptive learning rate 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.

5.3 Loss function

To compare the ground truth with the predictions made by the model, a metric is used to calculate the error, i.e., loss function. Many different functions exists to define this loss, each with their own appliances. For continuous data, the Mean Squared Error, MSE, is a popular option (Wang et al., 2022). The MSE of two sets of points of *n* elements, is defined as the average of the squared errors, where the error corresponds to the difference between ground truth Y_i and prediction $\hat{Y}_i$. Since the prediction of the proposed models is a series of values, the MSE calculated is an average over the full sequence.

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

Equation 3: MSE loss function

However, MSE does not handle trends very well. This is displayed in Figure 21. 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, a second metric was tested: Dynamic Time Dilation, DTW.

Met opmerkingen [LD13]: Add foundings

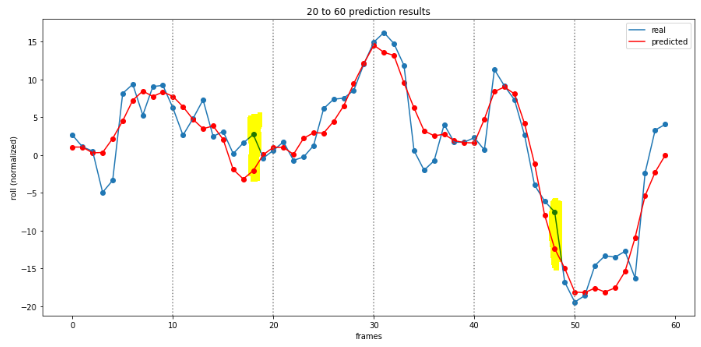


Figure 21: Prediction results with high error points marked

5.4 Hyperband

TODO, describe algorithm and concept

5.5 Inference time

TODO, describe method, concept and importance

6 Testing and Evaluation

To test the different models, the test dataset was used. Different methods were used to test the models. For example, prediction vs. real value graphs and average errors over the prediction sequence to name a few. Hyperband optimizations was only done to push the best models to their optimum. The simple models were experimented with to find the optimal values for their hyperparameters.

To analyze the performance of the models, mainly the MSE, RMSE and graphs depicting the real vs. predicted values were used. As a baseline, each model was also compared to a *zero-predictor*. This is a hypothetical model that predicts zero for every frame and follows no trend. This was done to evaluate whether the models captured the trends of the data well. If a model performs worse than a zero-predictor over the full test dataset, drastic changes need to be made to the design and hyperparameters. At that point, the model is behaving more like a pseudo-random number generator.

6.1 Single step models

Firstly, the three single-step models will be compared: the two single-output LSTMs and the multi-output LSTM. These models should perform very well as they only need to predict one value. The three models were trained for both 8 and 50 epochs and tested with an input sequence of 10 and 50 PR-values. Below, 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 data in the test data set.

MODEL (in -> out)	PITCH (denormalized RMSE)	ROLL (denormalized RMSE)
50 PR -> 1 Pitch - 8	4.69°	/
50 PR -> 1 Roll - 8	/	3.76°
50 PR -> 1 PR - 8	4.40°	3.54°
50 PR -> 1 Pitch - 50	1.85°	/
50 PR -> 1 Roll - 50	/	1.91°
50 PR -> 1 PR - 50	2.17°	2.03°
10 PR -> 1 pitch - 50	2.28	/
10 PR -> 1 roll - 50	/	2.25°
10 PR -> 1 PR - 50	2.34°	2.09°
Zero prediction	6.43°	7.30°

Table 3: denormalized RMSE for single-output LSTM models in different configurations

On the yellow rows, the results are shown for the models trained for 8 epochs with an input of 50 frames. Overall, the performance is very poor. The multi-output model performs slightly better than the single-output models on row one and two. The bottom row in orange shows the average error for a hypothetical zero-predictor model. As expected, these values are significantly worse than the trained models. This means that the models have indeed captured some form of underlying trend in the data and are making logical predictions. However, hyperparameter tweaking had to be done to increase performances.

Increasing the number of epochs allows the model to run through the full training dataset more. Therefore, it will update its internal parameters more and should be better performing. This is shown by the results on the blue rows. When increasing the number of epochs to 50, the RMSE values for pitch and roll both lower down to around 2°, which is a big improvement compared to previous results. The same configuration with 50 epochs was 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 50 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, the higher number of training iterations more than makes up for it. Only a marginal decrease in performance is shown compared to the 50 input 50 epoch models on the blue rows. From these results, it was concluded that input sequence has less effect on performance compared to the number of epochs and 50 epochs was chosen to be the standard for training the next models.

