Object Localization and Path Prediction Using Radar and Other Sources Towards Autonomous Shipping

Author:
Koen Vijverberg, BSc
koen.vijverberg@student.ru.nl

External supervisor:
Fons de Leeuw, BSc
fons@xomnia.com

Supervisor:
Prof. Dr. Elena Marchiori
elenam@cs.ru.nl

Second supervisor:
Prof. Dr. Tom Heskes
t.heskes@science.ru.nl

August 23, 2018
Abstract

In this master thesis we will explore the first steps on the road to creating an autonomous ship. We will create and test multiple path prediction algorithms, which can, based on previous positions, forecast the next positions of a ship. As a result, we will create a novel system that outperforms all benchmarked algorithms, although the performance seems to be unfit for live deployment yet. Besides path predictions, we will also train and test several recent bounding-box prediction algorithms on a custom dataset containing radar images. Results show that the newer neural network based systems significantly outperform the tested classical method, but, like the tested path prediction algorithms, they are not at the required level for autonomous deployment.
Acknowledgements

I would like to thank the people at Xomnia for giving me the opportunity to work on such an awesome and challenging project. Especially I would like to name Fons and David for your insights and thoughtful discussions, and Pieter, for introducing me to Xomnia, your tireless positivity is amazing. Thank you guys, your support helped a lot and kept me motivated.

Remco, thank you for allowing me to sail on board of your (huge!) container ships, those trips were truly an experience. Your willingness and drive to innovate the shipping sector is inspiring.

My parents and my girlfriend Jasmijn also proved to be invaluable while writing, providing a fresh look, and constructive feedback on my writings. This thesis definitely changed for the better because of you, from the bottom of my heart, thank you! And finally, I would like to thank my supervisors at the Radboud University, Prof. Dr. Elena Marchiori, and Prof. Dr. Tom Heskes for their help and advice.
# Contents

1 Introduction 4

2 Preliminaries 6  
  2.1 Scope of the project ........................................... 6  
  2.2 Navigating a ship .................................................. 6  
  2.3 Neural networks .................................................... 7  

3 Related work 12  

4 Data 13  
  4.1 Serial data .......................................................... 13  
    4.1.1 Example of NMEA data ....................................... 13  
    4.1.2 AIS ............................................................ 14  
    4.1.3 Inland AIS ................................................... 16  
    4.1.4 AIS preprocessing ........................................... 17  
  4.2 Inland ENC .......................................................... 21  
    4.2.1 Extracting S57 files ......................................... 23  
  4.3 Spatio-Temporal Graphs ........................................... 25  
  4.4 Radar ............................................................... 26  
    4.4.1 Radar decoding .............................................. 26  
    4.4.2 Radar image preprocessing .................................. 27  

5 Research 30  
  5.1 AIS - Path predictions ............................................ 30  
    5.1.1 Linear extrapolation ......................................... 31  
    5.1.2 Linear displacement .......................................... 32  
    5.1.3 Trajectory loss ............................................... 33  
    5.1.4 Vanilla RNN .................................................. 34  
    5.1.5 Social LSTM .................................................. 34  
    5.1.6 Attention LSTM .............................................. 36  
    5.1.7 Own method ................................................... 37  
    5.1.8 Batch generation and path prediction features .......... 40  
  5.2 Radar - Bounding-box predictions ................................ 42  
    5.2.1 Viola-Jones ................................................... 43
Chapter 1

Introduction

Inland shipping is big business in the Netherlands: in 2016 more than 17% of our cargo was transported over waterways, totalling 368 million tons [1][2]. Automating this shipping process will entail some important advantages, such as increasing safety (e.g. by reducing risks of collisions or damage because of improved connectivity and digitalization), cost reduction in terms of fuel and crew. It offers an opportunity for a more efficient use of the shipping and waterway capacity. Thus, automating the shipping process might contribute to sustainability as well. Eventually, efforts towards automatic shipping is mainly about competitive advantages: for sure inland shipping is closely watching digital, technical and economic developments. And not only within the sector of inland shipping itself, but also in alternative types of cargo transport.

Creating (semi-)autonomous mobility is a relatively new effort, enabled by recent improvements in machine learning and computer processing power. The efforts of Tesla CEO Elon Musk towards a self-driving car are widely known. It is not surprising that this idea is being applied to other types of mobility as well: autonomous trains [4], autonomous planes [7], and in this research: autonomous ships.

The purpose of this thesis is to explore how state of the art data science can contribute to the realization of autonomous shipping, and to build the first components on the road to a completely autonomous ship.

Creating a fully autonomous ship would make this research far too extensive: Rome was not built in one day and even Elon Musk has to cope with severe setbacks in his self-driving car project. Instead, we will limit ourselves to two smaller steps towards the realization of autonomous inland ships, namely the localization of ships in radar images and predicting boat
paths in a short future time frame. This leads to our main research question:

How can path prediction and object localization contribute to autonomous shipping?

This study takes the form of a case study. As such, it is practical and experimental in nature. The research object is the Factorfour, a container ship equipped with nautical sensor data gathering tools. We will collect both GPS and radar data on board of this ship. The GPS data will be obtained by interception of automated intership communication over FM frequency (AIS), providing GPS location, speed and heading of other nearby ships. The radar data will be gathered by linking a computer to the front radar of the ship.

In our opinion both radar & AIS data are vital for creating an autonomous ship\(^1\), thus we will experiment with both data sources in our research. This leads to our two research sub-questions:

1. Can we reliably predict a ship path using AIS information?

2. How accurately can we predict bounding boxes for ships in radar images?

For path prediction, we will experiment with different kinds of recurrent neural networks \([10][46]\), ranging from predicting GPS positions of one isolated ship, to combining radar-, environment- and GPS data of nearby ships. As a benchmark for the path prediction algorithms, we will employ linear extrapolation. To detect ships in radar images, we will employ state of the art neural networks architectures such as Fast(er) R-CNN \([39]\) and RetinaNet \([32]\). The bounding-box detectors will be benchmarked against the established Viola-Jones \([47]\) algorithm.

\(^1\)Section 2.2 explains in-depth how a shipper operates his ship.
Chapter 2

Preliminaries

2.1 Scope of the project

We will collect data from the Factofour (see figure 1), a container ship that is 135.0m long, 17.1m wide and weighs 6438 tonnes [9]. The Factofour is part of the fleet of the Shipping Factory, a company owned by Remco Pikaart. We can divide controlling the ship roughly into two categories: 1) Regular navigation over rivers and channels and 2) Special maneuvers: all other navigation activities, including docking, navigating in and near locks, loading and unloading cargo, etc. In this research we will focus exclusively on the first category, normal navigation of the ship, on the rivers Rhine and the Waal (see figure 2).

The Factofour transports containers back and forth between Cologne, Dusseldorf and Antwerp. The route can be seen in figure 2, although the route may vary slightly in the province of Zeeland, depending on water levels.

2.2 Navigating a ship

Steering a container ship requires careful planning, as collisions can cause a lot of damage [3] and these large and heavy ships tend to be slow to maneuver. When navigating a ship up- or downstream, the captain has to take into account the following elements: his own, and others ship sizes, speed and maneuverability characteristics. Also of importance are static objects like buoys, bridges, locks, the course of the river, the fairway\(^1\), and the depth of the river, which differs per season. In this research we will include all of these factors except the river depth, as we do not have any real-time or near real-time data information for this.

\(^1\)part of the river where ships are allowed to navigate, keeping the ships out of shallow water, and protecting the environment at the edges of the river
Figure 1: The Factofour, one of the three ships from the Shipping Factory, equipped with data-gathering tools for AIS and Radar data.

2.3 Neural networks

Neural networks are a type machine learning algorithm inspired by the brain: neurons. Since 2012 this type of algorithm has (re-)gained popularity due to its effectiveness on large datasets and complex problems [27]. We will make extensive use of neural networks in this research, since these are essential for the understanding and reproducability of our work. We will explain a few components of convolutional and recurrent neural networks.