A recurring trend was noticed among all these results. Roll prediction errors are overall lower than pitch errors. This could be explained by roll having a somewhat more predictable behavior than pitch. For example, pitch could be having a lot more micro changes that are hard to learn and predict accurately. Additionally, when looking at the zero predictions, the roll prediction error is higher than the pitch prediction one's, meaning that roll in general has more points that are farther away from zero than pitch. So even with roll's higher peak values, it is still easier to predict than roll. Coming back to the criteria set in the objectives, all models trained for 50 epochs fall within the [-3°, 3°] error window, however, they can't predict a 30 second sequence.

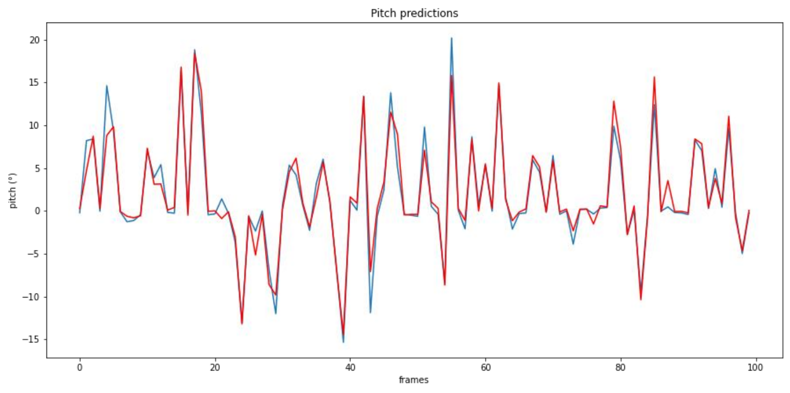


Figure 22: First 100 pitch predictions of single-step model

Figure 20 shows the result from the single-output model for pitch in the 10 input 50 output configurations. The predicted pitch is shown in red on top of the real value in blue for the first one hundred data points in the test dataset. This graph shows that the predicted pitch values seem to follow the trend very well but fall short of the real values by a couple degrees on some peaks. It is important to note here that since the test dataset was constructed by a random selection process as discussed in 3.5.2, the values on this graph are not in chronological order and are from different episodes. However, only one point is predicted per forward pass, therefore it is still possible to draw conclusions on how well the model can follow the trend in a sequence to predict a future value for that sequence. If multiple predictions of single future datapoints were to be very inaccurate, the red plot would not be overlapping the blue plot as well as it does.

Met opmerkingen [LDW14]: TODO: fix numbering

Met opmerkingen [LDW15]: TODO: recreate graphs with larger fontsize

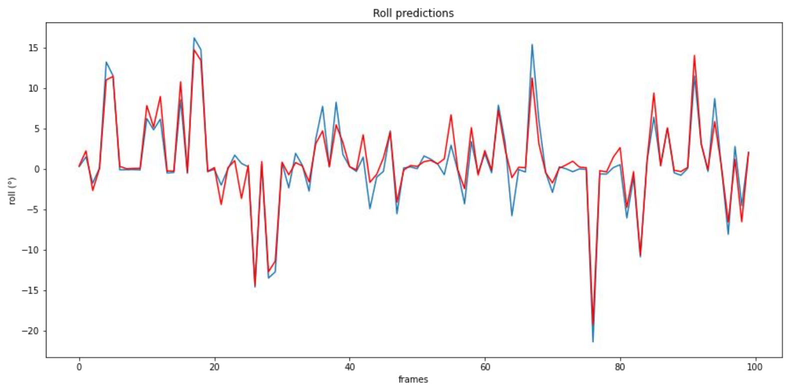


Figure 23: First 100 roll predictions of single-step model

Figure 21 shows the same graph for the single output model for roll in the same configuration. A similar conclusion to the previous model can be made. Interestingly enough, judging by these first one hundred points, the roll predictions are visually worse than those of roll. Where the pitch predictions only miss a few peaks on the blue line, the roll predictions fall short on almost every peak. However, based on the values in Table 2, this must be a coincidence on these first one hundred sequences.