A single neuron

Neural networks usually consist out of multiple neurons, connected to each other in some way. These individual neurons compute a simple function as shown in equation 2.1. Where the $x_i$’s represent the input, and $w_i$’s and $b$ are the trainable weights and bias respectively.

$$y(x) = \sum_{i=1}^{n} w_i x_i + b$$  \hspace{1cm} (2.1)

In most cases, there is also an activation applied after adding the bias, this is visualized in figure 3. The chosen activation function may vary, the hyperbolic tangent function (tanh) used to be popular, but in recent years the rectified linear unit (relu, equation 2.2) has gained popularity due to its faster computation, and non-saturation of the gradient [27].

$$y(x) = \max(x, 0)$$  \hspace{1cm} (2.2)
Figure 2: The main route the Factofour travels in the Netherlands, Germany and Belgium. From Germany to the coast province: the Rhine, the Waal, Boven Merwede, Nieuwe Merwede and Hollands Diep. From there on through the province of Zeeland to the port of Antwerp.

Figure 3: A visual representation of the operations applied in a single neuron. The $x$’s are the inputs, the $w$’s the weights, $b$ is the bias and $\phi$ represents the chosen activation function, if any.

**Feed-forward neural network**

The feed-forward neural networks, sometimes also called fully-connected neural networks, basics were invented by Rosenblatt et al. in 1957 [41]. The basic idea is to stack multiple layers of neurons together (multi-layer perceptrons), where each neuron is connected to all the neurons in the previous layer. The performance of a neural network is improved iteratively through a technique called backpropagation (RumelHart and Hinton [42]):
calculate the gradient with respect to the input, and adapting the weights and bias in every layer with gradient descent to bring the predictions more in line with the ground-truth. Figure 4 shows an example feed-forward neural network containing three layers: the input layer, a hidden layer, and the output layer. The input layer matches the number of inputs for a specific model, the hidden layer adds a configurable amount of latent features to the network, and the output layer matches the expected number of outputs for the model.

Figure 4: An example of a feed-forward neural network, containing one input, hidden and output layer. Each edge multiplies the incoming signal with its respective weight, the nodes add a bias to the sum of all the incoming edges to that specific node.

**Convolutional neural network**

The previously discussed multilayered perceptron is not well suited for image recognition tasks: since images are 2-dimensional, the number of parameters increases exponentially with the image size. A small image of $800 \times 800$ pixels contains $800 \times 800 \times 3 = 1,920,000$ values$^2$. Connecting such an input image into a multilayered perceptron with 5 hidden nodes would require approximately 38 MiB of weights, for just the first layer$^3$, exceeding the limitations of available system memory at the time [8]. Convolutional neural networks [28] proved to be a solution for this problem. They function by initializing a small weight kernel per input image, and multiplying this kernel in a sliding window fashion over the input, the memory requirements per computation layer improved vastly.

---

$^2$The $\times 3$ is because a color image contains 3 channels: red, green and blue, where a combination of the three can create any color.

$^3$1,920,000 $\times 5 \times 4 = 38,400,000$, 2 million inputs, all connected to 5 hidden nodes where we assume each weight to be a 32-bit floating point number.
The idea is that the initial convolutional layers capture lower level features of images, and subsequent layers higher level features required to make a correct prediction. For example, in a face detection network, the earlier layers might capture ears, noses and chins, and the final layers combine these features to discriminate between faces based on these captured features.

Of course, nowadays we have far more memory available for both the CPU\textsuperscript{4} and the GPU\textsuperscript{5} than available in 1998. In 2012 neural networks made a re-emergence in work by Krizhevsky et al. [27], since then neural networks have remained popular, and have obtained state of the art results on the ImageNet competition by stacking a large number of convolutional layers in sequence\textsuperscript{6}. Various neural network architectures will be tested in section (5.2) where we will predict bounding boxes of ships in radar images.

![LeNet-5 handwritten digit classification neural network architecture from 1998, authored by Lecun et al. [29].](image)

**Recurrent neural network**

Recurrent neural networks (RNN) are, unlike multilayered perceptrons and convolutional neural networks, not part of the feed-forward neural network family. The output of this type of neural network feeds back into the input, creating a loop. Figure 6 displays a simple representation of such a loop. This is useful in cases where an output is dependent, not just on the current input, but on preceding inputs as well. In our research we will mostly use Long-Short Term Memory (LSTM) [21] modules, but we will further elaborate on this topic in chapter 3 and section 5.1.

\textsuperscript{4}Central Processing Unit

\textsuperscript{5}Graphics Processing Unit, also called a graphics card, these are often used for neural network calculations for their massive parallel computation power.

\textsuperscript{6}Since then, various microarchitectures have been invented to train these kinds of neural networks, examples are Residual blocks (ResNet), Inception blocks, Dense blocks (DenseNet), and more.
Figure 6: A recurrent neural network (RNN), with on the left the schematic in- and output of the model, and on the right the unrolled loop for three time steps.
Chapter 3

Related work

Path prediction
There are a variety of methods available to predict motion, linear extrapolation being the simplest of all methods. Kalman Filters [23] can be employed to model uncertainty in the GPS measurements, partially negating the effects of noisy measurements. For pedestrian path prediction Helbing and Molnar introduced the Social Forces [20] model. There are a variety of path clustering methods [24][48][26][12], predicting future paths based on the similarity to earlier paths taken. Graves et al. [17] prove that recurrent neural networks can be applied for motion prediction, which inspired us to try to apply it to our specific domain of ship path predictions. The paper of Graves et al. makes extensive use of Long Short-term Memory (LSTM) [21] modules. Works of Alahi et al. [10] and Vemula et al. [46] are derived from the paper of Graves et. al. as well. They implement new methods to allow interaction of multiple entities in their neural network architectures.

Object detection
Conventional research in object detection relied on sliding-window approaches up until recently, examples of popular algorithms in this category are Deformable Part Model (DPM) [13] and subsequent improvements [43][14]. Another popular method for object detection is the Viola-Jones algorithm [47] from 2001. Recent popular object detection algorithms are mostly neural network based. Faster R-CNN [39] is the de facto benchmark: a two stage classifier which first localizes objects, and a second stage classifies them. Earlier works of Faster R-CNN relied on selective search [45] for the region proposals, this was replaced by a neural network in the latest version. With the desire for real time object detection, we saw the introduction of single stage classifiers such as: Single Shot detectors [34] (SSD), You Only Look Once [38] (YOLOv3), and the Feature Pyramid Networks [31], trading speed for accuracy.
Chapter 4

Data

The data used in this thesis originates from various sources, and most of it was processed in some form before usage. This chapter will describe each part and its transformations.

4.1 Serial data

The de facto standard to organize a commercial ship's data pipeline (gathering, sending, receiving, displaying) is through the use of the National Marine Electronics Association\(^1\) (NMEA) format. This format sends human readable ASCII sensor data over serial connections at a baud rate of 4800. This includes: engine control data, AIS data (more on this in section 4.1.2), weather information, positional information, depth of the ship at six points, the river pilot settings, which provides semi-automatic steering, and some other small sensors of lesser importance. Figure 7 shows an overview of the NMEA data collected on the Factofour. NMEA data is line-delimited (\(\textbackslash \text{r}\text{\backslash n}\)), each field of a NMEA message is comma-delimited (,). Only the first and last fields of NMEA data are specified in the standard, where a vendor is free to implement his protocol in the fields in-between. The first field is a two-letter code determining the type of message (e.g. gps-, weather or engine-data), preprendended by a dollar sign ($\$\$ or an exclamation mark (!), and possibly appended with more characters specifying additional information of the device. For example, GPS messages can be $GPAPA$ or $GPAPB$, originating from GPS device A and device B respectively.

4.1.1 Example of NMEA data

Hereafter is an excerpt of NMEA data caught on the Factofour’s main bus, a typical data rate is 40-50 messages per second, depending on various circumstances such as speed, location, current rate of turning, etc.

\(^{1}\text{https://www.nmea.org/} \)
4.1.2 AIS

Automatic Identification System\(^2\) (AIS) is a peer to peer information exchange system between ships, communicating positions, heading, speed and more miscellaneous data. It functions by installing an FM transmitter and receiver on each ship, enabling a communication range of about 15 kilometers. To limit the interference (multiple speakers and the same time) AIS devices implement the self-organized time-division multiple access (STDMA) standard. Figure 8 shows all communication channels that can be used by AIS.

The core of the AIS standard is, unlike NMEA, binary based. The bi-

\(^2\)https://en.wikipedia.org/wiki/Automatic_identification_system
Figure 8: Schematic overview of the communication of AIS in the real world. Namely: ship to ship bidirectional communication, ship to surface bidirectional communication, ship to ship communication via surface repeater station(s) and unidirectional ship to satellite links, where the satellite usually reports to a (fixed) base station. Credits: River Information Services (RIS), http://www.ris.eu/general/what_is_ris_/ais

Nary data is wrapped with extra data to make NMEA compliant: AIS sentences start with $AIVDO or $AIVDM delimiters, where $AIVDO are AIS messages from the ship itself, mainly meant for debugging purposes, $AIVDM messages are from other ships. There are 27 types of AIS messages, each for different purposes e.g. ship-ship position updates, ship-ship cargo information, ship-wall positional information exchange, buoy-ship and lock-ship communication, search and rescue data, etc. We will filter exclusively on ship-ship position reports, those are message types 1, 2, 3, 18 and
<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Position report</td>
<td>Scheduled position report (Class A shipborne mobile equipment)</td>
</tr>
<tr>
<td>2</td>
<td>Position report</td>
<td>Assigned scheduled position report; (Class A shipborne mobile equipment)</td>
</tr>
<tr>
<td>3</td>
<td>Position report</td>
<td>Special position report, response to interrogation; (Class A shipborne mobile equipment)</td>
</tr>
<tr>
<td>18</td>
<td>Standard Class B equipment position report</td>
<td>Standard position report for Class B shipborne mobile equipment to be used instead of Messages 1, 2, 3</td>
</tr>
<tr>
<td>19</td>
<td>Extended Class B equipment position report</td>
<td>Extended position report for class B shipborne mobile equipment; contains additional static information</td>
</tr>
</tbody>
</table>

Table 1: AIS position report types, quoted from http://arundale.com/docs/ais/ais_message_types.html

19, as shown in table 1. The official AIS standard is made by the International Maritime Organization (IMO) and the International Electrotechnical Commission (IEC), they require payment for the technical specifications. However, there are a few websites out there that provide a slightly out of date version for free\(^3\). Since all changes to AIS are backwards compatible (to ensure correct functioning of existing AIS devices, which are rarely updated), the decoding process stays the same for the ship to ship messages containing GPS positions.

### 4.1.3 Inland AIS

Besides the official AIS standard, the Central Commission for Navigation on the Rhine (CCNR)\(^4\) and the EU has stimulated to adopt an addition to the AIS standard, called “inland AIS”, which enables exchanging more information, meant for inland navigation over rivers and canals. This addition includes extra information such as its ENI (European Vessel Identification


\(^4\)A commission comprised of members from: Switzerland, France, Germany, Belgium and the Netherlands. This commission sets all the rules for shipping on the Rhine.
Number) transmission, additional search and rescue data (e.g. the number of crew members), additional cargo information and a few more small enhancements. All with the purpose to create a safer, more comprehensive and internationally compatible River Information Services (RIS) system, where each country had its own system previously.

The additions to the AIS standard are all implemented in AIS message types 6, 7 and 8, which are general purpose transmit types, created for extending the specification. The CCNR created a document with an overview of all changes made in (EU) legislation and the implementation specifications of inland AIS. The most quoted legislations are: Directive 2005/44/EC, Commission Regulation No 415/2007 and Commission Implementation Regulation No 689/2012.

4.1.4 AIS preprocessing

All collected AIS data is pushed through a preprocessing pipeline because it is noisy, and we are only interested in parts of the dataset. For example, we intent to use sensor data from the time intervals where the Factofour and other ships are traveling, and not stationary in the harbor. A container ship spends roughly 75-80% of its time in port, loading and unloading cargo. Including these parts of the dataset, would mean that the majority of data is comprised of ships performing special maneuvers like docking, undocking and navigating in complex situations. While those situations would be interesting to analyze, they are outside the scope of this research. Figure 9 depicts the complete pipeline from raw log files to ship path data. In the following sections we will describe each filtering step, explaining how and why it is performed. Table 2 shows the number of initial log files, the number of messages contained in those log files, and the number of messages retained after each filter. As an implementation detail: the data decoding and preprocessing was all scripted using Dask, a Python library that provides Graph-based operations on streaming data, effectively allowing to create out of memory operations on large datasets. This is useful because the total dataset size exceeds our system memory (18 GiB serial data vs 16 GiB RAM). An additional benefit is that operations in Dask can easily be parallelized, in contrast to native Python, which has to circumvent the Global Interpreter Lock (GIL) on compute-intensive workloads.

5https://www.ccr-zkr.org/files/documents/ris/vtt12_e.pdf
6Based on an interview with Remco Pikaart, owner of the Factofour.
7https://dask.pydata.org/en/latest/
8The Global Interpreter Lock (GIL) is a design choice of the Python programming language, it is a way to deal (partially) with the difficulties concurrency and writing thread-safe programs. In short: it allows for only one thread to run Python code (and thus modifying variables), a thread can yield the GIL to other threads when it is waiting for input or output (e.g. network or file operations), at some point the thread may receive
Figure 9: The complete pipeline for processing AIS data, from raw log files to ship path data.

<table>
<thead>
<tr>
<th>Step</th>
<th>Name</th>
<th>Number of AIS messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Log files</td>
<td>126,124</td>
</tr>
<tr>
<td>2</td>
<td>NMEA messages</td>
<td>230,067,103</td>
</tr>
<tr>
<td>3</td>
<td>AIS messages</td>
<td>16,193,945</td>
</tr>
<tr>
<td>4</td>
<td>Factofour travelling filt.</td>
<td>4,207,471</td>
</tr>
<tr>
<td>5</td>
<td>Speed and course filt.</td>
<td>3,260,847</td>
</tr>
<tr>
<td>6</td>
<td>Path interpolation filt.</td>
<td>2,255,729</td>
</tr>
<tr>
<td>7</td>
<td>Distance threshold filt.</td>
<td>2,243,408</td>
</tr>
</tbody>
</table>

Table 2: Initial number of log files, the number of messages contained in those log files, and the number of messages remaining after each respective filter. Figure 7 and section 4.1.4 describe all steps in a more elaborate way, this table only shows the most important ones plus the final AIS message count.

**Corrupt data**

The data generated on the Factofour is a continuous stream, this stream is written into one log file for each minute, on average each log file contains about 2800 NMEA messages. During the process of writing and transferring the log files to an external hard drive the log files may get corrupted. Of the 126,124 log files two turned out to be corrupt, and thus have been removed from our dataset.

**Invalid AIS data**

From each log file only AIS messages will be retained. The AIS messages will then be filtered based upon the following criteria: multi-line AIS messages the GIL back, and continue operations with the newly sent or received data. This reduces the idle waiting time of the CPU, but still only utilizes one processor core, while in compute-intensive task you’d ideally want to use all available cores. For a more in-depth explanation of the GIL I would like to refer to a presentation of David Beazley at PyCon in 2010, available at https://www.dabeaz.com/python/UnderstandingGIL.pdf

18
will be discarded, since all types of location update AIS messages fit into one NMEA line/message. AIS lines where the checksum (last 2 bytes of each NMEA message) does not match the message will be discarded due to their corruption. Some corrupt messages pass this test\(^9\), but fail upon decoding. These faulty AIS messages are caught using the try/except mechanism of Python, and thus do not influence the rest of the decoding.