With these results in mind, the multi-output model's performance is analyzed and compared in the same configuration. Figure 22 and Figure 23 show the prediction vs. real graphs for respectively normalized pitch and roll over all test datapoints. The dark sand-colored areas represent overlapping values between predictions and real values. Orange or blue areas mean that there are differences between predicted and real values. When comparing the two graphs, the same conclusion can be made that pitch is harder to predict than roll. The pitch prediction graphs shows a consistent blue area of missed peaks around the dark aera. This means that the predicted values fall completely within the dark area and thus fall short of the real blue values.

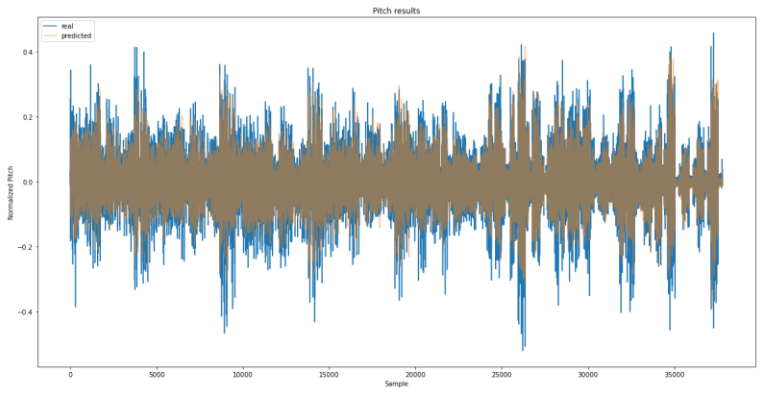


Figure 25: predicted pitch (orange) vs. real pitch (blue)

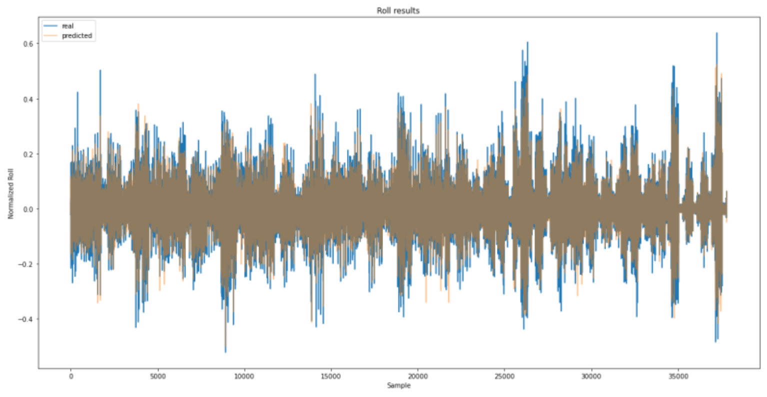


Figure 25: predicted roll (orange) vs. real roll (blue)

Finally, 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 doesn't know the validation data.

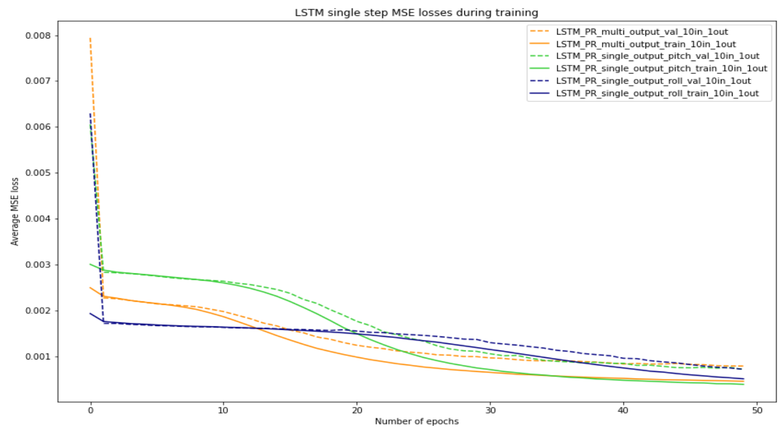


Figure 26: MSE loss during training of the single-step models

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.

6.2 Multi-step LSTM encoder decoder

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 error of approximately 30° of for both pitch and roll on the 10th frame. He used a ten input- and twelve output sequence length configuration 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° error is catastrophic when most data lies within a [-20°, 20°] interval. Based on these two reasons, it was assumed that Mykola had made an error during the calculation of this error.

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.