**Stationary ships**

The CCNR requires ships docked on, or near the Rhine, to keep their AIS device switched on when outside of port. If a ship is docked at port, it is up to the shippers discretion whether to keep the system on or not. We assume that docked ships will not effect regular ship traffic, and thus can be ignored. Another factor that we have to account for, is the fact that an average container ship spends 80\% of its time at port. As a result, most of the sampled paths are from ships moving inside the port, usually consisting of special maneuvers such as docking, undocking and moving through locks. Since this is outside the scope of the research, as mentioned in section 2, we will have to limit the amount of data sampled from these time periods. We will use a filter that selects time samples of five hours or longer, in which the Factofour was traveling continuously. Since the Factofour only travels that long when it is moving from Germany to Belgium, or vice versa, path segments of the Rhine and its descendants will be selected. Figure 10 and 11 visualize the windows where the Factofour was traveling, and how these intervals are selected for data sampling.

**GPS, speed and course sanity checks**

Several faulty AIS messages circumvent earlier checks, assumed is that this is due to faulty sensors or sensor connections to the AIS transmitting devices. For example, out of 16,193,945 AIS messages received, 125,812 have a speed of 102.3125 knots, and a rotation of 360.0 degrees, which are the maximum values of the speed over ground (sog) and course over ground (cog) respectively, according to the AIS standard. Important to note here, is that both values are not at the maximum number that can be achieved with the number of bits available to each type, but rather at the maximum value the AIS standard defines. This means there is a very low chance the error occurred in transmission, due to the fact that the bits are in the unique position of the exact maximum of the specification, and the checksum of the NMEA message is correct. To correct errors in the GPS data, we will do a simple sanity check using rough GPS boundaries around the route of the

---

\(^9\) Presumably due to bits being flipped, accidentally matching the checksum, and thus passing the checksum test. The fraction of messages where this happens is so small (< 0.1\%) that we do not deem any further investigation necessary.
Figure 10: Speed of the Factofour (in knots) vs time. The peaks filled with blue have been selected as valid intervals to sample AIS data from. The minimal travel time is 5 hours, to prevent selecting windows where the boat is traveling from dock to dock in harbor. The month of March is missing due to a full hard drive.

Factofour (see figure 2). We have only kept GPS coordinates with a longitude between 3.0 and 8.0, and a latitude between 50.0 and 53.0, have not encountered any coordinates outside these boundaries. Figure 12 displays a histogram of the reported velocities of the remaining AIS messages after this filter.

UTM coordinate transformations

Up until this point, all coordinates are in the latitude/longitude format, also known as WGS 84\textsuperscript{10}. This reference system assumes the earth as an oblate spheroid, which enables accurate distance calculations\textsuperscript{11}, but relatively slow compared to the Euclidean or Manhattan distance measures. Since we need frequent distance calculations between points later on in our research, it is better to convert all coordinates to a reference system where we can use one of the simpler measures. For that reason we have chosen for the UTM 31 coordinate system.

Path interpolation

While the reporting interval of AIS updates is defined [6], in practice these scheduled reports still arrive irregularly. This may happen because of a few reasons: 1) A transmission conflict may occur while sending AIS data over the FM, AIS type A devices can send and receive data over two bands, if

\textsuperscript{10}https://en.wikipedia.org/wiki/World_Geodetic_System
\textsuperscript{11}https://en.wikipedia.org/wiki/Vincenty’s_formulae
another ship is transmitting at the same time in the same band, the data of both will be lost, and they will have to re-negotiate their time-occupancy in the transmission schedule (using the earlier mentioned STDMA algorithm).

2) A transmission might be lost through other means of interference, e.g. a solid object between ships decreases the chance of a message getting through.

3) A ship might go out of range, the FM transmitter has on average a range of about 15 KM. Outside of this area the chances of receiving a message is lower, so the received messages from such a ship will be intermittent. To predict positions of multiple ships, we need them to have coordinates in discrete time steps, we choose an interval of 10 seconds since this is close to the median of the reporting intervals.

Filter drifting ships

Due to slight drifting and/or GPS errors, anchored ships often report movement, which can pollute the dataset. To counter this, the reported positions of each ship have been filtered by a minimum distance traveled per time step: 5 meters per 10 seconds. Also, the distance between the first and the last reported position of each ship path must lie at least 150m apart. Figure 12 displays the speed of all recorded vessels after this filtering step.

4.2 Inland ENC

The Inland ENC Harmonization Group (IEHG) was founded in 2003 with the purpose to merge country specific ENCs into one standard. As a result we now have so-called S57 files. These files contain vector data of 4 different geometry types: Point, Line, Polygon, MultiPolygon. All figures in the map consist of these primitives. Figures 13 and 14 show how these vector
Figure 12: A histogram of the reported ship velocity (in knots). Data points from ships with a speed of zero and improbable high speed have been filtered. The mean velocity is 7.7 knots, with a standard deviation of 9.0, the median velocity is 6.6 knots.

data files can be combined to create a comprehensive picture of the river and the relevant nautical data. These S57 files are called (Inland) Electronic Navigation Charts ((I)ENCs). They contain information on the river and the surrounding environment, which a captain needs to navigate properly. It marks details on the map such as:

- Water area: every water related location, plus depth information of those waters
- Land area
- Fairway: parts of the river where ships are allowed
- Bridge positions
- Locks
- Buoy positions
- Other navigation markers
4.2.1 Extracting S57 files

As visible in figures 13 and 14, there are a lot of S57 files required to cover the Netherlands with parts of Germany and Belgium. Each of these files contain on average 2000 polygons to mark the fairway and water areas, all of the files combined there are 100,000 polygons. During our training process we needed to create environment images of the river and fairway. Creating these Numpy matrices from the vector files turned out to be unwieldy, as they took about 17 seconds per image. In order to accelerate this process, we merged all polygons where possible, and created a bitmap image of both the water area and the fairway. These are stored in so-called GeoTIFF files, which also contain handy coordinate mapping metadata. By utilizing the GDAL library\(^{12}\), this metadata is automatically interpreted to create coordinate mapping functions from pixel to real-world coordinates, and vice versa. Reading these bitmap images is significantly faster, it now takes about 0.05 seconds to generate an environment image on our hardware. Figure 15 shows the obtained IENC map data in combination with AIS path data. The fairway is green, the water area blue and recorded ship paths are lines in red.

![Image of map data](image)

Figure 13: OpenCPN, an open-source nautical map viewer. Each cell is a separate S57 file which gets loaded at closer zoom levels. The source data is from Rijkswaterstaat (RWS), the Dutch authority responsible for managing public roads and waterways, and the German and Belgian equivalents.

\(^{12}\text{http://www.gdal.org/}\)
Figure 14: OpenCPN zoomed in on the river Waal, near Nijmegen, showing the exact river location, the Fairway, the river axis, a lock, distance markers, etc.

Figure 15: A combination of geographic information system (GIS) data and AIS data. The "depth area" and "fairway" vector shapes are extracted from S57 nautical map files, the AIS data was gathered on the Factofour and has been plotted on top of the vector data. Depth area (everywhere where water is) is plotted in blue, the fairway (where boats are allowed to sail) is plotted in green. The large quantities of ships (red lines) has effected the clearness of the map negatively.
Figure 16: An example of an environment image evaluated on a position on the river Waal, near Nijmegen. The view range is 1200m in both directions, and the image has been generated at a resolution of 200\times200 pixels, so the image shows the real world at a resolution of 6 meters per pixel.

4.3 Spatio-Temporal Graphs

The sections 5.1.6 and 5.1.7 make use of the Spatio-Temporal (ST) graph terminology. The idea is to represent each entity (e.g. a pedestrian, a ship, etc.) as a node in the graph, and connect the interacting entities. We distinguish between two types of edges: spatial edges and temporal edges. Spatial edges represent distances between ships at a certain time, where the weight of an edge is the relative distance between the two nodes. The movement of each entity over time is represented by the temporal edges, weighted by the distance travelled in that time step. Figure 17 shows a small ST-graph containing three ships and time steps, where the edges crossing the dotted lines are temporal edges, those who do not are spatial edges.
Figure 17: An example Spatio-Temporal graph containing three time steps. The edges between the same ships (crossing the dotted lines) are temporal edges, weighted with the distance moved from that time step to the next. The edges within each time step are spatial edges, these are weighted with the distance to each other.

4.4 Radar

Radar is an acronym for RAdio Detection And Ranging, and is a collective name for all devices using radio waves to detect and locate objects. The Factofour is equipped with a JRC JMA-610 X-band marine radar. This type of radar sends 6kW radio pulses at Frequency of 9-10 GHz, and waits for the echos of those pulses to detect objects. By using the known speed of radio waves: (close to) the speed of light, and the time between sending and receiving a pulse, the exact distance can be calculated. Besides knowing the distance to an object, the position of the object is important as well. To measure this, the radar turns at a speed of about 25 revolutions per minute, effectively creating a radar sweep every 2.5 seconds. The position of the receiver determines the position of the detected object. The radar has a configurable resolution, view distance and anti-clutter settings. For these settings we have chosen the same configuration as the shipper uses: 4096 slices of data (per revolution), a view range of 800-1200 meters and minimal clutter suppression. Figure 18 shows the radar device installed on the bow and stern of the Factofour.

4.4.1 Radar decoding

The radar devices on the Factofour send their data in UDP multicast packets over the ships internal network. These are, due to being UDP multicast packets, delivered to each device on the network, so capturing these packets is relatively easy. The contents of these packets are binary-based. The first 64 bits contain meta-data, the north-up rotation of the collected sweep, the heading-up rotation and the bearing pulse, which indicates which parts of the sweep this packet contains data on. These numbers are all 11-bits wide, with 4-bit padding in between, in big endian format. The consecutive data
part of the packet is a repetition of 3 bytes for a pixel value (0-8), and a single bit for determining if the received value is from echo data, or filled.

4.4.2 Radar image preprocessing

Once the radar data packets are decoded, they can be used to generate images. An example of one these can be found in figure 19. We manually label boats in these grayscale radar images to create a bounding-box prediction dataset. For training and prediction we stack a variable amount of preceeding radar images in the channel dimension. This compresses extra temporal data into a single image, which boosts the predictions of the neural-networks. Figure 20 shows schematically how this is performed for three channels.
Figure 19: An example of a radar image taken near the port of Antwerp. This image is preprocessed as shown in figure 20.
Figure 20: On the left: an image containing three channels: red, green and blue, enabling the creation of a color image by combining the pixel values of each color. On the right: a preprocessed radar image, with three consecutive radar frames stacked on top of each other. This is possible due to the fact that each individual radar image only has one color channel (grayscale). Combining the consecutive radar images enables storing temporal information into a single image. The additional temporal information is helpful for a neural network to classify ships due to the fact that ships in almost every case move relative to the radar source.
Chapter 5

Research

Just like in the previous chapter about data, we will split our research experiments into two parts: section 5.1 describes our work on path prediction algorithms; in section 5.2 we will dive into the details of the bounding-box algorithms evaluated on our radar images dataset.

5.1 AIS - Path predictions

We will use 10 previous positions of the ship, and try to predict the 10 following steps, at a resolution of 10 seconds per step. Figure 21 displays one such sampled ship path. We will sample paths from our AIS data, this means that we will train, validate and test on all ships in the neighborhood of the Factofour, and not just paths of the Factofour itself. Each following subsection explains a separate way to tackle this same problem, starting with linear extrapolation, training and testing small and simple ”vanilla” LSTMs models, and then moving to the more elaborate architectures: Social LSTM, Attention LSTM, and our own method.
5.1.1 Linear extrapolation

The linear extrapolation is our simplest model, but at the same time quite a good benchmark. Most of the time a ship sails straightforward, and this model performs excellent on those parts. However, this model does not take the environment into account, nor the presence of any ship nearby. Equation 5.1 describes the extrapolation of $x^{t+2}$ from two known points $x^{t+0}$ and $x^{t+1}$. Figure 22 is a visualization of linear extrapolation on one sample path.

$$x^{t+2} = x^{t+1} + (x^{t+1} - x^{t+0})$$  \hspace{1cm} (5.1)
5.1.2 Linear displacement

For the neural networks, we will predict the offset of the ship off the linear predicted path as shown in figure 23, we do this because linear extrapolations are great for short term predictions, which we have with our resolution of 10 seconds per step. Another benefit of defining the learning problem this way is that the predictions are close to zero, which is ideal for training a neural network [30].
5.1.3 Trajectory loss

We train the Vanilla LSTMs, Social LSTM, Attention LSTM and our own method all with the same loss function for fair comparison. As our previous method, we predict the linear displacement of each point. We initially started with a standard mean squared error loss function, but this proved to have two problems. First, the mean squared error is susceptible for outliers in the data, and GPS data is noisy: civilian GPS accuracy is theoretically $\leq 7.8m$ with a 95% probability [5], but in practice the average error is closer to 15 meters. Secondly, due to the noisy GPS position reports the linear displacement vectors are not smooth: e.g. while making a left turn in a river bend, the first two linear displacement vectors indicate (correctly) to the left, but the third may point slightly to the right. Overall the vector is correct, but the individual measurements are noisy.

As a solution we create a loss function which we call the ”trajectory loss”, consisting of two components: the mean absolute error per point, being less susceptible for outliers, and the absolute error of the cumulative sum of the displacements encountered up until each point. This second component is included to encourage the network to learn a smooth path: e.g. a near constant displacement when turning in a river bend, while still rewarding the network for taking the right path overall.

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| + \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{i} |y_i - \sum_{j=1}^{i} \hat{y}_i|$$  \hspace{1cm} (5.2)
5.1.4 Vanilla RNN

The first recurrent neural network we will test is the so-called “vanilla rnn”, this is a simple model with one Long Short Term Memory (LSTM) module, schematically pictures in figure 24.

We create two models of our vanilla RNN. Model (a) only uses the previous linear displacements as an input, model (b) uses the linear displacements as well, and adds temporal distances to the previous positions of the ship to the feature vector. Each of the described input features are embedded into feature vectors of length 128, the LSTM has a hidden size of 128. The last layer is fully connected to transform the output to a 2-d vector (x and y coordinates).

![Vanilla RNN diagram](image)

Figure 24: The vanilla recurrent neural network. Model a uses only the linear displacement features, model b adds temporal data to the input vector.

5.1.5 Social LSTM

Originally designed to model human movement in crowds, the Social LSTM paper [10] introduces a concept called ”social pooling”. This pooling layer embeds features of neighbors, mutually exchanging information of the path taken thus far. The architecture of the Social LSTM network can be found in figure 25. First, the temporal features are embedded, these are concatenated with a generated social pooling tensor (see next section). These embeddings are then forwarded through a LSTM module to predict the next position.
Figure 25: The Social LSTM architecture, the model utilizes two inputs: temporal and spatial locations, and predicts normalized absolute coordinates. The temporal locations are first embedded and then combined with the social pooling tensor. For the first iteration this is a zero-matrix, in subsequent time steps these are filled with the embeddings of neighbors depending on their relative spatial location. See sec. 5.1.5 for more information on the social pooling layer.