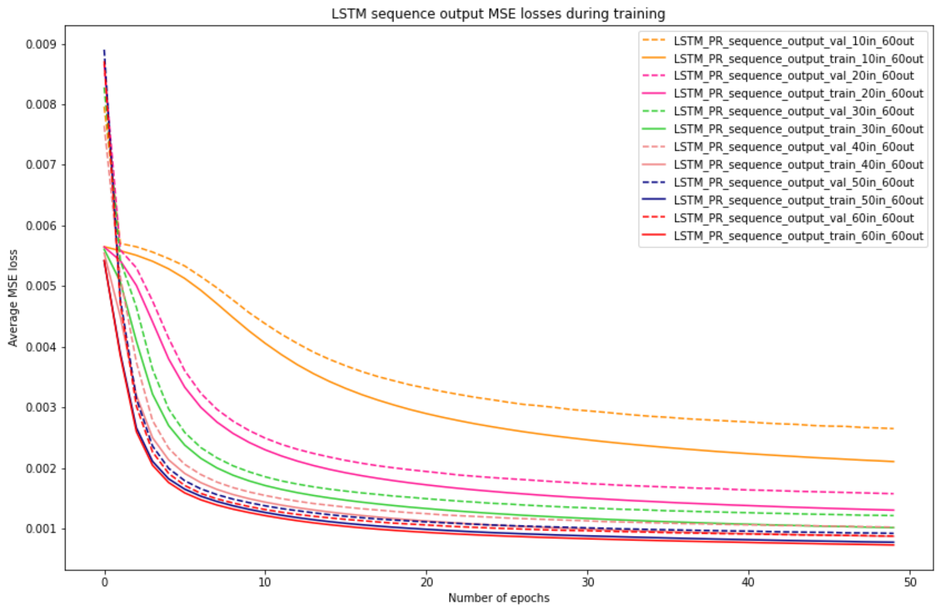


Figure 27: LSTM sequence model training and validation losses

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 in 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	PITCH (avg. error)	ROLL (avg. error)
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 4: Average pitch and roll error per IO-ratio

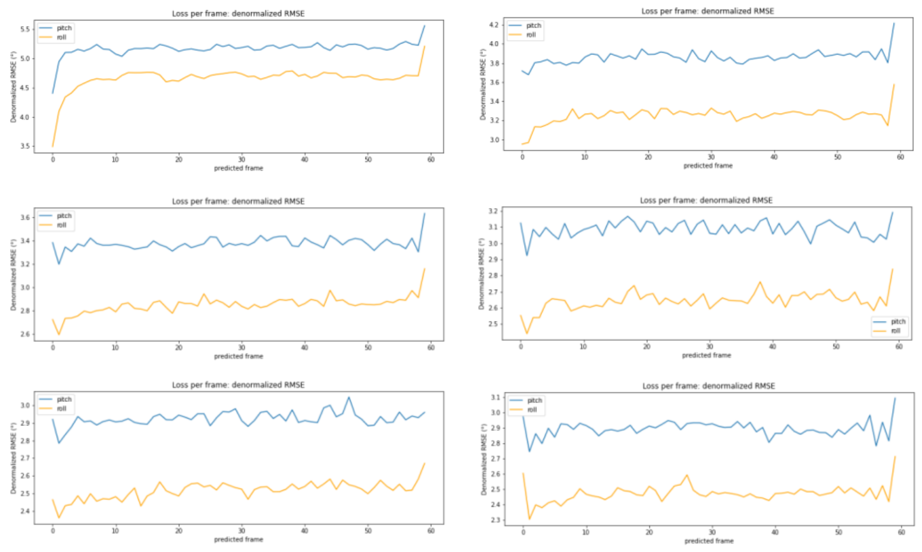


Figure 28: Loss per frame for different IO-ratios

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 show 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.