**Social Pooling layer**

The social pooling layer is a method to embed feature vectors of a variable number of neighboring entities (=ships) into a fixed size matrix. This "social" matrix has a dimension of $\text{pooling.size} \times \text{pooling.size} \times \text{num.features}$, where the relative position of a neighboring ship to the current ship determines the location in the first two dimensions, the last dimension is filled with the hidden state (from the RNN) of the neighboring ship. If there are multiple ships in the same position, their feature vectors are added. This happens quite often since the paper defines a $\text{pooling.size}$ of four. The hidden size is 128. Figure 26 visualizes the pooling layer for four pedestrians, where only the nearest pedestrians are combined in the pooling layer. Note that this differs from the implementation of the paper, there all pedestrians in the current frame are embedded in the pooling layer.
Figure 26: Overview of the social pooling concept, designed by Alahi et al. [10]. For each time step and individual, the features of neighbors are embedded in a 3-d matrix ($pooling_{size} \times pooling_{size} \times num\_features$), where the assigned position of the neighbors in the pooling matrix depends on the spatial position of that neighbor relative to the individual. If there are multiple neighbors in the same quadrant, their features are added.

5.1.6 Attention LSTM

The attention LSTM paper [46] was written as a successor to the Social LSTM. It introduces a concept called attention: a trainable weighting of nearby entities. This enables the network to prioritize certain entities in its decision making process. For example, an approaching ship might be given a higher priority than a closer ship that is moving in the same direction and speed. The inputs for the Attention LSTM are the same as for the Social LSTM: the ST-graph, containing spatial and temporal distances, and linear displacements. Figure 27 displays the high level architecture of the Attention LSTM architecture, slightly modified by the addition of the linear...
displacement pipeline, designed after the original model.

Figure 27: The attention LSTM model has three inputs: the linear displacement vectors (see figure 23), the temporal distances: distances to previous positions of the boat, and spatial distances: the distances from one ship to another. All inputs are embedded into 128-length vectors through a linear layer with ReLu activation. Then each variable is forwarded through an LSTM module to capture time dependencies. The spatial data is forwarded through the attention module to prioritize and include features of nearby ships into its own prediction process.

Figure 28: The attention module in detail. It has two inputs: the temporal LSTM data and the spatial LSTM data. Both are forwarded through a linear layer to reduce dimensionality. We take the softmax of the dot product of these two vectors, the output softmax is then used for a weighted sum of the embedded spatial data. This allows the model to weigh features from one ship over the other, creating the "attention" model.

5.1.7 Own method

Our own method builds upon the attention LSTM framework, and adds a pipeline to embed environment information into the feature vector. The following section explains in-depth how the environment embedding network is trained, and how we extract features from it.
Environment embedding

As mentioned in section 2.2, the surroundings of the boat are important for navigation decisions. Section 4.2.1 explains in-depth how we obtain environment information, and visualize it as in figure 16 and 21. To use these images in our recurrent neural networks, we need to extract features from the environment images. We have decided to use a small variational autoencoder [25] (VAE) network, created in a recent research by Ha. et al. [18]. An advantage of this method is that it allows for easy comparison of a generated image vs. the original, giving an indication of the feature embedding quality of the network.

The variational autoencoder consists of an encoder and decoder part. The encoder has four 2-d convolutional layers with a kernel size of 4 each, the number of filters per layer are 32, 64, 64 and 128, respectively. There are two fully-connected layers connected to the last convolutional layer, one outputting the mean vector, and the other the standard deviation vector, both output the same length: 128. The decoder part starts with a fully-connected layer with 1024 outputs, and re-creates an image out of this with 4 (2-d) transposed convolutions, the first two with a filter size of 5, the last two with a filter size of 6. The number of filters for the transposed convolutions are 64, 64, 32 and 2 respectively. Figure 29 shows an overview of the VAE network structure.

Figure 29: Schematic overview of the variational autoencoder. On the left: an input image, on the right: a re-created image by the network. Note that the image is somewhat blurry, and details of the river at the edges is lost. The network is trained on best re-creating the original image, minimizing the absolute error of each pixel, and minimizing the KL-divergence between the generated means & standard deviations and unit gaussian distributions.
**VAE loss function**

The loss function of the variational autoencoder is a composition of the Kullback-Leibler divergence and the mean squared pixel error. The first component, the Kullback-Leibler divergence measures the divergence between two probability distribution functions (PDFs). For two normal distributions, this is defined as:

\[
KL(p, q) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \quad (5.3)
\]

For the VAE, we define the KL-loss as the divergence between the predicted PDF and a unit gaussian distribution. Substituting \(\mu_2 = 0\) and \(\sigma_2 = 1\) into equation 5.3 yields our KL-loss function:

\[
KL(\mu_1, \sigma_1) = -\frac{1}{2}(\log(\sigma_1^2) + \sigma_1^2 + \mu_1^2 + 1) \quad (5.4)
\]

Where we predict \(\mu_1\) and \(\log(\sigma_1^2)\) with our network to minimize computations. The mean squared pixel error for an input image \(y\) with \(n_{pix}\) pixels, and a predicted image \(\hat{y}\), is defined as:

\[
Pix(y, \hat{y}) = \frac{1}{n_{pix}} \sum_{i=1}^{n_{pix}} (y_i - \hat{y}_i)^2 \quad (5.5)
\]

The combined loss function is defined as a weighted sum of these two components:

\[
L(y, \hat{y}, p, q) = w_{pix} \cdot Pix(y, \hat{y}) + w_{KL} \cdot KL(p, q) \quad (5.6)
\]

Where we choose \(w_{pix} = 10\) and \(w_{KL} = 1\).
5.1.8 Batch generation and path prediction features

At this point we have seen all features used for the path prediction methods, for clarity, all features that are created per batch are listed in table 3.

The batch generation is as follows: each dataset is assigned a (fixed) set of time intervals where it allowed to sample from. The valid intervals are selected as described in section 4.1.4, and visualized in figures 10 and 11. This list of valid intervals is chronologically split into three parts for the train, validation and test set. Splitting the dataset this way avoids data leakage, since each dataset simply cannot sample the same path as another.

Then, for each batch, a random ship is selected from the ships present in the time frame where the dataset is allowed to sample from\(^1\). Then, from this selected ship, a path of length 20 is selected\(^2\), and all ship (paths) that are completely defined for this time interval are included in the batch as well. These ships, time steps, and positions combined are the Temporal features.

The linear displacement features are then obtained by extrapolating each point, and recording the differences between the ground truth and the predicted point, as described in section 5.1.2. The spatial positions are simply the x and y distances between each ship in meters, a \(n \times n\) relation.

The environment images are obtained as described in section 4.2.1, extracting marine map data to vector and raster data, which allows us to create an image (in numpy matrix form) of the course of the river and fairway for any given position. The rotated environment images are rotations of the previously mentioned environment images, such that the direction of the ship is up, instead of north up. We generate the environment features based on these rotated environment images, as described in section 5.1.7, where each rotated environment image is pushed through the encoding part of the trained VAE network.

\(^1\)The ships positions are heavily preprocessed as described in section 4.1.

\(^2\)Where each time step is 10 seconds, as described in section 4.1.4, "Path interpolation".
<table>
<thead>
<tr>
<th>Feature shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporal position</td>
<td>$n \times t \times 2$ x,y coordinates in UTM-31 format of $n$ ships for $t$ time steps</td>
</tr>
<tr>
<td>Linear displacement</td>
<td>$n \times t \times 2$ x,y displacements in meters of $n$ ships for $t$ time steps</td>
</tr>
<tr>
<td>Spatial position</td>
<td>$n \times t \times n \times 2$ x,y distances in meters of $n$ ships for $t$ time steps to $n$ other ships.</td>
</tr>
<tr>
<td>Environment image</td>
<td>$n \times t \times 64 \times 64 \times 2$ environment image based on the ship location, for $n$ ships and $t$ time steps</td>
</tr>
<tr>
<td>Environment image rot.</td>
<td>$n \times t \times 64 \times 64 \times 2$ rotated environment image, direction up instead of north up</td>
</tr>
<tr>
<td>Environment features</td>
<td>$n \times t \times 128$ VAE encoded features based on the rotated environment image</td>
</tr>
</tbody>
</table>

Table 3: Features present in each batch. The first two dimensions are always $n$ ships, and $t$ time steps, where $n$ is variable, depending on the number of ships in range of the Factofour, $t$ is fixed at $t = 20$. We format our data this way for easier compatibility with the Pytorch framework.
5.2 Radar - Bounding-box predictions

At a later stage in an autonomous ship, ideally we would like to have multiple sensors detecting ships and other possible obstacles. This broadens the spectrum of which ships we can detect (since AIS only captures the presence of larger recreational and commercial vessels), and increasing the odds that a ship will get detected. In order to do this, we decided to implement computer vision on radar images. We decided to limit ourselves to bounding-box detection of ships in the radar images, postponing detection of buoys and other obstacles for future research. There are a multitude of existing algorithms for bounding-box detection, we chose to test the Viola-Jones [47] algorithm as a classical approach. Besides the Viola-Jones algorithm we test two neural network based methods: Faster R-CNN [39], and the recently published RetinaNet [32] model.
5.2.1 Viola-Jones

The Viola-Jones algorithm\(^3\) is a method for bounding-box detection. It was created in 2001 and focussed mainly on face detection. The algorithm has three major components: "Haar" features, Adaboost and the Cascade architecture.

**Haar features**

The Viola-Jones algorithm bases its predictions on common shifts in pixel intensity in the input images of the same class. For example, when detecting

---

\(^3\)The Viola-Jones paper has 17,428 citations (July 2018).
human faces, the nose bridge is almost always of lighter color than the surrounding area (see figure 31). This shift in pixel intensity is captured in so-called "Haar features", which are the central component of the Viola-Jones algorithm. The algorithm creates multiple Haar features to achieve the required accuracy, at the expense of processing speed.

![Figure 31: An example of a three-rectangle haar feature matching the nose bridge. The area under the white area is summed and subtracted from the sum under the black area, the resulting number is the feature value on that specific position of the haar feature, which is normally applied in a sliding window fashion over the entire input image.](https://example.com/figure31)

**Adaboost**

A typical Haar feature is contained in a $24 \times 24$ pixel area, and can contain two, three or four rectangles, which in total sums up to 162,336 possible positions\(^4\). A bit simplified, the Adaboost algorithm loops over all possible rectangle placements (limited to two rectangles in the paper), and creates a classifier on each Haar feature alone. The best performing feature gets "boosted", and is then part of the classifier, which is a linear combination of all boosted Haar features.

**Cascade classifier**

Haar features are not of equal importance: the 10 most important features can detect almost all faces, but with a lot of false positives \[^4\]. To increase the processing speed of the algorithm, which is linearly dependant on the number of Haar features, these can be applied in stages. The first stage includes the most important Haar features, and subsequent stages prune the false positive samples that passed the earlier stages. The samples rejected at any stage are discarded, and won’t be processed by subsequent layers. A typical Haar Cascade classifier for faces has 1600 features spread over 40 layers.

\[^4\]See [https://stackoverflow.com/questions/1707620/viola-jones-face-detection-claims-180k-features](https://stackoverflow.com/questions/1707620/viola-jones-face-detection-claims-180k-features), the paper erroneously reported 180,625 possible positions.
5.2.2 Faster R-CNN

Faster R-CNN [39] is a successor of the original R-CNN research [16] and fast R-CNN [15], where the main difference is that Faster R-CNN uses a region proposal network (RPN) to suggest regions of interest, where previous versions used selective search for this task. The Faster R-CNN network consists of two parts: the mentioned region proposal network, and an object classification network, providing class predictions of the object. The RPN and the classifier share the first few convolutional layers, the resulting feature map after those first convolutions is what both components use as input.

The RPN uses so-called anchors to propose regions of interest. The anchors determine the standard size of the proposed bounding-boxes, in the paper they are at three different scales: 1:1, 2:1 and 1:2, and three different resolutions: 8, 16, 32px. The RPN then predicts the transformation of the anchors, and also predicts an ”object-ness” score: if the object is part of the foreground or background. As a last step of the RPN, the predicted anchors are thresholded on their object-ness score and the proposed regions are forwarded through a non-maximum suppression step, to reduce the number of redundant proposals.

The final classification network may vary. In the paper the authors chose VGG-16 as their base network, but any other, like ResNet, Inception, or a custom network is possible. Before these can be applied, there is one thing to note: since the proposed bounding boxes are of different sizes, they need to be resized to a common resolution, the method used in the paper is called region of interest pooling (ROI pooling). The ROI pooling layer divides the bounding-box into $n \times n$ regions, and outputs the maximum value of each region. The paper implements a $7 \times 7$ ROI pooling, so for each bounding-box, the classification network receives a length 49 vector.

5.2.3 RetinaNet

RetinaNet [32] is unlike Faster R-CNN a single stage classifier, merging the proposal and classification components into a single network. This often results in a trade-off in accuracy for speed, as shown in recent popular papers such as Single Shot Detectors [34] and YOLOv3 [38]. Training single stage classifiers is somewhat harder than their two-stage counterparts due to the greater imbalance of foreground and background region proposals made by each network\(^5\). Regular approaches to counter this problem are class rebalancing methods such as hard-positive or negative mining, skipping common samples, etc. An other option is to use smarter loss function, which

\(^5\)E.g. Faster R-CNN outputs 6000 boxes at a resolution of 1000 × 600, whereas SSD outputs 24,564 boxes at a resolution of 512 × 512.
this paper applies: focal loss. The focal loss is defined as:

$$\text{Focal Loss}(p_t) = -(1 - p_t)^\gamma \log(p_t) \quad (5.7)$$

Where $p_t$ is

$$p_t = \begin{cases} 
    p, & \text{if } a = 1 \\
    1 - p, & \text{otherwise} 
\end{cases} \quad (5.8)$$

For $\gamma = 0$, this is equal to binary cross entropy. At gamma values $\gamma > 0$, the factor $(1 - p_t)^\gamma$ reduces the loss output based on how sure the network was of that specific sample. This is especially useful in cases where the classes are imbalanced, such as with the single stage object detection networks.

The object detection and classification network used in the paper is a variant of the Feature Pyramid Networks [31] architecture combined with ResNet [19]. First, an input image is forwarded through a ResNet-101 network, at the end of the ResNet module the features are intermittently upsampled by interpolation\(^6\) and combined with features from earlier layers in a pyramid-like fashion, as shown in figure 32. The idea behind this is that the later layers are more semantically rich: contain of what there is in the picture, and the earlier layers contain the move visual information: where objects are in the picture. These features are combined by addition, and subsequently forwarded through a $1 \times 1$ convolution, before being sent to the region proposal module borrowed from Faster R-CNN.

\[^6\]This paper does not use transposed convolutions to prevent the checkerboard effect this type of module often produces. [https://distill.pub/2016/deconv-checkerboard/]
Chapter 6

Results

6.1 Path predictions

We train on a continuous batch generator, sampling paths and all the metadata in an online fashion. The validation and test sets are both generated once by their batch generators, and then kept as offline datasets for easier and fairer comparison. For both the validation and test set we generate 200 batches, where they contain 2202 and 2265 ship paths respectively. The batch generation, and the features in each batch, are described in section 5.1.8. Table 4 shows a quick recap of the features used per model.

<table>
<thead>
<tr>
<th></th>
<th>Temporal position</th>
<th>Linear displacement</th>
<th>Spatial position</th>
<th>Environment features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear extrapolation</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vanilla LSTM (a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vanilla LSTM (b)</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Social LSTM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Attention LSTM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Own method</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4: Overview of the feature usage per model.