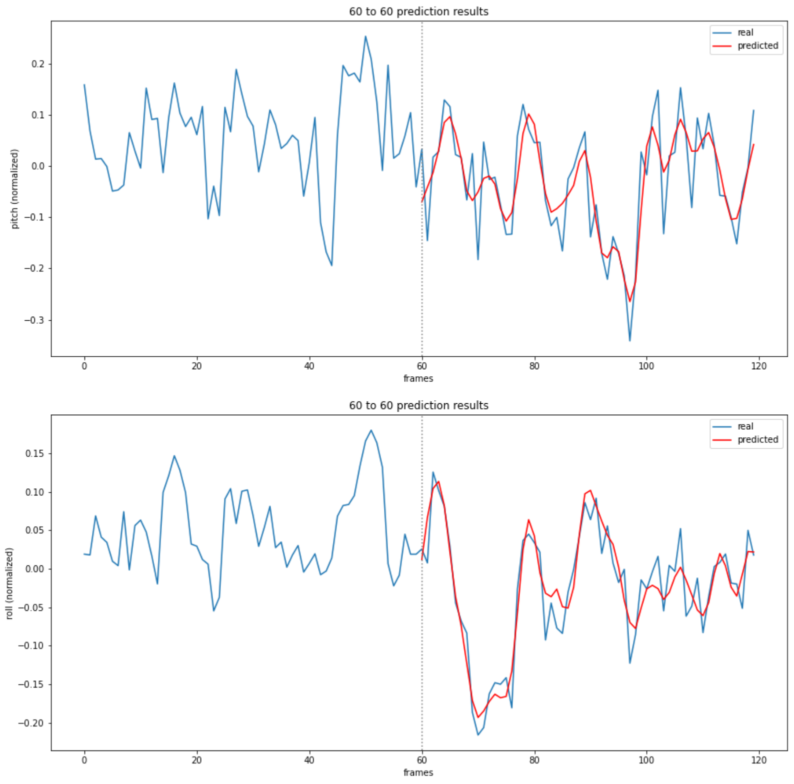
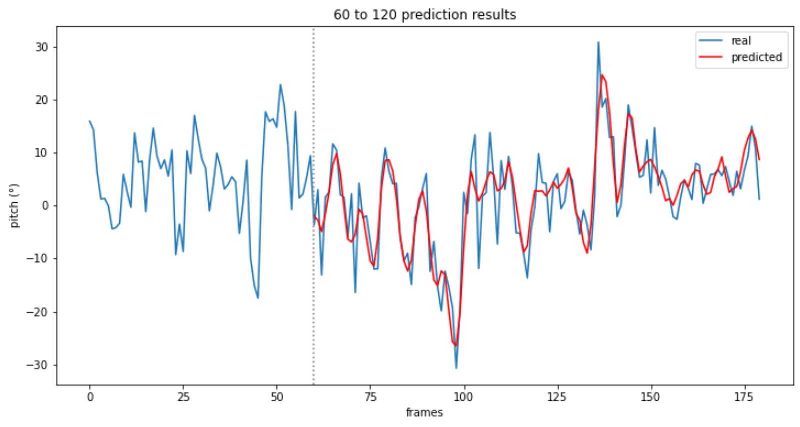


Figure 29: Prediction (red) vs. real (blue) for pitch (top) and roll (bottom)

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.

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 60th 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.



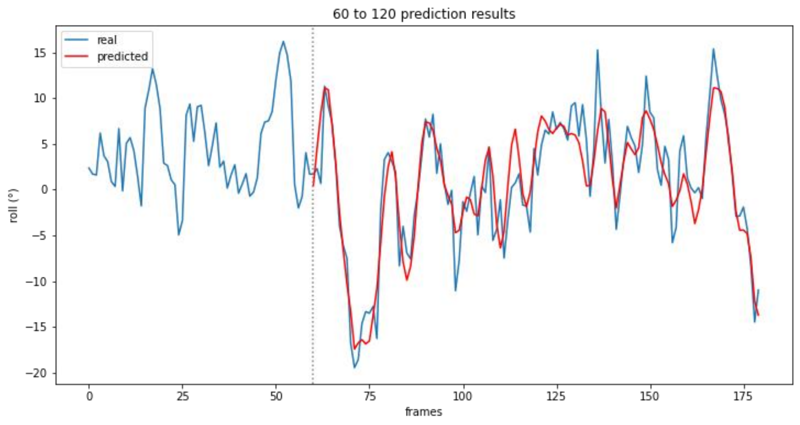


Figure 30: pitch (top) and roll (bottom) predictions for 60/120 configuration

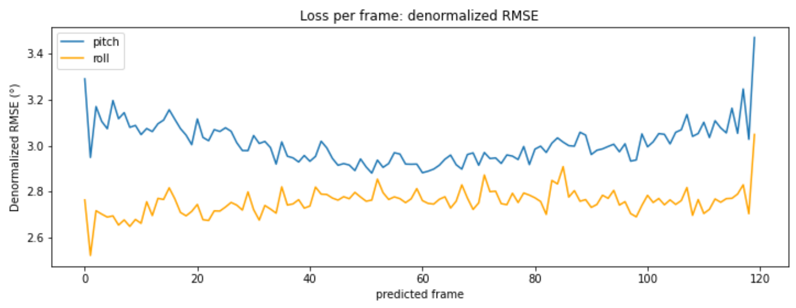


Figure 31: Loss per frame for 60/120 configuration

6.3 CNN to linear

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. The model was trained with an input sequence size of ten and an output size of sixty. These values were chosen in assumption that less images would be necessary compared to numeric data, to predict with the same level of accuracy as the LSTM sequence model.

TODO: describe results, fix training error before adding loss/epoch: 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 ass the same data and same training loop is used as the CNN LSTM image model.

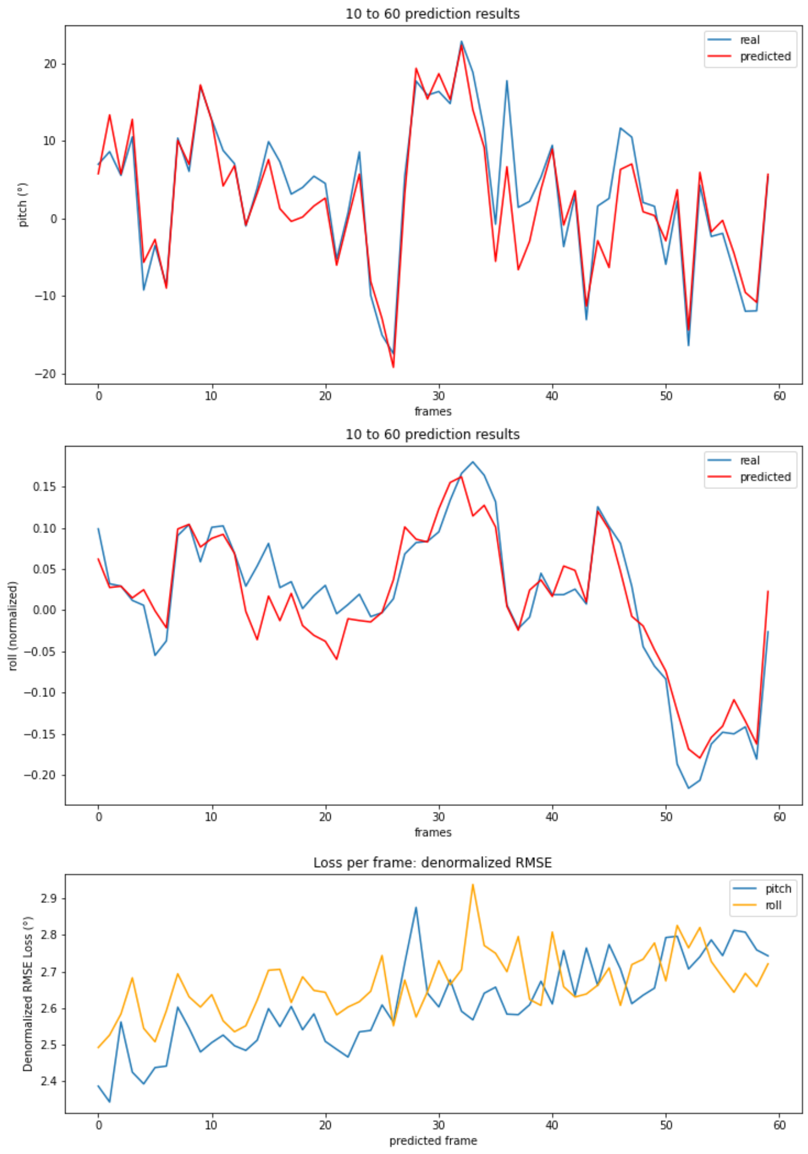


Figure 32: Predictions and LPF of CNN to linear model

6.4 CNN-LSTM image model

TODO: describe results

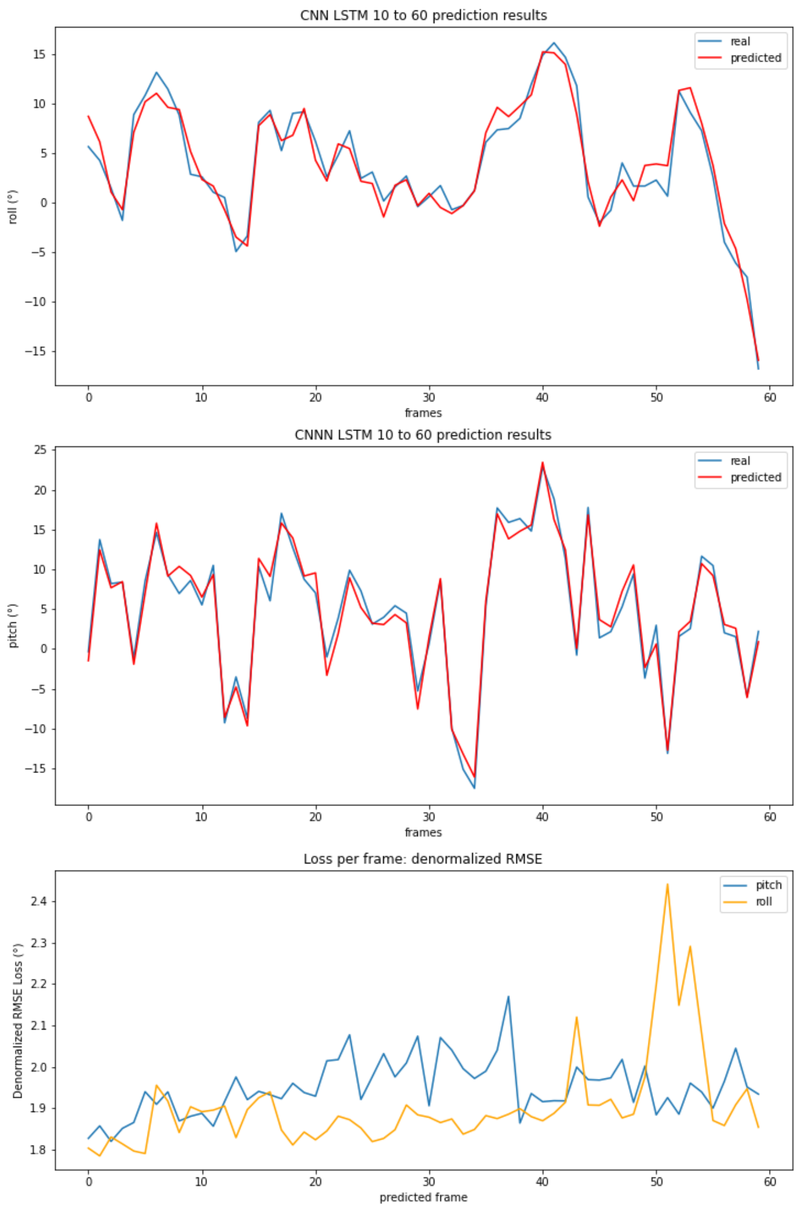


Figure 33: Prediction and LPF of CNN LSTM model for 10/60 IO-ratio

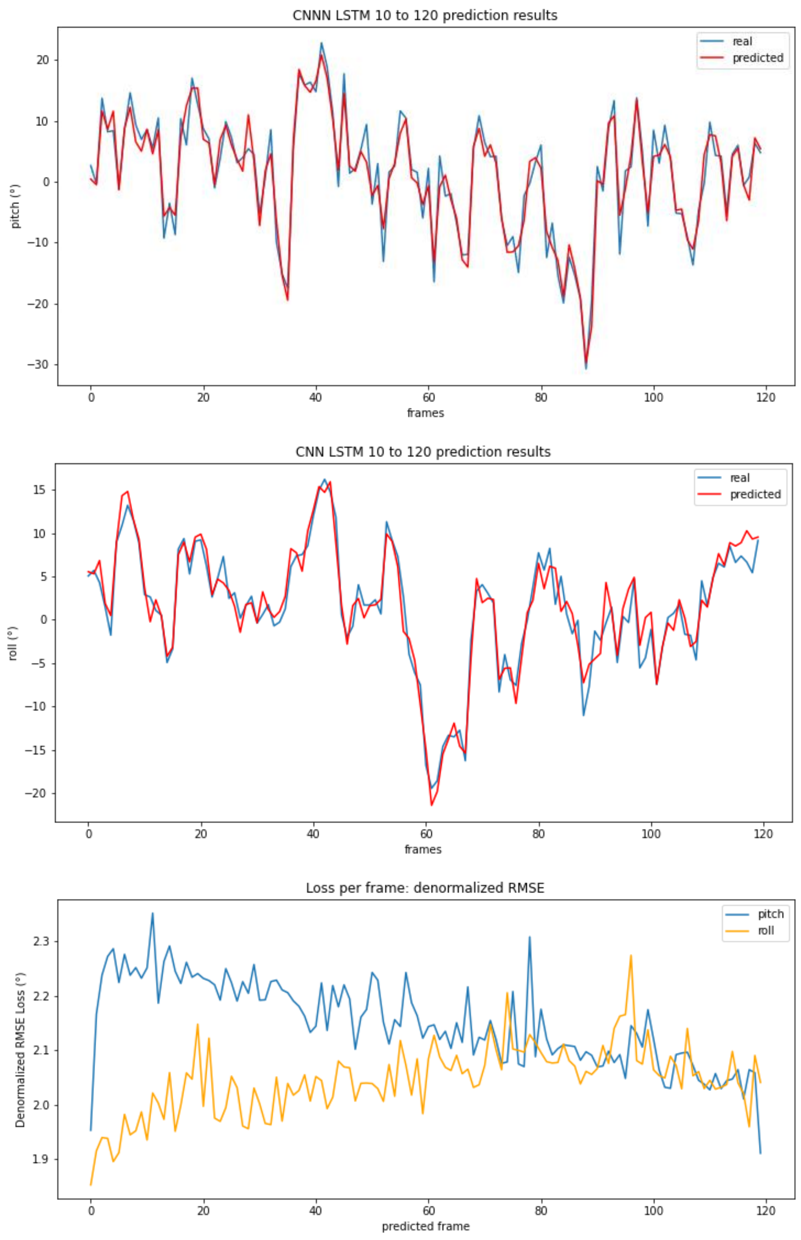


Figure 34: : Prediction and LPF of CNN LSTM model for 10/120 IO-ratio

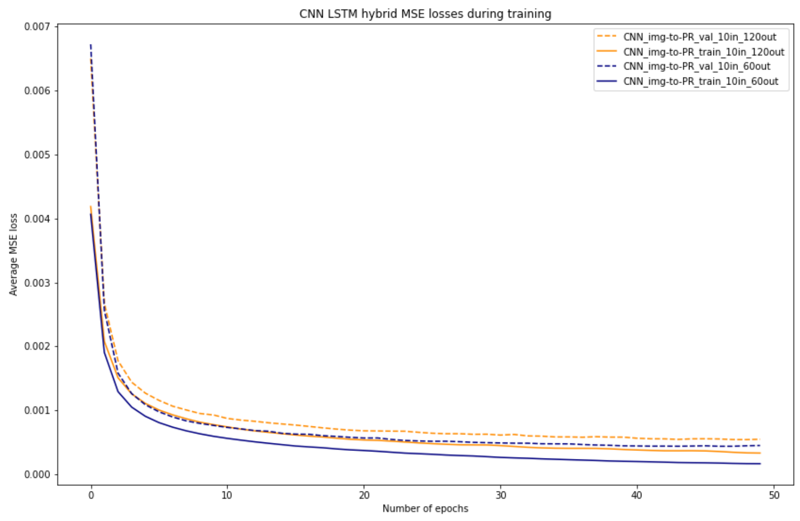


Figure 35: Training and validation loss for 10/60 and 10/120 configurations

6.5 CNN LSTM image and PR model

TODO, currently still takes way too long the make one forward pass (approximately 10 seconds) and has not been trained yet as a result of this. The torch.concat() method to append the 2 values of PR to the multidimensional (batch size, 1, flattened features size) Tensor is very inefficient and different solutions are still being experimented with.

6.6 Inference time

TODO, calculate inference time of each model and measure effect of IO-sequence sizes

6.7 Sequence models comparisons

TODO, compare all models to one another based on errors, complexity, requirements, inference time

6.8 Hyperband optimization

TODO, optimize best model's hyperparameters and analyze the results.

7 Conclusion

TODO

7.1 Discussion

TODO

7.2 Future improvements

TODO

8 References

Brownlee, J. (2017). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Machinelearningmastery. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

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

Doshi Sanket. (2019). *Various Optimization Algorithms For Training Neural Network*. Towards Data Science. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

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

Kaminskyi, 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

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/

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*, *7*(4), 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

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

Shen Kevin. (2018). *Effect of batch size on training dynamics*. Medium. https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

Wang, Q., Ma, Y., Zhao, K., & Tian, Y. (2022). 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

9 Appendix

TODO