We evaluate our methods based on two metrics: the mean absolute error, and the mean squared error. We trained on an adapted form of the mean absolute error (see eq. 5.2), the mean squared error is a good indication of
the consistency of each algorithm, since it punishes errors exponentially.

\[
\text{Mean Absolute Error}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{6.1}
\]

\[
\text{Mean Squared Error}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{6.2}
\]

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean absolute error</th>
<th>Mean squared error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear extrapolation</td>
<td>0.5614</td>
<td>1.406</td>
</tr>
<tr>
<td>Vanilla LSTM (a)</td>
<td>0.5418</td>
<td>1.334</td>
</tr>
<tr>
<td>Vanilla LSTM (b)</td>
<td>0.5493</td>
<td>1.408</td>
</tr>
<tr>
<td>Social LSTM</td>
<td>0.5455</td>
<td>1.391</td>
</tr>
<tr>
<td>Attention LSTM</td>
<td>0.5446</td>
<td>1.374</td>
</tr>
<tr>
<td>Own method</td>
<td>0.5250</td>
<td>1.301</td>
</tr>
</tbody>
</table>

Table 5: Mean absolute error and mean squared error of the tested algorithms on our test set. Our own method performs best, especially in the mean squared error metric. Although a reduction of 0.02 MAE seems relatively small, one needs to keep in mind that this is an average reduction per step. Over (potentially very long) paths this delta error is compounded, creating greater differences between algorithms.

6.2 Bounding box predictions

We evaluate our bounding-box predictions with four metrics: precision, recall, F1 score and the mean average precision. The mean average precision is calculated by ordering all predicted bounding boxes by their predicted probability. If the predicted bounding box overlaps more than 50% with a ground truth box, and the ground truth box was not positively predicted by an other bounding-box, it is marked as a true positive. The average over all recall levels as all bounding-boxes are evaluated is the mean average precision. The Viola-Jones algorithm does not include the mean average precision metric since it does not output probabilities.
Table 6: The number of training, test and validation cases in our radar bounding-box dataset.

\[
\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \quad (6.3)
\]

\[
\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \quad (6.4)
\]

\[
\text{Harmonic mean (F-score)} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (6.5)
\]

<table>
<thead>
<tr>
<th></th>
<th>train</th>
<th>validation</th>
<th>test</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>215</td>
<td>22</td>
<td>22</td>
<td>259</td>
</tr>
<tr>
<td>Number of bounding boxes</td>
<td>634</td>
<td>67</td>
<td>62</td>
<td>763</td>
</tr>
</tbody>
</table>

Table 7: The results from the various algorithms tested on the radar bounding-box dataset. Surprisingly Faster R-CNN (VGG-16\cite{44} based), performs best on the precision, recall and F1 scores, RetinaNet (ResNet-50\cite{19} based) has the best mean average precision, indicating that the class probability scores perform better in this model than in Faster R-CNN.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viola-Jones</td>
<td>12</td>
<td>6</td>
<td>50</td>
<td>0.667</td>
<td>0.194</td>
<td>0.301</td>
<td>-</td>
</tr>
<tr>
<td>Faster R-CNN</td>
<td>51</td>
<td>9</td>
<td>11</td>
<td>0.850</td>
<td>0.823</td>
<td>0.836</td>
<td>53.3</td>
</tr>
<tr>
<td>RetinaNet</td>
<td>43</td>
<td>135</td>
<td>19</td>
<td>0.242</td>
<td>0.694</td>
<td>0.359</td>
<td>61.0</td>
</tr>
</tbody>
</table>
Chapter 7

Conclusions

Path prediction

We noted with surprise that a simple LSTM model (a), with few inputs manages to beat all but our own algorithm on the mean absolute error metric. When considering the mean squared error as the deciding factor, the results are as expected: the more complex the model, the better it performs. Our own method performs best in both metrics, possibly enabled by the addition of the environment features (see sec. 5.1.7), which indicate the boats position, and where the fairway and river edges are, relative to its position.

Returning to our research sub-question: "Can we reliably predict a ship path using AIS information?". The answer is: yes, up to a certain degree. All neural network methods outperform the linear extrapolation, as expected, since they have the linear extrapolation as a baseline, and only predict displacements from the linear path. We do not deem the improvements large enough to create a fully autonomous ship (yet) from these models, but it definitely is a step towards that final goal.

Radar tracking

We tested three models in our radar bounding-box prediction problem: Viola-Jones, Faster R-CNN and RetinaNet. Our first tested model, the Viola-Jones algorithm, performs rather poorly. It only predicts a small amount of ships correctly, but at the same time it does not predict that many false positives, which compensates the low recall through a relatively high precision, therefore scoring second in the F1 score metric. Faster R-CNN scores highest in precision, recall and the F1 score. Retinanet’s precision is lowest of all tested methods due to the high number of false positives this model predicts. The somewhat higher recall compensates the F1 score slightly, scoring a second place in the F1 score metric. The fact that this model’s class prediction probabilities are better than Faster R-CNN con-
tributes to it scoring best on the mean average precision metric.

Our research sub-question for the radar tracking was: "How accurately can we predict bounding boxes for ships in radar images?". We deem the results for the bounding-box detection problem as quite good, but like the path prediction methods, insufficient for an inclusion into an autonomous shipping system. Ideally you would want a > 99% detection rate for such a system, which these models are not even close to achieving.

Answering our main research question: "How can path prediction and object localization contribute to autonomous shipping?". Employing our newly developed path prediction model reduced the mean absolute and squared error, but not significantly enough (yet) to create a fully autonomous ship. The object localization on radar images works quite well, but like the path prediction, it requires some additional improvements before being useful as a reliable sensor to detect nearby ships. Overall, we consider our research as a valuable step towards autonomous ships, but we are not there yet.
Chapter 8

Discussion

Path prediction

The simple LSTM model (a) performs best (except for our own method) on the mean absolute error metric. A possible explanation for this is that this model performs well on the cases where a boat keeps a straight line, or a constant turn, which is the case for the majority of samples. Due to the models inability to capture complex movements, it makes mistakes in those cases, for which it gets punished exponentially by the mean squared error metric, and thus scores lower there, although this needs some more investigation.

Overall, we saw a definite improvements from the linear extrapolation and simple LSTMs models, to the more complicated models. At the same time, reductions in both the mean squared error and the mean absolute error seem too small to be able to create a completely autonomous ship from the tested models (yet), although this is hard to determine since they are not tested in a live environment. Testing in a live environment is important because suggested routes by a model altering from the ground truth might be correct, but cannot be easily tested without knowledge of water levels, wind conditions, temporary traffic rules and other information which is not available in our offline dataset.

We recommend the gathering and inclusion of live river depth information, weather data and bridge and buoy positions as a possible next step in future research. Another possibility is to extract information from the maritime phone communication: shippers often interact over such systems to make their navigational intentions clear.
Radar tracking

Both Faster R-CNN and RetinaNet outperform the rather classical Viola-Jones algorithm, which is unsurprising since the latter cannot distinguish an object based on its context. This is important because radar images are quite noisy, there are often artifacts looking similar to a ship, while actually being an echo from an object on dry land. Both neural networks can determine some form of context from their anchor boxes, and deform them to match the ships bounding-box.

In comparison to the path prediction research, relative few time was spend on creating and testing the radar tracking models. As a result a lot of tunable parameters were left on their default settings, potentially missing out on easy performance gains. In future research, we would try different sizes of anchor boxes, eliminating the (almost) image-wide anchor boxes, and adding smaller versions. We would recommend doing experiments with a variable amount of radar images stacked as input, which was fixed at three images during our research, additional temporal information might be of use for the model. Another interesting thing to research would be to automatically annotate the radar images using the captured AIS data, which provides locations, headings and dimensions of ships. This would essentially create an infinite amount of training data, which we consider is an important limiting factor in our research.
Bibliography


57